“YodaBot”: A Robotics Final Project

RBE 2001

Section C01

Sierra Palmer, Nicholas Benoit, Max Luu

03/03/2017

Abstract

This paper discusses the process Team 2 took in developing YodaBot, a robot designed in order to accomplish the tasks set forth by the RBE 2001 final project challenge. The robot's objectives were to obtain used nuclear rods and replace them with new ones by depositing used rods in the Used Rod Storage and picking up new ones in the New Rod Supply Area, be able to navigate the field, and respond to the field's sensing via Bluetooth. YodaBot was able to accomplish the base requirements of the challenge by our team working through calculations and coming up with creative ideas to solve problems, such as designing a new gripper because the provided one would not work with the team's four bar design. It was able to pick up and deposit rods from both sides of the field and interact with the field at the beginning of the time period in order to determine where rods were positioned on the field. As a whole, the team learned a great deal from this challenge, not only within their field, but within aspects such as time management and team dynamic.

Table of Contents

List of Figures

Introduction and Background

As a challenge for the final project of Unified Robotics 1, teams are tasked with designing and creating a robot that is able to replace spent fuel rods located in nuclear reactors.

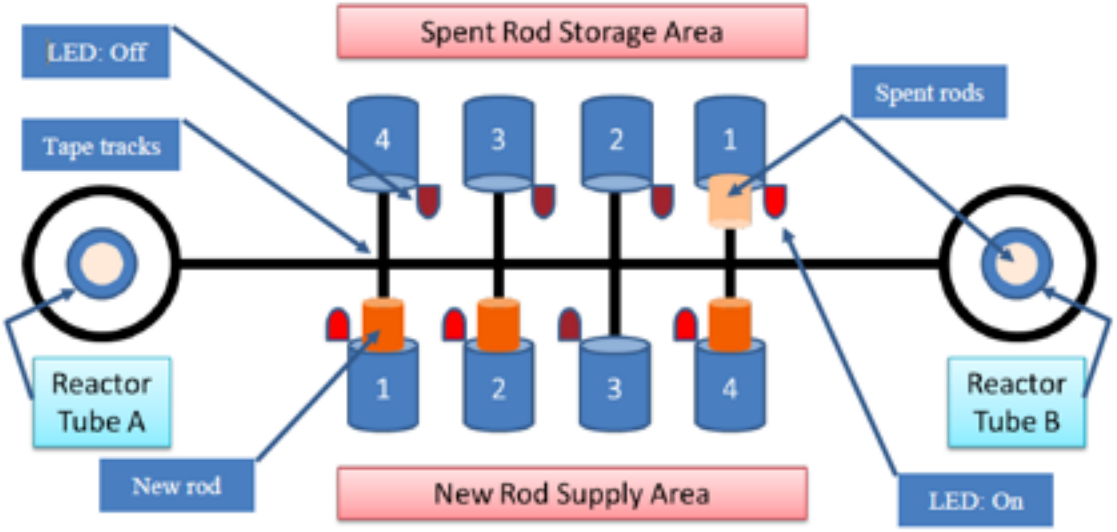

Figure 1: Playing field layout

The object of the challenge is to remove the used or "spent" fuel rods positioned perpendicular to the table and held within Reactor Tubes A and B and place them in the Spent Rod Storage Area. The Spent Rod Storage Area and New Rod Supply Area are of identical structures; both contain four horizontal tubes with a corresponding LED that is lit when it contains a fuel rod.

This challenge is a multi-task process that is to be completed within a time period of ten minutes. Within this period, the robot must drive (tele-operate) to a Reactor Tube and then remove the spent rod without substantial rub against the sides of the tube. From then on, the robot operates autonomously. It must then locate an empty rod storage container via either Bluetooth signals received from the field or from a sensor that detects the corresponding illuminated LED and store the spent rod. Using the same location technology, the robot must navigate to the New Rod Supply Area, collect a new rod from the container, and then return to the empty reactor to place the new rod. This process is then repeated for the Reactor Tube on the opposite side.

Throughout the duration of the challenge, the robot must be able to notify those in the surrounding area that there is radiation present. This means that whenever a rod is exposed, (i.e.

not housed in either a reactor tube or in a supply area), there must be a visual alert, such as an LED or a printed message on the LCD.

As the weeks progress in the term, teams must keep in mind certain aspects when designing and building their robot. These include being able to use most of the material taught during the course, being able to demonstrate good software development and coding practices while also dealing with autonomous and tele-operated robot functions, and using a four-bar mechanism to move the rod to its different orientations. It is also highly suggested that teams utilize Bluetooth to communicate with the field while using a message packet protocol.

Methodology

The first step in generating a design for this project was to read through the challenge for the final project and realize the different components that would have to go into accomplishing it. It was also a matter of breaking the challenge into subsections that would allow for easier design and analysis. As a team, it was decided that there were three main physical components to the robot: chassis, four bar, and gripper.

Chassis

When designing the chassis, it was decided that the drive train needed to be the first focus due to the fact that the robot required movement on the field. Different ideas for the drive train were introduced after taking into account the multiple obstacles the robot had to face. After looking at the line follow course on the field, it became clear that the robot would only require 90 or 180 degree turns. Because of this, the turning center would need to be in the center of the robot. To accomplish this, a six-wheel drive train with dropped center wheels was selected. The center wheel on either side would be non-slip 2.25” diameter wheels with a 1:3 gear ratio connected to a Vex 393 motor; they would be the only driven wheels. A 1:3 gear ratio was chosen to achieve the 2 inch per second goal speed. This will be further discussed in the Analysis section.The remaining four wheels, one on each corner, would be the 2.25” omni-directional wheels that would allow for easier maneuverability on turns. These wheels are mounted with metal Vex C-brackets in order to create a sturdy connection to the chassis sides.

The chassis itself is a seven inch by nine inch by one and a quarter inch plywood robot that has been laser cut to fit together. The name YodaBot came from comparing the sizes of other teams’ robots to this one and realizing it was much smaller. The joke then became that “Size matters not” as long as the robot can work. The chassis also contains a multi-purpose bottom

plate that snaps to the sides. It acts as a mounting plate for the two Vex 393 motors of the drive train, has an alignment section on the front in order for the four bar to be able to reach the fuel rod, and is also a mounting point for the 2 Pololu light sensors to be used for line following.

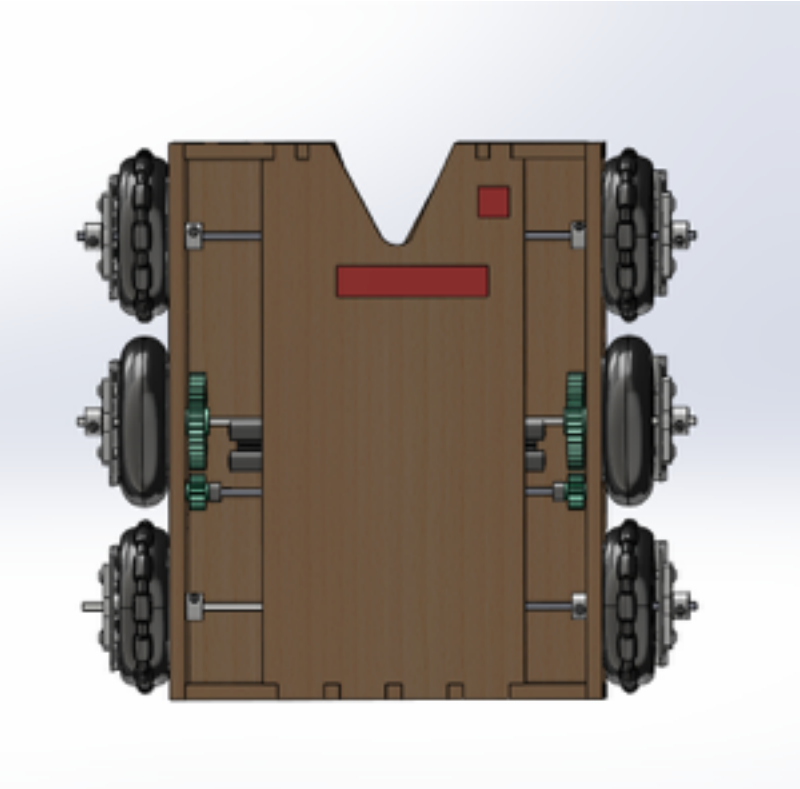

Figure 2
Bottom View of Chassis with Light Sensors (red bars) and Drive Train

Four Bar

The four bar went through many iterations before reaching the final product mounted on YodaBot. The first was a four bar design with a rack and pinion mounted on it that would move between horizontal and vertical positions. This allowed for the robot to grab the rod out of the reactor tube and pull up with no rub from the vertical position. The arm system would then rotate backwards so that the arm would be horizontal, and then, using the rack and pinion, the rod would be placed into the spent rod storage area. However, with the way this design was driven, the coupler never moved, creating a two bar, which would not accomplish one of the challenge requirements.

