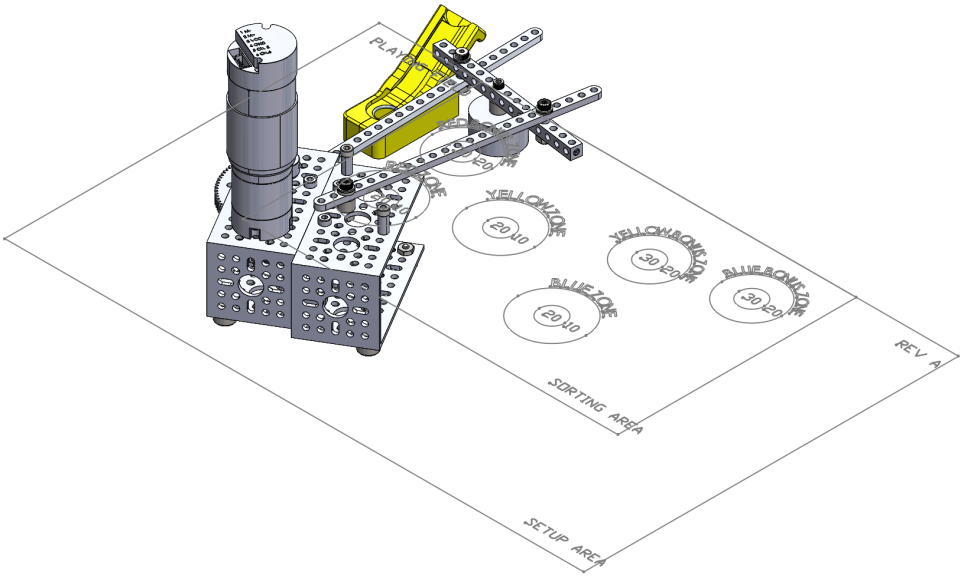
# ME350
# Fall 2021 Semester
# Final Report

Tanguy Dartus
Nathan Montgomery
William Wagner
Jason Williams

Instructors: Professor Michael Umbriac and Professor Chinedum Okwudire




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

*Tanguy Dartus*
*Nathan Montgomery*
*William Wagner*
*Jason Williams*

## Section 1: Design Goals and Requirements

### Design Requirements

Listed below are the hard design requirements for our linkage design for the pick and place project. The requirements for our team's design were generated based on the design and performance specifications given in the project description document. The requirements are listed below, and are additionally summarized in the requirements table below with minimum, maximum, and target values listed as needed for our team's design requirements.

1. Have a linkage transmission angle deviation as close as possible to 0 degrees.
2. Have a position response that fulfills the following specifications:
    a. Have an overshoot of less than 10% of the servo motor input step command value.
    b. Achieve a settling time that is less than 2 seconds.
3. Place discs within the circles of target zones as close as possible to the center.
4. Sort discs correctly based on color using the kit-provided color sensor and place each disc on the correctly colored target for the particular disc each time.
5. Maximize time remaining after all discs have been placed by the linkage in each round.

**Table 1.1:** Team design requirements and target, minimum, and maximum allowable values. These design requirements will directly impact our linkage design generation for the project.

| Requirement | Target Value or Measurement | Minimum/Maximum Values or Measurements |
|---|---|---|
| Transmission Angle Deviation | < 20 degrees (minimize) | 0 degrees/60 degrees |
| Settling Time | < 2 seconds (minimize) | 0 seconds/2 seconds |
| Overshoot | < 10% of step command value (minimize) | 0%/100% |
| Distance from Dropped Disc Center to Target Center | < 5 mm from target center (minimize) | 0 mm/250 mm |
| Disc Sorting Success Rate | 100% success rate (maximize) | 0%/100% |
| Round Time Remaining | > 10 seconds (maximize) | 0 seconds/90 seconds |

### Design Goals

The goals below represent soft specifications for our team's linkage design for the project. These goals do not have hard quantitative values that must be met, but are nonetheless still important to follow and will positively influence our team score and project performance the closer they are followed. These goals will help to narrow down future design options and further refine our final linkage design.

1. Design and build the mechanism to have predictable motion and behavior.
2. The mechanism should be able to run the desired amount of time for all rounds without significant decreases in performance occurring; it must be resilient to fatigue.
3. Minimize friction at linkage joints through careful joint design and built-in adjustments as needed.
4. Minimize play in linkage joints and unnecessary mechanism mass with precise and justified construction.
5. The mechanism should have a clean and professional appearance when designed and built.

6. Arduino code used for operating the mechanism should be understandable when read by others outside of the group; it should be organized, commented, and readable.

**Restrictions in Design**

Described below are key design restrictions that our team must consider for any linkage design. These restrictions include key competition rules for the project. Also included are rules that are needed to ensure proper team member health and safety during mechanism building, testing, and competition.

1. All project components (color sensor, feeder ramp, and mechanism) must lie within the designated setup area for the competition. The first figure below shows the full playing field, including the setup area and sorting area with targets. The second figure below shows the premade disk feeding ramp object that our linkage must collect disks from while both are placed in the setup area.

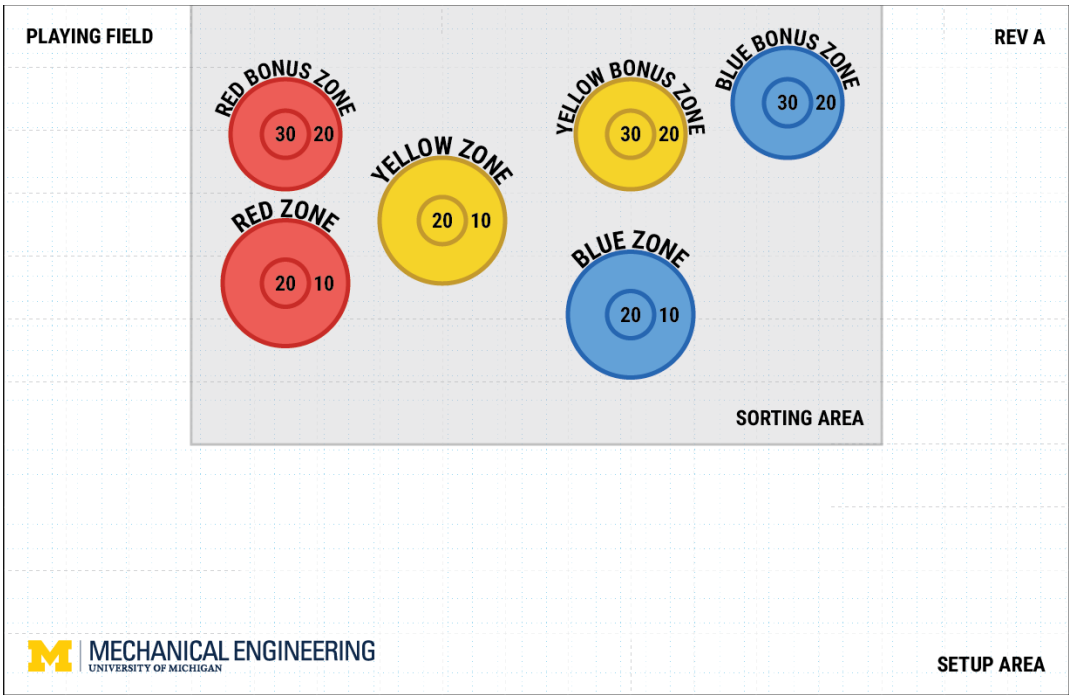



**Figure 1.1:** Project playing field (top view, 17” x 11” in size). The setup area where project components can be placed surrounds the sorting area (with the disc targets) on three sides.

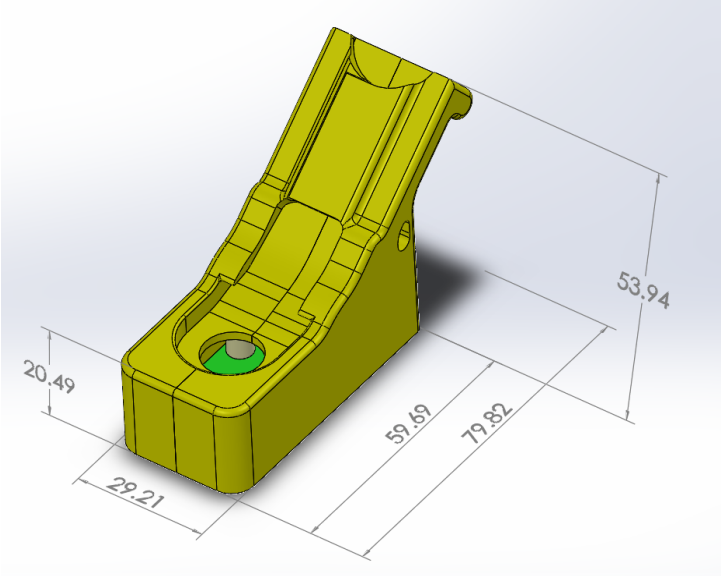



**Figure 1.2:** Integrated disc feeding ramp. This object is premade and must be placed by our team within the setup area such that our mechanism can properly grab disks from it.

2. The mechanism should contact a hard detail on both ends of its motion to provide a repeatable limit to its motions and to prevent overshoot of the mechanism.
3. The maximum voltage that can be supplied to the assigned mechanism motor is 12 V.
4. The mechanism motor must be securely mounted to the ground link, and adequate space for the motor and transmission must be provided.
5. Our mechanism's transmission must be built using only the gears and other mechanism parts provided in our project kit.
6. Our mechanism in its entirety must be built using only kit provided parts, additional outside parts for our project may not be purchased or used without explicit instruction from course staff.
7. The mechanism linkage must move to contact a limit switch in order to calibrate the encoder count when setting initial position.

## Section 2: Design Selection

### Selection Criteria, Weighting Scheme, and Values Assigned in Pugh Chart

In this section, we as a team decide which linkage design we will choose for our final project design. For this selection process, we started with deciding key criterion to judge our designs against, assigning weights to each criterion based on their importance. We then determine a weighted performance for our designs using a pugh chart. This finally allowed us as a team to make an objective decision on which design was best to use for the linkage in our final project.

The weighting scheme we chose to use for our criterion was from a 1 to 5 scale, with 1 being the least important and 5 being the most important criterion. We used this scale since it makes sense for the 10 different selection criteria we have. This scale made it easy for our team to agree on a few criteria with top 5 and lowest 1 ratings, with most of the others falling in the middle of the scale. The weight for each criterion is later used in our pugh chart to calculate weighted performance averages for each design.

Criterion and weight reasoning:

- **Transmission Angle Deviation**: The transmission angle deviation is the angle difference between coupler and the follower link. The reason that the transmission angle is important is that it affects the efficiency of the mechanism. A lower magnitude of transmission angle deviation leads to more effective force transfer through the linkage and better efficiency.
    - We have a transmission angle deviation at a value of 3/5 because it heavily impacts the effectiveness of the torque on the coupler.
- **Accuracy of Hitting Target Zones:** The accuracy of hitting the target zones is the deviation of the middle of the disk to the middle of the zone. It is very important that the middle of the disk is as close as it can get to the middle of the zone circle because it will give you more room for error in the mechanism.
    - The accuracy of hitting target zones has a weight of 4/5 because the most points are created when the center of the disk is in the circle of the higher zone.
- **Required Power:** The required power of the mechanism to function at the specific theta range and acceleration of the linkage. The required power has to be under the given 1 watt limit. We took a ratio between the angular acceleration and the time of rotation to determine the linkage power requirement and thus find the linkage that is the most effective at moving (accelerating quickly) while staying under the power limit of 1 W.
    - The required power has a weight of 2/5 because while important, the values were reasonably similar between the designs.
- **Linkage Inertia:** The linkage inertia will affect the accuracy of our motions. If it is too big we will need to stop the power supply earlier to counterbalance that inertia hence making controlling the mechanism pretty hard. The linkage inertia is related to the length of the linkage and their positioning to one another.
    - The linkage inertia has a weight of 2/5 because the linkage inertia is a main factor in the required power, time of motion and the friction in the joints, but isn't directly related to scored points in competition.
- **Total Required Range of Motion:** The total required range of motion is the angle of the linkage between the pickup location and the last colored zone of the motion. This will affect the time of the motion, a shorter angle means a faster motion time.

    - The total required range of motion has a weight of 3/5 because the total required range of motion affects the time of rotation and the required power of the linkage.
- **Friction:** Friction will impact the power required to realize the motion. Bigger friction means for the same amount of power the motion will be slower hence it is important to lower them as much as possible.
    - Friction was given a weight of 2/5 because friction is the main factor in determining the actual efficiency of the linkage, but this also isn't likely to vary much between designs. Friction is also difficult to determine without experimentation and thus is something we can only roughly estimate as a team. We can estimate it based on whether more large parts are present near joints or rotation points, as this would cause higher friction due to added weight.
- **Part Count:** The part count won't affect our final choice but the fewer parts we have the easiest it would be to control it and predict its behavior.
    - Part count was given a weight of 1/5 because it does not affect the effectiveness of the linkage, only the ease of assembly which is less of a concern.
- **Ease of Assembly:** The ease of assembly is something we take into account but isn't really important for the final choice so we assigned a low weight to it.
    - Ease of assembly was given a weight of 1/5 because it does not affect the effectiveness of the linkage, only the assembly time of the linkage which is not limited by any direct rules. We can roughly measure this by looking at the number of parts used and how easy it is to reach certain joints for tightening, etc, in a design. More parts reduced ease of assembly, as well as joints or attachment points being located in tight spaces that are hard to reach and build/adjust within the metal bracket parts.
- **Number of Bonus Zones Hit:** Determined by the number of bonuses the electromagnet moves over. The more the better.
    - Number of bonus zones hit was given a weight of 2/5 because the number of bonus zones affects the total number of points scored by the linkage.
- **Time for Full Range of Motion**: The time for full range of motion is important because it translates the efficiency of the mechanism.
    - The time for full range of motion was given a weight of 4/5 because it will affect the total power required and the amount of time left after the sorting of the disks is done, this also affects how quickly disks can be picked up and dropped to score points. For the values of total power required for each linkage we looked at the ADAMS analysis results. These values can be found in table 2.2 later below.

### Pugh Chart Results and Individual Analysis Results

In accordance with a typical pugh chart process, for each criteria an initial performance is assigned for each design with 0 meaning average design performance, 1 meaning better than average performance, and -1 meaning worse than average performance. These unweighted performance values are multiplied by the corresponding weight for the criterion for each design as specified above. These now weighted performance marks are then finally tallied up for each design. This results in weighted performance totals, where the highest total represents the best performing design based on the needs and considerations of our team. The table below is a full description of our pugh chart, with the criterion and weights, and the unweighted and weighted design performance totals.

**Table 2.1:** Pugh chart comparing different design ideas; the total results show that Jason's design is the most effective for our team needs.

| Quantitative Measure | Weight (4 = Most Important, 1 = Least Important) | Team Member Linkage Design Performance (1 = Better Than Average, 0 = Neutral, -1 = Worse Than Average) | | | |
|---|---|---|---|---|---|
| | | Jason | Will | Tanguy | Nathan |
| Transmission Angle Deviation | 3 | 1 | 0 | 0 | 1 |
| Accuracy of Hitting Target Zones | 4 | 0 | 1 | 0 | 0 |
| Required Power | 2 | 0 | 0 | 0 | 0 |
| Linkage Inertia | 2 | 1 | -1 | 0 | -1 |
| Total Required Range of Motion | 3 | 1 | 0 | 1 | -1 |
| Friction | 2 | 1 | 1 | -1 | -1 |
| Part Count | 1 | 0 | 0 | 0 | 0 |
| Ease of Assembly | 1 | 0 | -1 | 0 | 0 |
| Number of Bonus Zones Hit | 2 | 0 | -1 | 0 | 1 |
| Time for Full Range of Motion | 4 | 1 | 0 | 0 | -1 |
| **Weighted Performance Totals** | | 14 | 1 | 1 | -6 |

In order to determine the unweighted linkage design performance values in the above pugh chart (0, 1, -1) for each team member's design, the real values for certain criteria of each design had to be calculated and compared across all of the designs. These real values for different criteria are organized in the table below. These real values allowed for an average performance to be gauged. This then allowed us to properly say whether a particular team member's linkage design performed with, better than, or worse than, the average for all of our team's designs.

**Table 2.2:** Individual analysis results; the determined numerical values were used to determine scores in the pugh chart.

| Individual Analysis Quantity | Team Member Linkage Design Values | | | |
|---|---|---|---|---|
| | Jason | Will | Tanguy | Nathan |
| Maximum Transmission Angle Deviation (degrees) | 39 | 53 | 55 | 43 |
| Required Power Via ADAMS (W) | 1.00 | 1.00 | 0.85 | 0.70 |
| Input Link Length (mm) | 136 | 120 | 160 | 168 |
| Follower Link Length (mm) | 128 | 120 | 120 | 168 |
| Coupler Link Length (mm) | 72 | 96 | 72 | 64 |
| Total Required Range of Motion (degrees) | 63 | 80 | 65 | 105 |
| Number of Bonus Zones Hit (#) | 1 | 0 | 1 | 2 |
| Sweep Time for Full Range of Motion (s) | 0.29 | 0.54 | 0.32 | 0.46 |

**Final Design Winner**

Following our full selection process, Jason's design won as the final design choice because it was objectively the most effective in the most performance categories with high importance weights. This correlated to the highest pugh chart score. This can be seen in the pugh chart table above since Jason's design had the highest weighted performance total by far in our completed pugh chart.

Some elements of Jason's design that made it more effective and made it a good choice for our final linkage design include that the linkage reaches one bonus zone and hits it with above average accuracy. This design also has the best transmission angles of the group and minimizes the time required for the full range of motion. The inertia of Jason's linkage was also fairly good compared to other designs. Overall, Jason's design outperforms those from the other team members by a fair margin. We as a team agreed that it is the best choice to use as our final project design moving forward. We are overall very confident that this design will serve us well in the final competition.

## Section 3: Final Linkage Design and Analysis

**Final Design Characteristics and CAD Model Analysis**

In this section, we discuss our final linkage design and go in depth on the expected results we will get using it. We will look at what it achieves and compare that to the requirements of the project. We analyze much of these characteristics using the full CAD model of our final design, which includes all parts and joint assemblies. The tables below show the various geometric properties of our final linkage design as measured from our final design's CAD model.

**Table 3.1:** Transmission angle for each position.

| Metric | Pickup location | Red zone | Yellow zone | Blue zone |
|---|---|---|---|---|
| Angle (μ, degrees) | 116 | 93 | 71 | 51 |
| Deviation (\|90 - μ\|) | 26 | 3 | 19 | 39 |

**Table 3.2:** Final link lengths. The length limit values are maximum lengths that each link can be given the limited size of link parts available to use for building in our kit.

| | Input | Coupler | Follower |
|---|---|---|---|
| Length (mm) | 136 | 72 | 128 |
| Length Limit (mm) | 176 | 120 | 176 |

**Table 3.3:** Ground pivot locations relative to bottom left corner of the setup area. The x-coordinates are for horizontal distance and the y-coordinates are for vertical distance, relative to the bottom left corner of the field in a top down view.

| | X-coordinate (mm) | Y-coordinate (mm) |
|---|---|---|
| Input ground pivot | 115.81 | 92.75 |
| Follower ground pivot | 161.79 | 92.75 |

From Table 3.1 above, the maximum transmission angle deviation is 39 degrees. The goal is to minimize this value with the maximum allowable value being 60 degrees. This design does a decent job at achieving this goal. From Tables 3.2 and 3.3, we can conclude that because the ground pivots are so close to the sorting area, it allows the link lengths to be relatively short. This can reduce inertia and improve the speed of the sorting process.

Figures 3.1 through 3.5 below show the linkage at each hard stop as well as over each of the zones. The hard stops are used to avoid having a larger range of motion than that necessary to pick up the disks and place them. This minimizes time loss if the power control isn't calibrated perfectly. They are also used for safety - they ensure the mechanism is constrained and will not become dangerous.

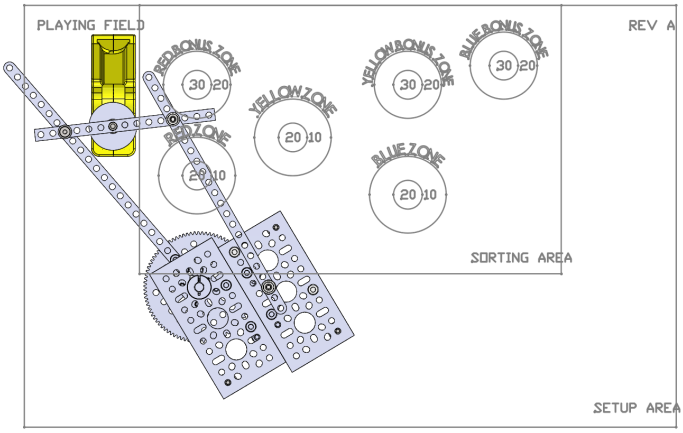



**Figure 3.1:** Mechanism over pickup location and at hard stop.

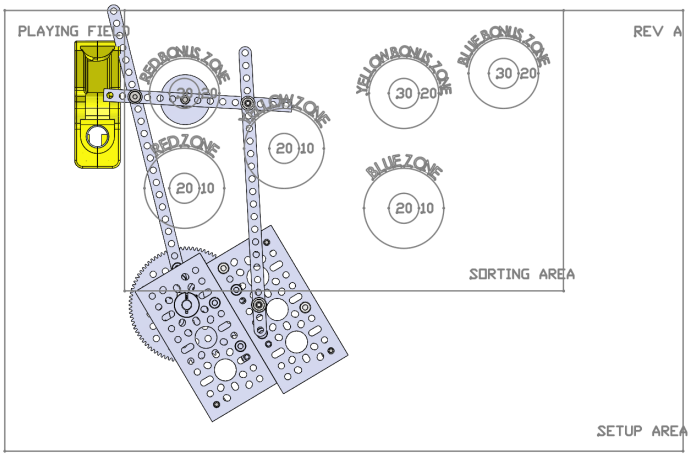



**Figure 3.2:** Mechanism over red bonus zone.

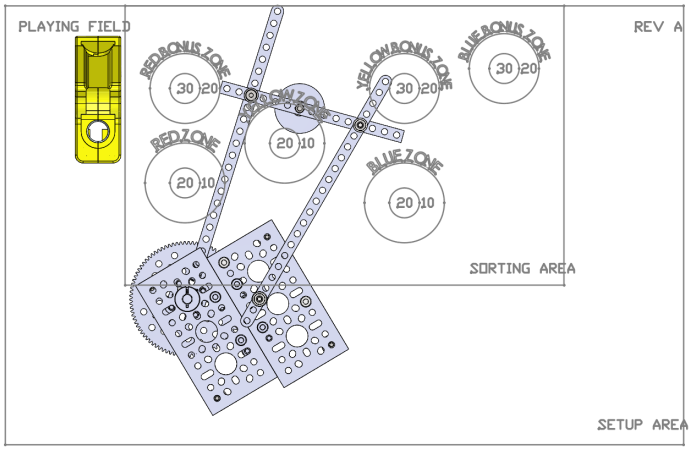



**Figure 3.3:** Mechanism over yellow zone.

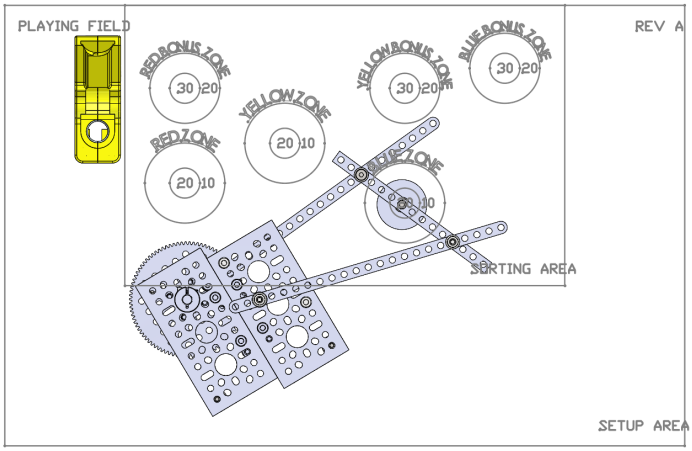



**Figure 3.4:** Mechanism over blue zone.

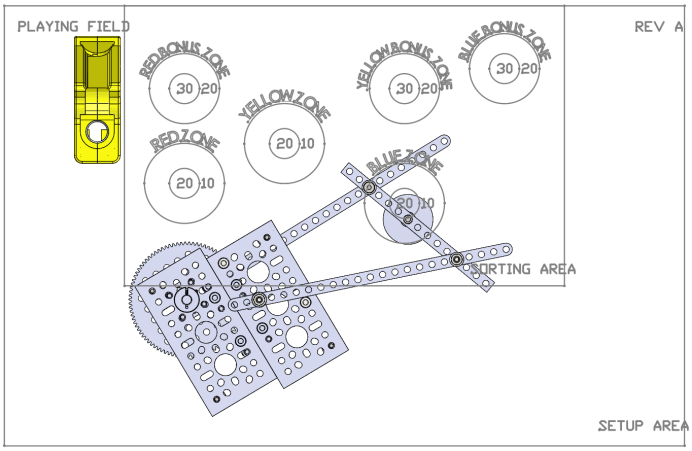



**Figure 3.5:** Mechanism at second hard stop.

Looking at our design, there are a few ways we can make it better. First of all, moving the disc feeder to the right, while still touching the setup area, will reduce the angle of motion and make the motion faster. We could also adjust the location of the ground pivots to ensure the discs are always dropped in the zone. With the

current design, the yellow discs will be dropped on the edge of the yellow zone which could cause some of the later discs to slide out of the zone.

