Final Report

RBE 2001 C02

Authors:

Jason Conklin **Teresa Saddler** **Benjamin Ward**

Submitted On: March 1, 2019

Abstract

An autonomous robotic system was designed and constructed to collect and replace solar panel simulacrum, constructed of 6061 aluminum and PLA plastic, on a roof structure, at various angles. The robot was required to respond to emergency start and fault-clearing commands, and to receive information from the field simulator about solar panel status.

A 4-bar linkage was selected as the primary actuation mechanism due to its reduced torque requirements compared to a simple arm and its restricted and well-defined path of motion. A drivetrain with two-wheel-drive and a ball castor was chosen for controllability, since it neatly centered the virtual turning center of the robot between the two driven wheels. For both mechanisms, transmissions were selected to maximize power delivered to the output and ensure controllable linear and angular speeds.

Total power draw was restricted based upon physical battery limitations, and total operation time was targeted at a minimum of 10 minutes.

Overall robot design and construction proceeded taking into consideration physical constraints, material and component cost, and cost of machining time. Material and machining costs were minimized via use of simple 2D and 3D CNC fabrication methods (laser cutting and 3d printing).

Programming proceeded by taking into account the overall task requirements, utilizing a flow diagram to organize common tasks into repeatable code blocks. Various sensors, including encoders and an infrared reflectance sensor array, were used to supplement and integrate knowledge about the robot's environment into the system.

Successful robot completion occurred, taking into account an extended deadline. All major tasks were completed in full. "Nice-to-have" tasks were not completed.

Table of Contents

List of Figures

List of Tables

1 Introduction

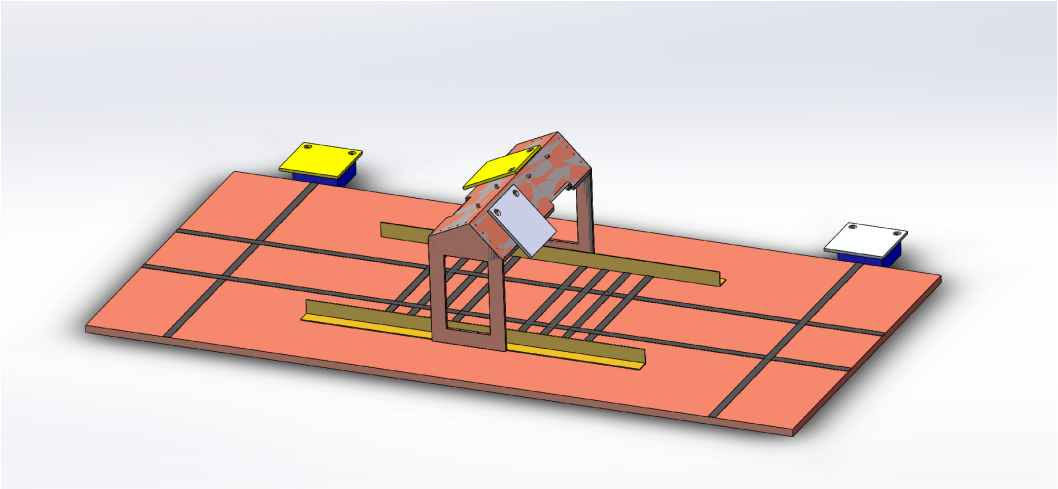


Figure 1: The full challenge field in one possible starting configuration

In this project we were required to design and create an autonomous robotic system which was able to place solar collector panels, represented by aluminum and plastic plates, on a roof structure at angles of 25 and 45 degrees and with a clearance height of 12.087in. The roof structure had two spots on each side, one empty and one populated with one of the solar collector panels, resulting in a starting configuration such as the one seen in Figure 1. Also located on the field are black tape lines in the positions represented in Figure 1 and guidance rails leading to the roof structure from either staging area. The robot was limited to the use of four motors, and was required to use a four-bar mechanism. The robot was required to autonomously collect a solar collector panel from one staging area and place it on the empty spot on the roof, then remove the solar collector panel on the other spot and place it in the staging area. This process had to be done for each side of the roof structure, with an optional goal of being able to carry out this routine without resetting the robot on the other side of the field by driving the robot under the roof structure.

While this task was carried out autonomously, the robot had to be able to respond to emergency start and restart commands and also to receive information on which spots on the roof structure were empty, the angle of the roof on the starting side, and the type of

collector on the starting side. Additionally, the robot had to be able to receive information about when it was safe to release and grip solar collector panels on the roof structure. These commands all had to be communicated via Wi-Fi from the field controller application.

2 Methodology

We began by creating a specific strategy to narrow the constraints for the robot design, and by clarifying parameters on the challenge, such as robot size and time limits. To determine our strategy, we studied the possible points we could achieve and assessed any trade-offs with complications the different challenge elements could introduce. After studying these trade studies, we decided on our main priorities and our additional objectives.

From there, we designed our four-bar linkage using the process of graphical linkage synthesis, wherein we used the provided field CAD models to determine the joint locations for the four-bar linkage. This was done by constraining the chosen grabbing mechanism cross-section to the required positions, specifically in positions that allowed it to place the solar collector plates on the staging area, and both sides of the roof structure. We then made construction lines connecting the joints of the grabber in each of the three positions and constructed perpendicular bisectors from these connecting lines. Where the bisectors from corresponding joints intersected was where the crank and follower joints would be located on the robot. Using this process, we were able to identify optimal joint locations and crank and follower link lengths.

Using this four-bar design, we did a force analysis on both the lifting and gripping mechanisms in maximum torque positions. To do this, we created free body diagrams of both the grabber mechanism and the four-bar mechanism and using the known forces acting on both mechanisms, found the forces on the joints and the required torques on the cranks. Using that required torque, we were able to calculate the forces acting on the gear teeth in the transmission of the four-bar linkage, and the torque requirements of the motors controlling the four-bar linkage and the grabber mechanism. Further, using the the power curve of the motor controlling the four-bar linkage, we were able to determine the speed of the motor, and therefore the speeds of the linkage components, when in the maximum

torque position.

Once we had verified our four-bar linkage design, we were able to impose further constraints on our drive system, which enabled us to fit our drivetrain around the physical linkage dimensions. We created a design and were able to analyze the maximum speed of our transmission considering the motor free speed and transmission efficiency, and also assess the power requirement of the motor at slip and stall.

Then, based on the mechanical systems on our robot, we selected appropriate sensors necessary to complete the task. Using our selected sensors and the previous calculations for the power requirements of the drivetrain, we were then able to calculate both the current and power requirements at steady-state and peak conditions for the system. Current and power draw of electronic components at steady-state and peak conditions were determined using knowledge of use and published information/specifications of the electronic components.

With the physical design completed and verified, we finalized the the CAD representation of the robot to ensure that the center of mass in all conditions remained in a stable position. To do this, we simply ensured that all component masses were correct and had SolidWorks determine the center of mass of the robot in all necessary positions, particularly in the maximum reach position of the aluminum solar collection panel.

Finally, based on the mechanical capabilities of the robot design, and our sensor selections, as well as the project requirements, we designed a state machine for programming the robot tasks. To do this, we determined at the lowest level the different "states" of the robot, or in other words, the different combinations of actions that needed to be triggered at any given moment in the robot operation. We then determined how these states related, and created a graphical representation of the state machine for guidance when programming showing the different states with arrows denoting their relationships.

3 Analysis

We carried out a trade study based on the different objectives of the project in order to hone our strategy. We based our trade study on the robot demo rubric provided to us, excluding the "Overall Design" and and "Innovation/Creativity" categories, a summary of which can

Table 1: Point distribution of challenge

Category	Description	Weighted Points
Reliability - Mechanical	Rate the reliability of the mechanical aspects of the robot.Solidly made and working well would rate a full score. Constantly jamming and/or falling apart would rate a 0.	15
Reliability - Control Program	Rate the reliability of the control (software) aspects of the robot. This is from the observed behaviors.	15
Mech - Dynamics	How well were forces/torques accounted for? Does the design indicate an awareness of and attempt to deal with forces, moment arms, etc.?	15
Mech - Gripper Design & Operation	Is the gripper design suitable for the task? How well does the gripper work? Does the gripper design detract from its operation?	15
Electrical - Neatness	Has there been an attempt to tidy up the wiring and keep it neat? Labelling wires should add to this score (but not detract if not present).	15
Line Following/Position Accuracy	If the robot tracks well, makes accurate turns, and positions itself well, then award a full score. If it can't stay on course or make an accurate turn no matter what, then award a 0.	30
Starting Info Communication	Robot is able to receive empty spot location, roof angle, and collector type information from field	10
Emergency Stop Command	Robot is able to receive emergency stop command and resume interrupted task	10
Collector Manipulation	Robot is able to manipulate both types of collectors	40
Staging Area Pick Up	Robot is able to pick up new collector from staging area	10
25 Degree Placement	Robot is able to place new collector on 25-degree roof with required pause before releasing	10
25 Degree Pick Up	Robot is able to remove old collector from 25-degree roof (with required pause before removing once gripper has closed on plate)	30
45 Degree Placement	Robot is able to place new collector on 45-degree roof with required pause before releasing	30
45 Degree Pick Up	Robot is able to remove old collector on 45-degree roof (with required pause before removing once gripper has closed on plate)	20
No Reset	Bonus points for completing both sides of the field in one autonomous mission	50

be found in Table 1. Using this point distribution, we were able to identify our priorities and decide whether objectives introduced undue complexity that made them undesirable and not worth their potential points.

Based on the point distribution, we made the decision to prioritize making as simple a robot as possible to complete the course in full. We also decided to aim for a robot that could complete the course without reset, because we did not anticipate that it would over-complicate the design or interfere with other design objectives; however, it was only a secondary objective since getting these bonus points required being able to complete the other course-related objectives.

After deciding our strategy, we began designing our four-bar linkage. To do this, we constructed our grabber geometry, based upon the "Bottom Out Grabber" design provided to us (which can be seen in Figure 2), in a sketch on top of the field CAD provided to us. We made three of these grabber sketches, and constrained them each to positions in which the robot needs to grip plates, specifically where the plate is on the 25 degree roof, the 45 degree roof and the staging area. We then connected the joints of these sketches using construction lines, and constructed perpendicular bisectors of these lines. The remaining joints for the four-bar linkage were located at the intersection of the corresponding bisectors. We then found an optimal configuration in which both of the found joints remained below the lowest point of the roof, as seen in Figure 3. Using these joints, we were able to both find the link lengths, which were 7.73" for the top link and 6.18" for the bottom link, and the required height of the grounded joints above the field.

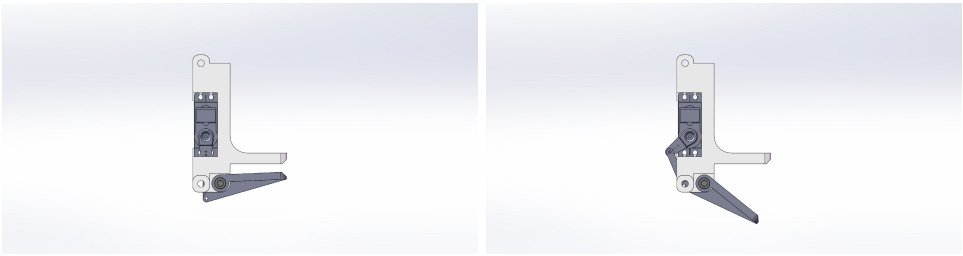


Figure 2: Grabber in Open and Closed Positions

We then did a force analysis of a four-bar linkage in this configuration with the bottom link positioned horizontal to the base plate. To do this, we created a free body diagram of the

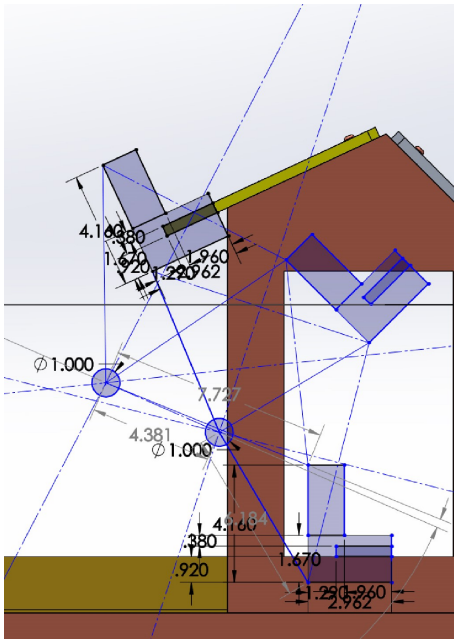


Figure 3: Graphical linkage synthesis results in SolidWorks

entire system as seen in Figure 4, and derived equations of equilibrium for the system. Then, we broke the system down into smaller components and carried out the same process. The calculations in full can be found in Appendix 2. Using our generated system of equations, we were able to find the moment around the crank joint, τ_{lift}, and the forces on each of the links, A_{xl}, A_{yl}, B_{xl}, B_{yl}, C_{xl}, C_{yl}, D_{xl} and D_{yl}:

$$A_{xl} = -2.41\ lbf \quad A_{yl} = -0.54\ lbf$$

$$B_{xl} = 2.41\ lbf \quad B_{yl} = 3.31\ lbf$$

$$C_{xl} = 2.41\ lbf \quad C_{yl} = 3.49\ lbf$$

$$D_{xl} = -2.41\ lbf \quad D_{yl} = -0.40\ lbf$$

$$\tau_{lift} = 21.00\ in \cdot lbf$$

We chose a transmission by determining the minimum gear ratio and the maximum controllable speed. The full calculations can be found in Appendix 3. To find the minimum gear ratio, we used the formula:

$$e_{min_{lift}} = \frac{\tau_S}{\tau_{lift}}\eta \tag{1}$$

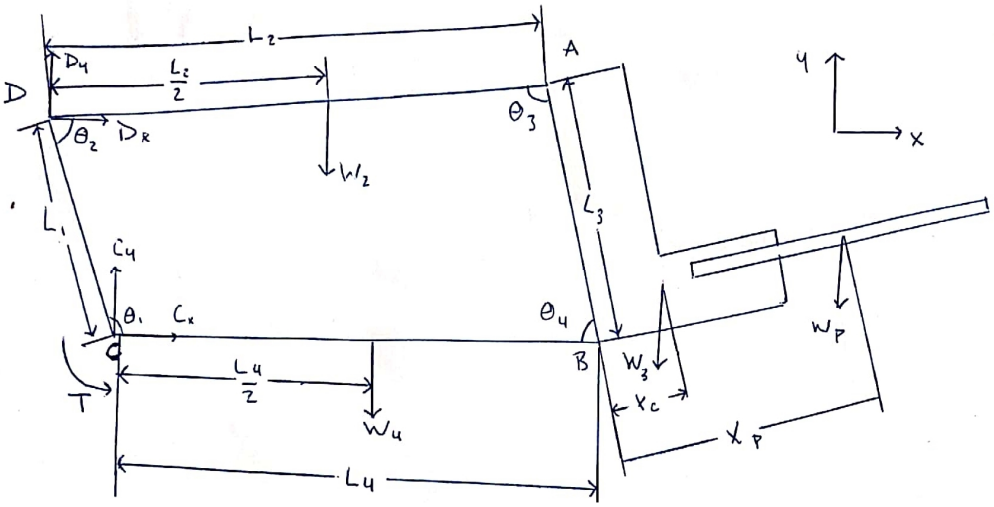


Figure 4: Free body diagram of the lift four-bar linkage

where τ_S is the stall torque of the motor and η is the efficiency. We found the minimum gear ratio $e_{min_{lift}} = 0.353$. To find the gear ratio with the maximum controllable speed, we used the equation:

$$e_{ideal} = \frac{\omega_{ideal}}{\omega_{max\ power}} \tag{2}$$

where ω_{ideal}, the maximum controllable speed, is defined as 20 $\frac{degrees}{sec}$ and $\omega_{max\ power}$ is the angular speed of the motor at maximum power, which is defined as:

$$\omega_{max\ power} = \frac{\omega_{free}}{2} \tag{3}$$

where ω_{free} is the motor free speed. We found the gear ratio with the maximum controllable speed to be $e_{ideal} = 0.033$. We picked a gear ratio of 324:7056 (or 1: 21.7777), which is between the minimum and controllable speeds and allows us to use standard 20DP gears which fit into our physical constraints with a double-stage 18:84 gear reduction.