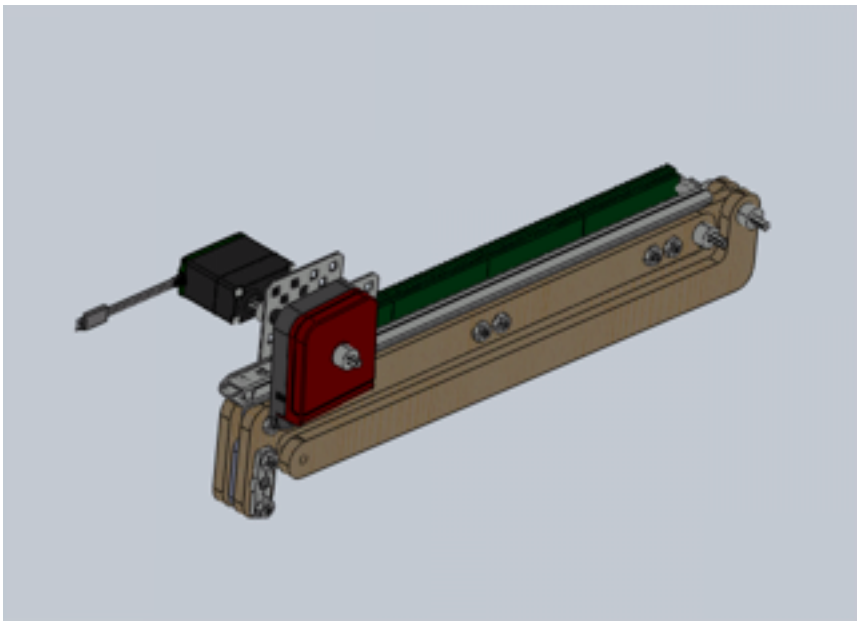

Figure 3
Original "Four-bar" Design

After this first iteration, discussions arose on how to build a four bar system that would work correctly. The second idea was to create a crossed four-bar that would allow for two positions. However, due to the small size of the robot, it would be unable to be mounted. This led to the third iteration of the four bar system.

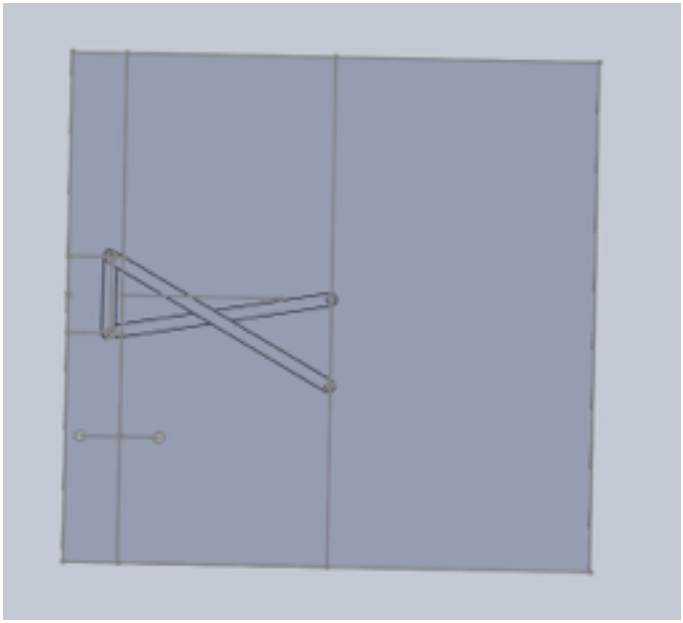

Figure 4
Second Iteration Four-Bar: Crossed Configuration

The third iteration of the four bar system consists of a large link of 3.94 inches and a short link of 2.25 inches. These two link differ in the vertical direction by 1.19 inches and in the horizontal direction by 0.03 inches. The coupler is the link used to assist with the pickup of the rod and is 3.0 inches in length. The 3.94 inch link is considered the crank in the system and is powered in the same configuration as the drive train, which is a 1:3 gear ratio driven by a Vex 393 motor. Calculations are discussed in the Analysis section.

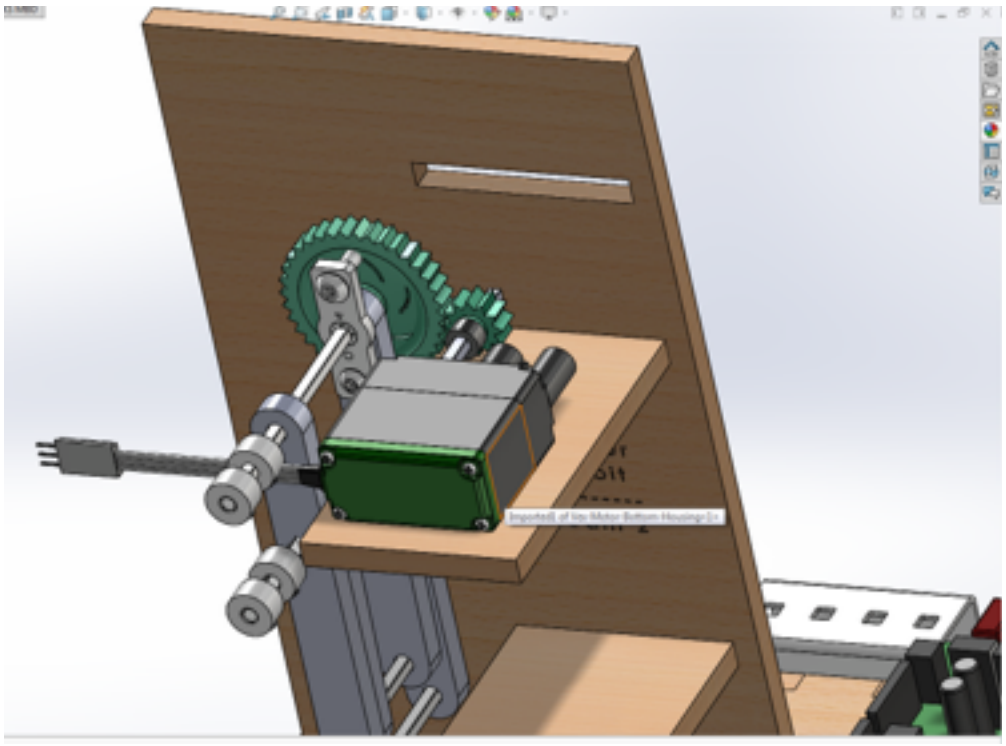

Figure 5
Motor Mount and Gearing for Final Four-Bar Design

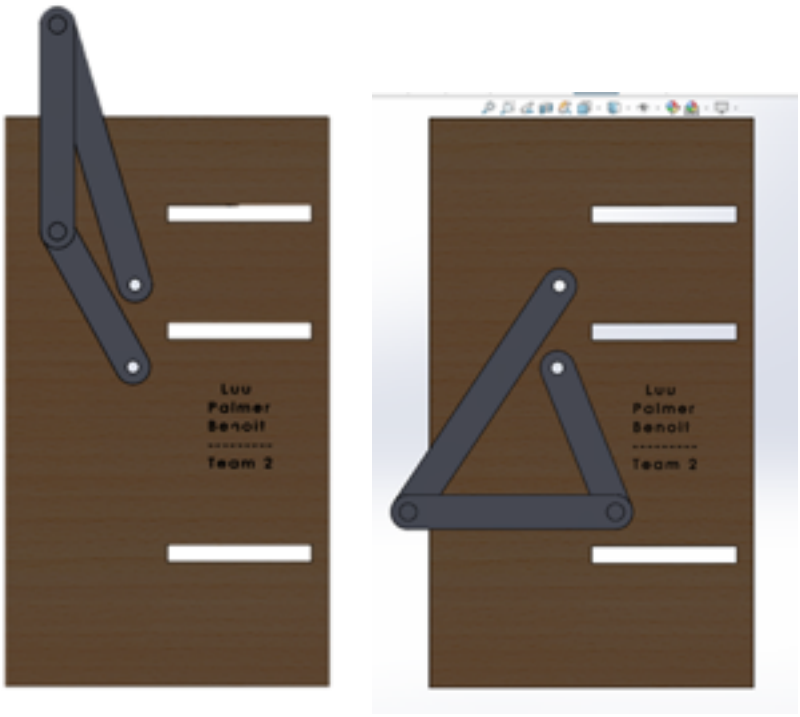

Figure 6
Top (left) and Bottom (right) Positions of the Four Bar

Gripper

Originally, the idea for the gripper was to utilize the one provided in the kits. However, after looking more in depth at the final four bar design, it was discovered that the system would be unable to remove the rod from the reactor tube. This is due to the fact that the system would not be able to move vertically in its bottom position to remove the rod and move to the top position. In order to alleviate this problem, a new gripper was designed.

The new gripper consists of an originally designed 3D printed piece with dimensions of 3 inches by 1.5 inches. It has a central slot that allows for a rod to be passed through and inside the four bar system by a series of rollers connected to a Vex 296 motor. The rollers are covered in orthodontic rubber bands in order to increase friction. The motor can rotate in both directions in order to pick up the rod from the reactor tube and then distribute it into the specified storage area.