Figure 3.6 below shows an isometric view of our main linkage design. As can be seen in the figure, the base for our linkage design is formed from taking two of the large “U” shaped metal brackets and placing them side by side the long way. This gives our linkage a sturdy square shaped base that is able to support all of the needed gears and joints for our linkage. The main driving gear for our input link is used to mount the link. The gear is attached to a shaft with a set screw hub, so both the hub and input link are securely mounted to the gear. This entire assembly is connected via friction to the input shaft. That shaft itself is attached to the base with two supporting bearings for low friction rotation, and a cylindrical clamp to control its height. Lastly, the construction for our hard stops can be seen as long screws attached to the base top with smooth metal cylinder coverings. This gives the hard stops a smooth and sturdy surface for the linkage to hit against as needed during movement for proper constrainment. Figure 3.7 shows a cross section view of the joint construction used between the coupler, input, and ground links. The use of the washer in between the links reduces the friction between them. It is also secured from top and bottom to ensure equal distribution of the reaction forces. This type of joint allows the mechanism to move easily while reducing the free play and friction. When you use plastic washers there should not be enough force between the screw and washer because it would cause the washer to deform and defeat the purpose of the washer. Plastic washers are mostly used for rotational joints. Metal washers are used when there needs to have more thrust force than the plastic washer can handle.

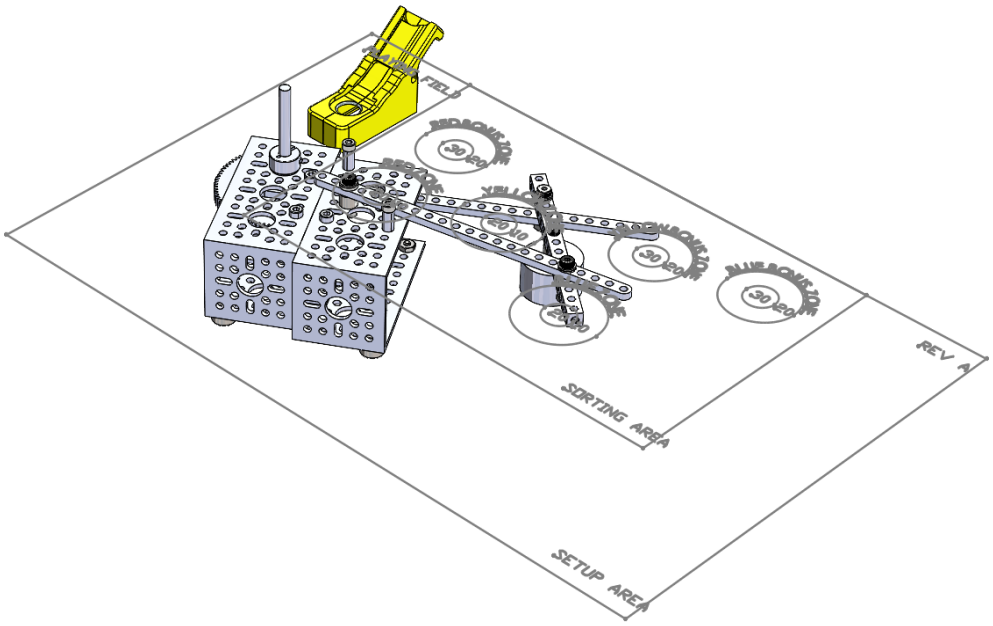



**Figure 3.6:** Perspective view of the final linkage mechanism and setup area.

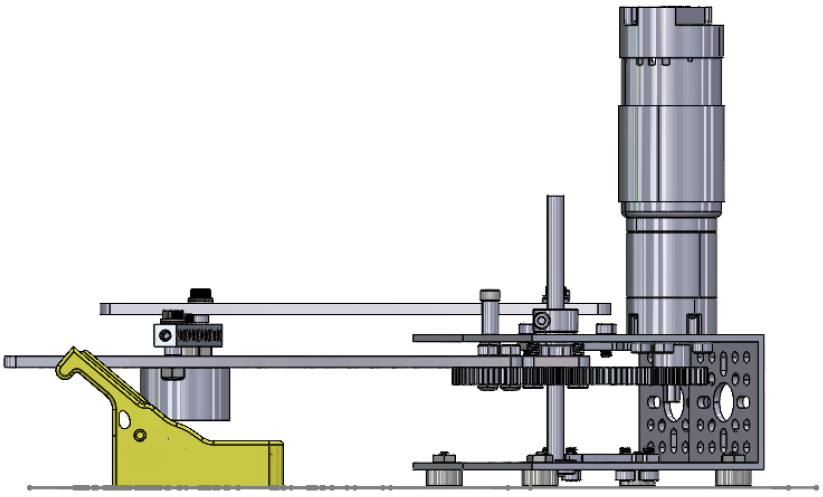


**Figure 3.7**: Side view of the final linkage mechanism.

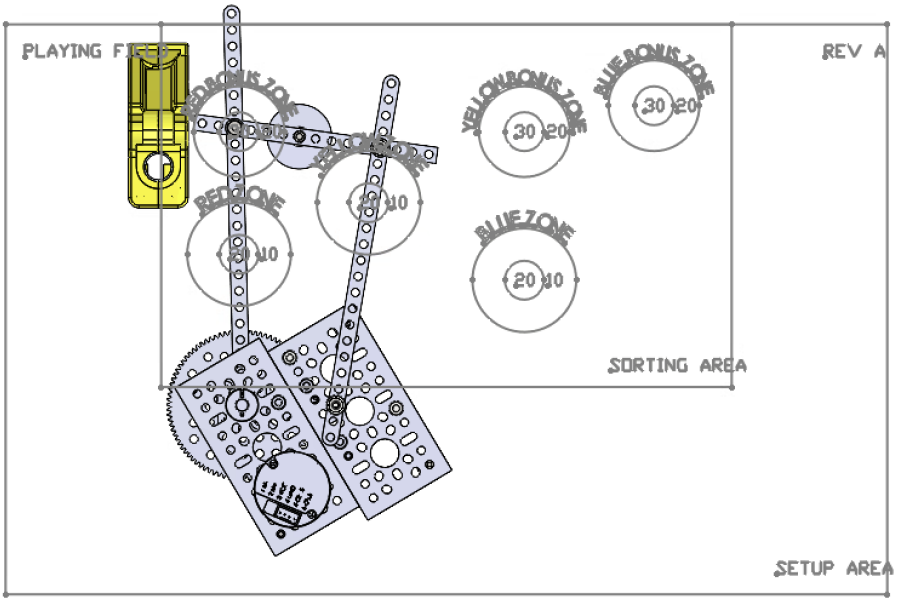



**Figure 3.8**: Top view of the final linkage mechanism and setup area.

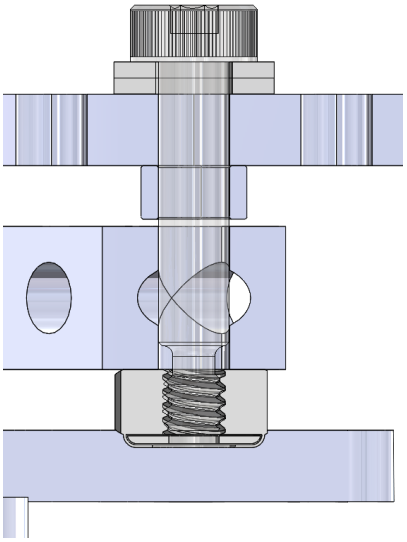


**Figure 3.9:** Cross section view of input to coupler and coupler to follower joints.

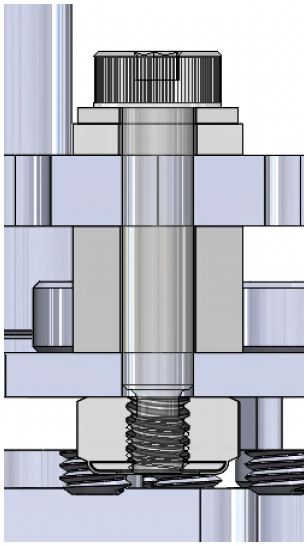


**Figure 3.10**: Cross section view of the follower to ground joint.

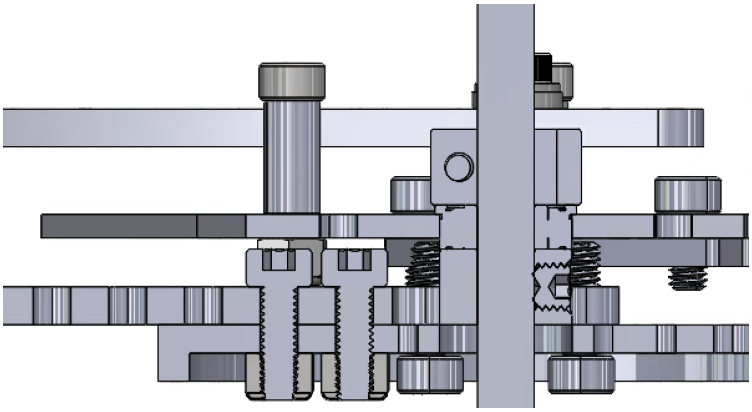


**Figure 3.11**: Cross section view of the input to ground joint.

**ADAMS Analysis**

We used the software ADAMS to conduct a dynamic simulation on the four bar linkage design. Figure 3.12 shows the mechanism imported into ADAMS with only the major components. Figures 3.13 through 3.16 show the results of the simulation. We attempted to minimize the time of motion by getting the maximum amount of power from the motor. We are allowed to use up to 1 W of power. The acceleration profile chosen was that of constant acceleration for half of the time followed by constant deceleration. Using the ADAMS simulation tool and reducing the time in increments we end up having a time of 0.29 seconds for the full range of motion. This time will differ from that of the real mechanism because ADAMS does not take into account the friction in our design or the weight of the components not included.

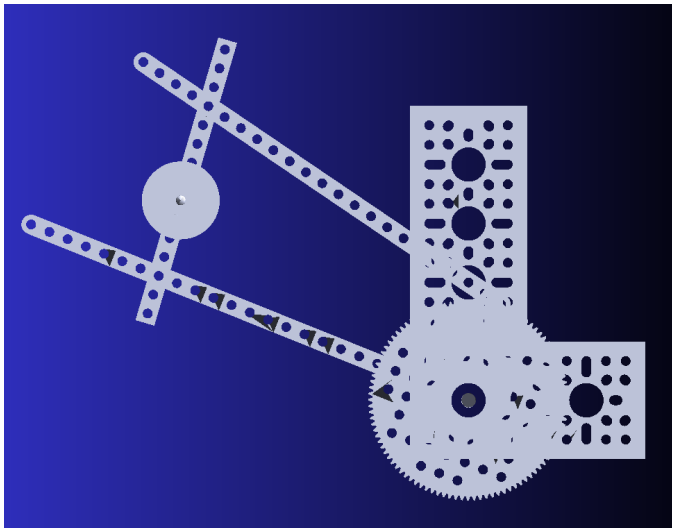


**Figure 3.12:** Linkage design in ADAMS, note that the base seen differs from our final design. The linkage shape used and analyzed is identical to that of our final design, and is the only thing considered in the analysis.

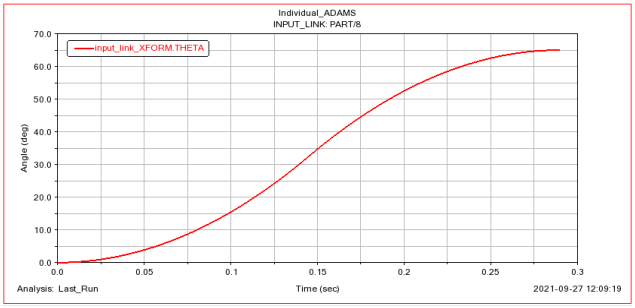



**Figure 3.13:** Angular position of input link.

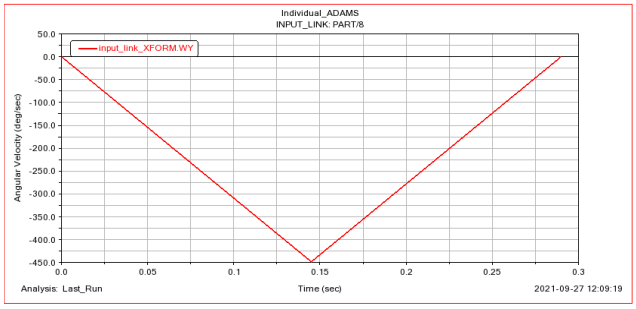
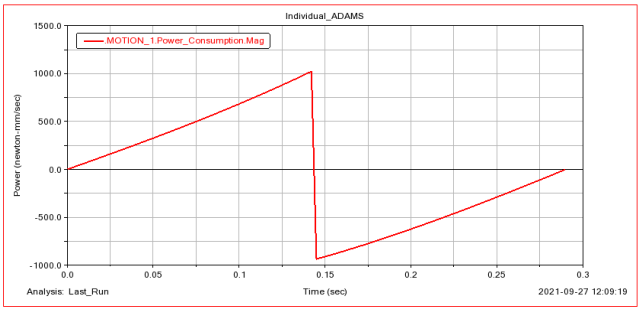



**Figure 3.14:** Angular velocity of input link.



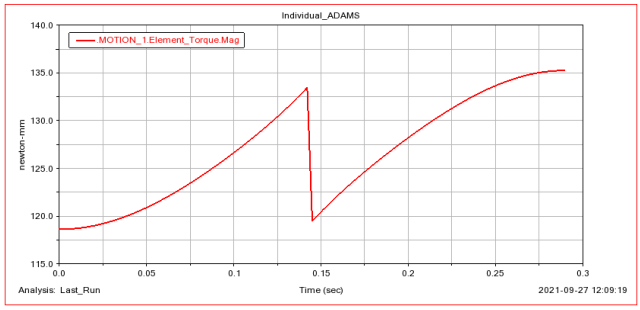



**Figure 3.15:** Power of motor.

**Figure 3.16:** Input torque of motor.

The behavior of our fully assembled linkage mechanism so far aligns well with our expectations from the ADAMS analysis, however these results can be expanded on to become more certain. Our linkage assembly thus far for our final design was made without any electronic components added. The linkage can be turned by hand only. However, our linkage was found to move very smoothly when the input axle was spun by hand. As such it can be fair to preliminarily say that friction should not affect performance too much for our linkage. We are not yet sure of how much motor power or input torque our real built linkage will exactly require. However, we can again say that they should align well with our ADAMS results since our linkage thus far is well constructed. ADAMS does not consider friction or extra weight due to gears or joints connected to the linkage, so we can expect the real power and torque requirements to definitely exceed the ADAMS results. Despite these issues, when turning our linkage we did not see any large spikes in friction or angular velocity, and the linkage smoothly swept over its full motion range. This is indicative of our good transmission angle deviation for our final design. Overall, we expect our ADAMS results to be good baseline predictions of our real linkage performance and behavior as we move into adding electronics to our linkage assembly.

## Section 4: DR 2 Design Updates

Once the final design from DR 1 for our linkage mechanism was built we tried running a series of basic motion tests on the linkage, both without the motor then later with it. We as a team were satisfied by the results we could achieve with our design. The mechanism was moving the electromagnet point accurately over the intended areas of the different field zones. As such we decided to keep our design the same way it was at the end of our DR 1 work. We did not make any significant changes to our design. The only minor adjustment we made to our design as part of lab work and testing was to move the disk pickup location slightly to better align with the linkage sweep path. This change only involved moving the disk ramp location slightly. This change did not affect any other design features/placement or behaviors of our linkage. That change instead should only improve competition performance slightly by reducing the chance of errors when picking up disks with the electromagnet.

In addition to maintaining our final linkage design from DR 1, we also made edits to the report sections completed for DR 1. These sections (1 through 3) are all included above at the beginning of this report, while the appendix section (appendix A) is found at the end of the report with the rest of the appendices. As specified by our GSI, required text and formatting changes were made to improve our report. All of the changes to text we made in our DR 1 content is highlighted in blue colored text.

## Section 5: Transmission Ratio Determination

In order to determine the gear ratio of our mechanism, we used the inertia matching as well as the resolution of the electromagnet. We will compare the results we get from both methods and decide which one fits the best for our mechanism.

The concept of inertia matching is trying to minimize the motor torque by selecting an appropriate transmission ratio.

In order to find the motor torque we first need to calculate the total moment of inertia:

$$I = I_m + \frac{I_L}{N^2}$$

With I the total moment of inertia, $I_m$ the moment of inertia of the motor, $I_L$ the moment of inertia of the load, N is the transmission reduction.

We can then calculate the angular acceleration of the motor.
The angular velocities of the motor and the load are related by the transmission ratio:

$$\omega_m = N * \omega_L$$

Deriving it will give us the acceleration of the motor:

$$\alpha_m = N * \alpha_L$$

Having the acceleration we can determine the start-up torque of the motor:

$$T = I * \alpha_m = (I_m + \frac{I_L}{N^2}) * N * \alpha_L$$

Solving for N give us:

$$N = \sqrt{\frac{I_L}{I_m}}$$

Applying it with inertia matching we can find the transmission ratio using the formula below:

$$N_{Inertia\ Matching} = \sqrt{\frac{I_{total}}{I_m}}$$

To use that formula we need to determine $I_{total}$ the total inertia of our mechanism, it depends on every part that is included in it. To do that we can sum the kinetic energies of all of them as shown below:

$$\frac{1}{2} I_{total} \omega_{input}^2 = \frac{1}{2} I_{input} \omega_{input}^2 + \frac{1}{2} I_{coupler} \omega_{coupler}^2 + \frac{1}{2} I_{output} \omega_{output}^2$$

Using the the parallel axis theorem and the constraints on angular velocity from the instant centers we can rewrite it as:

$$\frac{1}{2}I_{total}\omega_{input}^{\ 2} = \frac{1}{2}I_{input}\omega_{input}^{\ 2} + \frac{1}{2}I_{coupler}\left(\frac{R_1\omega_{input}}{R_2}\right)^2 + \frac{1}{2}I_{output}\left(\frac{R_1R_3\omega_{input}}{R_2R_4}\right)^2$$

Where we can calculate the different inertia as below:

$I_{input} = I_{input\ CG} + m_{input}(\frac{R_1}{2})^2$

$I_{output} = I_{output\ CG} + m_{output}(\frac{R_4}{2})^2$

$I_{coupler} = I_{coupler\ CG} + m_{coupler}(R_5)^2$

Where R1~5 are as shown below

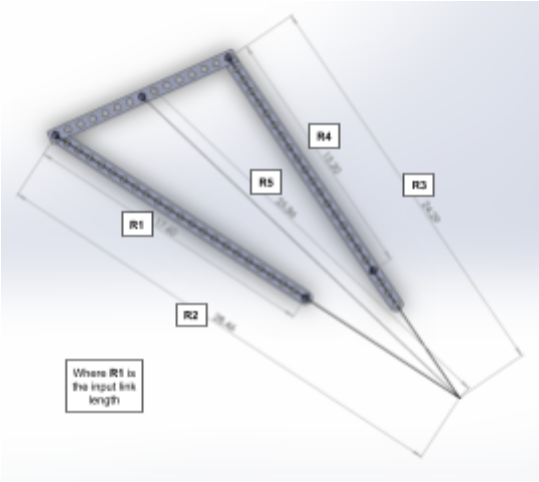



**Figure 5.1**: Sketch overlay for R1~R5 measurements where R1 is the input link. R5 is the distance between the rotation axis and CG of the coupler.

In order to find all those values we use Solidworks and the CAD of our mechanism. R1 and R4 are fixed but R2,3,4 depends on the CG hence they will be different at each zone.

$I_m$ depends on the characteristics of our motor, we can find it using a specific procedure. We didn't do it during the lab, the value is given $I_m$ = $5213.61\ g \cdot cm^2$

We can then apply the formula. We summarized the results in Table 5.1 below.

**Table 5.1:** Optimal transmission ratio for each zone location.

| | | Pickup Zone | Red Zone | Yellow Zone | Blue Zone |
|---|---|---|---|---|---|
| **Transmission Angle** | | 116 | 93 | 71 | 51 |
| **Instant Center Radii (mm)** | R1 (input link) | 13.60 | 13.60 | 13.60 | 13.60 |
| | R2 | 41.00 | 40.20 | 30.44 | 16.60 |
| | R3 | 38.80 | 39.43 | 32.25 | 20.33 |
| | R4 (output link) | 14.40 | 14.40 | 14.40 | 14.40 |
| | R5 | 40.20 | 39.78 | 30.87 | 17.61 |
| **Total Inertia of the Mechanism** | | 22147.18999 | 22589.08757 | 23885.53862 | 26880.45698 |
| **Transmission Ratio** | | 2.061057202 | 2.081517536 | 2.140416297 | 2.270644021 |

We are going to use the biggest transmission ratio hence $N_{Inertia\ Matching}$ = 2.270644022.

We also need to take into account the resolution of our electromagnet to choose our gear ratio. A large transmission ratio will make our mechanism more precise (neglecting backlash), since the same amount of motor rotation creates less displacement of the electromagnet.

We pick a resolution that gives a fine resolution because we use numerical derivatives to calculate the velocity of the mechanism. The equation that follows takes into account the encoder resolution, gearbox transmission ratio, and electromagnet deviation from center.

The resolution of the encoder on the motor is 28 pulses per revolution (PPR) with quadruple decoding on the motor shaft. Each revolution after the gearbox yields is equal to 28*13.7 = 383.6 encoder counts, hence we have 360/383.6 = 0.938 deg on the output shaft. We then need to divide this by the external transmission ratio N to have the angular resolution of the output link.

We introduce the desired count range which is a value that represents the number of encoder counts our linkage can move and still have the electromagnet over a given zone. For this project, it was determined it should be equal to 30.

Taking everything into account we have the following equation:

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

$$30\ = \left(\frac{28\ counts}{1\ rev}\right) \cdot 13.7 \cdot \left(\frac{13.95}{\frac{360\ degrees}{1\ rev}}\right) \cdot N$$

$$N = \frac{30}{\left(\frac{28\ counts}{1\ rev}\right) \cdot 13.7 \cdot \left(\frac{13.95}{\frac{360\ degrees}{1\ rev}}\right)}$$

$$N = 2.01823$$

We find that the external gear ratio using encoder resolution should be 2.018231357. (see calculations on the last page for the value)

Using the 90 tooth gear we selected in our design we are able to achieve a gear ratio of 3:1 or 4.5:1 depending on the pinion we choose. Those values are above what is necessary according to the results we get from the inertia matching and electromagnet resolution. We will decide to take the one which is the closest for our needs hence we decided to take a 3:1 ratio.

The results we found for our gear ratio aren't exact and are approximated because we used CAD to find them. This includes estimating the weight of each component using material properties and size instead of actually weighing them. The material chosen was an estimate as well and not the actual component material. In reality, the difference in weight might have a large effect on the gear ratio where a larger weight would result in a larger required gear ratio.

In the calculations we also didn't take into account the friction factors, these will lower the speed of the linkages in the mechanism hence increasing the inertia and the gear ratio.