Additionally, we found the speed of the linkage components in the maximum torque position by using the method of instantaneous centers. The full calculations can be found in Appendix 3. We found the instantaneous center of links 2 and 4, as seen in Figure 5, and using T_{lift} we found the instantaneous velocities of each link. We found the instantaneous

velocities at maximum torque V_A, V_B, and V_P to be:

$$V_A = 5.2 \ \frac{in}{sec}$$
$$V_B = 4.734 \ \frac{in}{sec}$$
$$V_P = 5.66 \ \frac{in}{sec}$$

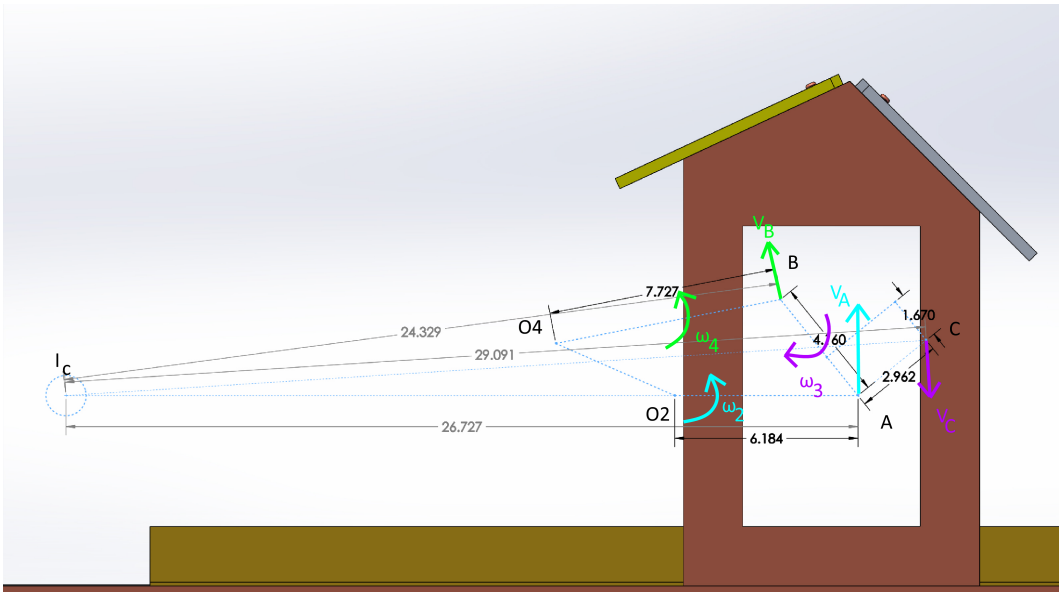


Figure 5: Dimensions for lift mechanism velocity analysis using method of instantaneous centers

Similarly, we completed a force analysis of the grabber mechanism, starting with creating a free body diagram of the entire system as seen in Figure 6. The full calculations can be found in Appendix 2. We proceeded to use the same method as in the four-bar linkage to calculate the torque at the crank joint $\tau_{grabber}$, and the forces at the joints A_{xg}, A_{yg}, B_{xg}, B_{yg}, C_{xg}, C_{yg}, D_{xg} and D_{yg}:

$$A_{xg} = -1.23 \ lbf \quad A_{yg} = 2.33 \ lbf$$
$$B_{xg} = 1.23 \ lbf \quad B_{yg} = 0.08 \ lbf$$
$$C_{xg} = 1.23 \ lbf \quad C_{yg} = 0.23 \ lbf$$
$$D_{xg} = -1.23 \ lbf \quad D_{yg} = 2.58 \ lbf$$

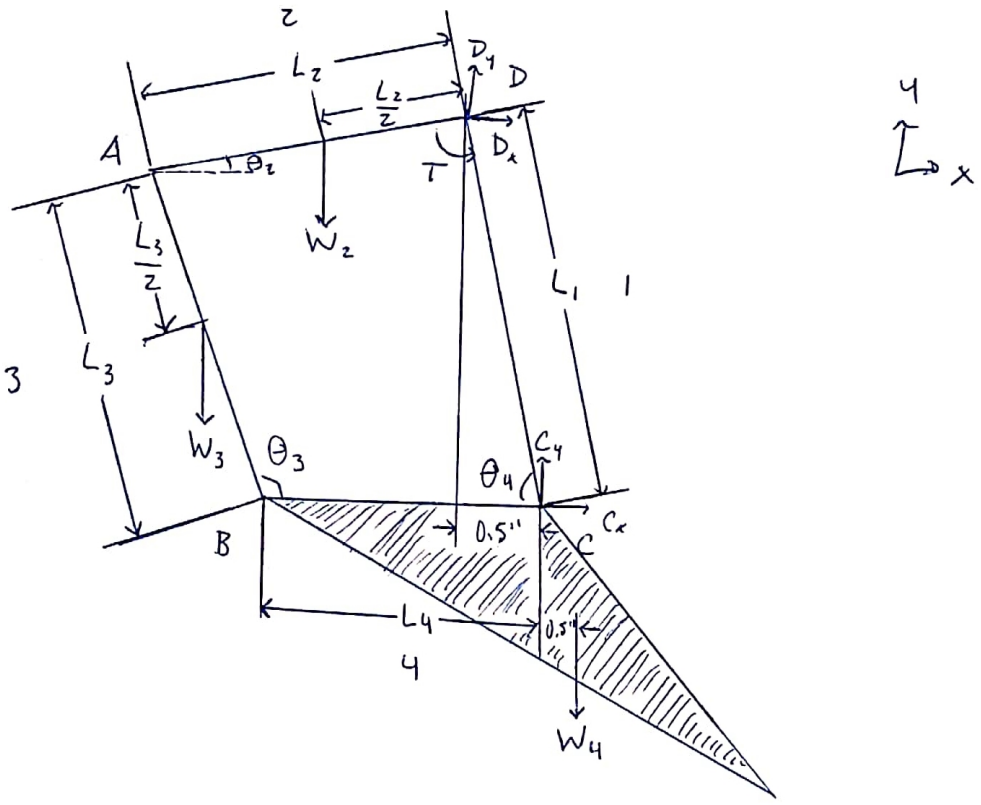


Figure 6: Free body diagram of the grabber four-bar linkage

$$\tau_{grabber} = -1.09\ in \cdot lbf$$

Using the measurements obtained from the graphical linkage synthesis, we then designed the linkage in CAD. To ensure clearance of the physical links, we made the top link curved, while keeping the direct distance between the joints on the top link the same as determined in the graphical linkage synthesis, as can be seen in Figure 7. The exploded view of our four-bar linkage and the bill of materials can be found in Appendix 1.

To ensure that our transmission would endure the challenge, we calculated the shear forces on the gear teeth and the factor of safety associated with our chosen material. Specifically, we looked at the last gear in the transmission, an 84-tooth connected to the bottom link of the four-bar linkage, because its material was the most had the lowest yield strength,

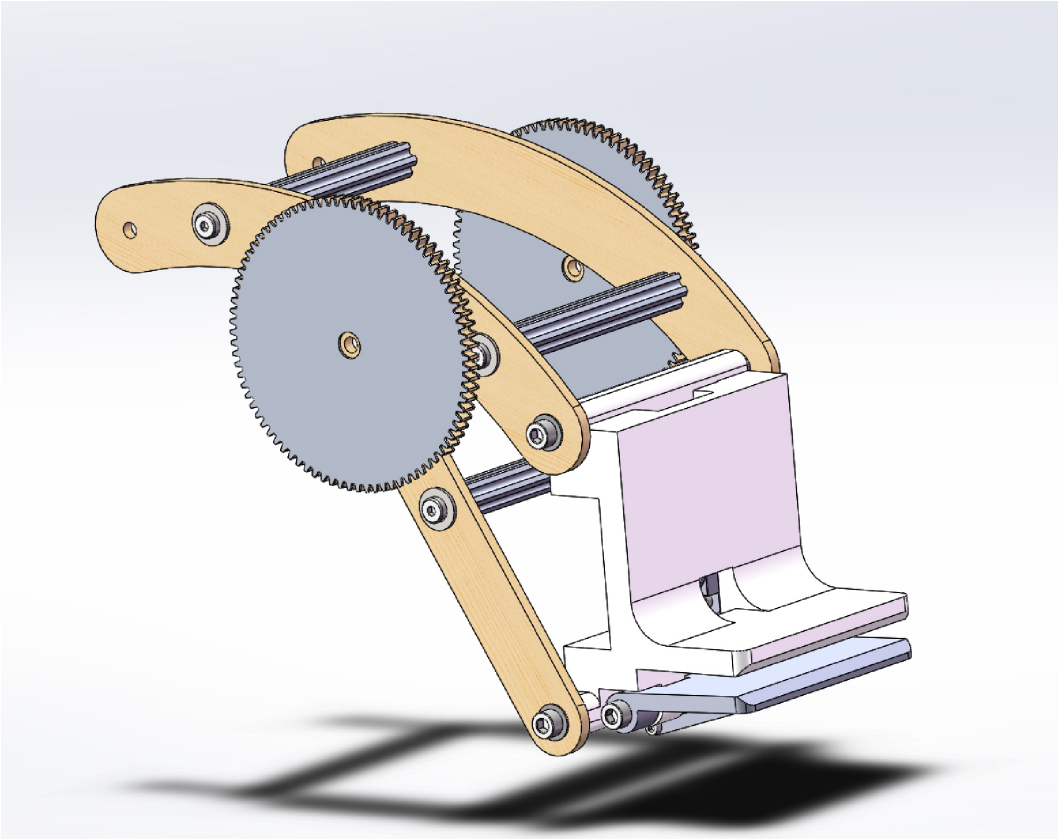


Figure 7: Linkage CAD model based upon linkage synthesis

and rather being thick and made only of PLA as the other gears were, it would be made of 1/4" (nominal) birch plywood plated on each face with thin sheets of PLA. To calculate the forces on these teeth, we used the equation:

$$F_{teeth} = \frac{\tau_{lift}}{r_{pitch}} \tag{4}$$

where r_{pitch}, the pitch radius, is defined as:

$$r_{pitch} = \frac{t}{2P} \tag{5}$$

where t is the number of teeth on the gear, and P is the diametral pitch of the gear. Based on the forces calculated, the sheer stress on the gear teeth had a factor of safety of over 2. Full calculations can be found in Appendix 3.

Maximum current draw was calculated based on torque applied to the motor at the

position of maximum torque on the crank (when it is extended parallel to the field), using the equation:

$$I_{linkage_{max}} = \frac{T_{motor_{load}}}{T_{motor_{stall}}}(I_{stall} - I_{free}) + I_{free} + I_{gripper_{max}} \tag{6}$$

Full calculations can be found in Appendix 3. We found the maximum current draw, $I_{linkage_{max}}$, to be:

$$I_{linkage_{max}} = 0.88\ A + 500\ mA = 1.38\ A$$

We continued by imposing constraints on our drivetrain. We intended to use the rails to guide our robot in a straight line as it approached the roof structure and ensure precision location with respect to the roof structure. In order to do this, we were able to constrain the width of the robot to 12 inches, not including the rail riders, which would align the center of the robot with the center of the solar collector plate.

With these constraints, we then were able to design our drivetrain and transmission. We chose to have front wheel drive with one ball castor in the back-center of the robot. To choose a transmission, we determined the minimum gear ratio which we calculated the minimum gear ratio using the formula:

$$e_{min_{drive}} = \frac{T_S}{T_{f_{max}}}\eta \tag{7}$$

where T_S is the motor stall torque, $T_{f_{max}}$ is the maximum torque resisting friction, and η is the efficiency.

We determined a maximum "ideal" gear ratio assuming a controllable speed of 10 $\frac{in}{sec}$, at maximum mechanical power:

$$T_{max_{power}} = \frac{T_S}{2} = 85\ ozf \cdot in \tag{8}$$

$$\omega_{max_{power}} = \frac{\omega_{free}}{2} = 100\ rpm \tag{9}$$

$$\omega_{ideal} = \frac{10\frac{in}{sec}}{d_{wheel}} = 183.364\ \frac{deg}{sec} \tag{10}$$

$$e_{ideal} = \frac{\omega_{ideal}}{\omega_{max_{power}}} = 0.306 \tag{11}$$

where ω_{free} is the free speed of the motor and d_{wheel} is the wheel diameter in inches.

We selected a 18:54 ratio (1:3) transmission in order to balance speed and controllability, with our focus being on controllability since speed was not an essential component of the challenge. This ratio is between the controllable ratio and the minimum ratio, and errs on the side of the controllable ratio. Using the weight of the robot as calculated in our SolidWorks model, the transmission, the robot design and the motor specifications, we were then able to calculate the maximum speed of the robot and the power requirement of the robot at both slip and stall. To calculate the max speed, we converted motor free (angular) speed to linear speed:

$$v =_{free} \frac{d_{wheel}}{2} \omega_{free} e_{drivetrain} = 10.908 \ \frac{in}{sec} \tag{12}$$

$$v_{max} = v_{free} \eta = 10.363 \ \frac{in}{sec} \tag{13}$$

where $e_{drivetrain}$ is the overall drivetrain gear ratio, and η is the efficiency. This physical design of our robot being finished, we were able to render it as a CAD model, as seen in Figures 8 and 9.