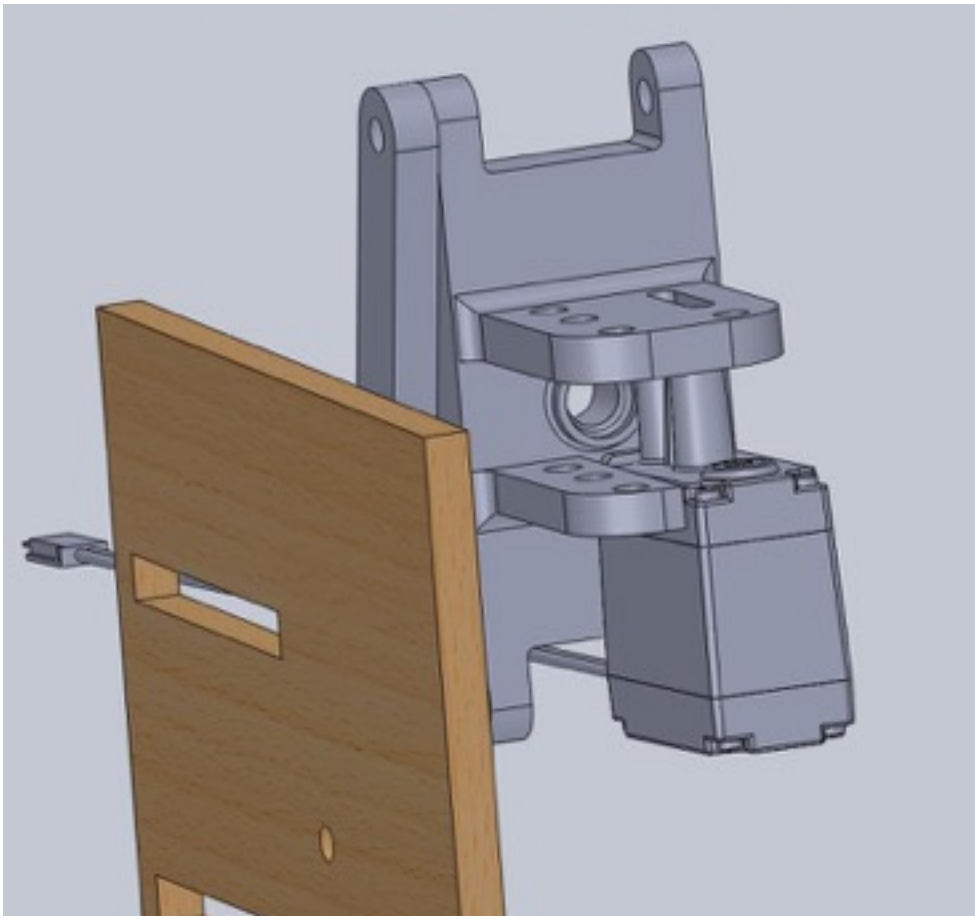

Figure 7
Gripper Mechanism Design Using Vex 269 motor (not shown: motor shaft roller and idler roller)

After the physical design process concluded, it was then time to turn sights towards developing a strategy in order to achieve the tasks given in the challenge. This would then be turned into the outline for the code, that would eventually become the final state machine. Due to the linearity of the challenge, the outline mimicked this pattern. This is displayed in a programming flowchart that is located in Appendix A.

Code

The Nuclear Reactor problem is very linear in nature, and this linear procedure allowed us to write very simple code. The structure of our Loop was a switch case ladder, with each case in the switch was a step in the process of refuelling the reactor. The decision making in the program was largely concerned with progressing to the next step of the program, as opposed to a system that was concerned with choosing between which step to progress to.

The code was written to conserve time, and code was reused as much as possible. Wherever a function was being called multiple times, it because a sub-function with an if-else ladder. All drive functions were grouped together, as were four bar functions and collector functions. This structure also allowed for easy implementation of the state machine structure.

Analysis

In designing the robot, the first problem to be solved would be to determine the desired speed of the drive train and the motors and gearing needed in order to achieve it. When looking at the challenge, the robot does not require to travel quickly as it only has to travel over short distances. Due to this, it was decided to have a goal speed of 2 inches per second. In order to calculate this, we had to look at the Power and Current table at Steady-State to determine the maximum speed the robot would be able to travel when connected to a Vex 393.

Speed (rpm)	Torque (N m)	Torque (in lbs)	Current (A)	Power (wt)	Efficiency	Heat (wt)
0	1.67	14.76	4.800	0.0	0%	35
7	1.56	13.78	4.505	1.1	3%	31
13	1.45	12.79	4.209	2.0	7%	28
20	1.33	11.81	3.914	2.8	10%	25
27	1.22	10.82	3.619	3.4	13%	23
33	1.11	9.84	3.323	3.9	16%	20
40	1.00	8.86	3.028	4.2	19%	18
47	0.89	7.87	2.733	4.3	22%	15
53	0.78	6.89	2.437	4.3	25%	13
60	0.67	5.90	2.142	4.2	27%	11
67	0.56	4.92	1.847	3.9	29%	9
73	0.44	3.94	1.551	3.4	31%	8
80	0.33	2.95	1.256	2.8	31%	6
87	0.22	1.97	0.961	2.0	29%	5
93	0.11	0.98	0.665	1.1	23%	4
100	0.00	0.00	0.370	0.0	0%	3

Figure 8
Power and Current Table at Steady-State for a VEX 393

According to the table as seen in Figure 8, if a motor is running at maximum speed, then it would be moving at a velocity of 13.09 in/sec.

Given:
Rpm = 100

Circumference = diameter*pi = 2.5*pi
100 (rev/min)*2.5(in/rev)*1/60(min/sec) = 13.09 in/sec

After this data, it was realized that this speed would be unnecessary and could hinder YodaBot's performance; the motor would have to be geared down. To create a gear ratio that would allow for the desired 2 in/sec to be within a more efficient range, it was decided to use a 1:3 ratio by utilizing a 36 tooth gear driven by a 12 tooth gear that is attached to the motor. With the new ratio, the maximum speed the robot would be able to achieve is 4.367 in/sec. Compare this to the 2 in/sec desired speed, and it is found that the system runs at about 46% of its maximum speed, allowing for about a 23% efficiency when comparing to the table in Figure 8. The calculations for the geared down drive train speeds as well as the output torque can be seen in Appendix B, where they were calculated in the computer program, MathCAD.

In these calculations, n_in denotes the 12 teeth of the driving gear, n_out denotes the 36 teeth of the driven gear, r denotes the diameter of the wheel, w_wheel denotes the angular speed of the wheel, w_in denotes the angular speed going into the system, and t_in denotes the torque going into the system.

The next source for major calculations was for the four bar system mounted on the robot. This system utilized the same Vex 393, so Figure 8 was once again used. There were three different calculations that had to be made for the system: position, force, and component speed.

To calculate the position of the four bar, which allowed us to solve for Theta 3 and Theta 4, a series of equations that can be found in Appendix C can be utilized. The calculations were done in MathCAD and can be found in Appendix D. The most difficult thing about this calculation was determining where in the four bar design Theta 3 and Theta 4 were located, especially with the design being so dissimilar to the design introduced as an example in the original equations.

The example design and the team's design may be found in Appendix C along with the original equations.

Most of the values, such as Kone, are calculated based on the given equations. However, the values of a, b, cc, d, and Theta2 are the values of the follow, crank, coupler, ground, and the angle between ground and the follow, respectively. This allowed for the determining of the angles of the four bar's starting position.

After determining the measurements of the starting position, it was then time to move towards the calculating the forces applied to each link. This would be a key factor in figuring out how much of an output torque the motor would require. This was solved by setting up nine equations of equilibrium to solve for the nine unknowns, which turn out to be the forces in the X and Y direction for each of the coupled points and the moment about the crank. The known parameters are the lengths of the crank, 3.49 in, coupler, 3 in, follower, 2.25 in, y-distance of ground, 1.19 in, x-distance of ground, 0.03 in, weight of link 2 or the follower, .00440925 lbf, weight of link 3 or the coupler, .220462 lbf, weight of link 4 or the crank, .00881849 lbf, and Theta 2, 3, and 5 based on calculations from measurements from the SolidWorks model. The equations of equilibrium solved in MathCAD can be found in Appendix E.

The final calculation that needed to be conducted in order to gain a full understanding of the workings of the four bar, which would be calculating the angular speed of the links in radians per second and the velocity of the linkage point in inches per second. It was assumed that the crank should move at a speed of 2 inches per second because it allows for the four bar system to move quick enough to where it will not put too much force on the system, but also will not go out of control due to its inability to control its speed. Appendix F contains the calculations done in MathCAD to find the velocity and angular speed of each entity in the system

Results

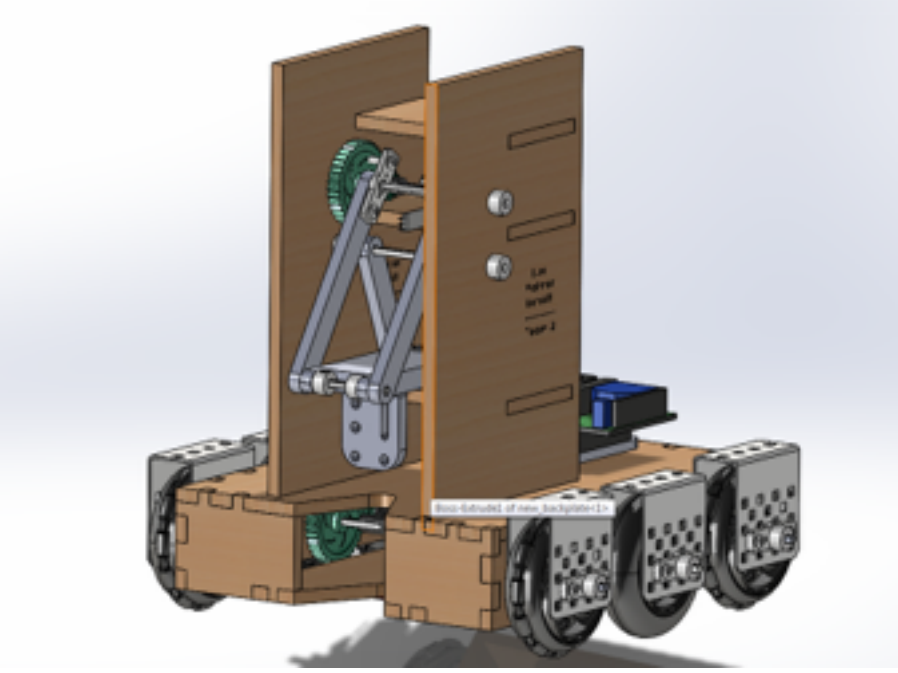

Figure 9

Final Robot CAD model

Prior to the final project presentation, the robot could complete multiple tasks individually. This included removing the nuclear waste rod from the reactor tube, line following to drop place it in the spend rod storage area, retrieving a new rod from the New rod supply area, and then placing it into the reactor tube. The robot was able to notify those in the surrounding area when it was holding a nuclear rod by utilizing a small speaker that would emit an alarm. The line following on the robot is slow, but allows for the robot to be completely autonomous for the duration of the challenge.