## Section 6: Final Transmission Design

The final transmission design developed and used by our team for the project was the last main system of our linkage mechanism that was designed and assembled. The features of our transmission are detailed in the figures below. The transmission for our linkage uses the 30 tooth pinion attached to the shaft of the motor via set screw. The pinion then meshes with the 90 tooth gear which the input link is attached to. The gear is attached to the gear shaft using a set screw in a screw hub which is itself attached to the gear with four 12mm M4 socket head screws. The input link is attached to the gear with two 5mm M4 socket head screws. The set screw restricts gear motion in the vertical direction while the gear shaft is restricted along the horizontal directions via two flanged ball bearings at the top and bottom of the base. These ball bearings allow for rotation along the vertical axis of the gear and by extension the input link. The motor itself is attached to our linkage base using four 12mm M4 socket head screws.

The height of the gear and pinion on their respective shafts is adjustable by way of loosening each set screw and moving them up or down before tightening again. This ensures that the teeth are initially meshed and the heights are in line, leading to maximum efficiency between the pinion and gear. In our design, torque is transmitted from the motor to the pinion, gear, and finally from the screws on the gear to the input link then to the rest of the linkage joints. Images of the final CAD design with added transmission ratio can be found below in Figures 6.1-6.3.

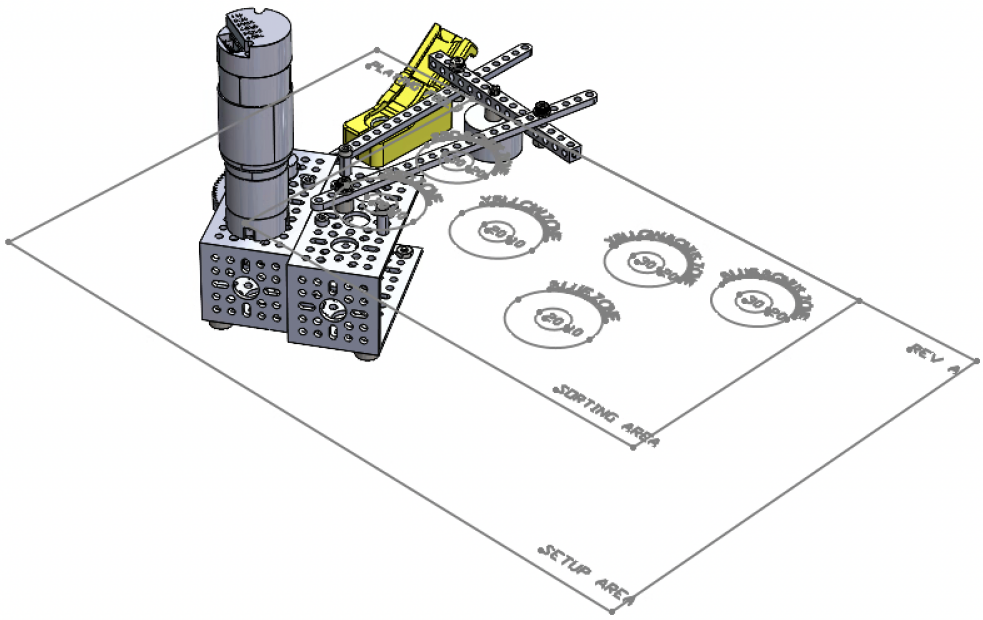



**Figure 6.1:** Isometric view of final CAD design.

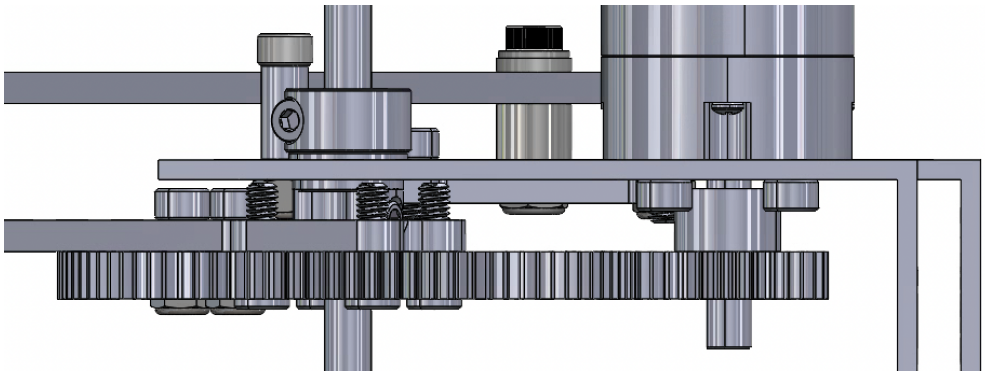


**Figure 6.2:** Side view of transmission design.

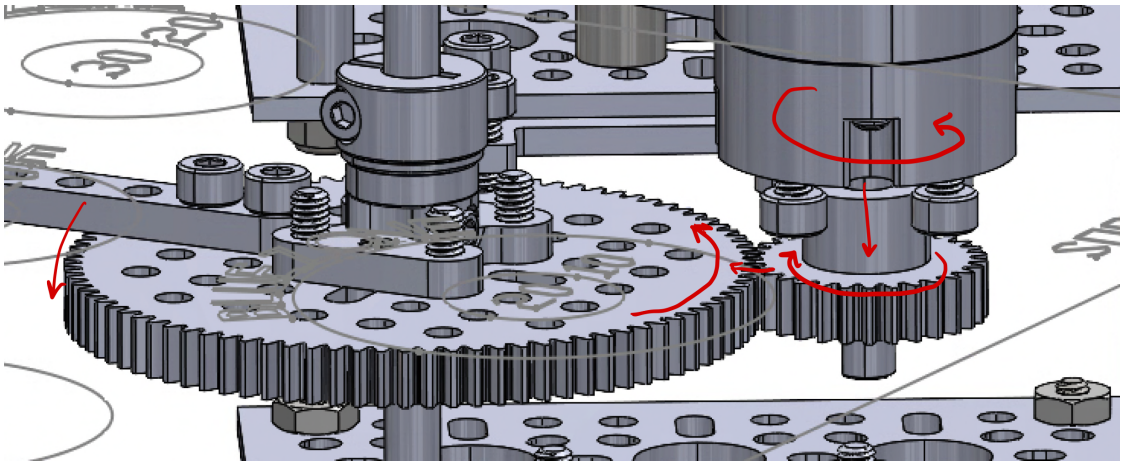


**Figure 6.3:** Isometric view of transmission design (red arrows show torque transfer).

## Section 7: Power Analysis

In order to determine an accurate and precise estimate for the optimal time of motion of our linkage mechanism as it makes a full sweep across the playing field, we conducted a thorough power analysis of our built linkage design. This involved accounting for the base statics and expected dynamics of our linkage mechanism as it is driven by our kit-provided motor. Our motor receives input electrical power that is translated into output mechanical power that acts to move the load of our linkage mechanism. Motor power is translated by our linkage transmission into finally moving the arms of our linkage to complete the task of picking and placing the colored disks as desired.

To start our power analysis, we organized a table of variables and formulas that would be required to go from a set of known static parameters for our motor, transmission, and linkage build to determining the expected motion behavior of our linkage mechanism once input electrical power was provided. The table below shows the full set of variables that our team knew initially or later calculated as part of our power analysis to determine the motion time for our linkage. The variables listed in the below table are also later shown and used in our step-by-step calculations that determine the linkage motion time. Included in the table are variables as used in calculations, their full names, and their calculated values. The table additionally includes the source or equation used to determine each variable value. Some variables were given to our team initially and known beforehand, while others were calculated with certain formulas, or determined from previous lab work.

**Table 7.1:** Full collection of parameters used in the power analysis for our linkage. Included for each parameter is the variable associated with it (as it appears in our calculations), the determined value, and the equation or source used to calculate the value.

| Parameter Name | Variable | Variable Value | Equation/Source Used to Calculate the Variable |
|---|---|---|---|
| Applied Inertia of the Motor | $J_{m,\,applied}$ | 5213.61 g*cm^2 | Given (From Lab 5) |
| Inertia of the Linkage | $J_{linkage}$ | 26880.46 g*cm^2 | Calculated Result From Lab 5 Work |
| Total Inertia of Linkage and Motor | $J$ | 73802.49 g*cm^2 | $J = (J_{m,\,applied} * N^2) + J_{Linkage}$ |
| Gear Ratio of Linkage Gear Train | $N$ | 3:1 ratio | $N = \frac{N_{teeth,gear}}{N_{teeth,pinion}}$ |
| Time of Linkage Motion | $t_{motion,\,ADAMS}$ | 0.29 seconds | From ADAMS Analysis in Lab 5 Work |
| Maximum Angular Speed of Load | $\omega_{max,\,load}$ | $74.72\ RPM$ | $\omega_{max,\,load} = \frac{\theta}{t_{motion}}$ |
| Maximum Angular Acceleration of Load | $a_{max,\,load}$ | $53.94\ rad/s^2$ | $a_{max,\,load} = \frac{2*\omega_{max,\,load}}{t_{motion}}$ |
| Input Link Angle of Motion (From pickup to blue zone) | $\theta$ | 65 degrees | $a = \frac{4\theta}{t^2}$ |

| | | | |
|---|---|---|---|
| Load Torque | $T_L$ | $0.398\ Nm$ | $T_L = \frac{4*\theta*J}{(t_{motion})^2}$ |
| Motor Output Torque | $T_m$ | $0.133\ Nm$ | $T_m = \frac{T_L}{N}$ |
| Motor Stall Torque at source voltage | $T_s$ | $0.14\ Nm$ | $T_s = \frac{T_{s,nom} * V_s}{V_{nominal}}$ |
| Motor Stall Torque at nominal voltage | $T_{s,nom}$ | 18.7 kg*cm<br>1836 mNm | Given (From Motor Specification Table) |
| Motor Torque-Speed Gradient | $K$ | 4.22 mNm/rpm | $K = \frac{T_s}{\omega_{no\ load}}$ |
| No Load Speed | $\omega_{no\ load}$ | 435 rpm | Given (From Motor Specification Table) |
| Source Voltage | $V_s$ | 9 V | Given |
| Nominal Voltage | $V_{s,nom}$ | 12 V | Given (From Motor Specification Table) |

Using our initially known parameters, we completed a series of calculation steps for our power analysis to determine other parameter values and eventually solve for the final desired value of linkage motion time. Our team's power analysis calculations followed the steps below. The same variables and formulas as listed in the above table were all used throughout our calculation steps.

Through these calculations, we will consider that the velocity profile is triangular.

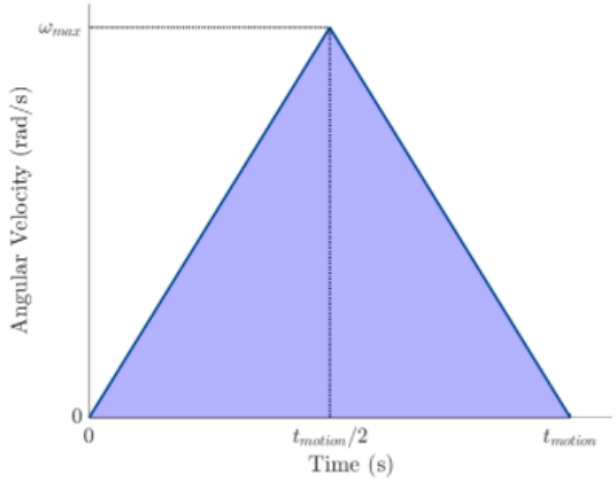



**Figure 7.1:** Velocity profile

***Step 1 :*** *Total inertia calculation.*

$$J = (J_{m,\,applied} * N^2) + J_{Linkage} = (5213.61 * 3^2) + 26880.46 = 73802.49\ g * cm\text{^}2$$

***Step 2 :*** *Required maximum angular velocity and acceleration.*

$$\theta = \int_0^{t_{motion,ADAMS}} \omega_{max,\,load}\, dt$$

$\theta$ is also simply the area under the velocity profile:

$$\theta = \frac{1}{2} * t_{motion,\,ADAMS} * \omega_{max,\,load}$$

$$\omega_{max,\,load} = \frac{2*\theta}{t_{motion,\,ADAMS}} = \frac{2*65*(\pi/180)}{0.29} = 7.82\ rad/s = 74.72\ RPM$$

Using the slope of the profile we have:

$$a_{max,\,load} = \frac{d}{dt} * \omega_{max,\,load} = \frac{2*\omega_{max,\,load}}{t_{motion,\,ADAMS}} = \frac{2*7.82}{0.29} = 53.94\ rad/s^2$$

***Step 3 :*** *Load torque calculation.*

The load torque in term of acceleration and inertia is equal to:

$$T_L = J * \alpha$$

Substituting with the value we calculated before we get :

$$T_L = J * a_{max,\,load} = \frac{2*J*\omega_{max,\,load}}{t_{motion,\,ADAMS}} = \frac{4*\theta*J}{(t_{motion,\,ADAMS})^2} = \frac{4*65*(\pi/180)*73802.49*10^{-7}}{(0.29)^2} = 0.398\ Nm$$

***Step 4 :*** *Motor torque output calculation.*

Using the transmission ratio we get:

$$N = \frac{T_L}{T_m}$$

$$T_m = \frac{T_L}{N} = \frac{0.398}{3} = 0.133\ Nm$$

$$\omega_m = N * \omega_{max,\,load}\ (placeholder\ used\ for\ algebraic\ simplification)$$

***Step 5 :*** *Stall torque calculation.*

For every electrical motor we have:

$$T_m = T_s - K * \omega_m$$

Applying it to our problem and substituting with the value we calculated before we get:

$$\frac{T_L}{N} = T_m = T_s - K * N * \omega_{max,\,load}$$

Our motor isn't operating at nominal voltage, hence we need to find the the new stall torque:

$$\frac{T_s}{T_{nominal}} = \frac{V_s}{V_{nominal}}$$

$$T_s = \frac{T_{s,nom} * V_s}{V_{s,nom}} = \frac{0.187*9}{12} = 0.14\ Nm$$

By substituting we get:

$$\frac{T_L}{N} = T_m = \frac{T_{s,nom} * V_s}{V_{s,nom}} - K * N * \omega_{max,\ load}$$

***Step 6 :*** *Equations used together for final motion time calculation.*

$$T_L = \frac{4*\theta*J}{(t_{motion})^2}$$

$$\frac{T_L}{N} = T_m = \frac{T_{s,nom} * V_s}{V_{s,nom}} - K * N * \omega_{max,\ load}$$

$$\omega_{max,\ load} = \frac{2*\theta}{t_{motion}}$$

***Step 7 :*** *End formula for smallest time of motion calculation.*

Solving this equation to find the smallest $t_{motion}$ we get:

$$t_{motion} = \frac{2*K*N*\theta*V_{s,nom}}{2*T_{s,nom} * V_s} + \frac{V_{s,nom}\sqrt{4K^2N^2\theta^2 + 4\frac{4T_{s,nom}V_s\theta J}{V_{s,nom}N}}}{2*T_{s,nom} * V_s}$$

$$t_{motion} = \frac{2*0.0403*3*1.1345*12}{2*1.836*9} + \frac{12*\sqrt{4*0.0403^2*3^2*1.1345^2 + 4*\frac{4*1.836*9*1.1345*0.00738}{12*3}}}{2*1.836*9} = 0.2339\ seconds$$

From our final result for our power analysis calculations, we end up with a linkage motion time of about 0.23 seconds. This time represents the fastest motion time that we can expect our own linkage design to make under full input motor voltage for a full sweep across the playing field. This result differs from our initial ADAMS analysis result carried out in earlier lab work since ADAMS only accounted for direct power input to the input link of our linkage. ADAMS also only accounted for inertia directly at the input link of our linkage and did not account for our transmission or apparent motor inertia. The new result we have from our above calculations do take all of these factors into account for an optimized motion time under more realistic conditions. This 0.23 second time for our linkage sweep can be expected for our real motor torque at its maximum input voltage and for our total mechatronics system inertia (with the motor, transmission, and linkage all accounted for). Another assumption we made was that friction in the transmission and linkage joins was negligible. Our linkage mechanism was designed and built such that the link arms and transmission axles can rotate very smoothly with negligible friction, and as such neglecting that friction was deemed acceptable by our team. Other assumptions include simplifying the linkage inertia to simple inertia, the velocity profile as perfectly triangular, and controller and H-bridge dynamics are not considered. That being said, our final value for the smallest possible motion time for our linkage at the maximum 9 volt motor input was 0.23 seconds.

## Section 8: Torque Transfer Analysis

In order to identify whether or not the mechanical system of our linkage will fail due to stresses experienced during torque transfer (as part of power transfer), we performed an analysis of torque transfer at each relevant joint and fastener location. This analysis is important because it ensures that our linkage system will not fail during use which would be a major safety and reliability issue. Failure of our linkage mechanism would prevent us from completing any competition tasks or scoring any points, which would very much negatively impact our team performance. In our linkage, torque is supplied by the motor and transferred through the pinion, set screw, gear, (all parts of the transmission) and then finally into the linkage links to actually move our linkage as desired.

Our assumptions in this analysis are as follows: we first assume that the input/coupler, coupler/follower, and follower/base joints have negligible friction. We also assume the motor shaft is round to simplify stress calculations. In reality the motor shaft has a small flat cut side to form a “D” shape, but assuming it has a circular cross section instead is much easier and does not change the stress calculation by a significant amount. Next our team assumed that the seating torque used in the assembly is the seating torque found in Table 8.2. We also assume that all of the friction is coming from the gears and their material. Finally, we assume there is no sliding in the process of torque transfer. Table 8.1 below was used to organize the various coefficients of friction used for our torque analysis and calculations. The coefficient values were found from common values listed in academic literature or manuals.

**Table 8.1:** Coefficient of friction values for different interactions of material types. Both sliding (kinetic) and static coefficients of friction are listed as required in our later calculations.

| Material Interface Type | Coefficient(s) of Friction |
|---|---|
| Aluminum on aluminum dry | 1.40 sliding, 1.05 - 1.30 static |
| Aluminum on mild steel dry | 0.61 static, 0.47 sliding |
| Nylon on steel | 0.40 static |
| Nylon on nylon | 0.15 - 0.25 static |
| Steel on steel clean and dry | 0.50 - 0.80 static, 0.42 sliding |
| Brass to steel | 0.51 static |

A key part of our torque analysis is considering how torque is transferred through our linkage design joins. Figures 8.1 through 8.5 below show an aerial view of the CAD model as well as cross sections of the various joints and connections in the mechanism. These are key joints where torque transfer occurs and therefore where failure could more likely occur.

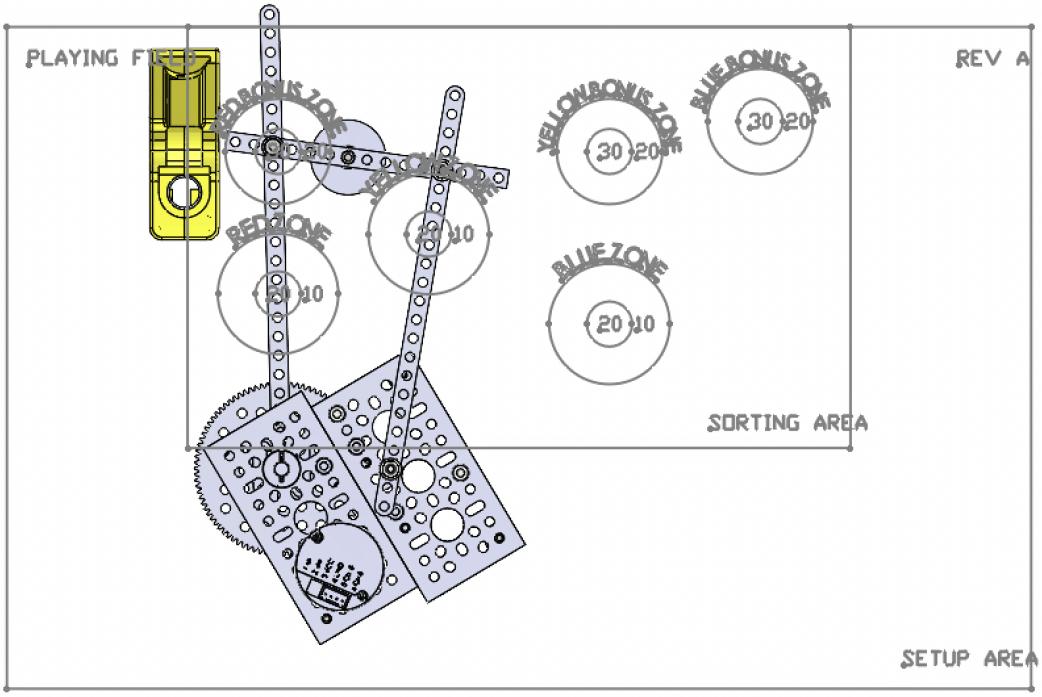



**Figure 8.1:** Top down view of CAD model.

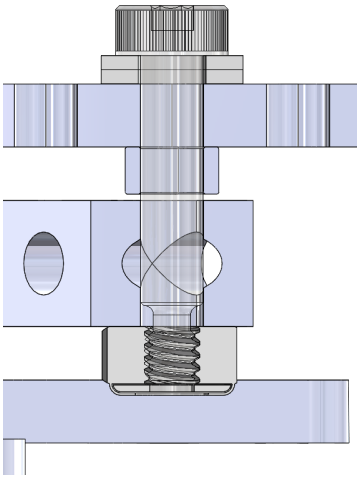


**Figure 8.2:** Input to coupler and coupler to follower joints cross sectional view.

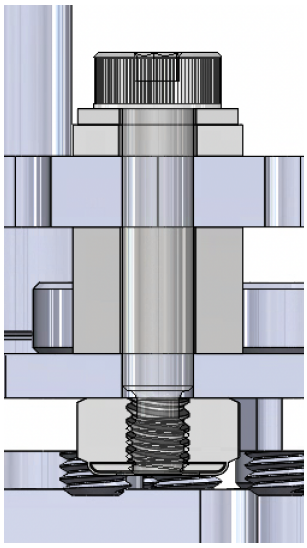


**Figure 8.3:** Follower to ground joint cross sectional view.

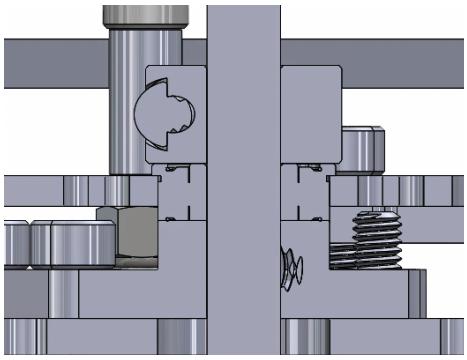


**Figure 8.4:** Set screw cross sectional view.

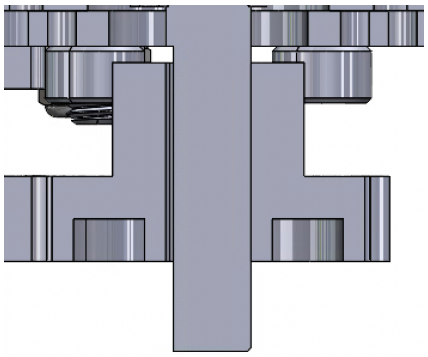


**Figure 8.5:** Pinion cross sectional view.

We are performing the torque transfer analysis only on the joints where the torque is being transferred and making sure that the friction between the shafts and the set screws are bigger than the transmitted torque. If the torques transmitted between the gears is greater than the friction in the shaftes then there will be slipping in the shafts. This slipping of a shaft will cause a loss of torque and power transfer in our linkage, and our team considers this to be a major failure in our linkage. If the motor has a higher force due to the torque

than the friction in the shaft then the motor will spin freely and not spin the gear causing failure of our linkage to function. The friction between the second gear and its shaft does not matter because that shaft is supposed to be free flowing while supported by the ball bearings. The table below shows the calculated torques and torsional holding powers that can be supported by the various components such as screws used to assemble and hold together our linkage mechanism joints.