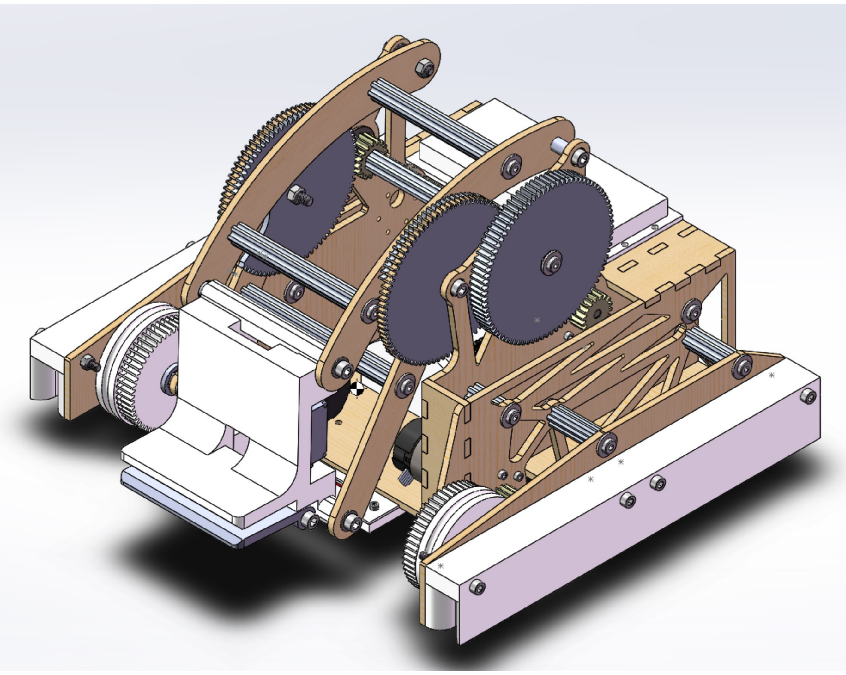


Figure 8: View of SolidWorks CAD model of final design of robot with the linkage extended

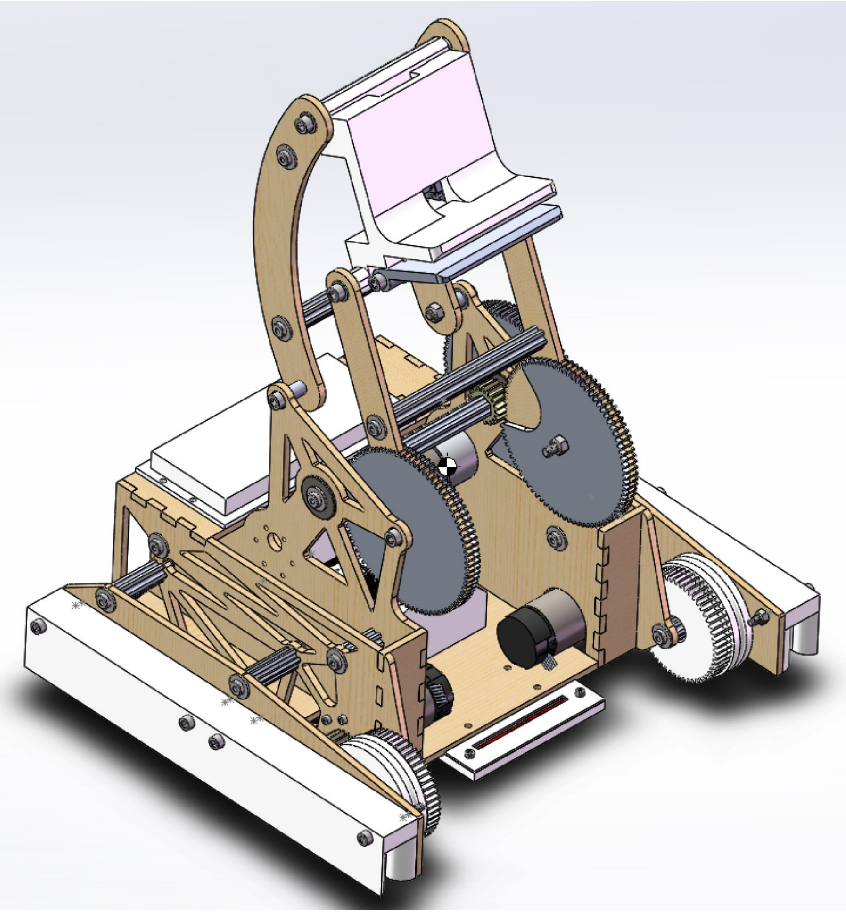


Figure 9: View of SolidWorks CAD model of final design of robot with the linkage down

Power requirements were determined using the calculated gear ratio. Since the maximum torque due to friction is less than the stall torque, the robot is guaranteed to slip instead of stall.

$$T_{output_{stall}} = \frac{T_S}{e.drivetrain} = 31.875\ in \cdot lbf \tag{14}$$

If the two motors on the drivetrain were to stall, the current drawn would be:

$$I_{stall} = 2I_S = 10\ A \tag{15}$$

where I_S is the stall current of one motor.

The current drawn at slip is maximized when the resistive torque due to friction is

maximum.

$$T_{load_{motor}} = T_{friction_{max}} e_{drivetrain} = 1.872\ in \cdot lbf \tag{16}$$

$$I_{load_{motor}} = \frac{T_{load_{motor}}}{T_S}(I_S - I_{free}) + I_{free} = 1.128\ A \tag{17}$$

$$I_{slip} = 2 * I_{load_{motor}} = 2.257\ A \tag{18}$$

Overall power requirements were based on several factors: peak drivetrain current draw, peak lift linkage current draw, and steady-state current for electronics components.

$$\begin{aligned} P_{max} = V_{ref}(I_{slip} + I_{linkage_{max}} + I_{steadystate}) &= V_{ref}(2.257\ A + 1.38\ A + 155\ mA) \\ &= V_{ref}(3.792\ A) = 45.504\ W \end{aligned} \tag{19}$$

where V_{ref} is 12V, aka 8x 1.5V 2Ah rechargeable batteries.

The steady-state current calculations are based on manufacturer specifications for idle current draw in each of the electronic components, given by:

$$I_{idle} = \sum_{i}^{n} I_i \tag{20}$$

where I_i is the idle current draw for the component i. In this case the components sum as outlined in Table 2.

Table 2: Current Draw of Electrical Components

Component	Current Draw
ESP32	25 mA
WiFi Reciever	100 mA
Encoders	30 mA

$$I_{idle} = 25\ mA + 100\ mA + 30\ mA I_{idle} = 155\ mA \tag{21}$$

Given an ideal 12V Battery pack with a capacity of 2Ah, the total energy the pack can provide is given by:

$$\begin{aligned} E = Pt = V_{ref} It \\ E = (12\ V)(2\ Ah)(\frac{3600\ s}{1\ h}) \\ E = 86.4\ kJ \end{aligned} \tag{22}$$

Now that we know the energy the pack can provide, we can divide by the power requirement to determine operation time. Steady state requirements:

$$\begin{aligned} t &= \frac{E}{P} = \frac{E}{VI} \\ t &= \frac{86.4\ kJ}{(12\ V)(0.155\ A)} * \frac{1\ hr}{3600\ s} \\ t &= 12.9\ hr \end{aligned} \tag{23}$$

As we can see in the equation above, running at only steady-state, the battery will last for about 12 hours, 54 minutes.

Peak requirements:

$$\begin{aligned} t &= \frac{E}{P} = \frac{E}{P_{max}} \\ t &= \frac{86.4\ kJ}{45.504\ W} * \frac{1\ min}{60\ s} \\ t &= 31.6\ min \end{aligned} \tag{24}$$

Even at a constant max current draw, the battery pack will still allow 31 minutes of operation, more than enough time to complete the challenge. This exceeds the goal of 10 minutes - the maximum time allocated to complete the entire list of tasks.

To program the robot, we implemented a simple state machine. To create this, we outlined and categorized the tasks that our robot had to carry out, the list of which can be found in Appendix 5. An innovation our team added was to "layer" certain states, allowing for reuse of the same state with slightly different parameters. In the case of line following, we needed the robot to reuse the same procedure each time it followed a line, but to alter its direction, correctional amount, and next state depending on its progress in completing the mission. We used this idea for several states, including line following, and transitions between perpendicular lines. Due to this "layered" approach, our robot followed a fairly simple procedure as outlined in Figure 10.

First, the robot would pickup a panel from the safe zone. Next, the robot would search for the perpendicular line leading to the appropriate position as received from the field controller. Again, the robot would follow a line to the appropriate perpendicular line to perform new collector panel dropoff, before repeating the process in reverse to arrive at the other position for old panel pickup, and finally, back to the safe zone for old panel

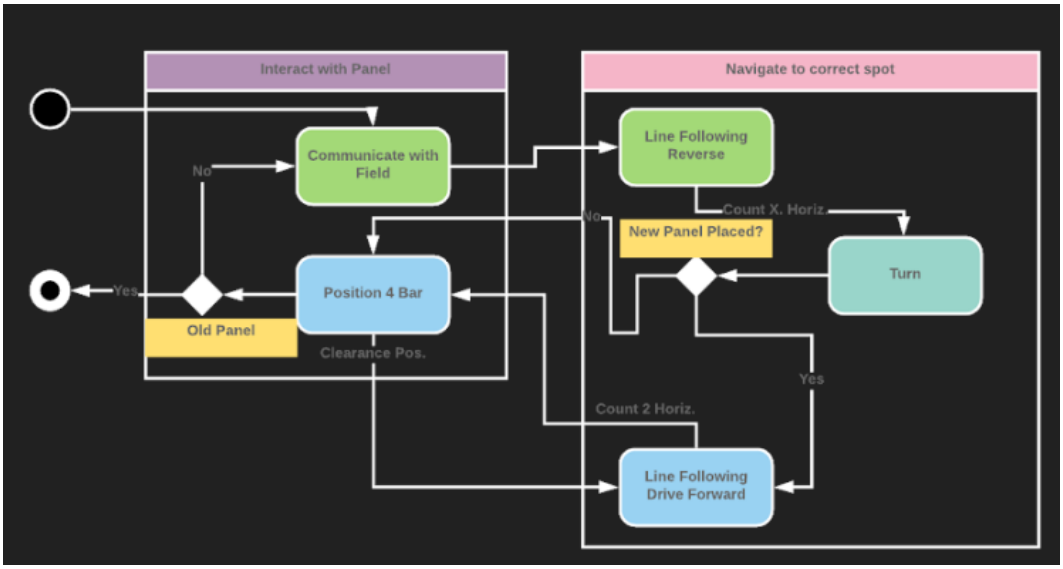


Figure 10: Basic Flow Diagram Approximating state Machine

dropoff. This basic process relies heavily on repeated task of line following, line counting, and switching between crossing lines, which Figure 10 loosely demonstrates.

4 Results & Discussion

We found during the testing and demonstrations of the different systems of our robot that many of our calculations and predictions were accurate, while others were inaccurate or required adjustment. In order to account for our inaccuracies, we had to make some minor changes to our robot design, as seen in Figure 11

This experience demonstrated to each of us that the real world is "messy" and that our predictions are not always perfect in describing the behavior of the dynamic systems and processes that make up a robot.

Concerning one of our primary strategies, following field lines for navigation, we had mostly positive results. The difference in reflectance of the black tape and light-tan wood of the field allowed our robot to quantitatively distinguish between them.

In order to accurately follow a line across the field, we implemented code that would correct the position of the robot relative to the line based on the values read from the sensors

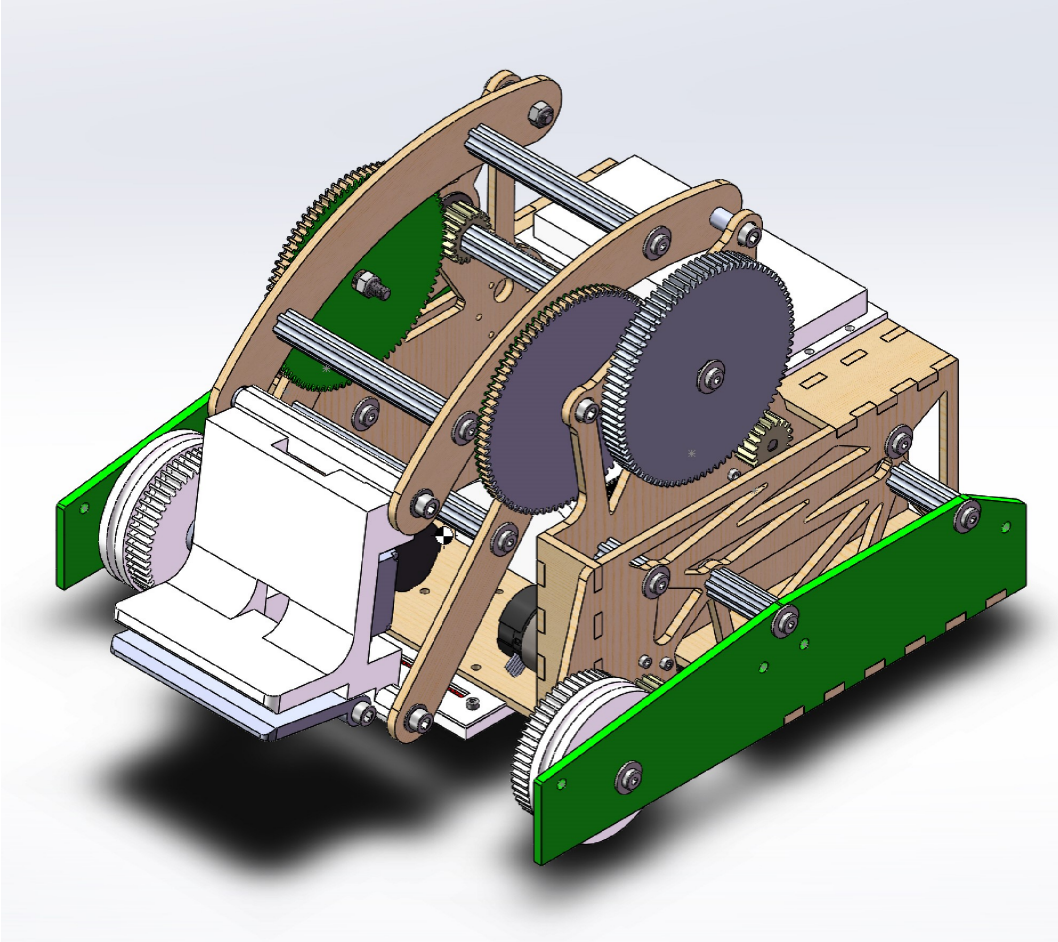


Figure 11: SolidWorks CAD model representing our robot after adjustments were made to account for errors in design, with the parts changed highlighted in green

at any time. Depending on the direction of travel, if the two infrared light sensors returned greatly differing values, then the appropriate wheel's position was adjusted to "straighten out" the the robot chassis relative to the line being followed. The basic logic behind this method of line following is demonstrated in Figure 12.