During the final project presentation, the robot was completely autonomous. The only time the robot had to be touched was during the calibration process, where the Pololu sensors had to be moved across the line in order to obtain the correct values to determine the difference between the lines and the white base board. The robot was able to collect the old rod from the reactor tube 3 out of 5 times. One of the largest reasons for the robot's inability to collect the rod every time was due to the lack of traction from the duct tape-wrapped rollers. This also hindered any other rod manipulation for the rest of the challenge. The robot was also only able to complete the challenge on one side of the course due to the way the code was written. Radiation notifications were also changed for the presentation, as the either "high radiation" or "low radiation" were printed on the LCD depending on which type of rod the robot contained. It also lacked any Bluetooth commands for the completion of the challenge.

Discussion

One of the major issues when starting the process for this project was that the team was extremely optimistic and had many ideas that it could work on. However, as the weeks progressed, it felt as though the designs were ahead of scheduled, so work slowed to a complete stop at some points. This caused for a delay in design deadlines being met, and going from a position of being ahead of schedule to being behind. As a result, goals that were set forth in the beginning had to be dropped in order to achieve the basic goals. Teammates also felt that there were times when they did not know what to work on, which also caused a delay in the process. The team has discussed that this method was not functional and that a new process should be implemented in the future. This new method includes not only group weekly goals and objectives, but also individual ones to be updated at the end of the week. This will allow for greater progression and overall better team dynamic for future projects.

The chassis worked well in that it was a sturdy base for the rest of the robot to be built upon. However, in the future, the most beneficial process would be for the robot to be fully designed before being laser cut. This includes any sensor mountings, wiring holes, etc. Because the design was not fully planned out prior to building, many devices were left out, such as the light sensor mountings or holes for the drive train wiring to go through. This left the team to

come up with creative ways for mounting the bottom plate and the light sensors, which consisted of the bottom plate being taped to the main chassis with electrical tape and the light sensors being mounted through the plate by using a drill press to drill multiple holes before the piece where the light sensor would be was punched out. It did not cause too many issues with the robot operationally, but was not as aesthetically pleasing as the original design was.

The drive train was able to achieve the maneuverability that was required of the challenge. The only thing the team felt could have been changed was to mount the dropped center wheel a little higher in order for the opposite omni-wheels to touch the drive surface as the robot turned.

The four bar was functional in terms of links and gearing. The team originally wanted to utilize PID with this system, but due to time constraints, it was decided to use limit switches in the bottom and top configurations in order to create a hard stop. If given more time, the PID would have been written and given more time to be developed. The other problem would be to fix the sliding of the links along the drive shaft during rotation. This could be fixed with the use of more shaft collars, which the team was unable to find in the lab closer to the deadline.

The gripper was the component that gave the team the most trouble. The original design worked to solve the problem of being able to pull in a nuclear rod. The final project was able to do that most of the time, but it still caused a lot of problems. Due to the team being behind on time, it was only realized around week four that this piece would be needed. This led to a quick design that would be able to work, which led to certain qualities of the piece to be forgotten, such as a place for the rod to be able to slide through when the four bar went into the bottom position. Another issue is that due to time constraints, the piece had to be 3D printed in ABS plastic, which led to weak mounting points on certain parts of the gripper setup. The part would crack if set down a certain way, and even shattered when dropped at one point, which was then fixed with hot glue. After the shattering, the piece never worked the same. The rollers themselves were also an issue. Utilizing just the orthodontic rubber bands did not give enough traction for the rod to be pulled through. This led to the rubber bands then being hot glued to the roller system, which also did not provide enough traction. The final idea was to wrap the rollers in duct tape with the adhesive side facing out. This worked for the most part, except for when the team forgot to switch out the duct tape before the final presentation. The best way for the team to be able to fix this issue would be to spend more time thinking out ideas. This component was an example of having to throw an idea together quickly and then doing whatever possible in order to make it work, rather than spending a lot of time on it in the beginning and thinking of all possible outcomes and issues and then designing the piece.

The code was the most reliable piece of the entire project. This was due to the intense amounts of planning through flowcharting and breaking each piece of the program down as much as possible. This allowed for much smaller tasks that were easier to accomplish than

having to work on a much more massive problem that could easily become complex. Again, this was so successful due to pre-planning and discussing and solving all issues that the team felt could arise during this process.

Overall, the team learned that the biggest component of this project is time management. Many of these problems could have been solved during the project and term if they had just been discussed and solved during the pre-planning stages rather than during spur-of-the-moment decisions.

Conclusion

For the duration of the Unified Robotics 1 Engineering Course, team two worked to the best of their abilities to develop a functional robot that could complete the challenge set before them. They needed to create a robot that would have the ability to pick up and maneuver radioactive rods, place them into different containers on the field, find its way around the field, and also be able to respond to the field in a way that would allow for those viewing the challenge to be able to recognize what was happening. Through a long planning process, a design was developed through the use of calculations that the team felt would best accomplish the tasks at hand. The results were not exactly what they had planned for, but also gave them an idea as to how to move forward from this challenge and to continue to grow and learn within their major field.

Comments

Overall, the team felt this was definitely a challenge, but an excellent learning opportunity. We were not able to achieve everything we had hoped for at the start of the term, but definitely learned some really important lessons not only in the field of robotics, but about being a member of a team and definitely time management. We would not have wanted it any other way, and now know what we need to do in the future in order to achieve greater things.

Appendix A

See attached PDF in the submission

Appendix B

Drive train calculations

Known Values

$n_{in} := 12$ $r := 2.5in$ $w_{wheel} := 33rpm$

$n_{out} := 36$ $w_{in} := 100rpm$ $\tau_{in} := 1.11N$

Calculations

$circ := \pi r$

$$w_{out} := \frac{n_{in} \cdot w_{in}}{n_{out}}$$

$$speed := w_{out} \cdot \frac{1}{60} \cdot circ$$

$$\tau_{out} := \tau_{in} \cdot \frac{w_{wheel}}{w_{out}}$$

Solved Values

$speed := 4.367 \frac{in}{sec}$

$\tau_{out} := 1.12 Nm$

Appendix C

Positional Analysis Equations and Drawings for the Four Bar

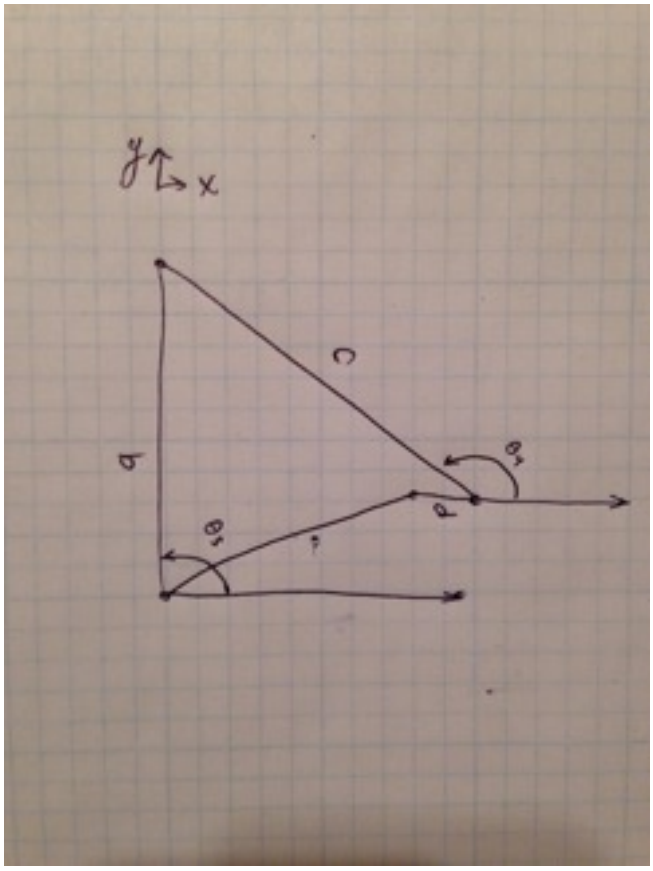

Figure 10

Four Bar Positional Analysis Sketch

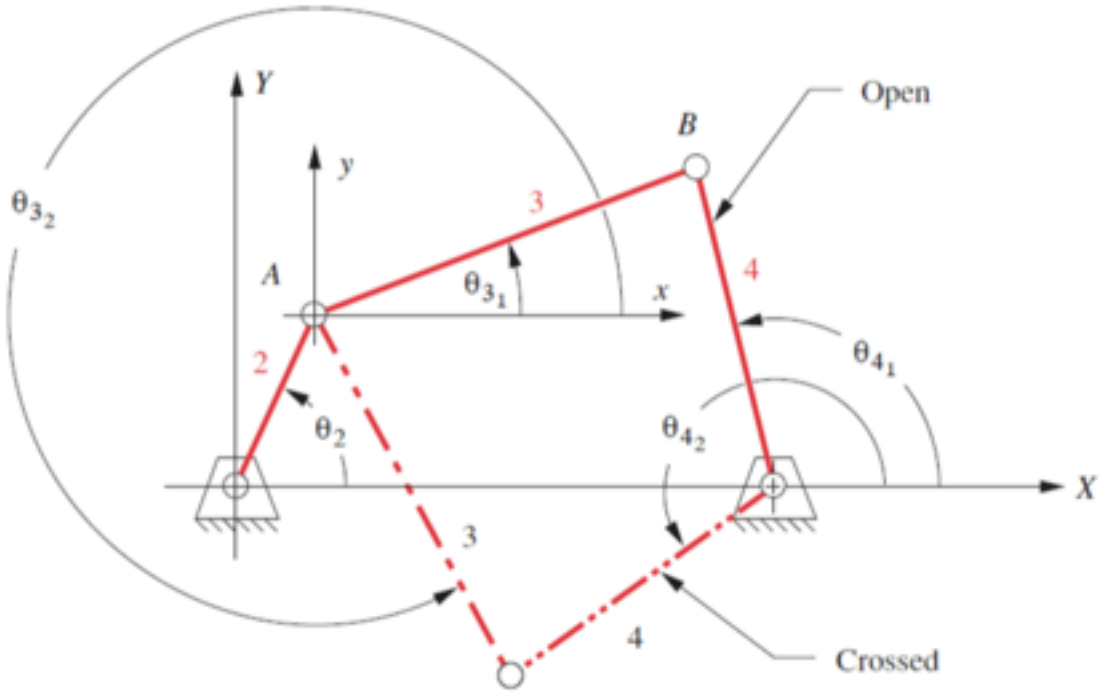

Figure 11

Given Four Bar Positional Analysis Configuration

From the last equation, calculate θ_4

$$\theta_4 = 2\arctan\left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}\right)$$

where...