**Table 8.2:** Torque calculations with friction coefficients.

| Nominal screw size | Seating torque (N-mm) | Normal Force on the gear (N) | Friction Coefficient | Friction (N) |
|---|---|---|---|---|
| M4 | 4067.454 | 2401.92 | .51 (Brass to Steel) | 1224.98 |
| M4 | 4067.454 | 2401.92 | .61 (Aluminum to Steel) | 1465.17 |

Below are the full calculations used to determine the values given in the above table. These calculations specifically apply to the joints in our linkage transmission and at the base of the input link where torque is actually being transferred.

$$F_{Holding\ power} = Normal\ Force(Friction\ Coefficient)$$

$$F_{motor} = \frac{T_{motor}}{radius} = 44.33\ N < F_{Holding\ power}\ of\ the\ brass\ to\ steel\ contact$$

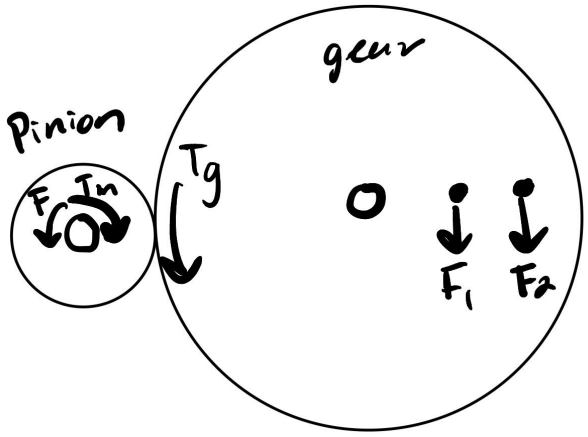



**Figure 8.6:** torque transfer of the whole linkage.

(The friction in the second axle has to be greater than the rotational inertia of the whole linkage times the angular acceleration)

$$F_{gear2} = J_{total} * \alpha_{max} \div shaft\ radius = (73802.49\ g * cm\text{^}2)(1 * 10^{-7}\ \frac{kg*m^2}{g*cm^2})(53.94\ rads/sec)/(.003)$$

$$= 132.66N < F_{holding\ power}$$

**Bolt shear and bending stress due to torque calculations and analysis**

1. Find the torque applied to the gear

$$T_{gear\ 2} = T_{motor} N_{gear\ ratio} =. 3988\ Nm$$

2. Torque equilibrium

$$T_{gear\ 2} = F_{bolt1} R_{bolt1} + F_{bolt2} R_{bolt2}$$

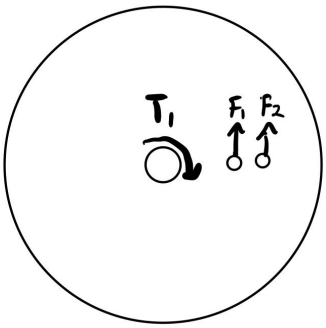



**Figure 8.7:** torque transfer from gear to bolts.

Assumptions:

The force on the screws are equal to each other'

Screws are #8-36 low carbon steel and have a kpsi (shear strength) of 60(413 Mpa) and a yield strength of 36 kpsi (248.211Mpa)

Screws have a shear stress area of .01474 in^2 (9.5096584 mm^2)

$$R_{bolt1} = 16mm,\ R_{bolt2} = 24mm$$

3. Find the force acting on the bolt

$$F_{bolt1} = F_{bolt2} \frac{T_{gear\ 2}}{(R_{bolt1} + R_{bolt2})} = F_{bolt} = 8.9745\ Newtons$$

4. Find the shear stress in the bolt

$$\frac{F_{bolt}}{Area} = \tau < shear\ strength = 413\ Mpa > \frac{8.9745}{9.5096584} =. 9437\ Mpa$$

No shear stress failure

5. Find the height of the application of the force on the bolt

$$h_{application} = 12mm - 2mm = 10mm$$

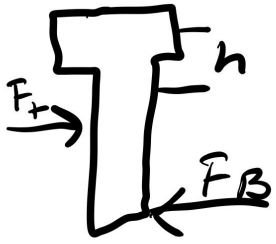



**Figure 8.8:** FBD of the bolts in the gear.

6. Find the moment due to the force

$$M_{bolt} = F_{bolt}(h_{application}) = 89.745\ Nmm$$

$$I_{bolt} = \frac{\pi(d_{bolt})^4}{64} = 12.566\ mm^4$$

7. Find the bending stress caused by the moment on the bolt

$$\sigma_{zz} = \frac{M\,R_{bolt}}{I_{bolt}} = \frac{89.745\ Nmm(2mm)}{12.566\ mm^4} = 14.177\ Mpa < yield\ strength = 248\ Mpa$$

Using these equations we can conclude that the screws will not fail due to the bending stress or shear stress.

Low C steel has lowest K psi so it will give the most conservative material to use.

**Yield and fracture of pinion and gear calculations and analysis**

Pinion:

1. Use the motor equation to find angular velocity of the motor and therefore the pinion.

$$T_m = T_s - K * \omega_m$$

$$0.133 = 0.14 - 0.00422 * \omega_m$$

$$\omega_m = \omega_{pinion} = 16.59\ RPM$$

2. Calculate actual power usage of motor

$$K_t = T/I = 1.834/9.2 = 0.199\ Nm/A$$

$$P_{mechanical} = H = T_m * \omega_m = 0.133 * 16.59 = 2.21\ W$$

3. Solve for tangential force experienced by each tooth of the pinion.

$$W_t = \frac{60{,}000*H}{\pi*d*n} = \frac{60{,}000*2.21}{\pi*24*16.59} = 106.01\ N$$

4. Calculate pitch line velocity.

$$V = \frac{\pi*d*n}{60} = \frac{\pi*0.024*16.59}{60} = 0.0208\ m/s$$

5. Calculate velocity factor using pitch line velocity.

$$K_v = \frac{6.1+V}{6.1} = \frac{6.1+0.0208}{6.1} = 1.003$$ (assuming cut or milled profile)

6. Find Lewis form factor from table 14-2 in the lecture notes.

$$Y = 0.359$$

7. Calculate the module using pitch diameter and number of teeth.

$$m = \frac{d}{N} = \frac{0.024}{30} = 0.0008\ mm$$

8. Calculate bending stress on each tooth of the pinion.

$$\sigma_{pinion} = \frac{K_v*W_t}{F*m*Y} = \frac{1.003*106.01}{0.006*0.0008*0.359} = 61.7\ MPa$$

This value is well below the typical yield strength of brass (around 340 MPa), so the pinion will not yield during motion.

Gear:
Steps follow closely to what was done with the pinion.

$$\omega_{gear} = \frac{\omega_{pinion}}{N} = \frac{16.59}{3} = 5.53\ RPM$$

$$W_t = \frac{60{,}000*H}{\pi*d*n} = \frac{60{,}000*2.21}{\pi*72*5.53} = 106.01\ N$$ (makes sense that it's the same as pinion)

$$V = \frac{\pi*d*n}{60} = \frac{\pi*0.072*5.53}{60} = 0.0208\ m/s$$

$$K_v = \frac{6.1+V}{6.1} = \frac{6.1+0.0208}{6.1} = 1.003$$ (assuming cut or milled profile)

$$Y = 0.442$$

$$m = \frac{d}{N} = \frac{0.072}{90} = 0.0008$$

$$\sigma_{gear} = \frac{K_v*W_t}{F*m*Y} = \frac{1.003*106.01}{0.006*0.0008*0.442} = 50.12\ MPa$$

This value is below the typical yield strength of aluminum (around 60 MPa), so the gear will not yield during motion.

From our calculations and torque analysis, we found that the key joints in our linkage mechanism where torque transfer occurs should not fail. This is because the set screws and axles will not slip and fail since the friction between the axles is greater than the torque provided by the motor and transferred via the gear train ratio. We again consider failure to be the slipping of the set screw on the axel since this will cause torque and power transfer to cease, which will prevent our linkage from moving or functioning. We also analyzed whether or not the teeth of the gear and pinion would yield under typical operational loads and found that the stress they experience is well below the yield stress of the material so they will not. As such, with our torque analysis completed and the integrity of our joints verified, we as a team can be confident that our linkage mechanism will function as intended.

## Section 9: Capabilities & Limitations of the Hardware

In order to fully understand the operation and limitations of the mechatronic assembly, we must understand each part's limitations. In this section, the hardware components covered are the color sensor, electromagnet, power supply, and H-bridge driver. All of these components are crucial to the performance of the linkage and therefore it is useful to lay out their potential points of failure and/or restrictions. We discuss the limitations of each component below and their capabilities, as well as how our team personally grappled with those limitations during our project work.

**Color Sensor**

First, a discussion of how the TCS3200 color sensor operates. It contains semiconductors that convert light to current, called photodiodes. There are four different types of photodiodes, ones with red filter, blue filter, green filter, and no filter. The color sensor has sixteen of each type arranged in an array. Each photodiode with a filter is sensitive to light of that color filter; photodiodes with a red filter are sensitive to red light and so on. The current read by the color sensor is converted into a square wave with a corresponding frequency proportional to the light intensity. The Arduino board then reads that square wave and the color is determined based on user defined ranges of each value.

In order to ensure that the correct color was read each time a new disc was introduced, we recorded the red, green, blue, and clear output ranges for each disc color. These values can be found in Appendix C. After calling the "getColor" function to return the RGB and clear values read by the sensor, the "evaluateColorSensor" function uses if statements to determine if these values lie within the predetermined ranges. If they do, this color is recorded and the system is told to move to the appropriate location.

The major limitation of the color sensor is the variability of sensor readings under different lighting conditions. If something casts a shadow over the sensor, the values for the same color will not be within the predetermined range and therefore it will be read as either the wrong color or no color at all. To ensure this was not an issue, we calibrated and tested under the same conditions, full light with nothing over the color sensor. Another minor limitation was that sometimes the values were negative, but this happened very infrequently. This might occur due to the overflow caused by the addition of numbers with the result being larger than what can be displayed with the number of bits of the sensor. Because this happened so rarely and it fixed itself in a matter of microseconds, we determined that it was not an issue that needed to be addressed. Overall, the color sensor performance was optimal and never limited the functionality of the entire system.

**Electromagnet**

The operation of the electromagnet involved turning on when at the disc pickup location, maintaining a constant twelve volts throughout the path of motion, and dropping the disc once at the drop off location using a decaying sinusoidal function to control voltage. This sinusoidal function was required to ensure that there was no residual magnetism as the electromagnet moved back to the disc pickup location. Sometimes it would still be magnetized enough to pick up a disc after a full test run so we reuploaded the code and waited a few seconds in between each run to make sure it didn't affect the next test. At one point the electromagnet failed to pick up the disc after repeated tests but we determined that one of the wires had come loose at the point it is connected to the H-bridge. Once we reconnected it, that problem went away.

**Power supply**

While the sensors could be powered by the Arduino board by itself, in order to ensure the torque and speed requirements of the motor could be met we introduced a separate power supply. We used the power

supply at twelve volts output and the maximum current it can supply was two amps. This is not a limitation because it is much higher than the required current to run the motor (less than two amps). As such our team did not have to seriously consider the maximum power, current, or voltage outputs for our power supply since the maximum outputs far exceeded our needs.

**H-Bridge driver**

The H-bridge used acts as a motor driver that allows the voltage to be applied across the motor in two directions, allowing the motor to rotate forward and backward. It achieves this by alternating between pairs of switches that are either open or closed (see Figure 9.1 below from the Lab 8 Tutorial). When switches one and four are closed, the motor runs in the forward direction and vice versa. The assumption is that these are ideal switches meaning that when they are open they have infinite resistance and when they are closed they have zero resistance. This is not the case in reality and this leads to energy loss. It also acts as a relay that allows the low current from the Arduino board to control the higher voltage and current from the power supply. The maximum current output is two amps (found via the given datasheet), which is the same as the power supply we used. However, the h-bridge is more limited because its current output is controlled by the Arduino board as part of our motor control system.

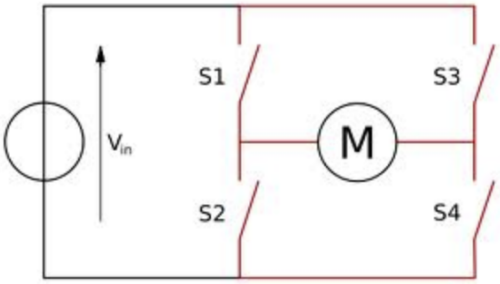



**Figure 9.1**: Simplified diagram of H-bridge

## Section 10: Arduino Implementation

To control the linkage system we used an Arduino Uno board as the electronic card of our system. It was in charge of acquiring the data from our different circuit elements as well as sending the output control data to the motor to alter its voltage and control it.

To ease the control of our system we used a state machine. Using this has many advantages. It makes the code clearer and easier to write because the conditions are easier and you don't need to worry about everything happening at the same time. You can control what is happening when. You can determine precisely when you need to jump to a new state. That way you are sure that what you wanted happened before transitioning.

In our system a state machine is used to be sure that first we calibrate the mechanism then we go to the pick up location before determining the color of the disk and picking it up to drop it at the desired position. We then go into a loop of going to the disk feeder, determining the color and dropping it. We don't need to worry anymore about the calibration because we will never go back to that state again.

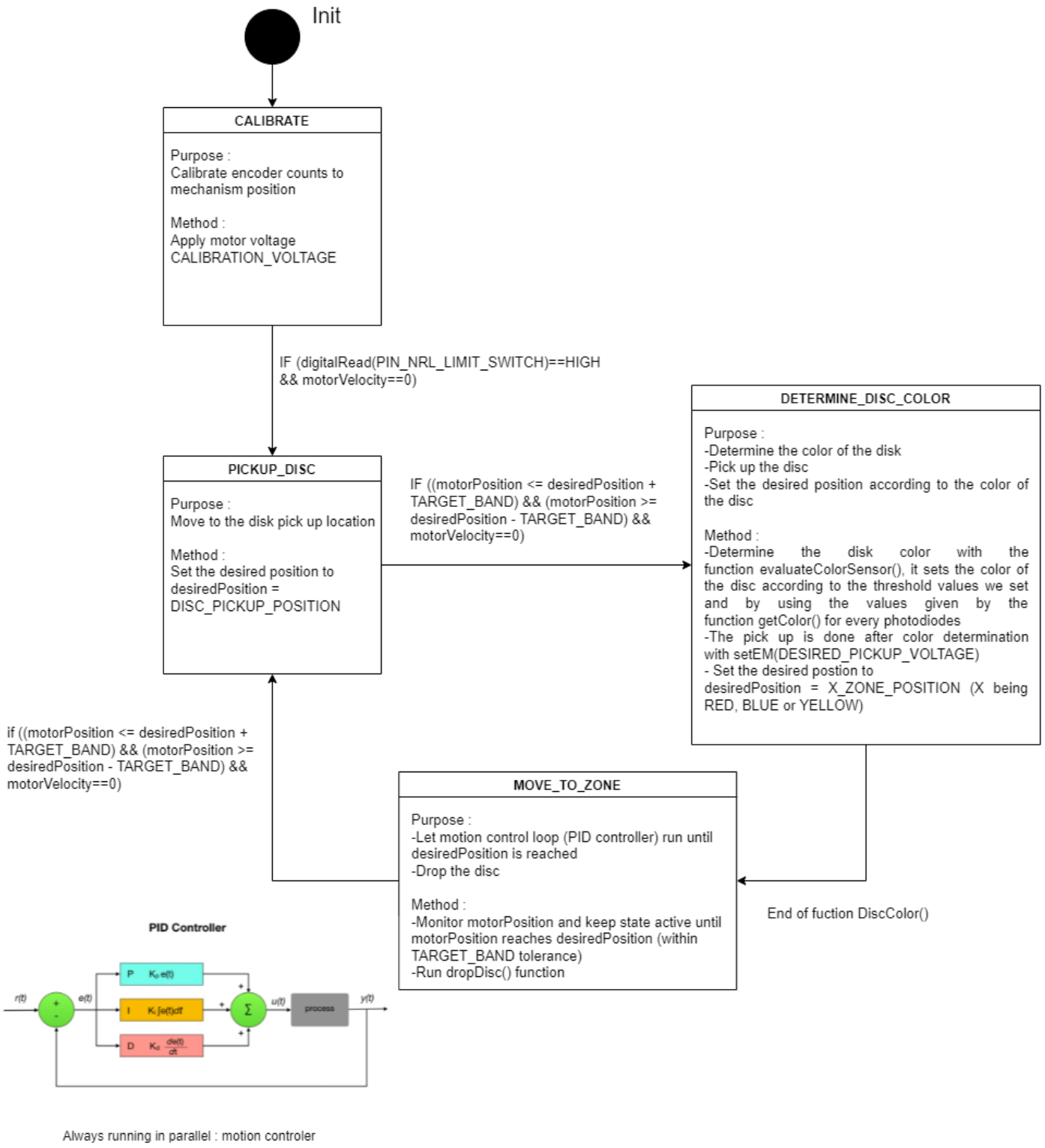



**Figure 10.1**: State machine diagram

Having a smooth, precise and fast motion is key to our project in order to maximize the score we get out of it. To achieve this we implemented a PID controller in the arduino code. This PID controller will be activated as soon as the mechanism is starting to move.

It is divided into three parts:

- Proportional $Kp\ e(t)$
- Integrative $Ki\ \int e(t)$
- Derivative $Kd\frac{de(t)}{dt}$

e(t) being the difference between the process output and the desired setpoint

We then add those together to have the full effect of the PID:

$$u(t)\ =\ Kp\ e(t)\ +\ Ki\ \int e(t)\ +\ Kd\frac{de(t)}{dt}$$

u(t) being the process input control value.

In the arduino code it will be implemented using:

```
desiredVoltage = KP * positionError +  KI * integralError +KD * velocityError
```

With:

```
positionError = desiredPosition - motorPosition
integralError = integralError + positionError * (float)(executionDuration) / 1000000
velocityError = 0 - motorVelocity
```

Hence we just need to look at the motorVelocity and motorPosition to control this PID.

Some changes we might have made to the Arduino code if given more time are in the following table.

**Table 10.1**: Proposed code changes

| Performance Issue/Behavior Change | Proposed Code Change |
|---|---|
| Mechanism completes all motions smoothly but would benefit from dynamic PID values because of the varied distances to each zone. | Implement dynamic control gains: change the values of KP, KI, and KD depending on what motion is supposed to happen. |
| The serial monitor prints out unnecessary statements that take time in the code, slowing down the mechanism. | Comment out these print statements and readjust the PID values accordingly (they might not be appropriate anymore). |
| The code has an excessive runtime and executes slower than expected for the given number of functions. | Optimize the code layout, and remove unnecessary lines, comments, or variables to save memory and boost code run speed. |

## Section 11: Final Testing Results & Design Critique

**Table 11.1**: Final testing score

| Color | Description | Number | Value of Item | Total |
|---|---|---|---|---|
| **Red** | Red discs dropped onto the red zone | 6<br>0 | 30 (Red center value)<br>20 (Red outer value) | 180 |
| **Yellow** | Yellow discs dropped onto the yellow zone | 6<br>0 | 20 (Yellow center value)<br>10 (Yellow outer value) | 120 |
| **Blue** | Blue discs dropped onto the blue zone | 6<br>0 | 20 (Blue center value)<br>10 (Blue outer value) | 120 |
| **Incorrect** | Discs dropped onto incorrect zone | 0 | 1/5 value of target | 0 |
| **Missed** | Number of discs that missed | 0 | 0 | 0 |
| **Time Bonus** | How much time you had left after dropping all discs. Max time is 90 seconds | 76 = 90 - 14 | +2 for every second | 152 |
| **Accuracy Bonus** | Did you sort all the discs correctly? | 1 | +50 points | 50 |
| **TOTAL** | Add up your scores from the previous rows | N/A | N/A | 622 |

**Mechanical Design:**

What worked well:

- The motion was exactly as we expected and each disc could reach the edge of the middle of the zone.
- Using SolidWorks was helpful to visualize the layout of the mechanism beforehand and make adjustments based on how things fit in 3D space.
- The different lab assignments progressed well from individual to team assignments, and allowed us as a group to quickly agree on an individual good design that we modified and used for our final design.

What didn't work well:

- The pinion occasionally came loose from the motor shaft which led to failure. We would have to fit it back on and tighten the set screw. However, this only happened twice in total and we determined that the design didn't need to be changed.
- We ended up moving the disc ramp location slightly so that the electromagnet picked up the discs slightly off center which helped the yellow discs reach the edge of the middle of the yellow zone.
- Some parts of the design formulation such as initial 2D linkage synthesis and ADAMS analysis could seem tedious at times when doing alone and might have been better as team assignments over individual ones, since later on more designs and files had to be shared in the group and modified. This caused organizational issues at times.

**Assembly:**

What worked well:

- The assembly worked very well because using the CAD and lab assignments we knew how the assembly would assemble without very much trouble.
- Each joint had little friction because of the washers and spacers used and because each of the links were level.

What didn't work well:

- The lengths of the wires limited the placement of the breadboard, Arduino, and H-bridge. It would have been beneficial to use longer wires to have more adaptability.
- The amount of wires present took up lots of space and made testing later on more difficult. Misplacing a wire also would be difficult to fix.

**Controls and Strategy:**

What worked well:

- We did the initial PID tuning but the mechanism wasn't reliable and would stutter occasionally. We found that using the dynamic testing PID values as a benchmark was very helpful and expedited the process of tuning the control parameters.
- It was fairly easy to quickly update Arduino code and reupload it during testing or PID tuning. This made it easy to get PID tuning done especially. The ease of use of Arudino was overall very helpful to refining our control system.

What didn't work well:

- The mechanism would stutter occasionally because of the PID controls. We chose the strategy where we only got one bonus zone but it limited the amount of points we can score. To maximize our points, our run time was 15 seconds.
- The toggle switch used to start and stop runs was very small and hard to use. At times it was even easy to forget which position it was toggled to which could be a possible safety hazard. Using a larger and easier to operate switch would be very beneficial for future kits.
- Feeding all of the disks into the ramp for testing and competition was very challenging and more of a quick reaction and dexterity task. This strongly affected our testing and performance since any mistake in feeding disks was easy to make and would cause the run to be a failure (due to incorrectly sorted disks, or missing a disk pickup).

## Section 12: Dynamic Performance

The performance of our closed-loop control system for our linkage was able to be made very fast and effective with PID tuning. Time spent on PID tuning by our team allowed our mechanism to move at a high speed between the disk pickup locations and color targets, while stopping quickly and accurately at each target. Our PID tuning also prevented any perceivable overshoot of our electromagnet and linkage. Overall our linkage motion was very fast and accurate at reaching each color target we aimed for.

These good motion qualities can be seen in the figures further below, which show the registered position of our linkage (via the motor encoder counts) versus time. These plots are from our dynamic performance testing work in the lab, and they all reflect the highly effective linkage performance we reached via PID tuning. The motion for all color zones (red, yellow, and blue) had quick settling times (near 250 ms for the blue zone and near 120 ms for the red and yellow zones) and fast rise times, high accuracy, and near zero overshoot. High accuracy can be seen since the steady state error for all color zones by our linkage was always very low. Our linkage for all runs for each zone had only at most a steady state error of 2 encoder counts, with most runs having 0 or 1 count for steady state error. Meanwhile, for all runs for each zone our linkage consistently had 0 counts of overshoot for the red and yellow zones, with 0 or 1 overshoot counts for the blue zone. These overshoot results are very good for our team and allowed us to stack almost every disk in each zone. Having more overshoot for the blue zone makes sense in this case since it is the furthest zone from our disk pickup location, and thus the motor is moving the fastest as it covers more ground to reach the blue zone.

Our observations during PID tuning for our linkage mechanism did overall agree well with the given PID tuning guidelines. We as a team closely followed the guidelines throughout our tuning, and by doing so we were able to achieve very good dynamic performance for our linkage motion. We saw that increasing Kp decreased the rise time and slightly reduced our steady state error as we completed our tuning. However increasing Kp did initially leave us with large overshoot. We eventually decided on a fairly high Kp value of 0.242 to give our mechanism a fast rising time and help to reduce steady state error. Meanwhile, we saw that increasing Kd helped to reduce overshoot, and importantly also prevented the linkage from oscillating around the target point for each zone once it was reached. This was especially important to our overall performance by reducing settling time, and we decided on a Kd value of 0.01105 which was able to reduce oscillations causing issues for our motion. This Kd value also was important to remove the overshoot that was initially present due to our large Kp value. This change successfully eliminated our overshoot almost entirely for all color zones. Lastly, we saw how changing Ki affected steady state error and oscillations. We decided on a low value of 0.00595 for our Ki which prevented oscillations from occurring and together with the Kp and Kd values allowed our steady state error for each zone to be very low.