The success of this method was mixed. In most cases, the "straightening out" action of this method of line following helped the robot assume a correct orientation for retrieval and deposit of the collector panels; however, this method also resulted in the occasional over-correction, which could slow the approach, or in certain cases, direct the robot off-course.

```
//chassis tilted clockwise relative to line
if(lineFollower->sensor1Val >= lineFollower->BLACK && lineFollower->sensor2Val <= lineFollower->WHITE){
    //straighten out -- rotate left wheel backward
    motor1->startInterpolationDegrees(motor1->getAngleDegrees() - 60, 1000, SIN);
}
//chassis tilted counterclockwise relative to line
else if(lineFollower->sensor1Val <= lineFollower->WHITE && lineFollower->sensor2Val >= lineFollower->BLACK){
    //straighten out -- rotate right wheel backward
    motor2->startInterpolationDegrees(motor2->getAngleDegrees() - 60, 1000, SIN);
}
```

Figure 12: Code Snippet Showing Line-following Logic

We used the Reflectance Sensor Array included in our kit of parts for another similar purpose: line counting. By recording when both sensors reported a value consistent with black tape simultaneously, we could detect the presence of a perpendicular line, and use this information to inform the robot's state machine program.

The four-bar mechanism we designed for lifting the collector panels to the appropriate positions functioned almost flawlessly after manufacture. The coupler / grabber combo proved excellently suited for the tasks of both placing and picking up collector panels. We found that during the coding process, we had no problems related to the linkage not reaching the correct positions for panel pickup and dropoff. As seen in Figure 13, the positions provided by the linkage configuration were so accurate that our robot simply positioned itself directly in front of the roof and lowered the collector panel by rotating the crank. This action positioned the collector panel onto its pegs.

One issue we encountered while testing our linkage was with our crank gears. The plywood we used for our crank link/driven gear combination had an actual thickness that was less than its nominal thickness, which led to decreased resistance to shear, and ultimately resulted in minor chipping of gear teeth. The problem was exacerbated by an unaccounted for toggle position of the linkage, which occurred due to the curved top links. While the curvature did provide clearance as desired, it also created a toggle point with joints on the bottom link, which if entered, caused the bottom link to not be able to drive the linkage any longer. If this was done while the bottom link was driven, the teeth of the stationary gear on the bottom link would be placed under a high amount of stress. In order to counteract this problem, we created plastic reinforcements, that when aligned and attached to our wooden

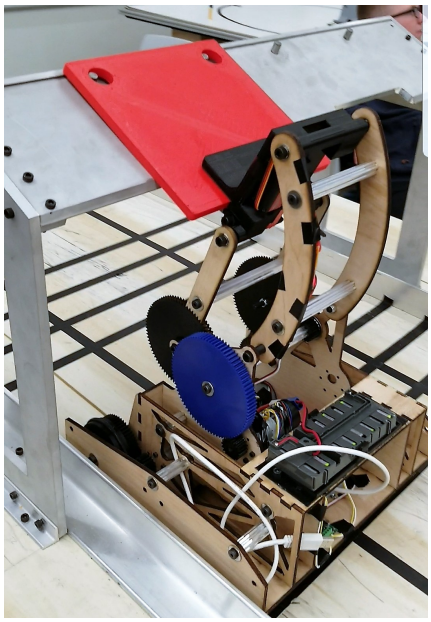


Figure 13: Our Robot Placing a Collector Panel

crank/gear, provided the strength necessary to resist shear. Additionally, to prevent the linkage from entering the toggle position, we prevented it in code using the encoder on the linkage motor.

We decided to use one of the provided grabber designs, which had mixed results. The design allowed us to easily control grabber position with a servo motor, but had physical limitations in length and width, that limited its ability to grip the collectors. By using strips of cardboard taped together, we modified the grabber's spacing, allowing for tight grip of the collector panels. This "custom-made" grabbing technique improved the overall capability of the robot, since loose collector panels were not an issue.

The manufacture of our robot went smoothly. We created a horizontal standoff-based design using Birch plywood and the popular-with-FIRST "churros". The churros are aluminum hex shafts that provide dual function as both standoffs and axles in our robot design. The churros are lightweight, strong, and simple to cut, which made them an excellent part for our build. We tapped the churros ourselves to allow for accurate placement of threads for assembly. The sheets of birch plywood that comprised the surfaces of the robot were glued together using wood glue for added strength. This combination materials and methods of

construction allowed us to assemble our robot rapidly, with not much time spent waiting for 3D printed parts. In addition, the robot chassis was strong, lightweight, and flexible, giving us a robust and damage-resistant chassis to work with.

Our robot performed well for the majority of assigned tasks, once it was fully assembled. Initially, we had difficulty with PID tuning, but that issue was solved by using different sets of PID gain values for different collector types. In addition, much trial-and-error was required to adjust the line following program to achieve correct orientation correction and horizontal line counting.

5 Conclusions

Overall, the project was a success as we were able to achieve all of our major objectives with minor adjustments made to our initial design. While we were unable to earn the points for presenting early or completing the entire challenge in one run, with the extended deadline we were not only able to complete the major tasks, but do so smoothly and without user-adjustment.

Given additional time, and continued access to the robot, the main improvements to be made on our robot are continuing work on the code so that all tasks can be completed in one run, and improvement of the line-following so that the robot runs smoother and faster. Additionally, the rail riders should be re-designed to work properly to allow the robot to forego line-following as it approaches the roof structure, and the gripper should be replaced with a more permanent solution to the spacing issues that made it difficult to pick up plates. As-is, our project, while not wholly successful, is very successful in completing the challenge tasks in isolation and with minor improvements would be a highly effective system for completion of the challenge tasks.

Comments

The dimensions were incorrect when using the field CAD for reference. In particular, we took the width of the rails from the CAD, and created our rail riders around this dimension. We only realized after printing that this was not accurate to the actual design of the field. The

CAD should be an accurate representation of the field. If it is not, proper documentation should be clearly provided accordingly. While the rails were not necessary for the success of our project, this is an unnecessary hassle and obstacle to teams.

Wear and tear on the field caused some problems with line following, especially where the field surface or tape made navigation difficult.

The term project was a very good application of concepts learned in class, especially the math-heavy sections. The PDR should have been earlier, but the CDR and FDR were well-timed relative to the fast pace of the course.

In our kit, we were supplied an analog servo instead of a digital servo, which caused us to waste over four valuable hours on Wednesday night due to a massive amount of time spent troubleshooting. The code we were given was not able to properly control the servo, while a new Arduino sketch worked fine. There should be very clear documentation about this possibility, and any supplier issues (cheap is not necessarily good!) should be resolved so it doesn't happen to another unwary team.

The servo also did not work with our code until we attached it again in our own code.

For purposes of the project and class in general, a semi-comprehensive powerpoint or other documentation covering concepts expected for class (especially the types of math for mechanism analysis) would have been very useful.

We had several problems with the ESP32. One of the pins we were told to use for the H-Bridge did not work (we switched it to another pin successfully). There should be more documentation on the ESP32 timers and how they related to PWM pins, and which can be used. We also had several instances of a smoking ESP32, even without making any obvious errors (once in the middle of Lab 4, once in the middle of testing for final project). We had to replace the ESP32 during Lab 4, which made it impossible to meet the original deadline - neither us nor Kevin could figure out why it was damaged.

The example robot base was wired incorrectly, and was a poor reference. Wiring diagrams on GitHub were also unclear.

Wire wrapping was an arbitrary and inefficient way to make connections. We often had issues with loose wires even when a sufficient number of well-done wraps. The wires were inevitably very messy, even when zip-tied or taped.

Appendices

Appendix 1: Four-Bar Mechanism Exploded View and Bill of Materials

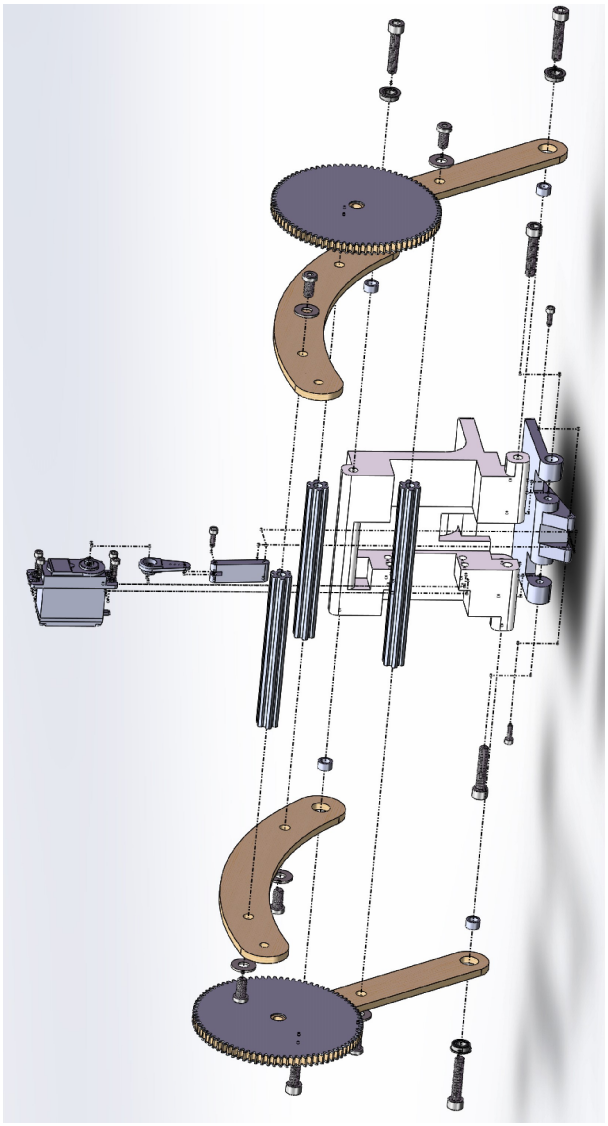
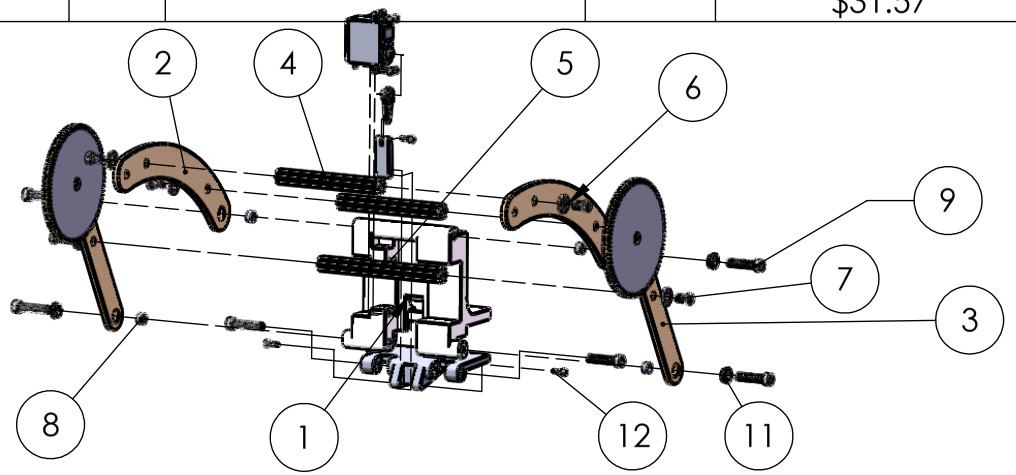


Figure 14: Rear view of exploded linkage

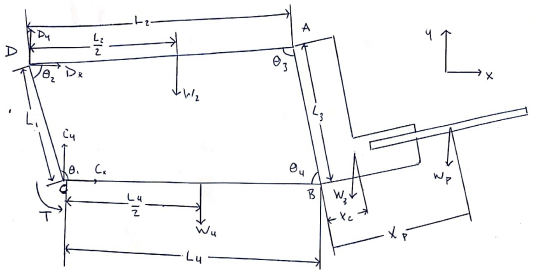
ITEM NO.	PART NUMBER	QTY.	MATERIAL	COST	EXT. COST	WEIGHT (LBS)	EXT. WEIGHT (LBS)
1	GRABBER	1			$0.00		0.00
	GRABBER_BODY	1	PLA	$3.31	$3.31	0.243	0.24
	GRIPPER	1	PLA	$1.01	$1.01	0.074	0.07
	LINK	1	PLA	$0.07	$0.07	0.005	0.01
	SERVO	1	N/A	$9.75	$9.75	0.137	0.14
	SERVO_HORN	1	PLA	$0.00	$0.00	0.002	0.00
	M6 X 30mm SCREW	2	BLACK OXIDE ALLOY STEEL	$0.16	$0.32	0.00	0.00
2	TOP_LINK	2	1/4" BIRCH PLYWOOD	$0.13	$0.26	0.050	0.10
3	BOTTOM LINK	2	1/4" BIRCH PLYWOOD	$0.22	$0.44	0.080	0.16
4	4-272IN_CHURRO	2	6005A ALUMINUM	$1.42	$2.84	0.021	0.04
5	5-072IN_CHURRO	1	6005A ALUMINUM	$1.69	$1.69	0.020	0.02
6	1/4" FLAT WASHER	6	STAINLESS STEEL	$0.03	$0.18	0.00	0.00
7	1/4"-20 SCREW	6	BLACK OXIDE ALLOY STEEL	$0.19	$1.14	0.00	0.00
8	SPACER-6mm_ID-0-2IN	4	PLA	$0.01	$0.04	0.001	0.00
9	M6 X 30mm SCREW	4	BLACK OXIDE ALLOY STEEL	$0.16	$0.64	0.00	0.00
10	84T_20_DP_REINFORCEMENT	4	PLA	$0.33	$1.32	0.024	0.08
11	6X12X4 FLANGED BEARING	4	N/A	$2.00	$8.00	0.001	0.00
12	M3 X 10mm SCREW	7	ALLOY STEEL	0.08	$0.56	0.00	0.00
					ASSEMBLY COST $31.57		ASSEMBLY WEIGHT 0.86