$$A = \cos\theta_2 - K_1 - K_2\cos\theta_2 + K_3$$

$$B = -2\sin\theta_2$$

$$C = K_1 - (K_2 + 1)\cos\theta_2 + K_3$$

$$K_1 = \frac{d}{a};\ K_2 = \frac{d}{c};\ K_3 = \frac{a^2 - b^2 + c^2 + d^2}{2ac}$$

$$K_4 = \frac{d}{b}; K_5 = \frac{c^2 - d^2 - a^2 - b^2}{2ab}$$

$$D = \cos\theta_2 - K_1 + K_4 \cos\theta_2 + K_5$$

$$E = -2\sin\theta_2$$

$$F = K_1 + (K_4 - 1)\cos\theta_2 + K_5$$

$$\theta_3 = 2\arctan\left(\frac{-E \pm \sqrt{E^2 - 4DF}}{2D}\right)$$

$$\theta_3 = 2\arctan\left(\frac{-E \pm \sqrt{E^2 - 4DF}}{2D}\right)$$

$$\theta_4 = 2\arctan\left(\frac{-B \pm \sqrt{B^2 - 4AC}}{2A}\right)$$

Appendix D

Positional Analysis of Four Bar

Known Values

$a := 2.25in \quad b := 3in \quad cc := 3.94in \quad d := 1.19in$

$Theta2 := 206.29deg$

K-Values

$$Kone := \frac{d}{a} \quad Ktwo := \frac{d}{cc} \quad Kthree := \frac{\left(a^2 - b^2 + cc^2 + d^2\right)}{2 \cdot a \cdot cc}$$

$$Kfour := \frac{d}{b} \quad Kfive := \frac{\left(cc^2 - d^2 - a^2 - b^2\right)}{2 \cdot a \cdot b}$$

$Kone = 0.529 \quad Ktwo = 0.302 \quad Kthree = 0.733$

$Kfour = 0.397 \quad Kfive = 3.333 \times 10^{-3}$

Solved Values

$Aa := \cos(Theta2) - Kone - Ktwo \cdot \cos(Theta2) + Kthree$

$Bb := -2 \cdot \sin(Theta2)$

$Cc := Kone - (Ktwo + 1) \cdot \cos(Theta2) + Kthree$

$Dd := \cos(Theta2) - Kone + Kfour \cdot \cos(Theta2) + Kfive$

$Ee := -2 \sin(Theta2)$

$Ff := Kone + (Kfour - 1) \cdot \cos(Theta2) + Kfive$

The Solution

$$Theta4one := 2\,atan\left[\frac{\left(-Bb + \sqrt{Bb^2 - 4 \cdot Aa \cdot Cc}\right)}{2 \cdot Aa}\right]$$

$$Theta4two := 2 \cdot atan\left[\frac{\left(-Bb - \sqrt{Bb^2 - 4 \cdot Aa \cdot Cc}\right)}{2 \cdot Aa}\right]$$

$$Theta3one := 2 \cdot atan\left[\frac{\left(-Ee + \sqrt{Ee^2 - 4Dd \cdot Ff}\right)}{2 \cdot Dd}\right]$$

$$Theta3two := 2 \cdot atan\left[\frac{\left(-Ee - \sqrt{Ee^2 - 4 \cdot Dd \cdot Ff}\right)}{2 \cdot Dd}\right]$$

Final Results

$Theta4one = -115.015 \cdot deg$

$Theta4two = 149.537 \cdot deg$

$Theta3one = -59.087 \cdot deg$

$Theta3two = 93.609 \cdot deg$

Appendix E

Equations of Equilibrium for the Four Bar

Known Parameters

$a := 2.25in \quad b := 3in \quad cc := 3.94in \quad d := 1.19in \quad ee := .03in$

$W_2 := .00440925lbf \quad W_3 := .220462lbf \quad W_4 := .00881849lbf$

$Theta2 := 206.29 \cdot deg \quad Theta3 := 67.95 \cdot deg \quad Theta5 := 56.23 \cdot deg$

Supply Initial Guesses for the Unknowns

$A_x := 2lbf \quad A_y := 2lbf \quad B_x := 2lbf \quad B_y := 2lbf$

$C_x := 2lbf \quad C_y := 2lbf \quad D_x := 2lbf \quad D_y := 2lbf$

$M_2 := 4in \cdot lbf$

From equations of equilibrium we have nine equations with nine unknowns

Given

From FBE of L2, L3, and L4 as a system:

$$0 = -M_2 - W_2 \cdot \left(\frac{a}{2}\right) \cdot (\cos(Theta3)) + W_3 \cdot \left[cc \cdot \cos(Theta5) - \left(\frac{b}{2}\right)\right] + W_4 \cdot \left(\frac{cc}{2}\right) \cdot (\cos(Theta5)) - A_y \cdot ee - A_x \cdot d$$

$$0 = -D_x - A_x$$

$$0 = -W_3 - W_4 - W_2 + D_y + A_y$$

From FBE of L2:

$$0 = -W_2 \cdot \left[\left(\frac{a}{2}\right) \cdot \cos(Theta3)\right] - B_y \cdot a \cdot \cos(Theta3) + B_x \cdot a \cdot \sin(Theta3)$$

$$0 = B_x - A_x$$

$$0 = -B_y + A_y - W_2$$

From FBE of L3

$$0 = W_3 \cdot \left(\frac{b}{2}\right) + C_y \cdot b$$

$$0 = C_x - B_x$$

$$0 = -C_y + B_y - W_3$$

$$\begin{pmatrix} SA_x \\ SA_y \\ SB_x \\ SB_y \\ SC_x \\ SC_y \\ SD_x \\ SD_y \\ SM_2 \end{pmatrix} := \text{Find}(A_x, A_y, B_x, B_y, C_x, C_y, D_x, D_y, M_2)$$

$SA_x = 0.046\,\text{lbf}$

$SA_y = 0.115\,\text{lbf}$

$SB_x = 0.046\,\text{lbf}$

$SB_y = 0.11\,\text{lbf}$

$SC_x = 0.046\,\text{lbf}$

$SC_y = -0.11\,\text{lbf}$

$SD_x = -0.046\,\text{lbf}$

$SD_y = 0.119\,\text{lbf}$

$SM_2 = 0.102\,\text{in}\cdot\text{lbf}$

Appendix F

Four Bar Component Speed Calculations

Four Bar Component Speeds

Known Values

$n_{out} := 36 \text{ teeth}$ $\quad BC := 3.94 \text{ in}$ $\quad AD := 2.25 \text{ in}$

$n_{in} := 12 \text{ teeth}$ $\quad I_{cb} := 3.37 \text{ in}$

$w_{in} := 100 \text{ RPM}$ $\quad I_{ca} := 3.016 \text{ in}$

Solve

$$w_{out} := \frac{(n_{in} \cdot w_{in})}{n_{out}} = 33.333 \text{ RPM}$$

$$w_2 := (w_{out})\left(\frac{1}{60}\right)[2 \cdot (3.14)] = 3.489 \frac{\text{rad}}{\text{sec}}$$