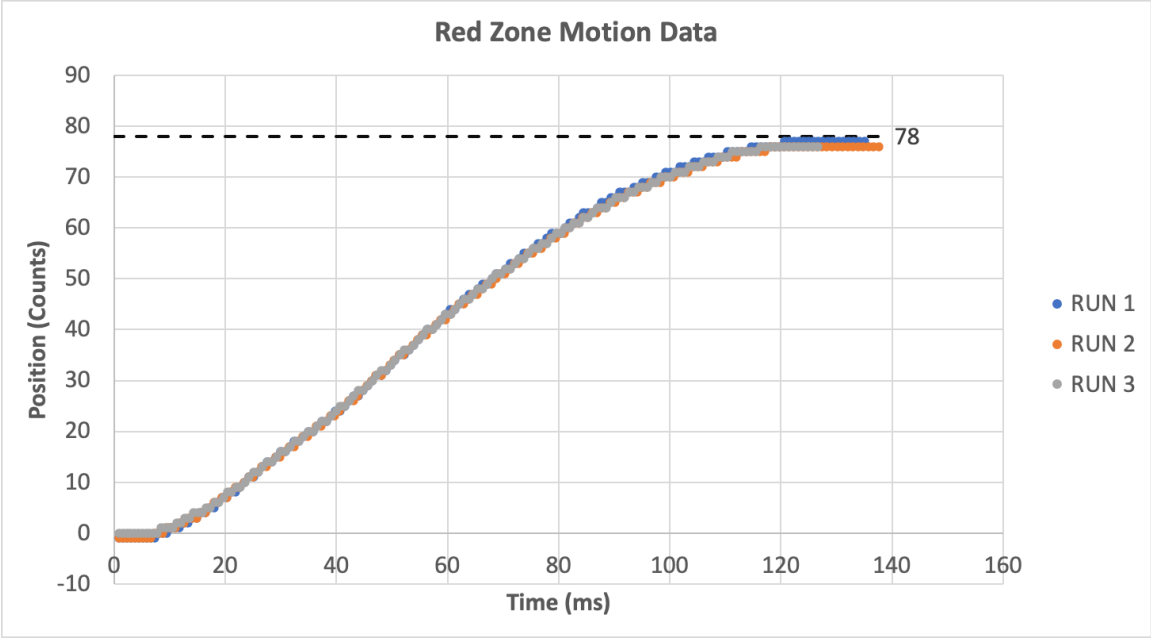



**Figure 12.1**: Red zone motion data results from dynamic performance testing

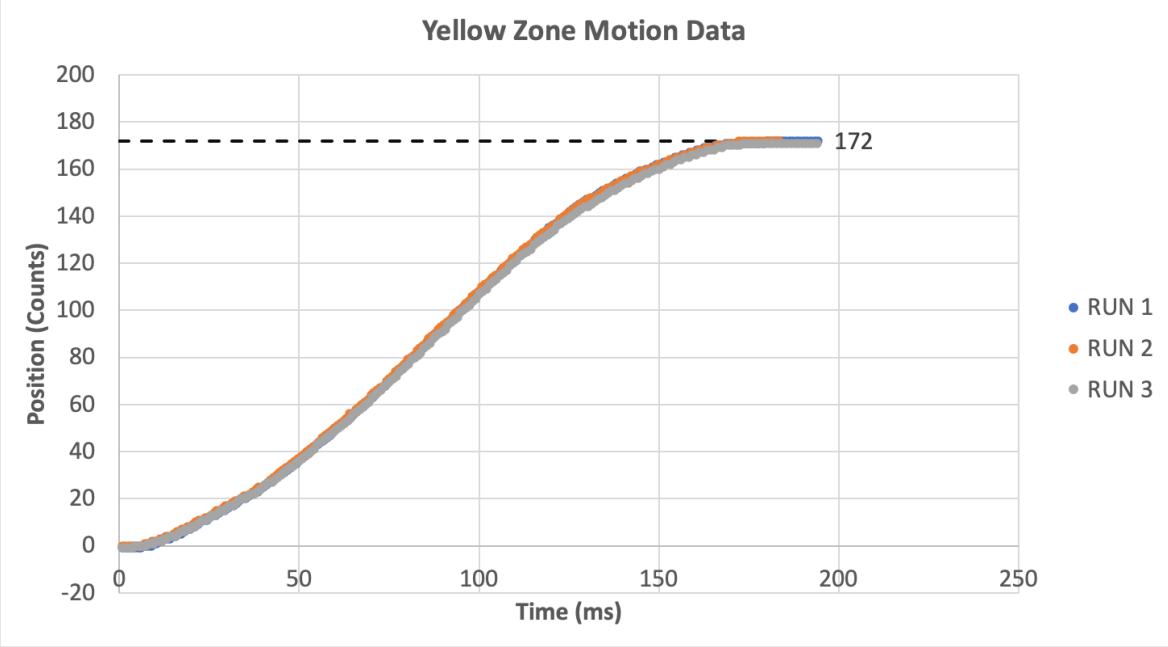



**Figure 12.2**: Yellow zone motion data results from dynamic performance testing

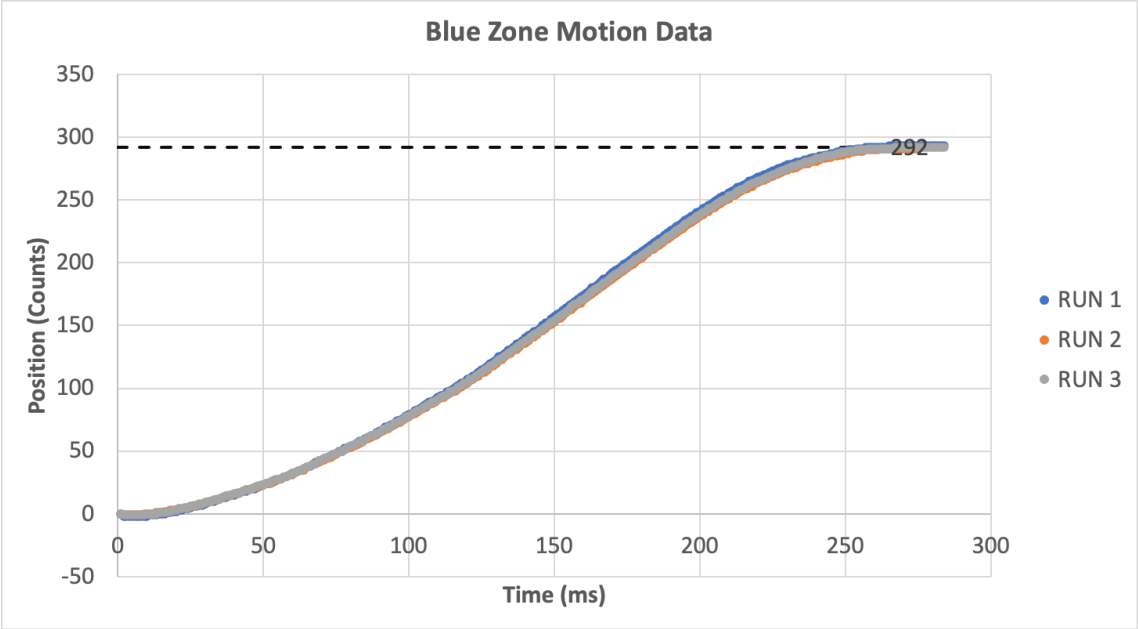



**Figure 12.3**: Blue zone motion data results from dynamic performance testing

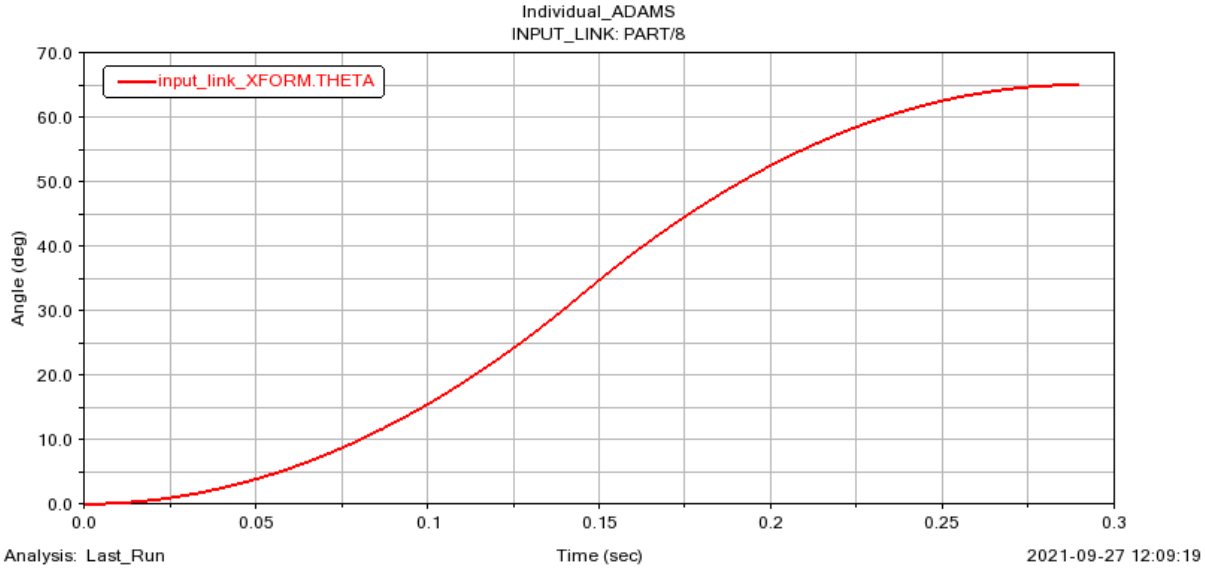



**Figure 12.4:** Trajectory from ADAMS

Discussion with the values we got on ADAMS:

Calculation of angle of motion from testing:

$$\theta = \frac{counts\ of\ motion}{count\ per\ revolution*N} * degree\ per\ revolution = \frac{292}{384.5*3} * 360 = 91.1°$$

This value of 91.1 degrees is different from the ADAMS analysis angle of 65 degrees. These values are fairly different but we are unsure what accounts for this difference.

The time of motion used for ADAMS is 290 ms and the settling time is around 270 ms. The settling time is equal to the rise time in this case because the curve is critically damped. This difference in time can partly be explained by the use of PID control which was not accounted for in ADAMS.

Additionally, our team answered the following questions further below.

1. Qualitatively speaking, would you say your controller resembles an underdamped, critically damped, or overdamped system?
2. Say you set your friction compensation voltage to 0V. Your linkage is settling with a steady-state offset of 20 counts. What gain should you increase first to reduce this error, given your goal is to minimize settling time, and why?
3. How does the required torque from the motor change as you change each PID gain?

Additional linkage mechanism dynamic performance answers.

1. Qualitatively speaking we would say that our linkage controller more closely resembles a critically damped system. We feel that this is true for all color zone targets our linkage must move to. This is because it very quickly and smoothly reaches the target position value from the starting location, much like a critically damped system would when released away from equilibrium. This behavior is easy to see in the position vs time charts in the figures earlier above for each zone. This critically dampened behavior is indicative of our good PID tuning work and results, and shows that our PID tuning was optimized to allow our linkage to reach targets quickly and accurately in each color zone. Finally, it also qualitatively is not fair to say our controller is underdamped or overdamped since for an underdamped case, large oscillations around the target or equilibrium values would be noticed. Such oscillations are not present at all for our controller and were well eliminated in our PID tuning. Our controller is not qualitatively overdamped since this would mean that our rise time towards our target position or equilibrium would be very long and slow. This is not the case either, since our rise times are very fast and not excessively slow.
2. In order to minimize the steady-state error, we have two choices. We can either increase Kp that is going to decrease the steady-state error or increase Ki that is going to eliminate it. The problem with the latter is that it will also increase the settling time which is something we want to keep as small as possible. Increasing Kp will only have a small effect on the settling time. Hence we should increase Kp if we want to reduce the steady-state error and reduce settling time. To minimize the settling time we could then increase Kd, that won't have an effect on the steady-state error but would decrease the settling time.
3. Increasing Kp increases the magnitude of the torque in proportion to the position error. This can also be justified by the fact that increasing Kp decreases rise time and the way to achieve this is to increase torque. The same can be said about Ki, increasing it decreases rise time and therefore increases torque magnitude. However, Kd has little effect on rise time so it also has little effect on the torque requirement.

## Section 13: Course Feedback & Evaluation

Reflect on the following based upon your lecture and project experience this semester. Limit discussions to 1 paragraph per question maximum.

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

Some parts and materials that would've been helpful to have in our kit include the following: a larger selection of gears in order to achieve a gear ratio closer to what was calculated in preassembly analysis; materials better suited to secure the limit switch to the base for testing; include tape for securing various parts in the kit itself. Lastly, it would also have been helpful to include longer extension cables for the breadboard and arduino circuitly.

2. What changes to the current project or new projects would you recommend for future semesters of ME 350?

Some changes to the current project include a longer disc ramp that holds more discs at once as well as letting the student have more input in creating the final testing code in order to improve understanding. Include better explanations of what is expected in each section of the reports. Another thing would be to change the scoring of the final testing so that the balance between speed and number of bonus zones hit is more even. Right now, there is a much higher incentive to hit more bonus zones and take longer than vice versa. Overall however we as a team enjoyed the project and think that its overall premise is valuable to ME students to experience and work on. We were able to learn a lot about mechatronics, control, linkages, and more during the project.

3. Do you think the current point distribution between all graded coursework (i.e. between lab and lecture) is fair? How could it be modified for future semesters?

We believe that the current point distribution is fair and does not need to be changed. We think that it makes sense to have the project take up a large part of the grade given how much class and lab time is dedicated to it. We also think the peer evaluation portion of the grade is fair and helps to encourage good teamwork as well.

# Appendix A: Assembly Manual

## Linkage Mechanical Assembly

This appendix section describes how our final linkage design is mechanically assembled before any electronic components are added. Each step describes the necessary actions and kit parts required to fully assemble our linkage mechanism by hand. Proper safety precautions should be followed and Z87 safety glasses should be worn during assembly. Also included below is a full list of parts needed for the assembly.

List of all parts required to assemble our linkage design:

1. Aluminum Patterned Angle-Bracket (x2)
2. Steel Socket Head Screw (M4 x 0.7mm, 8mm Length) (x8)
3. Steel I-Bracket (x2)
4. Threaded Rubber Feet (x4)
5. Steel Hex Nut (x4)
6. Steel Nylon-Insert Locknut (M4 x 0.7 mm, 5 mm High) (x4)
7. Aluminum Spacer (4mm ID, 6mm OD, 12mm Length) (x2)
8. Steel Socket Head Screw (M4 x 0.7mm, 18mm Length) (x2)
9. 1611 Series Flanged Ball Bearing (x2)
10. 90-Tooth Aluminum Gear (x1)
11. Steel Socket Head Screw (M4 x 0.7mm, 12mm Length) (x9)
12. Aluminum Set Screw Hub (x1)
13. 1102 Series Flat Beam (x2)
14. 1106 Series Square Beam (x1)
15. Steel Nylon-Insert Locknut (6-32, 11/64" High) (x2)
16. Aluminum Spacer (4mm ID, 6mm OD, 3mm Length) (x1)
17. 18-8 Steel Washer for M4 Screw Size, 4.3 mm ID, 9 mm OD (x3)
18. Steel Shoulder Screw (6-32, 5/8" Length, 5/32" Diameter) (x2)
19. Nylon Washer (0.173" ID, 1/16" Thick, 3/8" OD) (x1)
20. Nylon Spacer (0.166" ID, 9/32" Length, 3/8" OD) (x2)
21. 2100 Series Stainless Steel Round Shaft (6mm Diameter, 100mm Length) (x1)
22. 2910 Series Aluminum Clamping Collar (6mm ID x 19mm OD, 8mm Length) (x1)
23. Electromagnet (200N, R-1207-12) (x1)
24. Steel Socket Head Screw (6-32, 1-1/4" Length) (x1)
25. 5202 Series Planetary Gear Motor (13.7:1 Ratio) (x1)
26. 30-Tooth Brass Pinion Gear (6mm Bore, MOD 0.8, Set Screw) (x1)

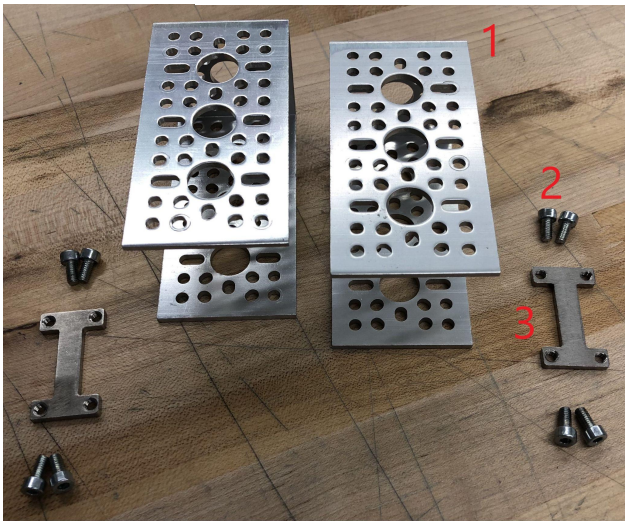
- **Step 1: Assembly of the bracket**

Parts used:

1. Aluminum Patterned Angle-Bracket (x2)
2. Steel Socket Head Screw (M4 x 0.7mm, 8mm Length) (x8)
3. Steel I-Bracket (x2)

Written description:

First take the two angle pattern brackets (1) and offset them by one hole row as shown in the pictures. Then take the two steel I brackets (3) and the eight short screws (2) and screw in the I brackets so that each side is attached to a larger bracket as shown. Repeat this for the top and bottom of the large brackets with the screws inserted to point inside on both the top and bottom.




a)

b)

c)

**Figure A.1**: a) Components before assembly; b) Top view of assembly; c) side view of assembly

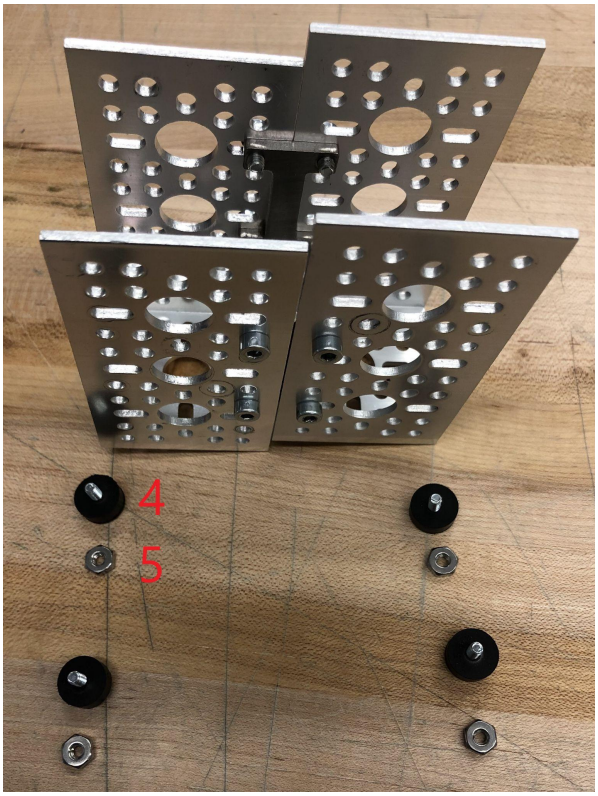
- **Step 2: Assembly of the linkage feet**

  Parts used:

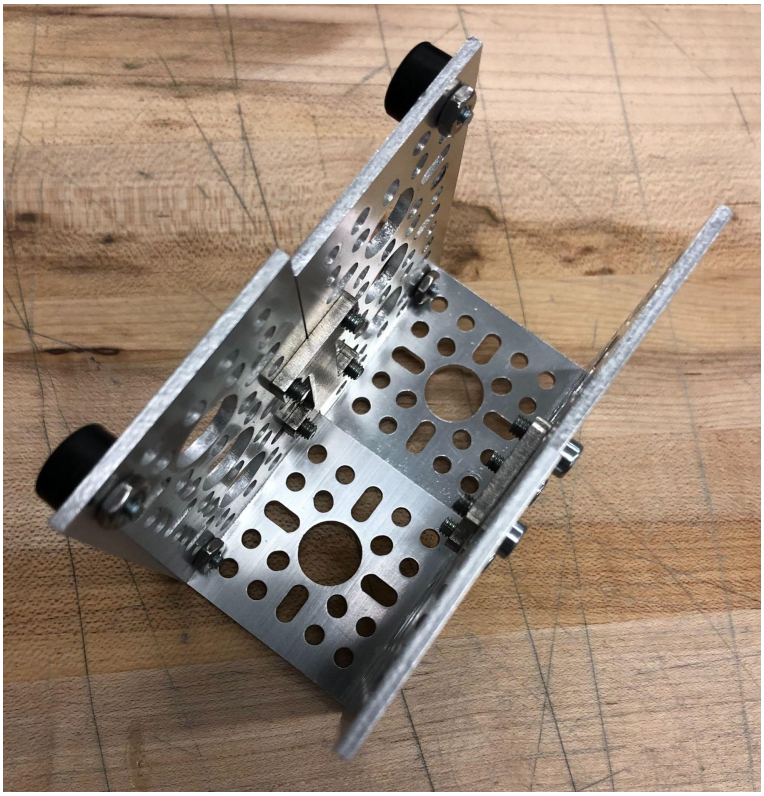
  4. Threaded Rubber Feet (x4)
  5. Steel Hex Nut (x4)

  Written description:

  First take the assembled base and make sure looking from the open end of the angle bracket, that the angle bracket on the right is closer than the one on the left. Then, flip over the assembly and install each of the four feet (4) in the corners of the base, and then tighten the feet into place with the four hex nuts (5). The feet should allow the current assembly to rest flat on a table as shown.




a)



b)

**Figure A.2**: a) Components before assembly; b) Completed assembly for this step

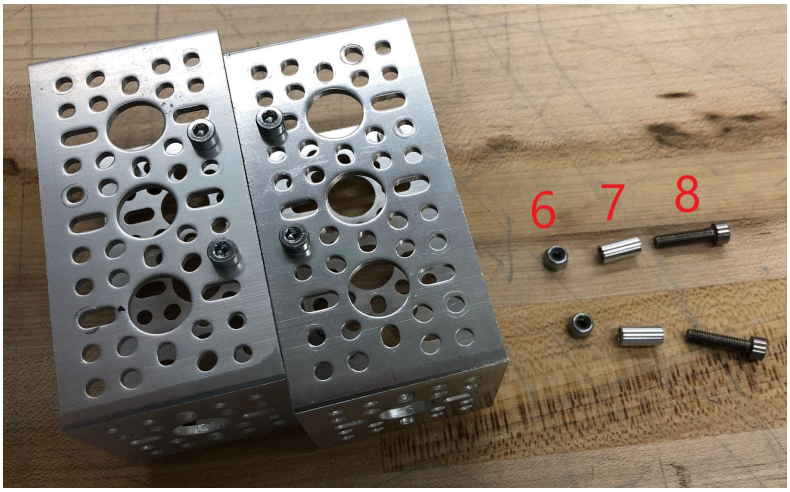
- **Step 3: Building and placing the hard stops**

  Parts used:

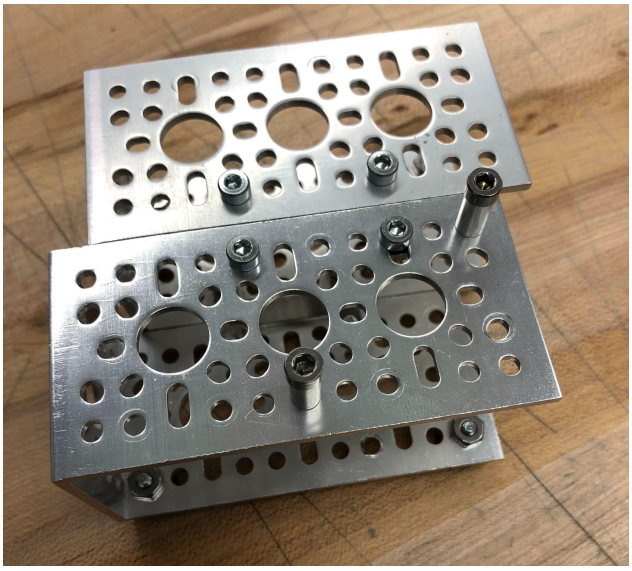
  6. Steel Nylon-Insert Locknut (M4 x 0.7 mm, 5 mm High) (x2)
  7. Aluminum Spacer (4mm ID, 6mm OD, 12mm Length) (x2)
  8. Steel Socket Head Screw (M4 x 0.7mm, 18mm Length) (x2)

  Written description:

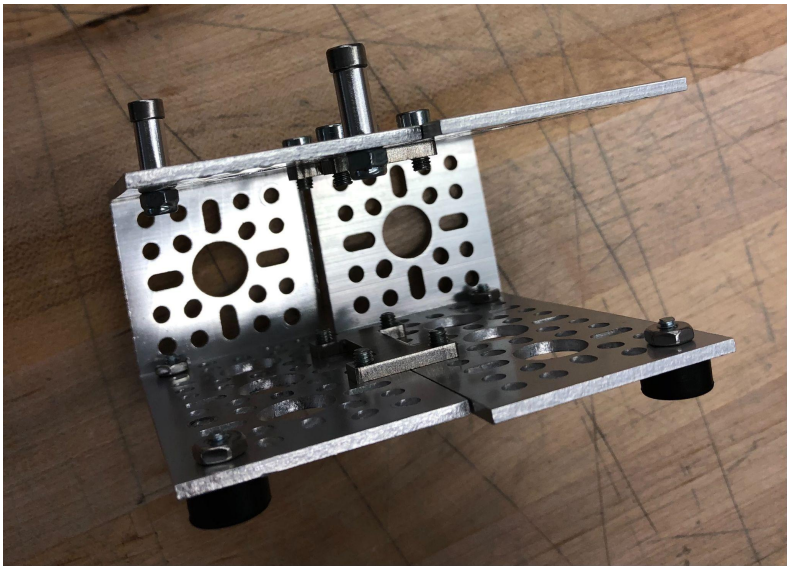
  First take the head screws (8) and put an aluminum spacer (7) on each head screw, and then place the head screws into the assigned holes on the bracket (from the rightmost picture). Then tighten the lock nuts (8) on the remaining exposed screw part of the head screw inside the large bracket surface. Ensure that the hard stops are both strongly secured to the main base.




a)



b)



c)

**Figure A.3**: a) Components before assembly; b) and c) Top and side views of completed step

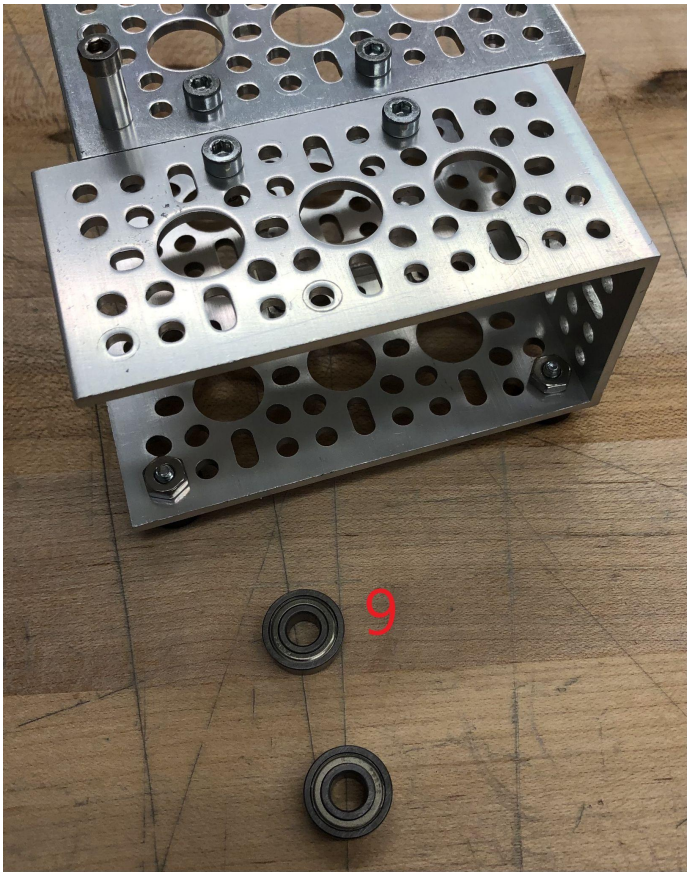
- **Step 4: Placing the ball bearings**

Parts used:

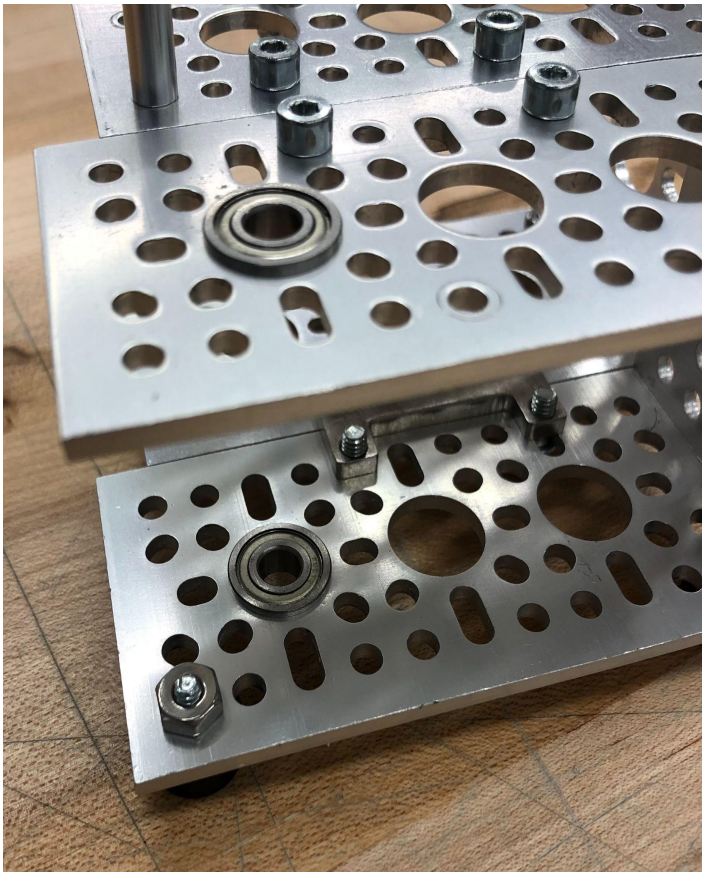
9. 1611 Series Flanged Ball Bearing (x2)

Written description:

First take the two ball bearings (9) and align them with the front right hole looking from the open end of the angle brackets. Now take the bearings and place them into the top and bottom holes with the flanges facing up. They will slide right in without needing much thrust force. Note that shaking the assembly or tilting it upside down may cause the bearings to fall out.




a)



b)

**Figure A.4**: a) Components before assembly; b) Completed assembly for this step

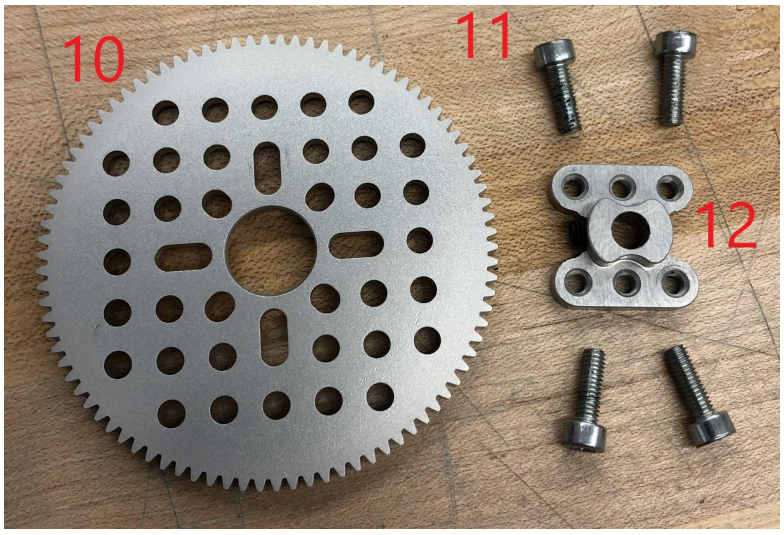
- **Step 5: Assembly of the input gear base**

Parts used:

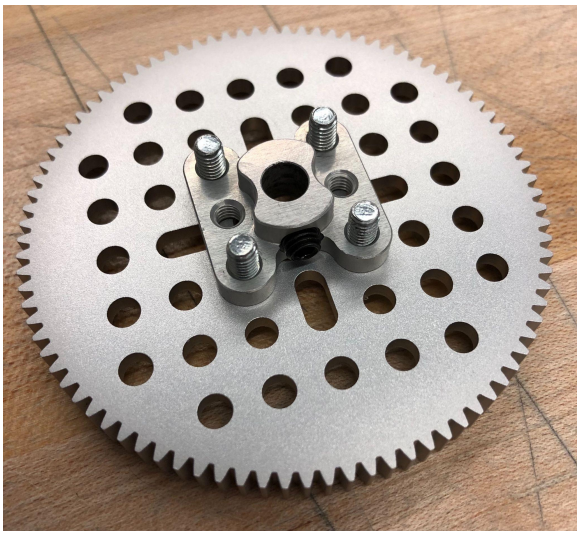
10. 90-Tooth Aluminum Gear (x1)
11. Steel Socket Head Screw (M4 x 0.7mm, 12mm Length) (x4)
12. Aluminum Set Screw Hub (x1)

Written description:

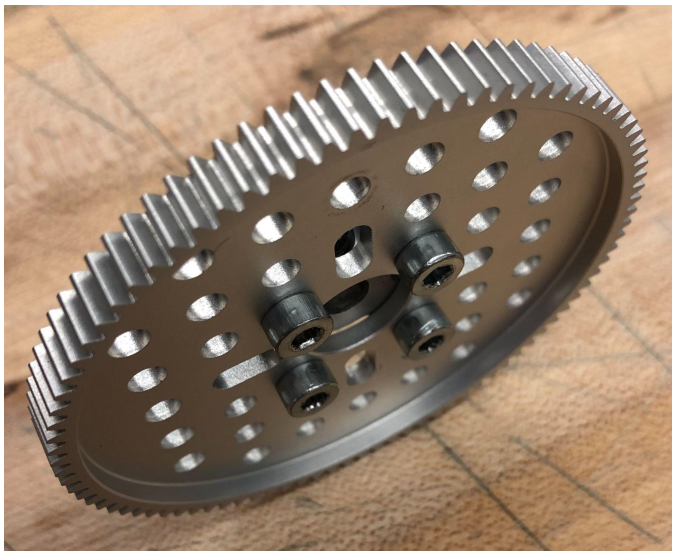
First place the 90-tooth aluminum gear (10) flat and then take the four screws (11) and the aluminum set screw hub (12). Now align the biggest hole on the gear to the set screw hub on the face of the gear with no indent from the teeth. Then align the screws how they are aligned from the picture coming from the indented side up through the set screw hub. Tighten each screw in a diagonal pattern so that the screw hub is secured flat against the gear about its center.




a)



b)



c)

**Figure A.5**: a) Components before assembly; b) and c) Top and side views of completed step

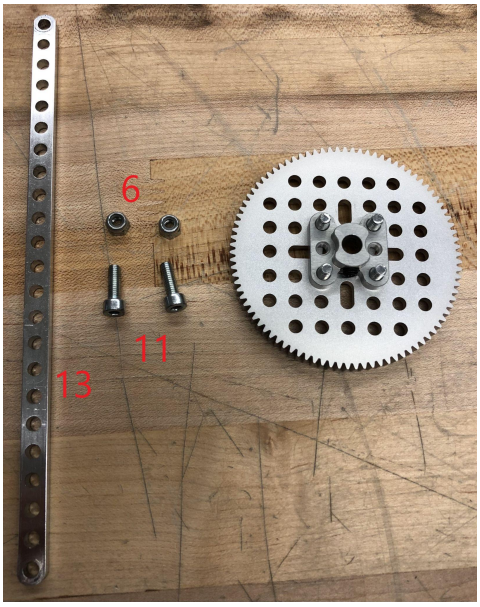
- **Step 6: Assembly of the input link on the gear base**

Parts used:

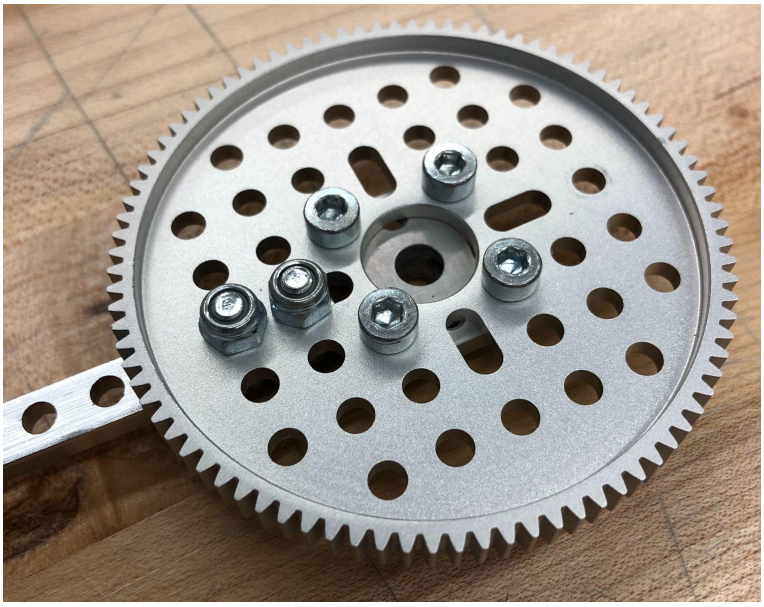
6. Steel Nylon-Insert Locknut (M4 x 0.7 mm, 5 mm High) (x2)
11. Steel Socket Head Screw (M4 x 0.7mm, 12mm Length) (x2)
13. 1102 Series Flat Beam (x1)

Written description:

First take the 1102 series flat beam (13) and align it with the curved edge of the set screw hub, and then put the two screws (11) through the link top on the flat side of the gear to exit out the indented side of the gear. Now screw the two nuts (6) onto each screw to lock the link into place.




a)



b)

c)

**Figure A.6**: a) Components before assembly; b) and c) Top and bottom views of completed step

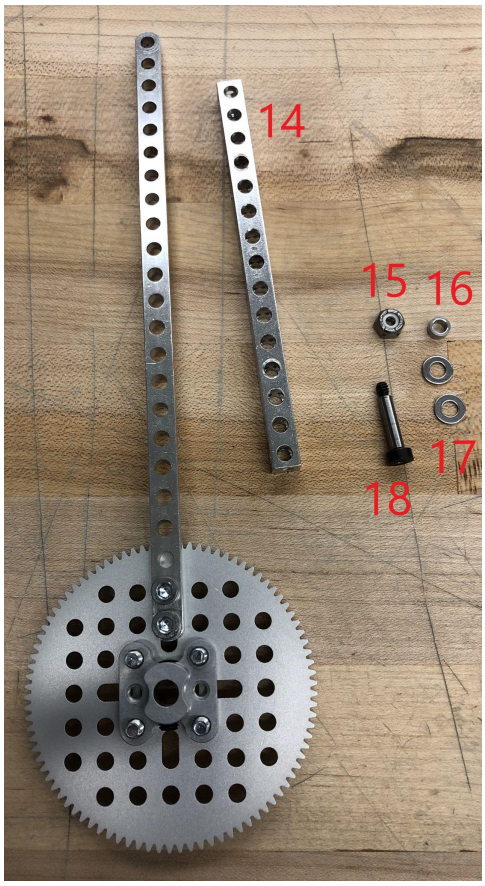
- **Step 7: Linking the coupler with the input link**

Parts used:

14. 1106 Series Square Beam (x1)
15. Steel Nylon-Insert Locknut (6-32, 11/64" High) (x1)
16. Aluminum Spacer (4mm ID, 6mm OD, 3mm Length) (x1)
17. 18-8 Steel Washer for M4 Screw Size, 4.3 mm ID, 9 mm OD (x2)
18. Steel Shoulder Screw (6-32, 5/8" Length, 5/32" Diameter) (x1)

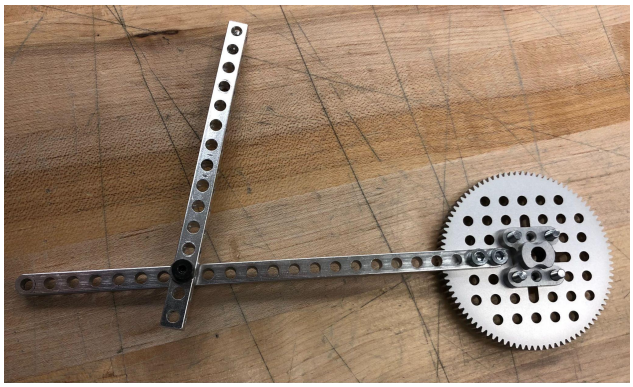
Written description:

First take the recently built part and the square beam (14) and place the screw (18) with two steel washers added (17) into the third hole from the square beam edge. Now take the aluminum spacer (16) and put it onto the screw. Now take the assembled part and put it into the eighth hole from the end farthest from the gear on the flat beam. Then take the lock nut (15) and screw it onto the threaded part of the screw through the input link. Make sure the stackup of the joint matches the bottom right image.




a)

b)



c)

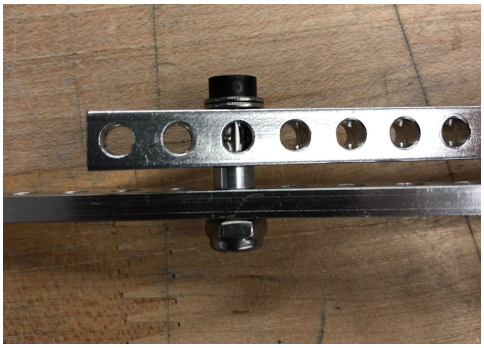


**Figure A.7**: a) Components before assembly; b) and c) Top and side views of completed step

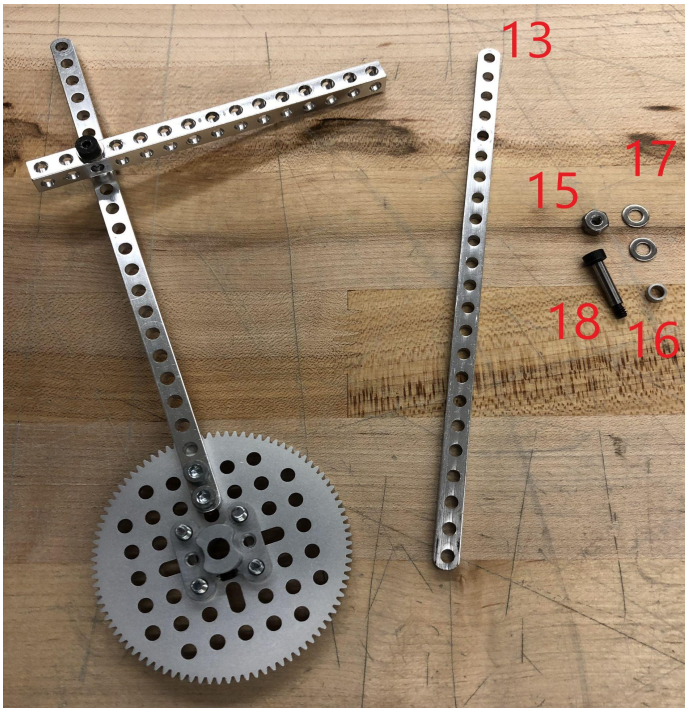
- **Step 8: Linking the follower to the linkages**

Parts used:

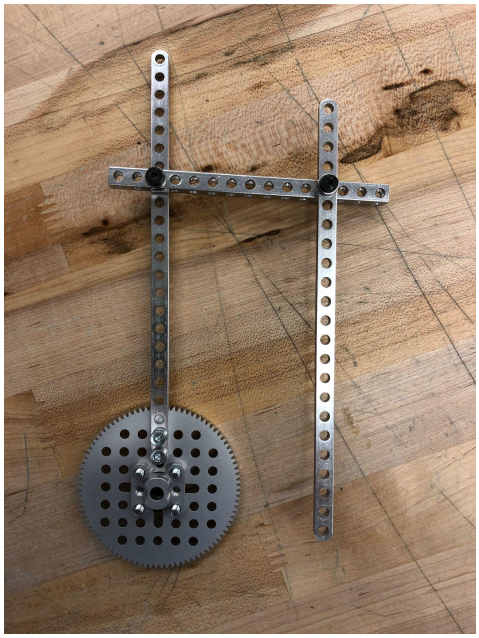
13. 1102 Series Flat Beam (x1)
15. Steel Nylon-Insert Locknut (6-32, 11/64" High) (x1)
16. Aluminum Spacer (4mm ID, 6mm OD, 3mm Length) (x1)
17. 18-8 Steel Washer for M4 Screw Size, 4.3 mm ID, 9 mm OD (x2)
18. Steel Shoulder Screw (6-32, 5/8" Length, 5/32" Diameter) (x1)

Written description:

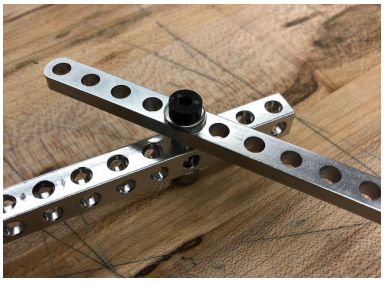
First take the screw (18) and add two steel washers (17) onto the screw. Then put the screw through the fifth hole from one end of the lone flat beam (13). Now add the aluminum spacer (16) to the screw and put it through the fourth hole from the right of the square beam end. Now put the lock nut (15) on the screw and tighten. Make sure the flat beam is on top of the coupler. Check that the joint stackup matches the image.




a)



b)



c)

**Figure A.8**: a) Components before assembly; b) and c) Top and side views of completed step

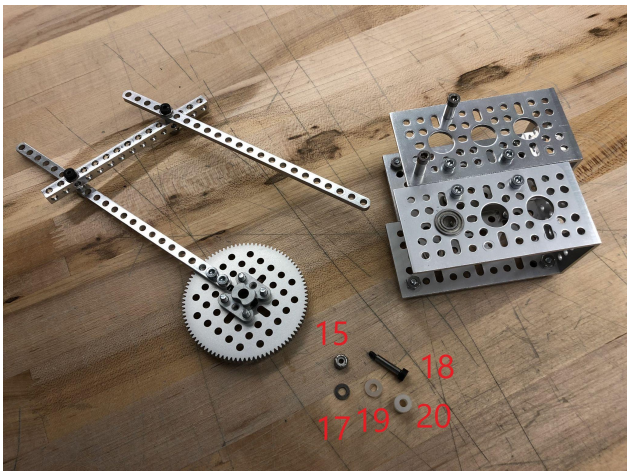
- **Step 9: Fixing the gear and the linkage to the brackets**

Parts used:

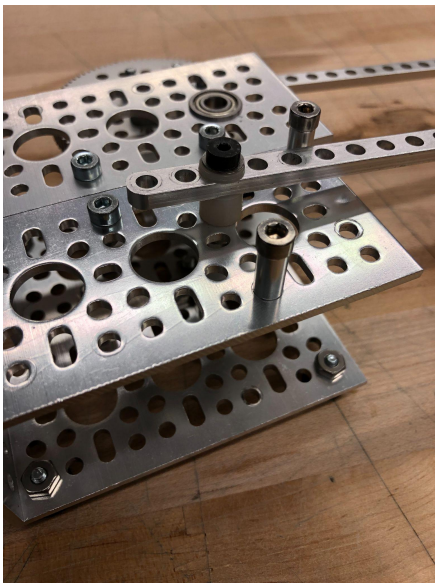
15. Steel Nylon-Insert Locknut (6-32, 11/64" High) (x1)
17.18-8 Steel Washer for M4 Screw Size, 4.3 mm ID, 9 mm OD (x1)
18. Steel Shoulder Screw (6-32, 5/8" Length, 5/32" Diameter) (x1)
19. Nylon Washer (0.173" ID, 1/16" Thick, 3/8" OD) (x1)
20. Nylon Spacer (0.166" ID, 9/32" Length, 3/8" OD) (x1)

Written description:

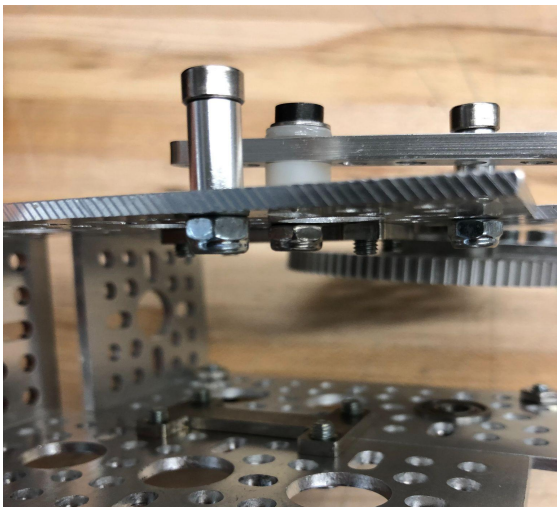
First take the newly assembled input, coupler, and follower links and put the shoulder screw (18), steel washer (17), and nylon washer (19) through the third hole sequentially from the follower link end. Now add the nylon spacer (20) to the screw assembly and put it into the assigned hole from the picture. Lastly add the locknut (15) to the screw assembly and tighten from below. Make sure the joint stackup and hole used matches the image below.




a)



b)



c)

**Figure A.9**: a) Components before assembly; b) and c) Top and side views of completed step

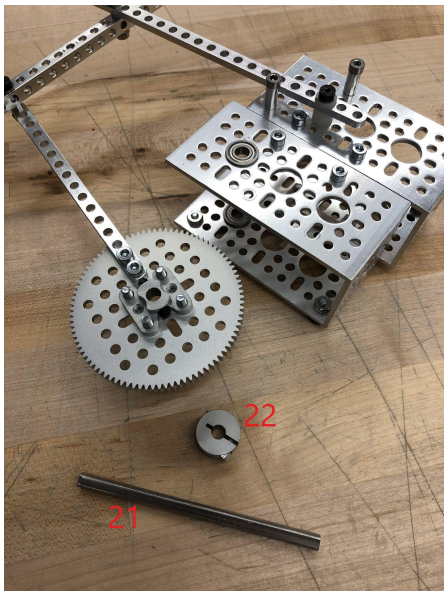
- **Step 10: Assembling the shaft**

Parts used:

21. 2100 Series Stainless Steel Round Shaft (6mm Diameter, 100mm Length) (x1)
22. 2910 Series Aluminum Clamping Collar (6mm ID x 19mm OD, 8mm Length) (x1)

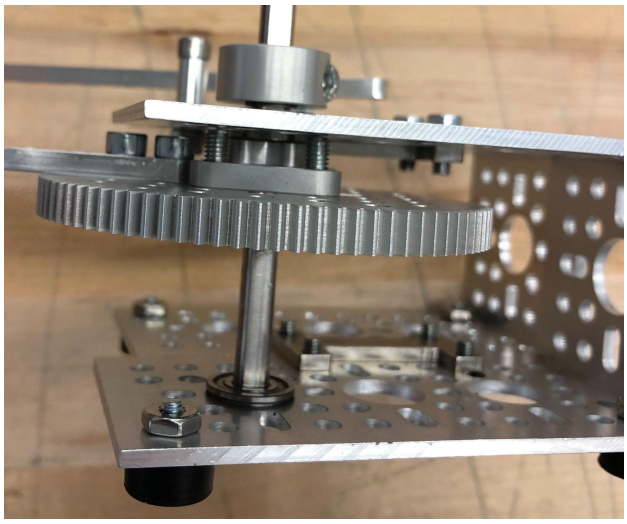
Written description:

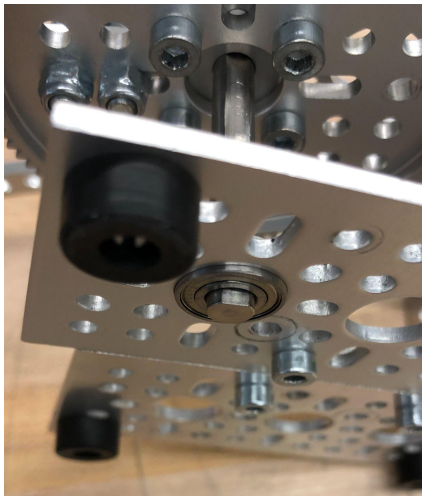
First take the input gear and align it with the two ball bearings so that the centers line up vertically. Next place the round steel shaft (21) through the two ball bearings and gear. Now put the clamping collar (22) on top of the round shaft. Next make sure that there is a little bit of the round shaft hanging through the bottom ball bearing and then tighten the clamping collar so that the shaft stays at that height. Next tighten the set screw hub so that the top of the set screw hub is close to the top of the angle pattern bracket. The heights where the collar and set hub are tightened should allow the linkage parts to remain horizontal and aligned.




a)

b)





c)

d)

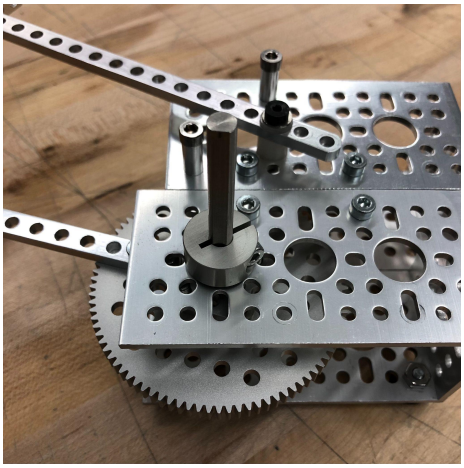


**Figure A.10**: a) Components before assembly; b) c), and d) Side, bottom, and top views of completed step

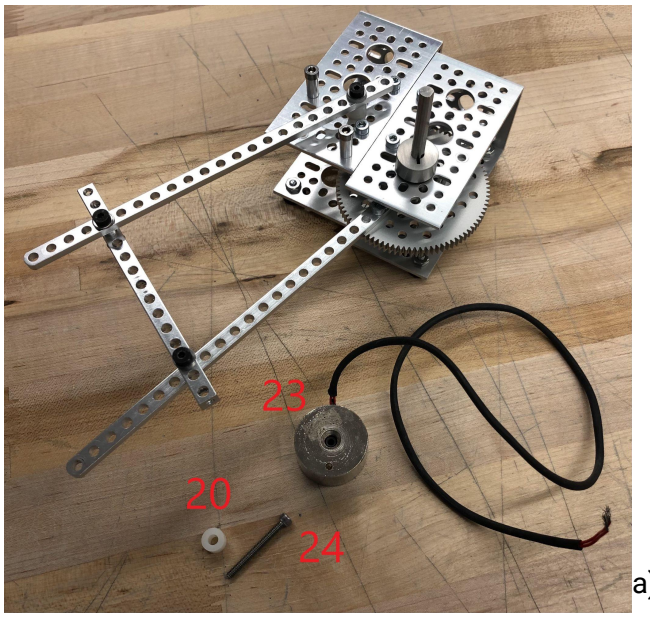
- **Step 11: Fixing on the electromagnet**

Parts used:

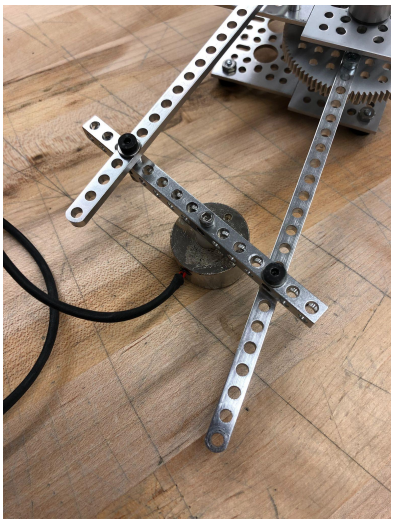
20. Nylon Spacer (0.166" ID, 9/32" Length, 3/8" OD) (x1)
23. Electromagnet (200N, R-1207-12) (x1)
24. Steel Socket Head Screw (6-32, 1-1/4" Length) (x1)

Written description:

Take the nylon spacer (20) and hold it under the fourth hole over from the input link and coupler link joint, and then take the long screw (24) and slide it through the same hole so that it passes through both the coupler link and the spacer. Then, take the electromagnet (23) and screw it onto the remaining end of the screw. Tighten down the screw by hand so the spacer and electromagnet are securely held onto the coupler link. The end result should match the pictures below.



a)


b)


c)

**Figure A.11**: a) Components before assembly; b) and c) Top and side views of completed step

- **Step 12: Finished design assembly (without transmission)**

  Parts used:
  None.

  Written description:
  The final linkage design assembly should at this point be completed. It should match the image below of the finished assembly. Take time to make sure that all parts are attached securely and that all joints in the linkage can move smoothly, adjusting them as needed. The finished linkage should be easy to swing though its range of motion by hand.

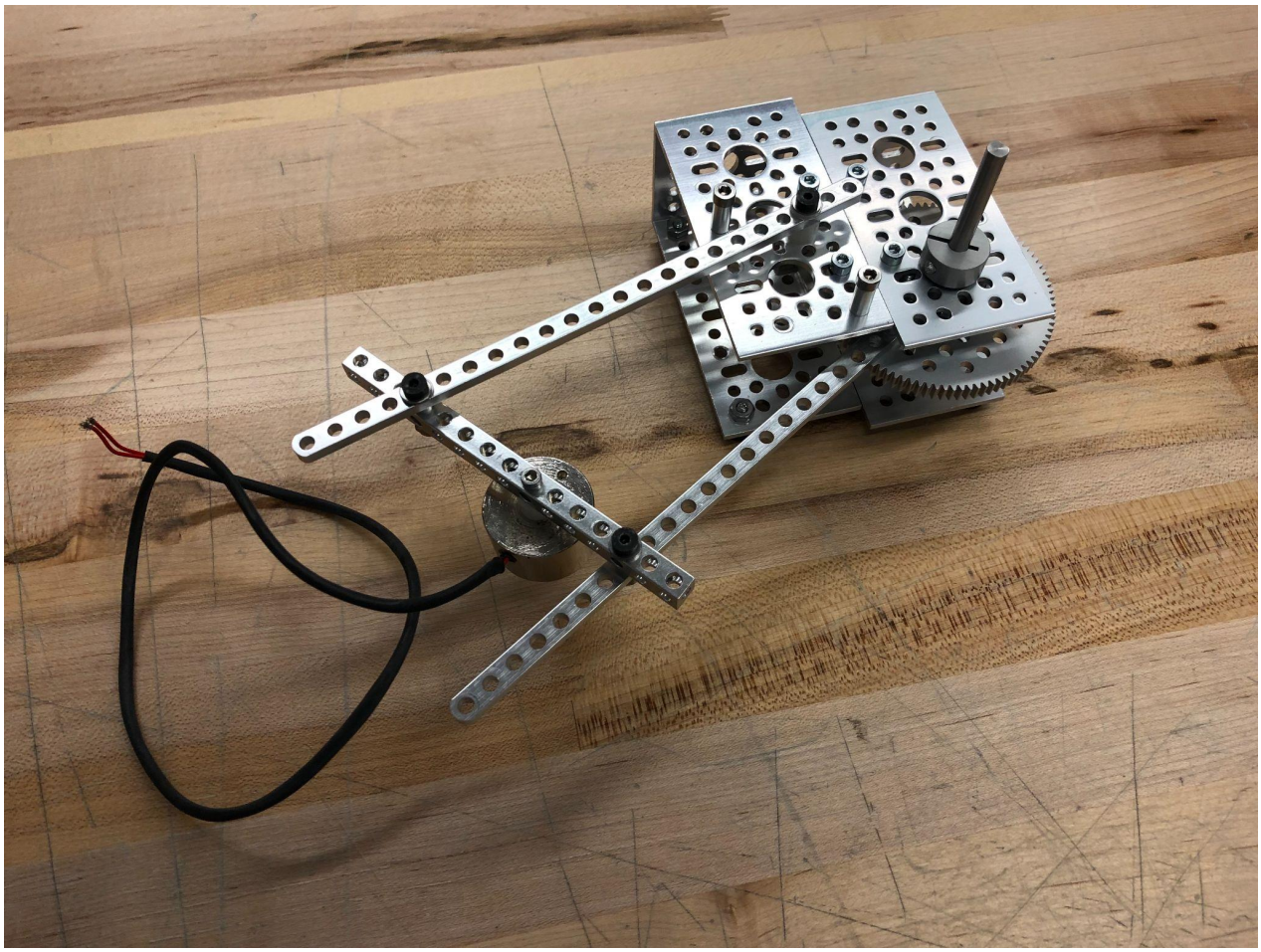


**Figure A.12**: Image of completed assembly (without transmission)

## Linkage Transmission and Motor Mounting

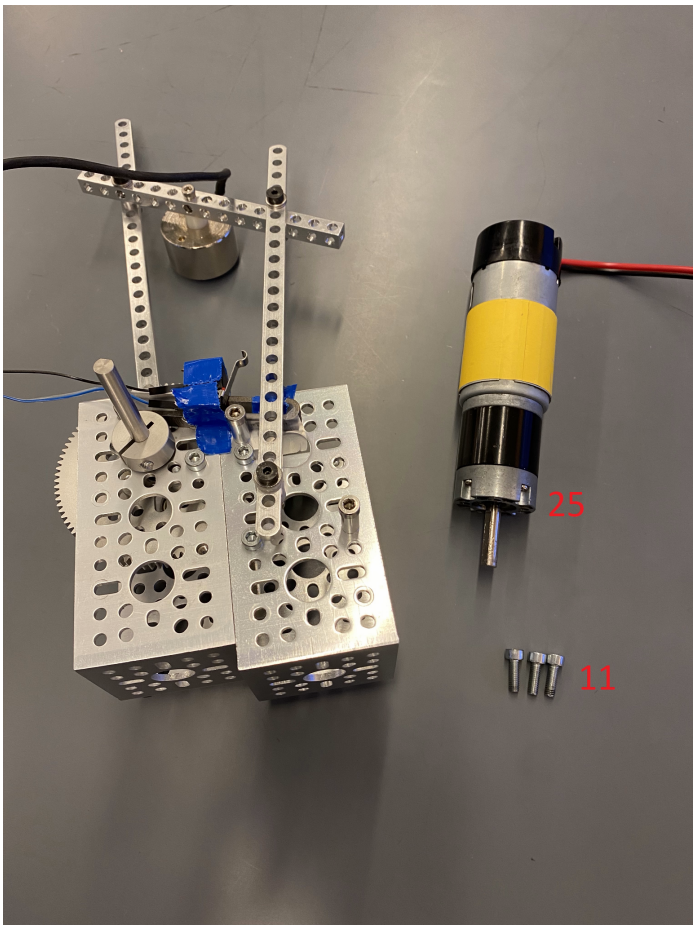
- **Step 13: Mount the motor**

Parts used:

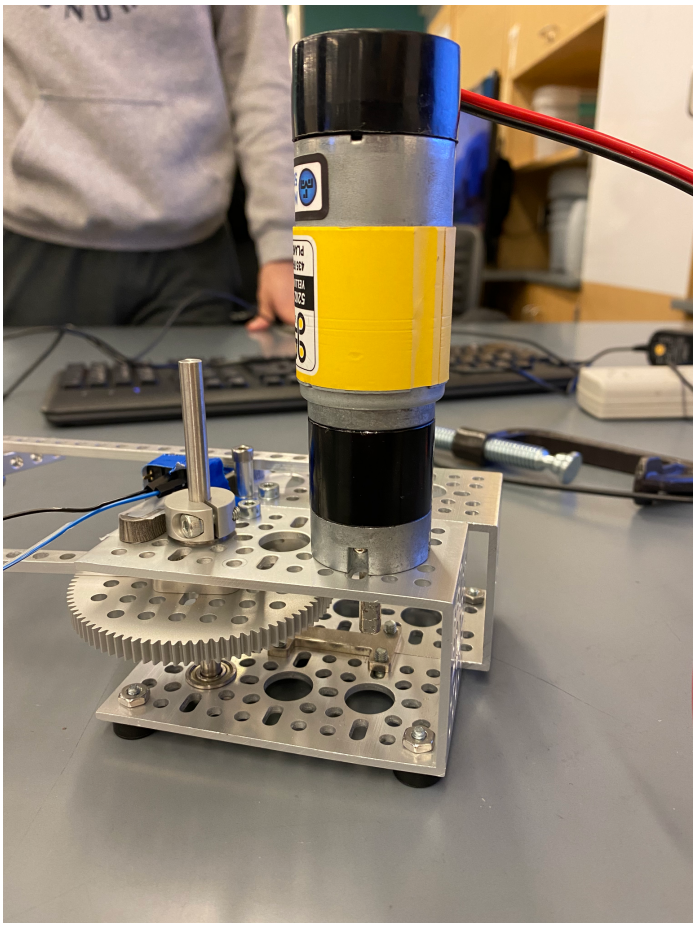
11. Steel Socket Head Screw (M4 x 0.7mm, 12mm Length) (x3)
25. 5202 Series Planetary Gear Motor (13.7:1 Ratio)

Written description :

First get the motor and align it like in the picture. To bolt in the motor (25) you will need to put the allen wrench through the bottom of the assembly to the motor and then put the bolt (11) on the allen wrench. Then align the bolt with the motor female bolt hole and bolt it in and repeat three times with the remaining bolts. (One of the female bolt holes was cross threaded so we could not bolt it in)




a)



b)

**Figure A.13 :** a) Components before assembly; b) side view of completed step

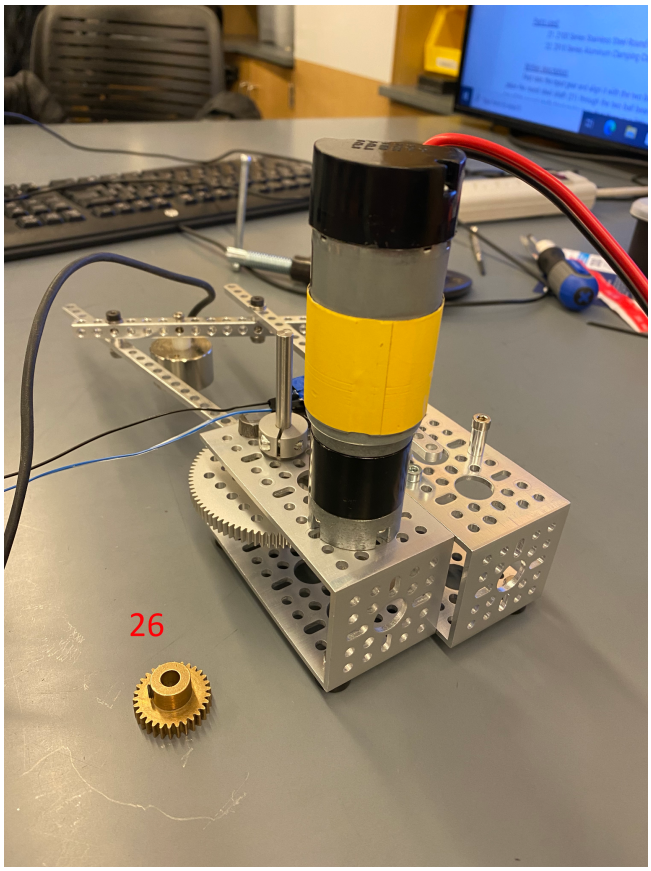
- **Step 14: Add pinion to motor shaft**

  Parts used:

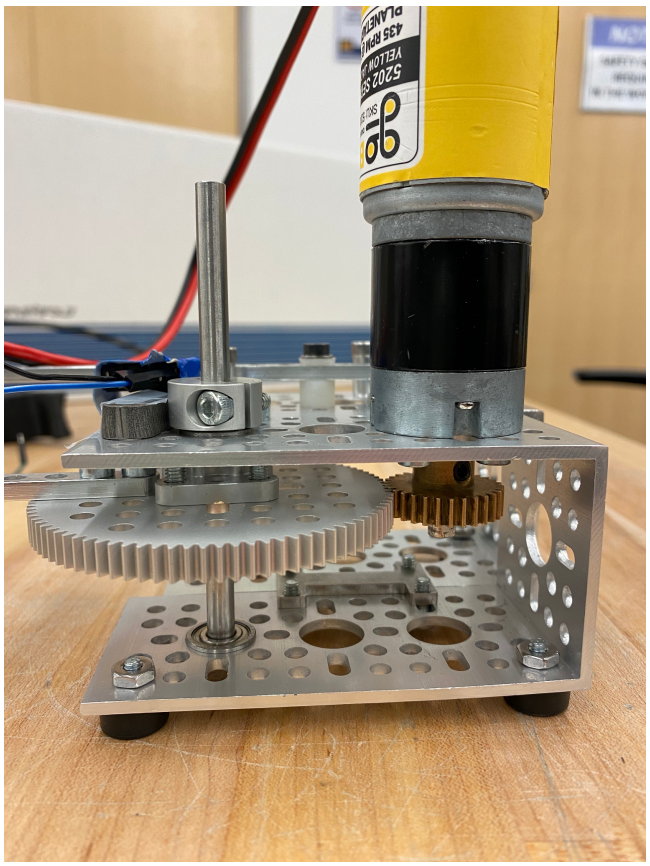
  26. 30-Tooth Brass Pinion Gear (6mm Bore, MOD 0.8, Set Screw)

  Written description:

  After securing the motor to the base, attach the pinion gear (26) to the motor with the flat side of the motor facing the set screw of the pinion. Then tighten until very snug.




a)



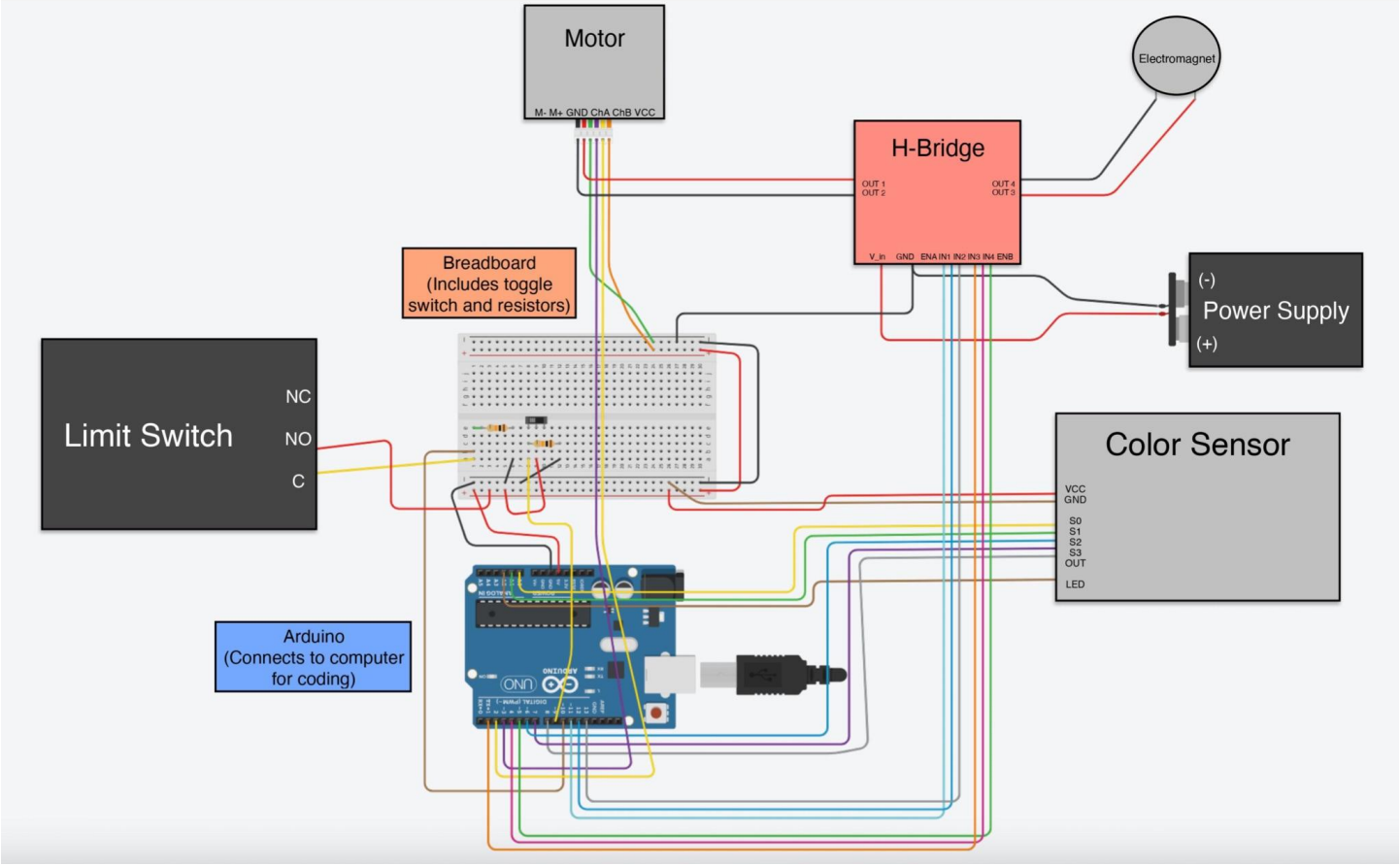
b)

**Figure A.14 :** a) Components before assembly; b) Side view of completed step

## Appendix B: Wiring Diagram and Calculations

### Wiring diagram

Below is a figure of the full and final wiring diagram for controlling and powering the movement of our team's linkage mechanism. All electronic components are included and their connectivity shown, though our physical linkage design is not shown. Instead components in the wiring diagram can be moved and placed as needed on or around the linkage for its operation. Also not pictured is a personal laptop/computer that is used to code and control the Arduino module. The computer used for this can vary and wires via a given Arduino to USB cable into the Arduino board as shown. Lastly, the power supply is set to the required voltage and plugs into a standard wall outlet.