TEAM 2	SCALE: 1:7	FINAL PROJECT LINKAGE	3/2/19

Appendix 2: Force Analysis

Lift Mechanism Force Analysis



Define Physical Constants

Weight of the Plate

$W_p := 2.3lbf$

Weight of Link 2

$W_2 := .14 \cdot lbf$

Weight of Link 3

$W_3 := .47lbf$

Weight of Link 4

$W_4 := .18lbf$

Length of Link 1

$L_1 := 4.38in$

Length of Link 2

$L_2 := 7.73in$

Length of Link 3

$L_3 := 4.16in$

Length of Link 4

$L_4 := 6.18in$

x-Coordinate of Center of Mass of Link 3 With Respect to the Joint of Links 3 and 4

$x_c := .75in$

y-Coordinate of Center of Mass of Link 3 With Respect to the Joint of Links 3 and 4

$y_c := 2in$

x-Coordinate of Center of Mass of Solar Collector Plate With Respect to the Joint of Links 3 and 4

$x_p := 5in$

y-Coordinate of Center of Mass of Solar Collector Plate With Respect to the Joint of Links 3 and 4

$y_p := .52in$

Angle Between Links 1 and 4

$\theta_1 := 156.38deg$

Angle Between Links 1 and 2

$\theta_2 := 34.6deg$

Angle Between Links 4 and 3

$\theta_4 := 51.06deg$

Angle of link 2 above the horizontal defined by link 4

$\theta_{2n} := \theta_2 - (180deg - \theta_1) = 10.98 \cdot deg$

Define variables

Forces at Joint A

$A_x := 0$ $A_y := 0$

Forces at Joint B

$B_x := 0$ $B_y := 0$

Forces at Joint C

$C_x := 0$ $C_y := 0$

Forces at Joint D

$D_x := 0$ $D_y := 0$

Torque at Crank

$T := 0$

Given

System Equations of Equilibrium

$\Sigma M_C=0$ $$0 = T - W_3\cdot\left(L_4 + x_c\cdot\cos\left(90deg - \theta_4\right)\right) - W_p\cdot\left(L_4 + x_p\cdot\cos\left(90\cdot deg - \theta_4\right)\right.$$

$$\left. - W_2\cdot\left(\frac{L_2}{2}\cdot\cos\left(\theta_{2n}\right) - L_1\cdot\sin\left(\theta_1 - 90deg\right)\right) - W_4\cdot\frac{L_4}{2}\right.$$

$\Sigma F_x=0$ $0 = C_x + D_x$

$\Sigma F_y=0$ $0 = C_y + D_y - W_2 - W_3 - W_4 - W_p$

Link 3 System of Equations

$\Sigma M_B=0$ $0 = -A_y\cdot L_3\cdot\cos\left(\theta_4\right) - A_x\cdot L_3\cdot\sin\left(\theta_4\right) - W_3\cdot x_c\cdot\cos\left(90deg - \theta_4\right) - W_p\cdot x_p\cdot\cos\left(90deg - \theta_4\right)$

$\Sigma F_x=0$ $0 = A_x + B_x$

$\Sigma F_y=0$ $0 = A_y + B_y - W_3 - W_p$

Link 2 System of Equations

$\Sigma M_D=0 \qquad 0 = L_2 \cdot \left[\left(\frac{-W_2}{2} - A_y \right) \cdot \cos(\theta_{2n}) + A_x \cdot \sin(\theta_{2n}) \right]$

$\Sigma F_x=0 \qquad 0 = D_x - A_x$

$\Sigma F_y=0 \qquad 0 = D_y - A_y - W_2$

Link 4 System of Equations

$\Sigma M_C=0 \qquad 0 = T - \left(\frac{L_4}{2} \right) \cdot W_4 - L_4 \cdot B_y$

$\Sigma F_x=0 \qquad 0 = C_y - B_y - W_4$

$\Sigma F_y=0 \qquad 0 = C_x - B_x$

Solve for Forces at Joints and Torque at Crank

$$\begin{pmatrix} A_{xx} \\ A_{yy} \\ B_{xx} \\ B_{yy} \\ C_{xx} \\ C_{yy} \\ D_{xx} \\ D_{yy} \\ T_{crank} \end{pmatrix} := \text{Find}(A_x, A_y, B_x, B_y, C_x, C_y, D_x, D_y, T)$$

Forces at Joint A

$A_{xx} = -2.414\,\text{lbf}$ $A_{yy} = -0.538\,\text{lbf}$

Forces at Joint B

$B_{xx} = 2.414\,\text{lbf}$ $B_{yy} = 3.308\,\text{lbf}$

Forces at Joint C

$C_{xx} = 2.414\,\text{lbf}$ $C_{yy} = 3.488\,\text{lbf}$

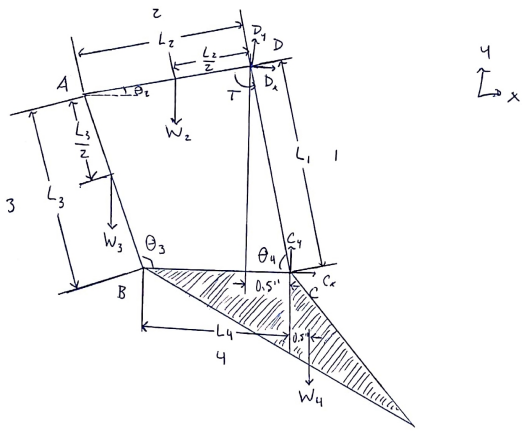
Forces at Joint D

$D_{xx} = -2.414\,\text{lbf}$ $D_{yy} = -0.398\,\text{lbf}$

Torque at Crank

$T_{crank} = 21.002 \cdot \text{lbf} \cdot \text{in}$ $T_{crank} = 336.032 \cdot \text{ozf} \cdot \text{in}$

Grabber Mechanism Force Analysis



Define Physical Constants

Weight of Link 2 $W_2 := .25lbf$

Weight of Link 3 with Plate $W_3 := 2.4lbf$

Weight of Link 4 $W_4 := .15lbf$

Length of Link 1 $L_1 := 1.714in$

Length of Link 2 $L_2 := .880in$

Length of Link 3 $L_3 := 1.26in$

Length of Link 4 $L_4 := .707in$

Angle Between Link 2 and the Horizontal Defined by Link 4

$\theta_2 := 34.05deg$

Angle Between Links 3 and 4 $\theta_3 := 114.48deg$

Angle Between Links 1 and 4 $\theta_4 := 73.04deg$

Define variables

Forces at Joint A $A_x := 0$ $A_y := 0$

Forces at Joint B $B_x := 0$ $B_y := 0$

Forces at Joint C $C_x := 0$ $C_y := 0$

Forces at Joint D $D_x := 0$ $D_y := 0$

Torque at Crank

$T := 0$

Given

System of Equations of Entire System

$\Sigma M_C=0 \qquad 0 = T + W_3 \cdot \left(L_4 - .5in + \frac{L_3}{2} \cos(180deg - \theta_3) \right) + W_2 \cdot \left(\frac{L_2}{2} \cdot \cos(\theta_2) \right) - W_4 \cdot \left(0.5in + \frac{L_4}{2} \right)$

$\Sigma F_x=0 \qquad 0 = C_x + D_x$

$\Sigma F_y=0 \qquad 0 = C_y + D_y - W_2 - W_3 - W_4$

Link 3 System of Equations

$\Sigma M_B=0 \qquad 0 = -A_y \cdot L_3 \cdot \cos(\theta_4) - A_x \cdot L_3 \cdot \sin(\theta_4) - W_3 \cdot \frac{L_3}{2} \cdot \cos(180deg - \theta_3)$

$\Sigma F_x=0 \qquad 0 = A_x + B_x$

$\Sigma F_y=0 \qquad 0 = A_y + B_y - W_3$

Link 4 System of Equations

$\Sigma M_C=0 \qquad 0 = -\left(\frac{L_4}{2} \right) \cdot W_4 + L_4 \cdot B_y$

$\Sigma F_x=0 \qquad 0 = C_y - B_y - W_4$

$\Sigma F_y=0 \qquad 0 = C_x - B_x$

Solve for Forces at Joints and Torque at Crank

$$\begin{pmatrix} A_{xx} \\ A_{yy} \\ B_{xx} \\ B_{yy} \\ C_{xx} \\ C_{yy} \\ D_{xx} \\ D_{yy} \\ T_t \end{pmatrix} := \text{Find}(A_x, A_y, B_x, B_y, C_x, C_y, D_x, D_y, T)$$

Forces at Joint A

$A_{xx} = -1.229\,\text{lbf}$ $A_{yy} = 2.325\,\text{lbf}$

Forces at Joint B

$B_{xx} = 1.229\,\text{lbf}$ $B_{yy} = 0.075\,\text{lbf}$

Forces at Joint C

$C_{xx} = 1.229\,\text{lbf}$ $C_{yy} = 0.225\,\text{lbf}$

Forces at Joint D

$D_{xx} = -1.229\,\text{lbf}$ $D_{yy} = 2.575\,\text{lbf}$

Torque at Crank

$T_t = -1.086\,\text{in}\cdot\text{lbf}$

Appendix 3: Linkage Transmission Calculations

Lift Gear Ratio

Lift Gear Ratio Calculations

Assumed Transmission Stage Efficiency

$\eta := 95\%$

Motor Free Speed

$\omega_{free} := 200rpm$

Motor Stall Torque

$T_{motor\_stall} := 170 \cdot ozf \cdot in$

Minimum Gear Ratio Calculations

$e_{min} := \frac{T_{motor\_stall}}{T_{crank}} \cdot \eta = 0.481$

$\frac{1}{e_{min}} = 2.081$

Ideal gear ratio calculations

Motor Torque at Max Power

$T_{max\_power} := \frac{T_{motor\_stall}}{2} = 85 \cdot ozf \cdot in$

Motor Speed at Max Power

$\omega_{max\_power} := \frac{\omega_{free}}{2} = 100 \cdot rpm$

Estimated controllable angular speed

$\omega_{ideal} := \frac{60deg}{3s} = 20 \cdot \frac{deg}{s}$

$e_{ideal} := \frac{\omega_{ideal}}{\omega_{max\_power}} = 0.033$

$\frac{1}{e_{ideal}} = 30$

We picked a gear ratio of 324:7056 (aka 1:~21.7777), which is between the minimum and controllable speeds and allows us to use standard 20dp gears which fit into our physical constraints w/ a double-stage 18:84 gear reduction.

$e_{chosen} := \frac{18 \cdot 18}{84 \cdot 84} = 0.046$

$\frac{1}{e_{chosen}} = 21.778$

Gear Tooth Shear Forces

Given:

Diametral Pitch of Linkage Gear

$P := 20 \cdot \frac{1}{in}$

Shear Strength of Birch Plywood

$\tau_{y\_birch\_ply} := 5800kPa = 0.841 \cdot ksi$

Maximum Torque at Crank

$T_{max\_crank} := 21.00in \cdot lbf$

Number of Teeth on Linkage Gear

$t := 84$

Minimum Factor of Safety

$FoS := 1$

Motor Stall Torque

$T_{stall\_crank} := 239.398in \cdot lbf$

Thickness of Tooth at Pitch Diameter

$t_p := \frac{\pi}{2P} = 0.079 \cdot in$

Calculation of Maximum Shear Stress Allowed by Minimum Factor of Safety

$\tau_{max\_allowable} := \frac{\tau_{y\_birch\_ply}}{FoS} = 841.219 \cdot psi$

Pitch Radius

$r_{pitch} := \frac{t}{2P} = 2.1 \cdot in$

Force Applied at Tooth Surface by Linkage at Maximum Torque Position

$F_{applied} := \frac{T_{max\_crank}}{r_{pitch}} = 10 \cdot lbf$

Width of Gear Calculations

Required Minimum Width

$b_{req} := \frac{F_{applied}}{t_p \cdot \tau_{y\_birch\_ply}} = 0.151 \cdot in$

Actual Width of Gear

$b_{actual} := .195in \cdot 2 = 0.39 \cdot in$

Factor of Safety Calculation

$FoS_{actual} := \frac{b_{actual}}{b_{req}} = 2.577$

While this is a factor of safety greater than one, it remains a little too close for comfort, requiring us to modify our design slightly in order to ensure that our gears don't break in operation.

Lift Mechanism Current Draw

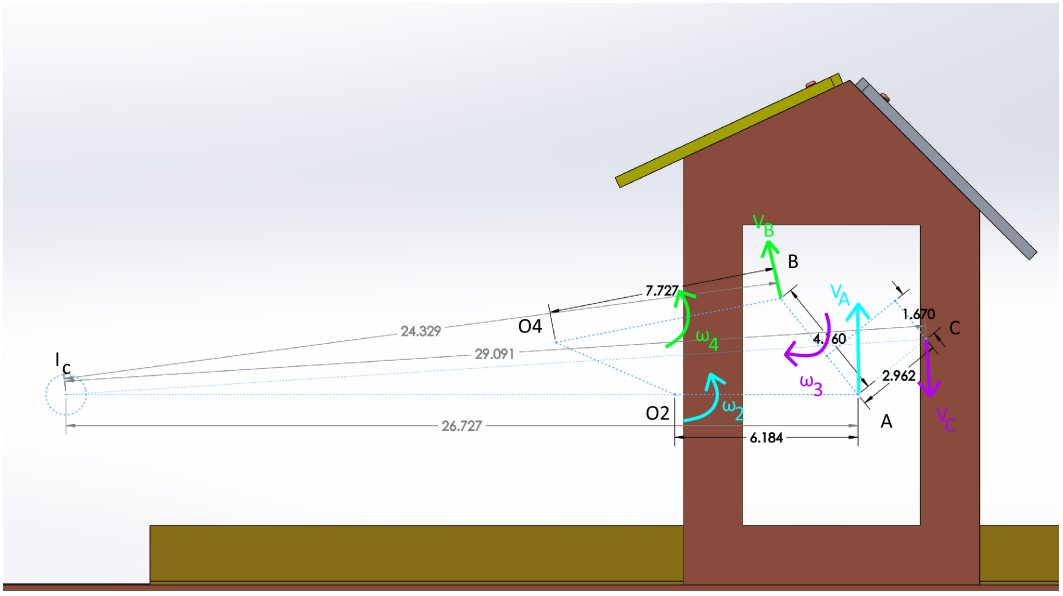
$$T_{motor} := T_{crank} \cdot e_{chosen} = 1.311 \cdot lbf \cdot in$$

$I_s := 5A$ Motor stall current

$I_f := 300mA$ Motor free-running current

$$I_L := \frac{T_{motor}}{T_{motor\_stall}} \cdot (I_s - I_f) + I_f = 0.88\,A$$ Overall current draw @ position of max torque