$$v_b := (BC) \cdot (w_2) = 13.746 \frac{\text{in}}{\text{sec}}$$

$$w_3 := \frac{v_b}{I_{cb}} = 4.079 \frac{\text{rad}}{\text{sec}}$$

$$v_a := I_{ca} \cdot w_3 = 12.302 \frac{\text{in}}{\text{sec}}$$

$$w_4 := \frac{v_a}{AD} = 5.468 \frac{\text{rad}}{\text{sec}}$$

Appendix G

Exploded View of Four Bar

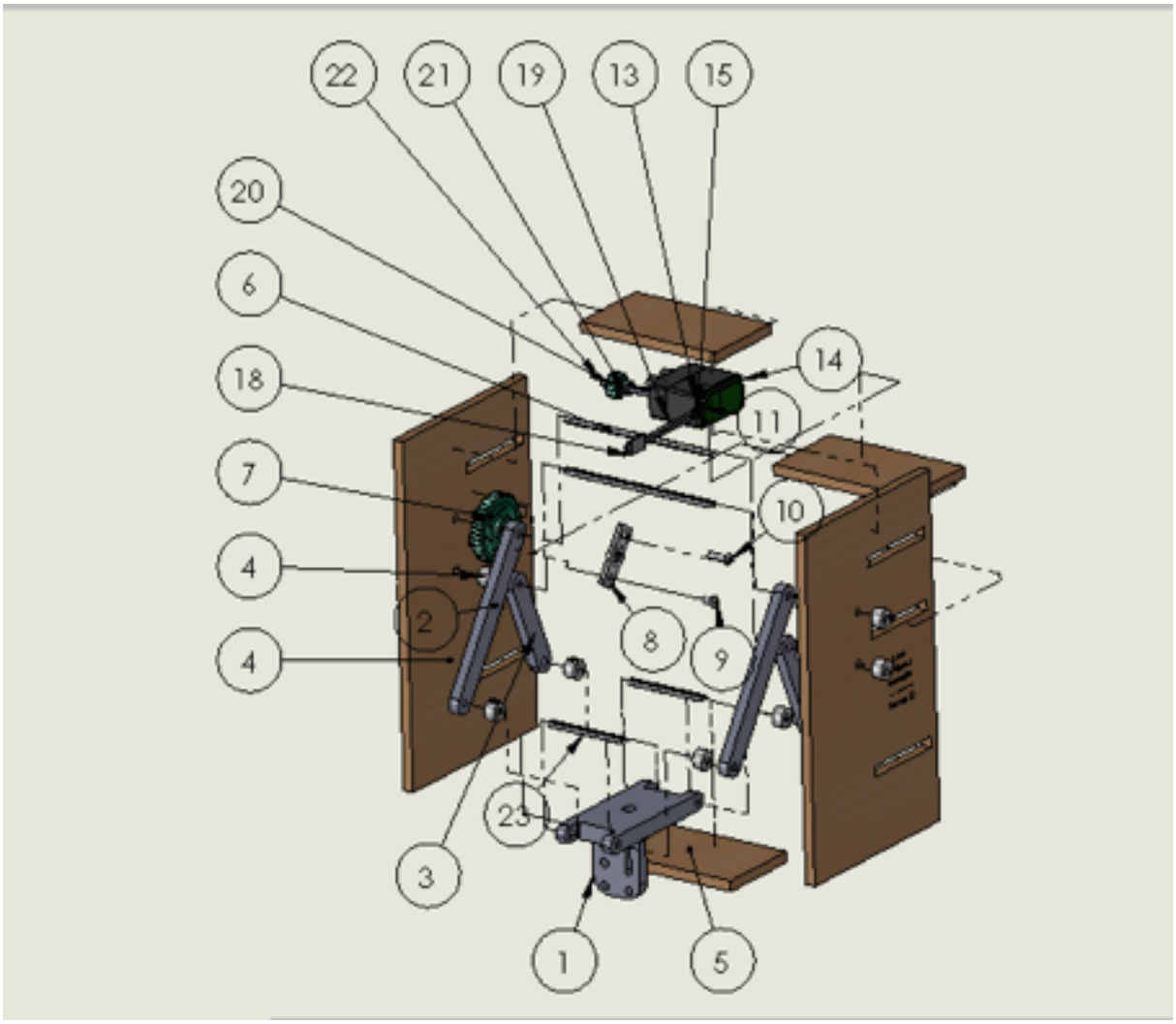

Figure 12

Exploded View of the Four Bar

Appendix H

Full State Machine

```
// ************ Library Inclusion Statements ************
// Include servo library
#include <Servo.h>
// Include light sensor library
#include <QTRSensors.h>
// Include LCD library
#include <LiquidCrystal.h>
// Include the standard input/output library
#include <stdio.h>
// Include the Arduino library
#include "Arduino.h"
```

```
// Include the Messages library
#include "Messages.h"
// ************ End Library Inclusion Statements ************
```

```
// ************ State Machine ************
/* Drive State Machine
 * State 0: Stop Robot Drive Functions
 * State 1: Drive to Reactor (INITIAL)
 * State 2: Lower the Fourbar to the Nuclear Reactor
 * State 3: Retrieve Spent Fuel Rod from Reactor
 * State 4: Raise the Fourbar with Spent Rod
 * State 5: Drive to Spent Fuel Rod Storage
 * State 6: Expel Spent Fuel Rod into Storage Tube
 * State 7: Drive from the Spent Rod Storage to the Center Line
 * State 8: Drive to New Fuel Rod Storage
 * State 9: Retrieve New Fuel Rod from Storage Tube
 * State 10: Drive to Reactor (WITH NEW FUEL ROD)
 * State 11: Lower Fourbar over Reactor
 * State 12: Expel New Fuel Rod into Reactor
 * State 13: Raise Fourbar to Show Placement of Fuel Rod
 *
 * State 14: Drive to Second Reactor
 * State 15: Drive to Spent Fuel Rod Storage
 * State 16: Drive to Reactor (WITH NEW FUEL ROD)
 */
const char S_STOP_ROBOT = 0;
const char S_DRIVE_TO_REACTOR = 1;
const char S_LOWER_FOURBAR = 2;
const char S_SLURP_FUEL_ROD = 3;
const char S_RAISE_FOURBAR = 4;
const char S_DRIVE_TO_SPENT_STORAGE = 5;
const char S_UNSLURP_FUEL_ROD = 6;
const char S_RETURN_TO_CENTER = 7;
const char S_DRIVE_TO_NEW_STORAGE = 8;
const char S_SLURP_NEW_FUEL_ROD = 9;
const char S_RETURN_TO_REACTOR_NEW = 10;
const char S_LOWER_FOURBAR_AGAIN = 11;
const char S_UNSLURP_NEW_FUEL_ROD = 12;
const char S_RAISE_FOURBAR_FINAL = 13;

const char S_DRIVE_TO_SECOND_REACTOR = 14;
const char S_DRIVE_TO_SPENT_STORAGE_FROM_SECOND_REACTOR = 15;
const char S_RETURN_TO_SECOND_REACTOR_NEW = 16;

// The current state of the state machine
// Starts by driving to reactor
char state = S_DRIVE_TO_REACTOR;

// State machine control variables
// Which tube is open to recieve spent fuel rod
int openTube = 0;

// Which tube is full (has a new fuel rod in it)
int fullTube = 0;

// Which tube the robot went to to deliver the spent fuel rod
int place = 0;
```

```
// Used to store the number of lines that need to be crossed in order to drive
autonomously
int goal = 0;
boolean secondReactor = false;
// ************ End State Machine ************



// ************ Light Sensor Initialization ************
// Declare number of sensors used to line follow
#define NUM_SENSORS 6
// Declare number of sensors used to count lines
#define NUM_COUNTERS 1
// Declare max time until time-out (in microseconds)
#define TIMEOUT 2500
// Declare IR LED pin
#define EMITTER_PIN 2

// sensors 1 through 6 are connected to digital pins 3 through 8, respectively
QTRSensorsRC lineFollower((unsigned char[]) {6, 7, 8, 9, 10, 11},
  NUM_SENSORS, TIMEOUT, EMITTER_PIN);
unsigned int sensorValues[NUM_SENSORS];

// sensor 7 is connected to digital pin 12
QTRSensorsRC lineCounter((unsigned char[]) {12},
  NUM_COUNTERS, TIMEOUT, EMITTER_PIN);
unsigned int counterValues[NUM_COUNTERS];

// Create line counter variable
int lines = 0;
// Line barrier flag
boolean canCountLine = true;
// ************ End Light Sensor Initialization ************



// ************ Motor Initialization ************
// Motor pin connections
const char collectorMotorPin = 26;
const char fourbarMotorPin = 27;
const char rightMotorPin = 28;
const char leftMotorPin = 29;

// Create motors
Servo collectorMotor;
Servo fourbarMotor;
Servo rightMotor;
Servo leftMotor;
// ************ End Motor Initialization ************



// ************ LCD Initialization ************
// Declare LCD pins
LiquidCrystal lcd(40,41,42,43,44,45);
```

```
// ************ End LCD Initialization ************



// ************ Limit Switch Initialization ************
// Limit switch mounted to the front of the chassis
// Used for alignment with reactor/storage tubes
const char limitSwitchPin = 24;
// Current value of the switch
boolean switchVal = HIGH;

// Limit switch at the bottom of the fourbar
// Used to determine if the fourbar is at the bottom limit of its rotation
const char bottomLimitSwitchPin = 23;
// Current value of the switch
boolean bottomLimitSwitchVal = HIGH;

// Limit switch at the top of the fourbar
// Used to deterimine if the fourbar is at the top limit of its rotation
const char topLimitSwitchPin = 22;
// Current value of the switch
boolean topLimitSwitchVal = HIGH;
// ************ End Limit Switch Initialization ************



// ************ Speaker Pin Initialization ************
// Digital pin for the piezo electric speaker
const int speakerPin = 4;

// Pitch control variables
const int pitchLow = 200;
const int pitchHigh = 1000;

// Increment between pitches moving high -> low or low -> high
int pitchStep = 10;

// The current pitch played by the speaker
int currentPitch;

// Siren control variables
// The current time in ms
int now = 0;
// State of the alarm loop, acts as a delay
boolean timeTaken = false;
// Controls whether the alarm is on
boolean alarmOn = false;
// ************ End Speaker Pin Initialization ************



//  ************ Heartbeat Initialization ************
Messages msg;
unsigned long timeForHeartbeat;
//  ************ End Heartbeat Initialization ************
```

```
// ************ Setup ************
void setup() {
  //Serial.begin(9600);
  Serial.begin(115200);
  Serial.println("Starting");
  msg.setup();
  timeForHeartbeat = millis() + 1000;

  // ************ Attach LCD ************
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0,0);
  // ************ End Attach ************



  // ************ Motor Setup ************
  lcd.print("Motor Setup");
  delay(1000);
  // Declare servo pins
  pinMode(collectorMotorPin, OUTPUT);
  pinMode(fourbarMotorPin, OUTPUT);
  pinMode(rightMotorPin, OUTPUT);
  pinMode(leftMotorPin, OUTPUT);

  // Attach motors to servo pins
  collectorMotor.attach(collectorMotorPin, 1000, 2000);  // Collector motor, powers
collection mechanism at end of fourbar
  fourbarMotor.attach(fourbarMotorPin, 1000, 2000);      // Fourbar motor, controls
fourbar rotation
  rightMotor.attach(rightMotorPin, 1000, 2000);          // Right motor, powers the
right side drivetrain
  leftMotor.attach(leftMotorPin, 1000, 2000);            // Left motor, powers the
left side drivetrain
  lcd.clear();
  // ************ End Motor Setup ************



  // ************ Limit Switch Setup ************
  pinMode(limitSwitchPin, INPUT_PULLUP);                 // Drive function limit
switch, mounted to front of chassis
  pinMode(bottomLimitSwitchPin, INPUT_PULLUP);           // Bottom of fourbar limit
switch, mounted to the bottom of the tower
  pinMode(topLimitSwitchPin, INPUT_PULLUP);              // Top of fourbar limit
switch, mounted to the top of the tower
  // ************ End Limit Switch Setup ************



  // ************ Calibrate the Light Sensor ************
  lcd.print("Calibrating");                              // Indicate over the LCD that