Model Numbers:
Color Sensor - 9520
H-Bridge - L298N
Breadboard - 2183-4000-ND
Arduino Uno - 1050-1041-ND
Limit Switch - 255-6141-ND
Toggle Switch - EG1903-ND
Motor - 5202-0002-0014
Power Supply - N/A (Amazon)
Electromagnet - R-1207-12

**Calculations**

**Table B.1: Resolution analysis**

| Position | Minimum Angle [deg] | Maximum angle [deg] | Difference |
|---|---|---|---|
| Pickup Zone | 51.11 | 65.06 | **13.95** |
| Red Zone | 73.33 | 91.56 | **18.23** |
| Yellow Zone | 104.98 | 121.87 | **16.89** |
| Blue Zone | 138.73 | 153.42 | **14.69** |

$$Desired\ Count\ Range\ = \left(\frac{PPR}{1\,rev}\right) \cdot N_{internal} \cdot \left(\frac{smallest\ difference\ [deg]}{\frac{360\ degrees}{1\,rev}}\right) \cdot N$$

$$30\ = \left(\frac{28\ counts}{1\,rev}\right) \cdot 13.7 \cdot \left(\frac{13.95}{\frac{360\ degrees}{1\,rev}}\right) \cdot N$$

$$N\ = \frac{30}{\left(\frac{28\ counts}{1\,rev}\right) \cdot 13.7 \cdot \left(\frac{13.95}{\frac{360\ degrees}{1\,rev}}\right)}$$

$$N\ = 2.01823$$

## Appendix C: Arduino Variables and Data

**Table C**: Summary of Variables in Arduino Code

| Item | Parameter | Description | Value | Unit |
|---|---|---|---|---|
| **1** | *DISC_PICKUP_POSITION* | Encoder count corresponding to disc pickup ramp | 2 | counts |
| **2** | *RED_ZONE_POSITION* | Encoder count corresponding to red zone | 78 | counts |
| **3** | *BLUE_ZONE_POSITION* | Encoder count corresponding to blue zone | 292 | counts |
| **4** | *WAIT_POSITION* | Encoder count when waiting to pick a target | 2 | counts |
| **5** | YELLOW_ZONE_POSITION | Encoder count corresponding to yellow zone | 172 | counts |
| **6** | *FRICTION_COMP_VOLTAGE* | Motor voltage required to overcome friction | 3.4 | volts |
| **7** | *RED Disc, red range* | Range of values that color sensor receives for the red disc red count | 9000-13000 | µs |
| **8** | *RED Disc, green range* | Range of values that color sensor receives for the red disc green count | 5500-7500 | µs |
| **9** | *RED Disc, blue range* | Range of values that color sensor receives for the red disc blue count | 7800-9500 | µs |
| **10** | *RED Disc, clr range* | Range of values that color sensor receives for the red disc clr count | 23000-31000 | µs |
| **11** | *YELLOW Disc, red range* | Range of values that color sensor receives for the yellow disc red count | 10000-14500 | µs |
| **12** | *YELLOW  Disc, green range* | Range of values that color sensor receives for the yellow disc green count | 9000-12000 | µs |
| **13** | *YELLOW  Disc, blue range* | Range of values that color sensor receives for the yellow disc blue count | 8000-10000 | µs |
| **14** | *YELLOW  Disc, clr range* | Range of values that color sensor receives for the yellow disc clr count | 25000-33000 | µs |
| **15** | *BLUE Disc, red range* | Range of values that color sensor receives for the blue disc red count | 6500-8500 | µs |
| **16** | *BLUE  Disc, green range* | Range of values that color sensor receives for the blue disc green count | 7500-8200 | µs |
| **17** | *BLUE Disc, blue range* | Range of values that color sensor receives for the blue disc blue count | 10500-13000 | µs |
| **18** | *BLUE  Disc, clr range* | Range of values that color sensor receives for the blue disc clr count | 26500-34000 | µs |
| **19** | *KP* | **P**ID: Proportional gain | 0.242 | V/count |

| | | | | |
|---|---|---|---|---|
| **20** | *KI* | PI**D**: Integral gain | 0.00595 | V/ (count*s) |
| **21** | *KD* | PI**D**: Derivative gain | 0.01105 | V*s/ count |

## Appendix D: Arduino Code

```
// ME350 Disc Sorting Sketch - Version 2.3
// Based on Target Dropping Sketch V4.1 by Karthik Urs, Joe Saginaw, David Mark, Julia
Slatin
// updated 3-26-2021 by Michael Fisher: color sensor int->long fix, motor voltage cap

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

//----------------------------- State Machine: -----------------------------//
// CONSTANTS:
// Definition of states in the state machine
const int CALIBRATE               = 1;
const int PICKUP_DISC             = 2;
const int DETERMINE_DISC_COLOR    = 3;
const int MOVE_TO_ZONE            = 4;

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

//----------------------------- Color Sensor: -----------------------------//
// CONSTANTS:
// Definition of disc colors:
const int RED = 1;
const int YELLOW = 2;
const int BLUE = 3;
const int NONE = 4;

// VARIABLES:
// Stores output frequency from the color sensor
unsigned long red = 0;
unsigned long green = 0;
unsigned long blue = 0;
unsigned long clr = 0;

// THRESHOLDS:
// Color sensor data compared against these to determine disc color

// RED DISC BOUNDS
int RED_R_LB = 9000;  // Lower bound for red channel
int RED_R_UB = 13000;  // Upper bound for red channel
int RED_G_LB = 5500;  // Lower bound for green channel
int RED_G_UB = 7500;  // Upper bound for green channel
int RED_B_LB = 7800;  // Lower bound for blue channel
int RED_B_UB = 9500;  // Upper bound for blue channel
int RED_CL_LB = 23000; // Lower bound for clear channel
```

```
int RED_CL_UB = 31000; // Upper bound for clear channel

// YELLOW DISC BOUNDS
int YELLOW_R_LB = 10000;  // Lower bound for red channel
int YELLOW_R_UB = 14500;  // Upper bound for red channel
int YELLOW_G_LB = 9000;  // Lower bound for green channel
int YELLOW_G_UB = 12000;  // Upper bound for green channel
int YELLOW_B_LB = 8000;  // Lower bound for blue channel
int YELLOW_B_UB = 10000;  // Upper bound for blue channel
int YELLOW_CL_LB = 25000; // Lower bound for clear channel
int YELLOW_CL_UB = 33000; // Upper bound for clear channel

// BLUE DISC BOUNDS
int BLUE_R_LB = 6500;  // Lower bound for red channel
int BLUE_R_UB = 8500;  // Upper bound for red channel
int BLUE_G_LB = 7500;  // Lower bound for green channel
int BLUE_G_UB = 8200;  // Upper bound for green channel
int BLUE_B_LB = 10500;  // Lower bound for blue channel
int BLUE_B_UB = 13000;  // Upper bound for blue channel
int BLUE_CL_LB = 26500; // Lower bound for clear channel
int BLUE_CL_UB = 34000; // Upper bound for clear channel

//------------------------------ 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 long 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

//------------------------------ High-level behavior of the controller:
------------------------------//
// CONSTANTS:
// Target positions:
```

```
const int CALIBRATION_VOLTAGE     = -4;                   // [Volt] Motor voltage used
during the calibration process, set to be
                                                          //        positive or negative
based on limit switch position.
const int DISC_PICKUP_POSITION    = 2;                    // [encoder counts] Motor
position corresponding to disc pickup location
const int RED_ZONE_POSITION       = 78;                  // [encoder counts] Motor
position corresponding to the red zone
const int YELLOW_ZONE_POSITION    = 172;                  // [encoder counts] Motor
position corresponding to the yellow zone
const int BLUE_ZONE_POSITION      = 292;                  // [encoder counts] Motor
position corresponding to the blue zone
const int LOWER_BOUND             = 0;                    // [encoder counts] Position
of the left end stop
const int UPPER_BOUND             = -360;                 // [encoder counts] Position
of the right end stop
const int TARGET_BAND             = 10;                    // [encoder counts] "Close
enough" range when moving towards a target.

//------------------------------ PID Controller  ------------------------------//
// CONSTANTS:
const float KP                    = 0.242;  // [Volt / encoder counts] P-Gain
const float KI                    = 0.00595; // [Volt / (encoder counts * seconds)]
I-Gain
const float KD                    = 0.01105; // [Volt * seconds / encoder counts] D-Gain
const float SUPPLY_VOLTAGE        = 12;    // [Volt] Supply voltage at the HBridge
const float FRICTION_COMP_VOLTAGE = 3.4;   // [Volt] Voltage needed to overcome friction
const float MOTOR_VOLTAGE_CAP     = 11;    // [Volt] Max voltage that the can be
delivered to the motor

// 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

//------------------------------ Electromagnet: ------------------------------//
//CONSTANTS:
const float DESIRED_PICKUP_VOLTAGE = 12.0; // [Volts]
const float DESIRED_MOVE_VOLTAGE = 12.0;   // [Volts]
const int   EM_PICKUP_DURATION = 70;     // [milliseconds]
const int   EM_DROP_DURATION = 80;       // [milliseconds]

//VARIABLES:
```

```
int pwm_voltage;  // pwm voltage delivered to electromagnet (0 to 255)
int dropT = 0;    // current time when drop starts
int dropT_c = 0;  // current time while dropping
float decayV = 0; // sinusoidal voltage

//------------------------------ 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    = 9;  // Connected to toggle switch (turns mechanism on
and off)
const int PIN_NRL_LIMIT_SWITCH    = 10; // 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  = 12; // 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_NR_PWM_ELECMAG      = 5;  // Connected to H Bridge (controls electromagnet
voltage pwm)
const int PIN_NR_ELECMAG_DIR1     = 1;  // Connected to H Bridge (controls electromagnet
direction)
const int PIN_NR_ELECMAG_DIR2     = 4;  // Connected to H Bridge (controls electromagnet
direction)
const int PIN_COLORSENSE_S2       = 6;  // Connected to color sensor (determines color
filter)
const int PIN_COLORSENSE_S3       = 7;  // Connected to color sensor (determines color
filter)
const int PIN_COLORSENSE_OUT      = 8;  // Connected to color sensor (output signal from
sensor)
const int PIN_COLORSENSE_S0       = A0; // Connected to color sensor (determines
frequency scaling)
const int PIN_COLORSENSE_S1       = A1; // Connected to color sensor (determines
frequency scaling)
const int PIN_COLORSENSE_LED      = A2; // Connected to color sensor (turns LEDS on/off)

// 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:
  // Toggle and Limit Switch
  pinMode(PIN_NR_ON_OFF_SWITCH,    INPUT);
  pinMode(PIN_NRL_LIMIT_SWITCH,    INPUT);
  // Motor encoder and power
  pinMode(PIN_NR_ENCODER_A,        INPUT_PULLUP);
  pinMode(PIN_NR_ENCODER_B,        INPUT_PULLUP);
  pinMode(PIN_NR_PWM_OUTPUT,       OUTPUT);
  pinMode(PIN_NR_PWM_DIRECTION_1,  OUTPUT);
  pinMode(PIN_NR_PWM_DIRECTION_2,  OUTPUT);
  // Electromagnet
  pinMode(PIN_NR_ELECMAG_DIR1,     OUTPUT);
  pinMode(PIN_NR_ELECMAG_DIR2,     OUTPUT);
  pinMode(PIN_NR_PWM_ELECMAG,      OUTPUT);
  // Color Sensor
  pinMode(PIN_COLORSENSE_S2,       OUTPUT);
  pinMode(PIN_COLORSENSE_S3,       OUTPUT);
  pinMode(PIN_COLORSENSE_OUT,      INPUT);
  pinMode(PIN_COLORSENSE_S0,       OUTPUT);
  pinMode(PIN_COLORSENSE_S1,       OUTPUT);
  pinMode(PIN_COLORSENSE_LED,      OUTPUT);

  // 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

  // Color sensor settings
  digitalWrite(PIN_COLORSENSE_S0, HIGH); // Set to 20% scaling
  digitalWrite(PIN_COLORSENSE_S1, LOW);
  digitalWrite(PIN_COLORSENSE_LED, HIGH); // Turn on LEDS

  // 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);

  // Set duty cycle of electromagnet. mapped to 0-255
  analogWrite(PIN_NR_PWM_ELECMAG, 0);
  digitalWrite(PIN_NR_ELECMAG_DIR1, LOW);
  digitalWrite(PIN_NR_ELECMAG_DIR2, LOW);
```

```
  dropDisc();
}
//----------------------------- 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 = LOWER_BOUND;
```

```
        // Reset the error integrator:
        integralError = 0;
        // Calibration is finalized. Transition into DETERMINE_DISC_COLOR state
        Serial.println("State transition from CALIBRATE to DETERMINE_DISC_COLOR");
        state = PICKUP_DISC;
      }
      // Otherwise we continue calibrating
      break;

    //*************************************************************************//
    // In the PICKUP_DISC state, we move to the disc pickup location
    case PICKUP_DISC:
      // The target position is the disc pickup location
      Serial.println("Moving to disc pickup"); // Comment out later to reduce lag
      desiredPosition = DISC_PICKUP_POSITION;

      if ((motorPosition <= desiredPosition + TARGET_BAND) && (motorPosition >=
desiredPosition - TARGET_BAND) && motorVelocity==0) {
        //Transition into DETERMINE_DISC_COLOR
        Serial.println("State transition from PICKUP_DISC to DETERMINE_DISC_COLOR");
        state = DETERMINE_DISC_COLOR;
      }

      // Otherwise we continue moving towards the pickup position
      break;

    //*************************************************************************//
    // In the DETERMINE_DISC_COLOR state, we read the color sensor, lift the disc, and
then move to the appropriate zone
    case DETERMINE_DISC_COLOR:

      if((motorPosition <= desiredPosition + TARGET_BAND) && (motorPosition >=
desiredPosition - TARGET_BAND) && motorVelocity==0) {
      // We have reached the color sensor

        // Check for the disc color and output to serial:
        // If we get a color reading, set the position and move to that color zone
        // Consider taking an average of multiple readings
        delay(5);
        getColor();
        int discColor = evaluateColorSensor();
        Serial.print(" Disc color is: ");
        switch (discColor) {
           case RED:
              Serial.println("RED.");
              desiredPosition = RED_ZONE_POSITION;
              Serial.println("State transition from DETERMINE_DISC_COLOR to
MOVE_TO_ZONE");
              // We have reached the pickup location
              // Turn on the electromagnet to pick up a disc
              Serial.println("Electromagnet is ON");
```

```
            setEM(DESIRED_PICKUP_VOLTAGE);
            delay(EM_PICKUP_DURATION); // Wait for the electromagnet to pick up
            state = MOVE_TO_ZONE;
            break;

         case YELLOW:
            Serial.println("YELLOW.");
            desiredPosition = YELLOW_ZONE_POSITION;
            Serial.println("State transition from DETERMINE_DISC_COLOR to
MOVE_TO_ZONE");
            // Turn on the electromagnet to pick up a disc
            Serial.println("Electromagnet is ON");
            setEM(DESIRED_PICKUP_VOLTAGE);
            delay(EM_PICKUP_DURATION); // Wait for the electromagnet to pick up
            state = MOVE_TO_ZONE;
            break;

         case BLUE:
            Serial.println("BLUE.");
            desiredPosition = BLUE_ZONE_POSITION;
            Serial.println("State transition from DETERMINE_DISC_COLOR to
MOVE_TO_ZONE");
            // Turn on the electromagnet to pick up a disc
            Serial.println("Electromagnet is ON");
            setEM(DESIRED_PICKUP_VOLTAGE);
            delay(EM_PICKUP_DURATION); // Wait for the electromagnet to pick up
            state = MOVE_TO_ZONE;
            break;

         case NONE:  //Serial.println("NONE.");
            Serial.println("Electromagnet is OFF");
            digitalWrite(PIN_NR_ELECMAG_DIR1, LOW);
            digitalWrite(PIN_NR_ELECMAG_DIR2, LOW);
            break;
      }
    }

    // Otherwise, we stay in DETERMINE_DISC_COLOR
    break;

  //*********************************************************************//
  // In the MOVE_TO_ZONE state, we select an color zone and move toward it, or
  // move toward Target 3 (a default position) if there is no active target
  case MOVE_TO_ZONE:
    setEM(DESIRED_MOVE_VOLTAGE);

    //Comment out later to reduce lag
    if (desiredPosition == RED_ZONE_POSITION) {
      Serial.println("Moving to Red Zone");
    }
    else if (desiredPosition == YELLOW_ZONE_POSITION) {
```

```
        Serial.println("Moving to Yellow Zone");
      }
      else if (desiredPosition == BLUE_ZONE_POSITION) {
        Serial.println("Moving to Blue Zone");
      }

      if ((motorPosition <= desiredPosition + TARGET_BAND) && (motorPosition >=
desiredPosition - TARGET_BAND) && motorVelocity==0) {
        // We have reached the desired disc drop off zone

        // Turn the electromagnet off to drop the disc
        Serial.println("Arrived at drop off zone, drop disc");
        dropDisc();
        delay (30); // Wait 1 second for the disc to drop

        Serial.println("State transition from MOVE_TO_ZONE to PICKUP_DISC");
        state = PICKUP_DISC;
      }
      // Otherwise, we keep moving to the disc drop off zone
      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 = LOWER_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;

    //** Friction Compensation 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;
```

```
    //Also turn off the electromagnet
    digitalWrite(PIN_NR_ELECMAG_DIR1, LOW);
    digitalWrite(PIN_NR_ELECMAG_DIR2, LOW);

    // Produce some debugging output:
    Serial.println("The toggle switch is off.  Motor Stopped. Electromagnet off. ");
  }
  // End of  else onOffSwitch==HIGH

  //** Send signal to motor **//
  // Convert from voltage to PWM cycle:
  motorCommand = int(abs(desiredVoltage * 255 / SUPPLY_VOLTAGE));
  // Clip values larger than motor voltage cap
  if (abs(desiredVoltage) > abs(MOTOR_VOLTAGE_CAP)) {
    motorCommand = 255*(MOTOR_VOLTAGE_CAP/SUPPLY_VOLTAGE);
  }
  // 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; MOVE_TO_DISC_PICKUP = 2;
DETERMINE_DISC_COLOR = 3; MOVE_TO_ZONE = 4]: ");
  Serial.print("State#: ");
  Serial.print(state);

  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("      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);


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


////////////////////////////////////////////////////////////////////
// This function returns the RGB and Clr values read by           //
// the color sensor.                                              //
////////////////////////////////////////////////////////////////////
int getColor(){
  // Setting RED (R) filtered photodiodes to be read
  digitalWrite(PIN_COLORSENSE_S2,LOW);  digitalWrite(PIN_COLORSENSE_S3,LOW);
  red = 1000000/pulseIn(PIN_COLORSENSE_OUT, LOW); // [micro sec] Time of one pulse from
sensor
  Serial.print("R = ");  Serial.print(red); // Print red frequency value
  delay(30);

  // Setting GREEN (G) filtered photodiodes to be read
  digitalWrite(PIN_COLORSENSE_S2,HIGH);  digitalWrite(PIN_COLORSENSE_S3,HIGH);
  green = 1000000/pulseIn(PIN_COLORSENSE_OUT, LOW); // [micro sec] Time of one pulse from
sensor
  Serial.print(" G = ");  Serial.print(green); // Print green frequency value
  delay(30);

  // Setting BLUE (B) filtered photodiodes to be read
  digitalWrite(PIN_COLORSENSE_S2,LOW);  digitalWrite(PIN_COLORSENSE_S3,HIGH);
  blue = 1000000/pulseIn(PIN_COLORSENSE_OUT, LOW); // [micro sec] Time of one pulse from
sensor
  Serial.print(" B = ");  Serial.print(blue); // Print blue frequency value
  delay(30);

  // Setting CLEAR (Cl) filtered photodiodes to be read
  digitalWrite(PIN_COLORSENSE_S2,HIGH);  digitalWrite(PIN_COLORSENSE_S3,LOW);
```

```
  clr = 1000000/pulseIn(PIN_COLORSENSE_OUT, LOW); // [micro sec] Time of one pulse from
sensor
  Serial.print(" Clr = ");  Serial.print(clr); // Print clear frequency value
  delay(30);
}
//------------------------------ End of function getColor
------------------------------//

////////////////////////////////////////////////////////////////////
// This function returns the color of the disc that is read         //
// based on the color calibration ranges set by the student         //
////////////////////////////////////////////////////////////////////
int evaluateColorSensor(){
  // initialize disc type with 'NONE'.  Override later if a disc color was detected
  int discColor = NONE;

  // Check if the disc is RED
  if ((red > RED_R_LB) && (red < RED_R_UB) && (green > RED_G_LB) && (green < RED_G_UB) &&
(blue > RED_B_LB) && (blue < RED_B_UB) && (clr > RED_CL_LB) && (clr < RED_CL_UB)){
    discColor = RED;
  }
  // Check if the disc is YELLOW
  if ((red > YELLOW_R_LB) && (red < YELLOW_R_UB) && (green > YELLOW_G_LB) && (green <
YELLOW_G_UB) && (blue > YELLOW_B_LB) && (blue < YELLOW_B_UB) && (clr > YELLOW_CL_LB) &&
(clr < YELLOW_CL_UB)){
    discColor = YELLOW;
  }
  // Check if the disc is BLUE
  if ((red > BLUE_R_LB) && (red < BLUE_R_UB) && (green > BLUE_G_LB) && (green <
BLUE_G_UB) && (blue > BLUE_B_LB) && (blue < BLUE_B_UB) && (clr > BLUE_CL_LB) && (clr <
BLUE_CL_UB)){
    discColor = BLUE;
  }

  // Note: If none of these if-statements is true, discColor remains 'NONE'
  return discColor;
}
//------------------------------ End of function evaluateColorSensor()
------------------------------//


////////////////////////////////////////////////////////////////////
// This function makes the electromagnet drop the disc using        //
// a sinusoidally decaying function to control the voltage          //
////////////////////////////////////////////////////////////////////
void dropDisc() {
  digitalWrite(PIN_NR_PWM_DIRECTION_1,LOW);
  digitalWrite(PIN_NR_PWM_DIRECTION_2,LOW);
  dropT = millis();
  dropT_c = millis();
  while (dropT_c - dropT < EM_DROP_DURATION) {
```

```
    decayV = DESIRED_PICKUP_VOLTAGE*exp(-(dropT_c-dropT)/60)*sin((dropT_c-dropT)/6);
    setEM(decayV);
    Serial.println("Dropping");
    dropT_c = millis();
  }
}
//------------------------------ End of function dropDisc()
-------------------------------//


////////////////////////////////////////////////////////////////////
// This function sets the voltage of the electromagnet based        //
////////////////////////////////////////////////////////////////////
void setEM(float involt) {
  int prev_volt = pwm_voltage;
  pwm_voltage = int(abs(involt/SUPPLY_VOLTAGE*255));
  if (prev_volt == pwm_voltage) return;
  analogWrite(PIN_NR_PWM_ELECMAG, pwm_voltage);
  if (involt > 0){
    digitalWrite(PIN_NR_ELECMAG_DIR1, HIGH);
    digitalWrite(PIN_NR_ELECMAG_DIR2, LOW);
  }
  else if (involt < 0){
    digitalWrite(PIN_NR_ELECMAG_DIR1, LOW);
    digitalWrite(PIN_NR_ELECMAG_DIR2, HIGH);
  }
  else {
    digitalWrite(PIN_NR_ELECMAG_DIR1, LOW);
    digitalWrite(PIN_NR_ELECMAG_DIR2, LOW);
  }
}
//------------------------------ End of function setEM() ------------------------------//
```