Lift Mechanism Instantaneous Center Velocity Analysis



$BI_C := 24.329in$ $BC := 7.727in$ $CI_C := 29.091in$ $AI_C := 26.727in$

$\omega_2 := 175rpm \cdot e_{chosen} = 0.841 \frac{1}{s}$

$V_A := \omega_2 \cdot L_4 = 5.2 \cdot \frac{in}{sec}$

$\omega_3 := \frac{V_A}{AI_C} = 0.195 \cdot \frac{rad}{sec}$

$V_B := \omega_3 \cdot BI_C = 4.734 \cdot \frac{in}{sec}$

$\omega_4 := \frac{V_B}{L_2} = 0.612 \cdot \frac{rad}{sec}$

$V_C := \omega_3 \cdot CI_C = 5.66 \cdot \frac{in}{sec}$

Appendix 4: Center of Mass Locations

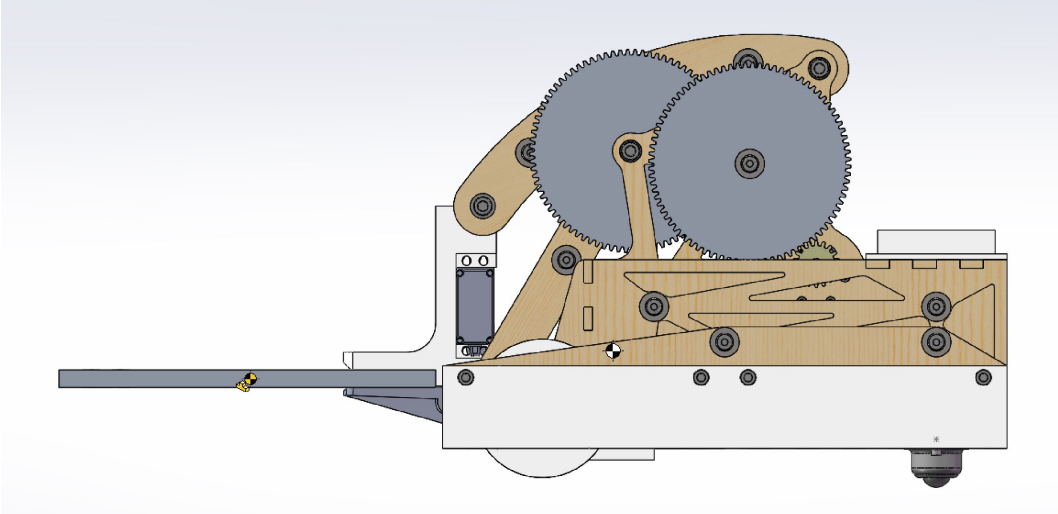


Figure 15: Robot CAD model with center of mass marked when in staging area pick-up configuration holding aluminum solar collector panel

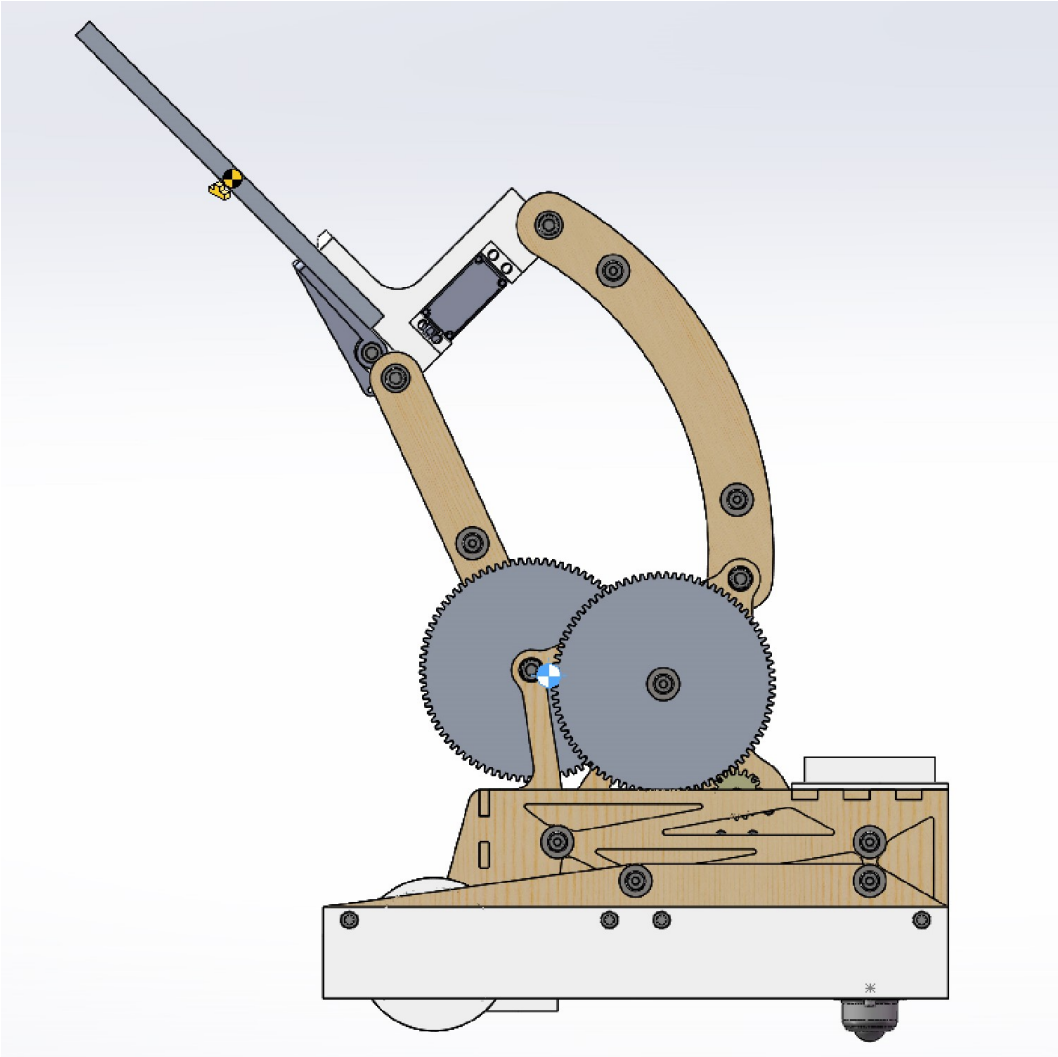


Figure 16: Robot CAD model with center of mass marked when in 25deg roof structure angle configuration holding aluminum solar collector panel

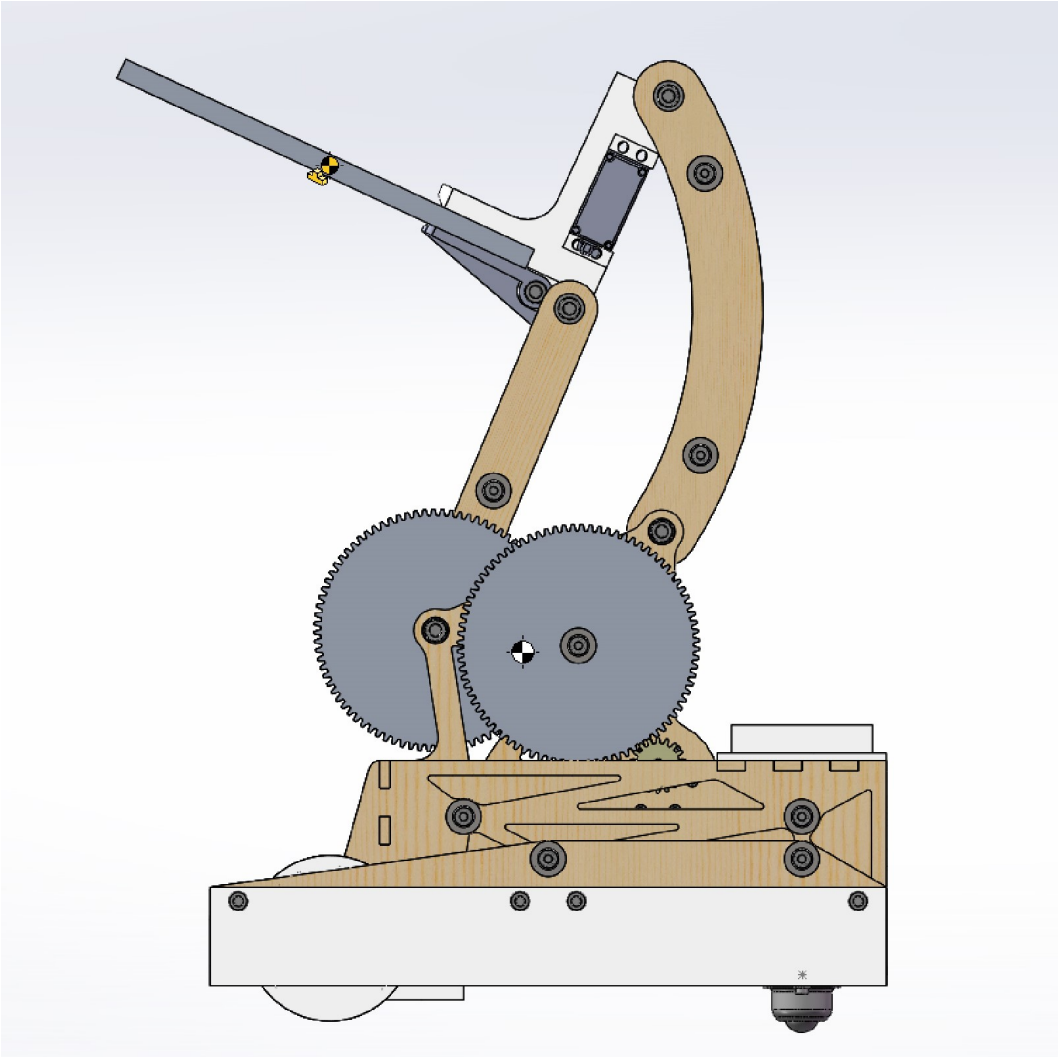


Figure 17: Robot CAD model with center of mass marked when in 45deg roof structure angle configuration holding aluminum solar collector panel

Appendix 5: High-level Sequence of Events

The general sequence of events once a pick order is received is as follows:

- Send Pick Order
- Grab station plate
- Pick up
- Navigate to the position commanded
- Wait for dropoff approval
- Drop off
- Let go
- Back up to line
- Navigate to the next position
- Wait for pickup approval
- Pick up
- Navigate to base station
- Wait for dropoff approval
- Drop off

At the end of the sequence, the robot may either remain at the base station or travel under the field roof to repeat the process on the other side.