the light sensor is calibrating
  for (int i = 0; i < 400; i++)                          // make the calibration take
about 10 seconds
  {
```

```
    lineFollower.calibrate();                              // reads all sensors 10 times
at 2500 us per read (i.e. ~25 ms per call)
  }
  lcd.clear();                                             // Clear LCD to indicate
calibration has ended
  // ************ End Light Sensor Calibration ************



  // ************ Print Sensor values ************
  // Print the calibration minimum values measured when emitters were on
  // Line sensor values
  for (int i = 0; i < NUM_SENSORS; i++) {
    lcd.setCursor(0,0);
    lcd.print("Sensor ");
    lcd.print(i);
    lcd.print(" min: ");
    lcd.setCursor(0,1);
    lcd.print(lineFollower.calibratedMinimumOn[i]);
    delay(1000);
    lcd.clear();
  }

//  lcd.setCursor(0,1);
//  // Line counter values
//  for (int i = 0; i < NUM_COUNTERS; i++) {
//    lcd.print(lineCounter.calibratedMinimumOn[i]);
//    delay(1000);
//    lcd.clear();
//  }

  // Print the calibration maximum values measured when emitters were on
  // Line sensor values
  for (int i = 0; i < NUM_SENSORS; i++) {
    lcd.setCursor(0,0);
    lcd.print("Sensor ");
    lcd.print(i);
    lcd.print(" max: ");
    lcd.setCursor(0,1);
    lcd.print(lineFollower.calibratedMaximumOn[i]);
    delay(1000);
    lcd.clear();
  }

//  lcd.setCursor(0,1);
//  // Line counter values
//  for (int i = 0; i < NUM_COUNTERS; i++) {
//    lcd.print(lineCounter.calibratedMaximumOn[i]);
//    delay(1000);
//    lcd.clear();
//  }
  delay(1000);
  // ************ End Print Sensor Values ************/

  // Clear LCD, display that Setup is ending and the Loop is about to begin
  lcd.clear();
```

```
  lcd.setCursor(0,0);
  lcd.print("Enter Loop");
  delay(1000);
  lcd.clear();
}
// ************ End Setup ************



// ************ Loop ************
void loop() {
  // In each iteration of the loop, update the switch value variables
  switchVal = digitalRead(limitSwitchPin);
  topLimitSwitchVal = digitalRead(topLimitSwitchPin);
  bottomLimitSwitchVal = digitalRead(bottomLimitSwitchPin);

//  // Uncomment for debugging purposes
//  // Print the values of the switches
//  Serial.println(switchVal);
//  Serial.println(topLimitSwitchVal);
//  Serial.println(bottomLimitSwitchVal);



  // ************ Heartbeat ************
  // Broadcast a heartbeat message to the field computer
  if (msg.read()) {
    msg.printMessage();
  }
  if (millis() > timeForHeartbeat) {
    timeForHeartbeat = millis() + 1000;
    msg.sendHeartbeat();
  }
  // ************ End Heartbeat ************



  // ************ Alarm Control Switch ************
  // If the alarmOn variable is set to true, run the alarm loop
  // If the alarmOn variable is set to false, turn the alarm off
  if (alarmOn = true) {
    soundAlarm();
  } else {
    stopAlarm();
  }
  // ************ End Alarm Control Switch ************



  // ************ State Machine Loop ************
  switch (state) {
    // State 0: Stop Robot Drive Functions
    case S_STOP_ROBOT:
      lcd.setCursor(0,0);
      lcd.print("State: 0");
      stopRobot();
```

```
  break;

// State 1: Drive to Reactor (INITIAL)
case S_DRIVE_TO_REACTOR:
  lcd.setCursor(0,0);
  lcd.print("State: 1");
  driveToReactor();
  break;

// State 2: Lower the Fourbar to the Nuclear Reactor
case S_LOWER_FOURBAR:
  lcd.setCursor(0,0);
  lcd.print("State: 2");
  lowerFourbar();
  break;

// State 3: Retrieve Spent Fuel Rod from Reactor
case S_SLURP_FUEL_ROD:
  lcd.setCursor(0,0);
  lcd.print("State: 3");
  slurpFuelRod();
  break;

// State 4: Raise the Fourbar with Spent Rod
case S_RAISE_FOURBAR:
  lcd.setCursor(0,0);
  lcd.print("State: 4");
  raiseFourbar();
  break;

// State 5: Drive to Spent Fuel Rod Storage
case S_DRIVE_TO_SPENT_STORAGE:
  lcd.setCursor(0,0);
  lcd.print("State: 5");
  driveToSpentStorage();
  break;

// State 6: Expel Spent Fuel Rod into Storage Tube
case S_UNSLURP_FUEL_ROD:
  lcd.setCursor(0,0);
  lcd.print("State: 6");
  unslurpFuelRod();
  break;

// State 7: Drive from the Spent Rod Storage to the Center Line
case S_RETURN_TO_CENTER:
  lcd.setCursor(0,0);
  lcd.print("State: 7");
  returnToCenter();
  break;

// State 8: Drive to New Fuel Rod Storage
case S_DRIVE_TO_NEW_STORAGE:
  lcd.setCursor(0,0);
  lcd.print("State: 8");
  driveToNewStorage();
```

```
  break;

// State 9: Retrieve New Fuel Rod from Storage Tube
case S_SLURP_NEW_FUEL_ROD:
  lcd.setCursor(0,0);
  lcd.print("State: 9");
  slurpNewFuelRod();
  break;

// State 10: Drive to Reactor (WITH NEW FUEL ROD)
case S_RETURN_TO_REACTOR_NEW:
  lcd.setCursor(0,0);
  lcd.print("State: 10");
  returnToReactor();
  break;

// State 11: Lower Fourbar over Reactor
case S_LOWER_FOURBAR_AGAIN:
  lcd.setCursor(0,0);
  lcd.print("State: 11");
  lowerFourbarAgain();
  break;

// State 12: Expel New Fuel Rod into Reactor
case S_UNSLURP_NEW_FUEL_ROD:
  lcd.setCursor(0,0);
  lcd.print("State: 12");
  unslurpNewFuelRod();
  break;

// State 13: Raise Fourbar to Show Placement of Fuel Rod
case S_RAISE_FOURBAR_FINAL:
  lcd.setCursor(0,0);
  lcd.print("State: 13");
  raiseFourbarFinal();
  break;

// State 14: Drive to Second Reactor
case S_DRIVE_TO_SECOND_REACTOR:
  lcd.setCursor(0,0);
  lcd.print("State: 14");
  driveToSecondReactor();
  break;

// State 15: Drive to Spent Fuel Rod Storage
case S_DRIVE_TO_SPENT_STORAGE_FROM_SECOND_REACTOR:
  lcd.setCursor(0,0);
  lcd.print("State: 15");
  driveToSpentStorageReactorTwo();
  break;

// State 16: Drive to Reactor (WITH NEW FUEL ROD)
case S_RETURN_TO_SECOND_REACTOR_NEW:
  lcd.setCursor(0,0);
  lcd.print("State: 16");
  returnToSecondReactor();
```

```
      break;
  }
  // ************ State Machine Loop ************
}
// ************ End Loop ************



// ************ Drive to Reactor ************
// driveToReactor: void -> void
// This function tells the robot to line follow forwards until the front
// bump sensor is triggered
void driveToReactor() {
    lcd.setCursor(0,1);
    lcd.print("SubState: 1");
    lineFollow();
    if (switchVal) {
      stopRobot();
      state = S_LOWER_FOURBAR;
    }
}
// ************ End Drive to Reactor ************



// ************ Lower Fourbar ************
// lowerFourbar: void -> void
// This function tells the robot to lower the fourbar from the starting position
// to the minimum position, when the bottom switch is triggered
void lowerFourbar() {
  lcd.setCursor(0,1);
  lcd.print("Substate: 1");
  moveFourbarBottom();
  slurp();
  if (!bottomLimitSwitchVal) {
    stopFourbar();
    state = S_SLURP_FUEL_ROD;
  }
}
// ************ End Lower Fourbar ************



// ************ Slurp Fuel Rod ************
// slurpFuelRod: void -> void
// This function tells the collector motor to turn on and retreive the spent fuel rod
void slurpFuelRod() {
  lcd.setCursor(0,1);
  lcd.print("Radiation Alert");
  slurp();
  delay(7000);
  stopCollector();
  alarmOn = true;
  state = S_RAISE_FOURBAR;
}
// ************ End Slurp Fuel Rod ************
```

```
// ************ Raise Fourbar ************
// raiseFourbar: void -> void
// This function tells the fourbar to raise from the current position to the maximum
position
// when the top limit switch is triggered
void raiseFourbar() {
  lcd.setCursor(0,1);
  //lcd.print("Substate: 1");
  lcd.print("Radiation Alert");
  moveFourbarTop();
  if (!topLimitSwitchVal) {
    stopFourbar();
    if (!secondReactor) {
      state = S_DRIVE_TO_SPENT_STORAGE;
    } else {
      state  = S_DRIVE_TO_SPENT_STORAGE_FROM_SECOND_REACTOR;