Appendix 6: Code

```
/*
 * StudentsRobot.cpp
 *
 *  Created on: Dec 28, 2018
 *      Author: hephaestus
 */

#include "StudentsRobot.h"



StudentsRobot::StudentsRobot(ServoEncoderPIDMotor * motor1,
                ServoEncoderPIDMotor * motor2, HBridgeEncoderPIDMotor * motor3,
                Servo * servo) {
        Serial.println("StudentsRobot::StudentsRobot called here ");
        this->servo = servo;
        this->motor1 = motor1;
        this->motor2 = motor2;
        this->motor3 = motor3;

        // Set the PID Clock gating rate. Thie must be 10 times slower than the motors
update rate
        motor1->myPID.sampleRateMs = 30; // 330hz servo, 3ms update, 30 ms PID
        motor2->myPID.sampleRateMs = 30; // 330hz servo, 3ms update, 30 ms PID
        motor3->myPID.sampleRateMs = 1;  // 10khz H-Bridge, 0.1ms update, 1 ms PID
        // Set default P.I.D gains
        motor1->SetTunings(0.15, 0.0001, 1.6);
        motor2->SetTunings(0.15, 0.0001, 1.6);
        motor3->SetTunings(0.01, 0.0001, 0.6);

        // After attach, compute ratios and bounding
        double motorToWheel = 3;
        motor1->setOutputBoundingValues(0, //the minimum value that the output takes (Full
reverse)
                        180, //the maximum value the output takes (Full forward)
                        90, //the value of the output to stop moving
                        1, //a positive value added to the stop value to creep forwards
                        1, //a positive value subtracted from stop value to creep backward
                        16.0 * // Encoder CPR
                                        50.0 * // Motor Gear box ratio
                                        motorToWheel * // motor to wheel stage ratio
                                        (1.0 / 360.0) * // degrees per revolution
                                        motor1->encoder.countsMode, // Number of edges
that are used to increment the value
                        117.5 * // Measured max RPM
                                        (1 / 60.0) * // Convert to seconds
                                        (1 / motorToWheel) *  // motor to wheel ratio
                                        360.0); // convert to degrees
        motor2->setOutputBoundingValues(0, //the minimum value that the output takes (Full
reverse)
                        180, //the maximum value the output takes (Full forward)
                        90, //the value of the output to stop moving
                        1, //a positive value added to the stop value to creep forwards
                        1, //a positive value subtracted from stop value to creep backward
                        16.0 * // Encoder CPR
                                        50.0 * // Motor Gear box ratio
                                        motorToWheel * // motor to wheel stage ratio
                                        (1.0 / 360.0) * // degrees per revolution
                                        motor2->encoder.countsMode, // Number of edges
that are used to increment the value
                        117.5 * // Measured max RPM
                                        (1 / 60.0) * // Convert to seconds
```

```
                                        (1 / motorToWheel) *  // motor to wheel ratio
                                        360.0); // convert to degrees
        motor3->setOutputBoundingValues(-255, //the minimum value that the output takes
(Full reverse)
                        255, //the maximum value the output takes (Full forward)
                        0, //the value of the output to stop moving
                        1, //a positive value added to the stop value to creep forwards
                        1, //a positive value subtracted from stop value to creep backward
                        16.0 * // Encoder CPR
                                        50.0 * // Motor Gear box ratio
                                        1.0 * // motor to arm stage ratio
                                        (1.0 / 360.0) * // degrees per revolution
                                        motor3->encoder.countsMode, // Number of edges
that are used to increment the value
                        117.5 * // Measured max RPM
                                        (1 / 60.0) * // Convert to seconds
                                        360.0); // convert to degrees

chassis = new DrivingChassis(motor1, motor2, WHEEL_TRACK, WHEEL_RADIUS);
lineFollower = new LineFollow(chassis);

        // Set up the line tracker

        pinMode(LINE_SENSE_ONE, ANALOG);
        pinMode(LINE_SENSE_TWO, ANALOG);
        pinMode(EMITTER_PIN, OUTPUT);
 //pinMode(SERVO_PIN, OUTPUT);
 pinMode(MOTOR3_ENABLE_PIN, OUTPUT);
 pinMode(MOTOR3_DIR, OUTPUT);
servo->setPeriodHertz(50);
servo->attach(SERVO_PIN);
 servo->write(GRABBER_CLOSED);
}
/**
 * Seperate from running the motor control,
 * update the state machine for running the final project code here
 */



 int timeIdx = 0;
 long time1 = 0;
 double values[400][2];

void StudentsRobot::updateStateMachine() {

        long now = millis();


        switch (status) {
        case StartupRobot:
        Serial.println("State Machine Startup");

        myCommandsStatus = Ready_for_new_task;
    digitalWrite(EMITTER_PIN, HIGH); // turn on IR LEDs
    digitalWrite(MOTOR3_ENABLE_PIN, HIGH); // enable HBridge

    lastStatus = StartupRobot;
    nextStatus = PickupPanelStart;
    status = WAIT_FOR_PICKORDER; // wait for control station info
        break;

        case StartRunning:
  myCommandsStatus = Heading_to_pickup;
  //servo->write(GRABBER_OPEN);
```

```
  float val;
  if(roofAngle == 45){
   val = ENCODER_POS_45;
  }else if(roofAngle == 25){
    val = ENCODER_POS_25;
  }
  lineFollower->lineCount = 0;
  lineFollower->NUM_LINES_LANE / roofPos;
  motor3->startInterpolationDegrees(val, 5000, SIN);
  nextStatus = LF_Forward4_Init;
  status = WAIT_FOR_MOTORS_TO_FINISH;

  break;

        case PlacePanel:
                myCommandsStatus = Waiting_for_approval_to_dropoff;

                //determine appropriate angle to dropoff panel
                float placeAngle;
                if(roofAngle == 45){
                        placeAngle = ENCODER_POS_45;
                }else if(roofAngle == 25){
                        placeAngle = ENCODER_POS_25;
                }
                //move grabber to correct position
                motor3->startInterpolationDegrees(placeAngle, 2000, SIN);
                nextStatus = Transition_Post_Dropoff;
                lastStatus = PlacePanel;
                status = WAIT_FOR_APPROVE; //wait for approval to release
                break;
        case PlacePanelFinal:
                myCommandsStatus = Dropping_off;
                chassis->driveForward(-75, 1000); // backup as panel goes down to have
correct spacing
                motor3->startInterpolationDegrees(0, 5000, SIN); //slowly place panel down
                nextStatus = Halting;
                lastStatus = PlacePanelFinal;
                status = WAIT_FOR_MOTORS_TO_FINISH;
                break;

        case PickupPanel:
                myCommandsStatus = Waiting_for_approval_to_pickup;
                chassis->driveForward(75, 1000); // drive forward from line into correct
position
                nextStatus = Transition_Post_Pickup;
                lastStatus = PickupPanel;
                status = WAIT_FOR_APPROVE; //wait for operator approval
                break;
        case Transition_Post_Dropoff:
                myCommandsStatus = Dropping_off;
                servo->write(GRABBER_OPEN); // release panel
                numLines=NUM_LINES_RETURN_DROPOFF; //adjust line target for return
                lineFollower->lineCount = 0; //reset line counter
                motor3->startInterpolationDegrees(ENCODER_POS_MAX, 5000, SIN); //move
grabber off of panel
                chassis->driveForward(-50,1000); //backup from line
                nextStatus = LF_Backup2_Init; //begin backup
                lastStatus = Transition_Post_Dropoff;
                status = WAIT_FOR_MOTORS_TO_FINISH;
                break;
        case Transition_Post_Pickup:
                myCommandsStatus = Heading_to_safe_zone;
                servo->write(GRABBER_CLOSED); //grip panel
                lineFollower->lineCount = 0; //reset line counter
```

```
                lineFollower->numLines =NUM_LINES_LANE;  //adjust line target for return

                //pick up panel
                motor3->startInterpolationDegrees(ENCODER_POS_MAX, 5000, SIN);
                nextStatus = LF_Backup3_Init;
                lastStatus = Transition_Post_Pickup;
                status = WAIT_FOR_MOTORS_TO_FINISH;
                break;
        case PickupPanelStart:
                //lift panel slowly
          motor3->startInterpolationDegrees(ENCODER_POS_MAX, 5000, SIN)
          lastStatus = PickupPanelStart;
          nextStatus = LF_Backup_Init;
          status = WAIT_FOR_MOTORS_TO_FINISH;
          break;

        /* The following two states work in conjunction with one another.
        LF_Backup_Init moves the robot backward slightly, and then transitions to
        LF_Backup_Detect, which checks for the current line following status, and then
transitions back to
        LF_Backup_Init. The loop is broken once the correct number of horizontal lines
have been detected
        Additionally, after a horizontal line is detected, the CountLineReverse state may
be entered to avoid double-counting
        */

        case LF_Backup_Init:
                Serial.println("State: Backup_Init");
                chassis->driveForward(-10, 0); // move backward slightly
                lastStatus = LF_Backup_Init;
                status = LF_Backup_Detect; //next, check line following status

                break;
  case LF_Backup_Detect:

    Serial.println("State: Backup_Detect");
    //Uncomment line below for debugging purposes
    //lineFollower->readSensors();

    //check for horizontal line
    if(lineFollower->sensor1Val >= lineFollower->BLACK && lineFollower->sensor2Val >=
lineFollower->BLACK){
      lineCount++; //increment line counter

      if(roofPos == lineCount){ //check if current line is the right one
       nextStatus = LF_Transition_1; //begin the switch to crossing line
       status = STOP;
      }
      else{ //take appropriate action to resume looking for hoz. lines
        status = CountLineReverse;
      }
    }
    else{

    //chassis tilted clockwise relative to line
    if(lineFollower->sensor1Val >= lineFollower->BLACK && lineFollower->sensor2Val <=
lineFollower->WHITE){
        //straighten out -- rotate left wheel backward
        motor1->startInterpolationDegrees(motor1->getAngleDegrees() - 60, 1000, SIN);
    }
    //chassis tilted counterclockwise relative to line
    else if(lineFollower->sensor1Val <= lineFollower->WHITE && lineFollower->sensor2Val >=
lineFollower->BLACK){
        //straighten out -- rotate right wheel backward
        motor2->startInterpolationDegrees(motor2->getAngleDegrees() - 60, 1000, SIN);
```

```
    }
    lastStatus = LF_Backup_Detect;
    nextStatus = LF_Backup_Init; //loop
    status = WAIT_FOR_MOTORS_TO_FINISH;
    }
    break;

  case LF_Transition_2: //part 2 of the transition between backing up from staging area
and dropoff
          Serial.println("State: LF_Transition_2");
          lineFollower->lineCount = 0; //reset line counter
          lineFollower->numLines =NUM_LINES_LANE;//adjust lines target for roof lane
    chassis->driveForward(50, 1000); //scoot off of line
    nextStatus = LF_Forward_Init;
    status = WAIT_FOR_MOTORS_TO_FINISH;
    lastStatus = LF_Transition_2;
    break;
  case LF_Transition_1: //part 1 of the transition between backing up from staging area
and dropoff
    Serial.println("State: LF_Transition_1");
    chassis->turnDegrees(95, 4000); //turn to line up with roof lane line
    nextStatus = LF_Transition_2;
    status = WAIT_FOR_MOTORS_TO_FINISH;
    lastStatus = LF_Transition_1;
    break;


case LF_Transition2_1: //part 1 of the transition between backing up from dropoff and
navigating to pickup lane

    Serial.println("State: LF_Transition2_1");
    chassis->driveForward(15, 1000);
    nextStatus = LF_Transition2_2;
    status = WAIT_FOR_MOTORS_TO_FINISH;
    lastStatus = LF_Transition2_1;
    break;
  case LF_Transition2_2: //part 2 of the transition between backing up from dropoff and
navigating to pickup lane

    Serial.println("State: LF_Transition2_2");
    chassis->turnDegrees(90, 4000); // turns from current lane back onto crossing line
  //set number of lines opposite of pos value so robot goes to correct second spot
    lineFollower->numLines =1;
  if(roofPos == 1){
    lineFollower->numLines =2;
  }
    nextStatus = LF_Forward2_Init;
    status = WAIT_FOR_MOTORS_TO_FINISH;
    lastStatus = LF_Transition2_2;
    break;

case LF_Transition3_1: //part 1 of the transition between moving forward from the crossing
line and to pickup lane
    lineFollower->lineCount = 0; //reset line count
    Serial.println("State: LF_Transition3_1");
    chassis->driveForward(-10, 1000); //move VTC to line before turn
    nextStatus = LF_Transition3_2;
    status = WAIT_FOR_MOTORS_TO_FINISH;
    lastStatus = LF_Transition3_1;
    break;
  case LF_Transition3_2: //part 2 of the transition between moving forward from the
crossing line and to pickup lane

    Serial.println("State: LF_Transition3_2");
    chassis->turnDegrees(90, 4000); //turn onto the pickup lane line
```

```
    numLines=NUM_LINES_LANE;
    nextStatus = LF_Forward4_Init;

    status = WAIT_FOR_MOTORS_TO_FINISH;
    lastStatus = LF_Transition3_2;
    break;


    /* The following two states work in conjunction with one another.
        LF_Forward_Init moves the robot forward slightly, and then transitions to
        LF_Forward_Detect, which checks for the current line following status, and then
transitions back to
        LF_Forward_Init. The loop is broken once the correct number of horizontal lines
have been detected
        Additionally, after a horizontal line is detected, the CountLine state may be
entered to avoid double-counting
        */

  case LF_Forward_Init: //state used by the robot to get from cross line to the correct
position for collector placement

    Serial.println("State: LF_Forward_Init");
    followLineDrive(&status, &nextStatus,LF_Forward_Detect, WAIT_FOR_MOTORS_TO_FINISH,50);
    lastStatus = LF_Forward_Init;
    break;
  case LF_Forward_Detect:
        Serial.println("State: LF_Forward_Detect");
    lastStatus = LF_Forward_Detect; //state that checks and corrects line alignment until
correct number of horizontal lines crossed
    //Uncomment below for debugging
    //lineFollower->readSensors();
    followLineDetect(&status, &nextStatus, WAIT_FOR_MOTORS_TO_FINISH, LF_Forward_Init,
                        CountLine, LF_Transition_1,-60);
    break;
    case LF_Forward2_Init:

    Serial.println("State: LF_Forward2_Init");
    followLineDrive(&status, &nextStatus,LF_Forward2_Detect,
WAIT_FOR_MOTORS_TO_FINISH,dirMult*50);
    lastStatus = LF_Forward2_Init;
    break;
  case LF_Forward2_Detect:
    lastStatus = LF_Forward2_Detect;
    Serial.println("State: LF_Forward2_Detect");
    followLineDetect(&status, &nextStatus, WAIT_FOR_MOTORS_TO_FINISH, LF_Forward2_Init,
                        CountLine, LF_Transition3_1,-60);
    break;



  case LF_Forward3_Init:

    Serial.println("State: LF_Forward_Init");
    followLineDrive(&status, &nextStatus,LF_Forward3_Detect,
WAIT_FOR_MOTORS_TO_FINISH,25);
    lastStatus = LF_Forward3_Init;
    break;
  case LF_Forward3_Detect:
    lastStatus = LF_Forward3_Detect;
    Serial.println("State: LF_Forward3_Detect");
    lineFollower->readSensors();
    followLineDetect(&status, &nextStatus, WAIT_FOR_MOTORS_TO_FINISH, LF_Forward3_Init,
                        CountLinee, PickupPanel,-60);

case LF_Backup3_Init:
```

```
    lastStatus = LF_Backup3_Init;
    Serial.println("State: Backup3_Init");
    followLineDrive(&status, &nextStatus,LF_Backup3_Detect,
WAIT_FOR_MOTORS_TO_FINISH,-10);

    break;
  case LF_Backup3_Detect:
  lastStatus = LF_Backup3_Detect;
    Serial.println("State: Backup_Detect");
    lineFollower->readSensors();
    followLineDetect(&status, &nextStatus, WAIT_FOR_MOTORS_TO_FINISH, LF_Backup3_Init,
                        CountLineReverse, LF_Transition3_1,-60);
    break;


case LF_Forward4_Init:

    Serial.println("State: LF_Forward4_Init");
    followLineDrive(&status, &nextStatus,LF_Forward4_Detect,
WAIT_FOR_MOTORS_TO_FINISH,25);
    lastStatus = LF_Forward4_Init;
    break;
  case LF_Forward4_Detect:
          followLineDetect(&status, &nextStatus, WAIT_FOR_MOTORS_TO_FINISH,
LF_Forward4_Init,
                                CountLine, PlacePanelFinal, -60);    break;


   case CountLine:
   motor1->stop();
   motor2->stop();
    Serial.printf("LINE COUNT %d\n", lineCount);
    //lastStatus = CountLine;
    chassis->driveForward(25, 1000);
    //nextStatus = LF_Forward_Detect;
    nextTime=millis()+2000;
    status = WAIT_FOR_TIME;
    lastStatus = CountLine;
   break;
   case CountLineReverse:
   motor1->stop();
   motor2->stop();
    //Serial.printf("LINE COUNT %d\n", lineCount);
    //lastStatus = CountLine;
    chassis->driveForward(-25, 1000);
    //nextStatus = LF_Forward_Detect;
    nextTime=millis()+2000;
    //status = WAIT_FOR_MOTORS_TO_FINISH;
    status = WAIT_FOR_TIME;
    lastStatus = CountLineReverse;
   break;
   case LF_Backup2_Init:
    lastStatus = LF_Backup2_Init;
    Serial.println("State: Backup2_Init");
    followLineDrive(&status, &nextStatus,LF_Backup2_Detect,
WAIT_FOR_MOTORS_TO_FINISH,-50);

   break;
  case LF_Backup2_Detect:
  lastStatus = LF_Backup2_Detect;
    Serial.println("State: Backup_Detect");
    lineFollower->readSensors();
    followLineDetect(&status, &nextStatus, WAIT_FOR_MOTORS_TO_FINISH, LF_Backup2_Init,
                        CountLineReverse, LF_Transition2_1,-60);
    break;
```

```
        case WAIT_FOR_TIME:
                // Check to see if enough time has elapsed
                if (nextTime <= millis()) {
                        // if the time is up, move on to the next state
                        status = nextStatus;
                }
    lastStatus = WAIT_FOR_TIME;
                break;
        case WAIT_FOR_MOTORS_TO_FINISH:
   //Serial.println("State: WAIT_FOR_MOTORS_TO_FINISH");
   //Serial.println(motor3->getAngleDegrees());
                if (chassis->isChassisDoneDriving() && motor3->isInterpolationDone()) {
                        status = nextStatus;
                }
    lastStatus = WAIT_FOR_MOTORS_TO_FINISH;
                break;
   case WAIT_FOR_APPROVE:
   //Serial.println("State: WAIT_FOR_MOTORS_TO_FINISH");
   //Serial.println(motor3->getAngleDegrees());
     if (approve) {
       approve = false;
       status = nextStatus;
     }
     lastStatus = WAIT_FOR_APPROVE;
     break;

   case WAIT_FOR_PICKORDER:
   //Serial.println("State: WAIT_FOR_MOTORS_TO_FINISH");
   //Serial.println(motor3->getAngleDegrees());
     if (pickorder) {
       pickorder = false;
       status = nextStatus;
     }
     lastStatus = WAIT_FOR_APPROVE;
     break;
   case STOP:
     Serial.println("State: STOP");
     motor3->stop();
     motor2->stop();
     motor1->stop();
     status = nextStatus;
     lastStatus = STOP;
     break;
        case Halting:
   servo->write(GRABBER_OPEN);
                myCommandsStatus = Fault_obstructed_path;
                // save state and enter safe mode
                Serial.println("Halting State machine");
                digitalWrite(EMITTER_PIN, 0);
                motor3->stop();
                motor2->stop();
                motor1->stop();
                status = Halt;
                break;
        case Halt:
                // in safe mode
                break;
        }
}

/**
 * This is run fast and should return fast
 *
```

```
 * You call the PIDMotor's loop function. This will update the whole motor control system
 * This will read from the concoder and write to the motors and handle the hardware
interface.
 * Instead of allowing this to be called by the controller yopu may call these from a
timer interrupt.
 */
void StudentsRobot::pidLoop() {
        motor1->loop();
        motor2->loop();
        motor3->loop();
}
/**
 * Approve
 *
 * @param buffer A buffer of floats containing nothing
 *
 * the is the event of the Approve button pressed in the GUI
 *
 * This function is called via coms.server() in:
 * @see RobotControlCenter::fastLoop
 */
void StudentsRobot::Approve(float * buffer) {
        // approve the procession to new state
        Serial.println("StudentsRobot::Approve");
  approve = true;
        /*if (myCommandsStatus == Waiting_for_approval_to_pickup) {
                myCommandsStatus = Waiting_for_approval_to_dropoff;
        } else {
                myCommandsStatus = Ready_for_new_task;
        }*/
}
/**
 * ClearFaults
 *
 * @param buffer A buffer of floats containing nothing
 *
 * this represents the event of the clear faults button press in the gui
 *
 * This function is called via coms.server() in:
 * @see RobotControlCenter::fastLoop
 */
void StudentsRobot::ClearFaults(float * buffer) {


        Serial.println("StudentsRobot::ClearFaults");
 Serial.printf("status: %d\n",  status);
        //myCommandsStatus = Ready_for_new_task;
        status = lastStatus;
}

/**
 * EStop
 *
 * @param buffer A buffer of floats containing nothing
 *
 * this represents the event of the EStop button press in the gui
 *
 * This is called whrn the estop in the GUI is pressed
 * All motors shuld hault and lock in position
 * Motors should not go idle and drop the plate
 *
 * This function is called via coms.server() in:
 * @see RobotControlCenter::fastLoop
 */
void StudentsRobot::EStop(float * buffer) {
```

```
        // Stop the robot immediatly
        Serial.println("StudentsRobot::EStop");
 Serial.printf("status: %d\n",  status);
        myCommandsStatus = Fault_E_Stop_pressed;
        status = Halting;

}
/**
 * PickOrder
 *
 * @param buffer A buffer of floats containing the pick order data
 *
 * buffer[0]  is the material, aluminum or plastic.
 * buffer[1]  is the drop off angle 25 or 45 degrees
 * buffer[2]  is the drop off position 1, or 2
 *
 * This function is called via coms.server() in:
 * @see RobotControlCenter::fastLoop
 */
void StudentsRobot::PickOrder(float * buffer) {
        pickorder = true;

        material = buffer[0];
        roofAngle = buffer[1];
        roofPos = buffer[2];


  if(roofPos == 1){
    dirMult = -1;
  }else{
    dirMult = 1;
  }

  if(material == ALUMINUM){
    motor3->SetTunings(KP_ALUM, KI_ALUM, KD_ALUM);
  }
  else if(material == PLASTIC){
    motor3->SetTunings(KP_PLASTIC, KI_PLASTIC, KD_PLASTIC);
  }
        //myCommandsStatus = Waiting_for_approval_to_pickup;
}
```

```
/*
 * config.h
 *
 *  Created on: Nov 5, 2018
 *      Author: hephaestus
 */

#ifndef SRC_CONFIG_H_
#define SRC_CONFIG_H_

#define TEAM_NAME "Team2"

#define USE_WIFI

//
#define WHEEL_TRACK 260
#define WHEEL_RADIUS (3.165*25.4/2.0)
//Line Following
#define NUM_LINES_LANE 4
#define NUM_LINES_CROSS 1
#define TURN_ANGLE 90
#define OFFSET = -10;

//PID vals
#define KP_PLASTIC 0.001
#define KI_PLASTIC 0.0001
#define KD_PLASTIC 0.9
#define KP_ALUM 0.01
#define KI_ALUM 0.0001
#define KD_ALUM 0.6

//Materials
#define ALUMINUM 1.00
#define PLASTIC 0.00
// Pins

/**
 * Drive motor 1 Servo PWM pin
 */
#define MOTOR1_PWM 15
/**
 * Drive motor 2 Servo PWM pin
 */
#define MOTOR2_PWM 4
/**
 * Drive motor 3 10Khz full duty PWM pin
 */
#define MOTOR3_PWM 12
/**
 * Pin for setting the direction of the H-Bridge
 */
#define MOTOR3_DIR 26
#define MOTOR3_ENABLE_PIN 13

//Encoder pins
#define MOTOR1_ENCA 18
#define MOTOR1_ENCB 19

#define MOTOR2_ENCA 17
#define MOTOR2_ENCB 16

#define MOTOR3_ENCA 27
#define MOTOR3_ENCB 14
```

```
#define ENCODER_POS_45 1850.0
#define ENCODER_POS_25 3580.0
#define ENCODER_POS_MAX 3400.0
// Line Sensor Pins
#define LINE_SENSE_ONE                  36
#define LINE_SENSE_TWO                  39
#define EMITTER_PIN         34  // emitter is controlled by digital pin



/**
 * Gripper pin for Servo
 */
#define SERVO_PIN 5
#define GRABBER_OPEN 110
#define GRABBER_CLOSED 180

#endif /* SRC_CONFIG_H_ */
```

```
/*
 * DrivingChassis.cpp
 *
 *  Created on: Jan 31, 2019
 *      Author: hephaestus
 */

#include "DriveChassis.h"
#include <math.h>

/**
 * Compute a delta in wheel angle to traverse a specific distance
 *
 * arc length   =       2*      π*      R*      (C/360)
 *
 * C  is the central angle of the arc in degrees
 * R  is the radius of the arc
 * π  is Pi
 *
 * @param distance a distance for this wheel to travel in MM
 * @return the wheel angle delta in degrees
 */
float DrivingChassis::distanceToWheelAngle(float distance) {
        float deltaAngle = (distance / mywheelRadiusMM) * (180 / M_PI);
        return deltaAngle;
}

/**
 * Compute the arch length distance the wheel needs to travel through to rotate the base
 * through a given number of degrees.
 *
 * arc length   =       2*      π*      R*      (C/360)
 *
 * C  is the central angle of the arc in degrees
 * R  is the radius of the arc
 * π  is Pi
 *
 * @param angle is the angle the base should be rotated by
 * @return is the linear distance the wheel needs to travel given the this CHassis's wheel
track
 */
float DrivingChassis::chassisRotationToWheelDistance(float angle) {
        float arcLength = 2 * M_PI * (mywheelTrackMM / 2) * (angle / 360);
        return arcLength;
}

DrivingChassis::~DrivingChassis() {
        // do nothing
}

/**
 * DrivingChassis encapsulates a 2 wheel differential steered chassis that drives around
 *
 * @param left the left motor
 * @param right the right motor
 * @param wheelTrackMM is the measurment in milimeters of the distance from the left wheel
contact point to the right wheels contact point
 * @param wheelRadiusMM is the measurment in milimeters of the radius of the wheels
 */
DrivingChassis::DrivingChassis(PIDMotor * left, PIDMotor * right,
                float wheelTrackMM, float wheelRadiusMM) {
        myleft = left;
        myright = right;
        mywheelTrackMM = wheelTrackMM;
```

```
        mywheelRadiusMM = wheelRadiusMM;


}

/**
 * Start a drive forward action
 *
 * @param mmDistanceFromCurrent is the distance the mobile base should drive forward
 * @param msDuration is the time in miliseconds that the drive action should take
 *
 * @note this function is fast-return and should not block
 */
void DrivingChassis::driveForward(float mmDistanceFromCurrent, int msDuration) {
        myleft->startInterpolationDegrees(myleft->getAngleDegrees() +
distanceToWheelAngle(mmDistanceFromCurrent), msDuration, SIN);
        myright->startInterpolationDegrees(myright->getAngleDegrees() +
distanceToWheelAngle(mmDistanceFromCurrent), msDuration, SIN);
}

/**
 * Start a turn action
 *
 * This action rotates the robot around the center line made up by the contact points of
the left and right wheels.
 * Positive angles should rotate to the left
 *
 * This rotation is a positive rotation about the Z axis of the robot.
 *
 * @param degreesToRotateBase the number of degrees to rotate
 * @param msDuration is the time in milliseconds that the drive action should take
 *
 *  @note this function is fast-return and should not block
 */
void DrivingChassis::turnDegrees(float degreesToRotateBase, int msDuration) {
        //if(degreesToRotateBase > 0){
                myright->startInterpolationDegrees(myright->getAngleDegrees() +
distanceToWheelAngle(chassisRotationToWheelDistance(degreesToRotateBase)), msDuration,
SIN);
    myleft->startInterpolationDegrees(myleft->getAngleDegrees() -
distanceToWheelAngle(chassisRotationToWheelDistance(degreesToRotateBase)), msDuration,
SIN);
        /*}else{
    myright->startInterpolationDegrees(myright->getAngleDegrees() -
distanceToWheelAngle(chassisRotationToWheelDistance(degreesToRotateBase)), msDuration,
SIN);
                myleft->startInterpolationDegrees(myleft->getAngleDegrees() +
distanceToWheelAngle(chassisRotationToWheelDistance(degreesToRotateBase)), msDuration,
SIN);
        }*/




}

/**
 * Check to see if the chassis is performing an action
 *
 * @return false is the chassis is driving, true is the chassis msDuration has elapsed
 *
 *  @note this function is fast-return and should not block
 */
bool DrivingChassis::isChassisDoneDriving() {
        return (myleft->isInterpolationDone() && myright->isInterpolationDone());
```

```
}
```

```
/*
 * LineFollow.cpp
 *
 *  Created on: Feb 22, 2019
 *      Author: jaconkliun
 */

#include "LineFollow.h"
#include "Arduino.h"
#include "StudentsRobot.h"

LineFollow::~LineFollow() {
  // do nothing
}
        LineFollow::LineFollow(DrivingChassis *achassis){
                this->chassis = achassis;
    this->sensor1Val = 0;
    this->sensor2Val = 0;
    this->lineCount = 0;
    this->numLines = 0;
        }
        /**
         * read read sensor(s) and update values
         */
        void LineFollow::readSensors(){
                sensor1Val = analogRead(LINE_SENSE_ONE);
                sensor2Val = analogRead(LINE_SENSE_TWO);
    Serial.printf("Sen1:%d,Sen2:%d\n", sensor1Val, sensor2Val);
        }
        /**
         * followLine take appropriate chassis action based on sensor values
         */
        void LineFollow::followLineDetect(RobotStateMachine * myStatus, RobotStateMachine
* myNextStatus, RobotStateMachine waitStatus, RobotStateMachine driveStatus,
RobotStateMachine countStatus, RobotStateMachine doneStatus, int correctionAmt){
                if(sensor1Val >= BLACK && sensor2Val >= BLACK){
                      lineCount++;
                      if(lineCount >= numLines){
                      *myNextStatus = driveStatus;
                      *myStatus = doneStatus;
                      }
                      else{
                          *myStatus = countStatus;
                      }

                    }
                    else{
                      if(sensor1Val >= BLACK && sensor2Val <= WHITE){

chassis->myleft->startInterpolationDegrees(chassis->myleft->getAngleDegrees() +
correctionAmt, 1000, SIN);
                    }
                    else if(sensor1Val <= WHITE && sensor2Val >= BLACK){

chassis->myright->startInterpolationDegrees(chassis->myright->getAngleDegrees() +
correctionAmt, 1000, SIN);
                    }
                    *myNextStatus = driveStatus;
                    *myStatus = waitStatus;
                    }
        }


        void LineFollow::followLineDrive(RobotStateMachine * myStatus, RobotStateMachine *
```

```
myNextStatus, RobotStateMachine detectStatus, RobotStateMachine waitStatus,int
forwardAmt){
                chassis->driveForward(forwardAmt, 1000);
                *myNextStatus = detectStatus;
                *myStatus = waitStatus;
        }

        /**
         * getLineCount return current number of counted perpendicular lines
         * @return number of perpendicular lines counted since last resetCount()
         */
        int LineFollow::getLineCount(){
                return lineCount;
        }
        /**
         * resetLineCount reset the number of counted perpendicular lines
         */
        void LineFollow::resetLineCount(){
                lineCount = 0;
        }
```

```
/*
 * Messages.h
 *
 *  Created on: 10/1/16
 *      Author: joest
 */
#include "Arduino.h"
#include "RBEPID.h"
#include <math.h>

//Class constructor
RBEPID::RBEPID() {

}

//Function to set PID gain values
void RBEPID::setpid(float P, float I, float D) {
        kp = P;
        ki = I;
        kd = D;
}

/**
 * calc the PID control signel
 *
 * @param setPoint is the setpoint of the PID system
 * @param curPosition the current position of the plan
 * @return a value from -1.0 to 1.0 representing the PID control signel
 */
float RBEPID::calc(double setPoint, double curPosition) {
        // calculate error
        float err = setPoint - curPosition;
        // calculate derivative of error
        float derErr = err - last_error;
        // calculate integral error. Running average is best but hard to implement

  //check for sign change and reset integral buffer
  if((last_error != 0) && (err / last_error) < 0){
    clearIntegralBuffer();
  }

        sum_error += err;
        // sum up the error value to send to the motor based off gain values.
        float out = (kp * err) + (ki * sum_error) + (kd * derErr);
  last_error = err;

  out = fmin(out, 1);
  out = fmax(out, -1);
        return out;
}

/**
 * Clear the internal representation fo the integral term.
 *
 */
void RBEPID::clearIntegralBuffer() {
        sum_error = 0;
}
```

Appendix 7: Contributions

Table 3: Team member contributions for lab and final project

Student Name	Contribution to: (%)	
	Lab	Final Project
Jason Conklin	33.3	33
Teresa Saddler	33.3	34
Benjamin Ward	33.3	33