    }
  }
}
// ************ End Raise Fourbar ************



// ************ Drive to Spent Storage ************
// driveToSpentStorage: void -> void
// This function directs the robot from Reactor A to an empty storage tube
void driveToSpentStorage() {
  /* Sub-states:
   * Sub-state 1: Backup from the reactor
   * Sub-state 2: Determine the goal storage tube from bluetooth data
   * Sub-state 3: Turn the robot 180 degrees
   * Sub-state 4: Line follow forward until the robot has encountered the goal line
   * Sub-state 5: Turn to face the storage tube
   * Sub-state 6: Line follow until the front limit switch is triggered
   */
  const char S2_BACKUP = 1;
  const char S2_DETERMINE_GOAL = 2;
  const char S2_TURN_AROUND = 3;
  const char S2_DRIVE_TO_LINE = 4;
  const char S2_TURN_LEFT = 5;
  const char S2_DRIVE_TO_STORAGE = 6;

  // Initial state
  static char state2 = S2_BACKUP;

  switch (state2) {
    // Sub-state 1: Backup from the reactor
    case S2_BACKUP:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 1");
      lcd.print("Radiation Alert");
      driveBackward();
      lines = 0;
```

```
  lineCount();
  if (lines > 0) {
    delay(2000);
    state2 = S2_DETERMINE_GOAL;
  }
  break;

// Sub-state 2: Determine the goal storage tube from bluetooth data
case S2_DETERMINE_GOAL:
  lcd.setCursor(0,1);
  //lcd.print("SubState: 2");
  lcd.print("Radiation Alert");
  openTube = 2;
  place = openTube;
  state2 = S2_TURN_AROUND;
  break;

// Sub-state 3: Turn the robot 180 degrees
case S2_TURN_AROUND:
  lcd.setCursor(0,1);
  //lcd.print("SubState: 3");
  lcd.print("Radiation Alert");
  turnLeft();
  lines = 0;
  lineCount();
  if (lines > 0) {
    stopRobot();
    lines = 0;
    turnLeft();
    delay(500);
    state2 = S2_DRIVE_TO_LINE;
  }
  break;

// Sub-state 4: Line follow forward until the robot has encountered the goal line
case S2_DRIVE_TO_LINE:
  lcd.setCursor(0,1);
  //lcd.print("SubState: 4");
  lcd.print("Radiation Alert");
  lineFollow();
  lineCount();
  if (lines >= openTube) {
    stopRobot();
    driveForward();
    delay(900);
    stopRobot();
    lines = 0;
    state2 = S2_TURN_LEFT;
  }
  break;

// Sub-state 5: Turn to face the storage tube
case S2_TURN_LEFT:
  lcd.setCursor(0,1);
  //lcd.print("SubState: 5");
  lcd.print("Radiation Alert");
```

```
      turnLeft();
      lines = 0;
      lineCount();
      if (lines > 0) {
        delay(1000);
        driveForward();
        delay(1000);
        stopRobot();
        lines = 0;
        state2 = S2_DRIVE_TO_STORAGE;
      }
      break;

    // Sub-state 6: Line follow until the front limit switch is triggered
    case S2_DRIVE_TO_STORAGE:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 6");
      lcd.print("Low Radiation");
      lineFollow();
      if (switchVal) {
        stopRobot();
        lcd.clear();
        state = S_UNSLURP_FUEL_ROD;
      }
      break;
  }
}
// ************ End Drive to Spent Storage ************


// ************ Unslurp Fuel Rod ************
// unslurpFuelRod: void -> void
// This function turns on the collector motor to expel the spent fuel rod
// from the mechanism into the storage tube
void unslurpFuelRod() {
  lcd.setCursor(0,1);
  lcd.print("SubState: 1");
  unslurp();
  delay(9000);
  stopCollector();
  alarmOn = false;
  state = S_RETURN_TO_CENTER;
}
// ************ Unslurp Fuel Rod ************



// ************ Return to Center Line ************
// returnToCenter: void -> void
// This function directs the robot back to the center line of the track
void returnToCenter() {
  /* Sub-states:
   * Sub-state 1: Tell the robot to back up from the storage tubes
   * Sub-state 2: Have the robot turn 180 degrees
   * Sub-state 3: Line follow until the robot encounters a line
   */
```

```
  const char S3_BACKUP = 1;
  const char S3_TURN_AROUND = 2;
  const char S3_DRIVE_TO_CENTERLINE = 3;

  // Initial state
  static char state3 = S3_BACKUP;

  switch (state3) {
    // Sub-state 1: Tell the robot to back up from the storage tubes
    case S3_BACKUP:
      lcd.setCursor(0,1);
      lcd.print("SubState: 1");
      driveBackward();
      delay(1000);
      stopRobot();
      state3 = S3_TURN_AROUND;
      break;

    // Sub-state 2: Have the robot turn 180 degrees
    case S3_TURN_AROUND:
      lcd.setCursor(0,1);
      lcd.print("SubState: 2");
      turnLeft();
      lineCount();
      if (lines > 0) {
        delay(750);
        stopRobot();
        lines = 0;
        state3 = S3_DRIVE_TO_CENTERLINE;
      }
      break;

    // Sub-state 3: Line follow until the robot encounters a line
    case S3_DRIVE_TO_CENTERLINE:
      lcd.setCursor(0,1);
      lcd.print("SubState: 3");
      lineFollow();
      lines = 0;
      lineCount();
      if (lines > 0) {
        stopRobot();
        state = S_DRIVE_TO_NEW_STORAGE;
      }
      break;
  }
}
// ************ End Return to Center Line ************



// ************ Drive to New Fuel Rod Storage ************
// driveToNewStorage: void -> void
// Direct the robot to move from the center line to a storage tube indicated by
bluetooth
void driveToNewStorage() {
  // Used to tell the state machine which way to turn
```

```
  boolean useS4_TURN_TO_FACE_STORAGE_RIGHT = false;

  /* Sub-state:
   * Sub-state 1: Determine the storage tube that has a new fuel rod
   * Sub-state 2: Direct the robot to turn left 90 degrees
   * Sub-state 3: Direct the robot to turn right 90 degrees
   * Sub-state 4: Line follow forwards until the robot encounters the desired line
   * Sub-state 5: Direct the robot to turn right 90 degrees towards the storage tube
   * Sub-state 6: Direct the robot to turn left 90 degrees towards the storage tube
   * Sub-state 7: Line follow to the storage tube until the front limit switch is
triggered
   */
  const char S4_DETERMINE_GOAL = 1;
  const char S4_TURN_LEFT = 2;
  const char S4_TURN_RIGHT = 3;
  const char S4_DRIVE_TO_LINE = 4;
  const char S4_TURN_TO_FACE_STORAGE_RIGHT = 5;
  const char S4_TURN_TO_FACE_STORAGE_LEFT = 6;
  const char S4_DRIVE_TO_STORAGE = 7;

  // Initial state
  static char state4 = S4_DETERMINE_GOAL;

  switch (state4) {
    // Sub-state 1: Determine the storage tube that has a new fuel rod
    case S4_DETERMINE_GOAL:
      lcd.setCursor(0,1);
      lcd.print("SubState: 1");
      fullTube = 2;
      if (fullTube == place) {
        state4 = S4_DRIVE_TO_STORAGE;
      } else if (fullTube < place) {
        state4 = S4_TURN_RIGHT;
      } else {
        state4 = S4_TURN_LEFT;
      }
      break;

    // Sub-state 2: Direct the robot to turn left 90 degrees
    case S4_TURN_LEFT:
      lcd.setCursor(0,1);
      lcd.print("SubState: 2");
      turnLeft();
      lines = 0;
      lineCount();
      if (lines > 0) {
        stopRobot();
        lines = 0;
        goal = fullTube - place;
        useS4_TURN_TO_FACE_STORAGE_RIGHT = true;
        state4 = S4_DRIVE_TO_LINE;
      }
      break;

    // Sub-state 3: Direct the robot to turn right 90 degrees
    case S4_TURN_RIGHT:
```

```
  lcd.setCursor(0,1);
  lcd.print("SubState: 3");
  turnRight();
  lineCount();
  if (lines > 2) {
    stopRobot();
    lines = 0;
    goal = place - fullTube;
    useS4_TURN_TO_FACE_STORAGE_RIGHT = false;
    state4 = S4_DRIVE_TO_LINE;
  }
  break;

// Sub-state 4: Line follow forwards until the robot encounters the desired line
case S4_DRIVE_TO_LINE:
  lcd.setCursor(0,1);
  lcd.print("SubState: 4");
  lineFollow();
  lineCount();
  if (lines > goal) {
    stopRobot();
    lines = 0;
    if (useS4_TURN_TO_FACE_STORAGE_RIGHT) {
      state4 = S4_TURN_TO_FACE_STORAGE_RIGHT;
    } else {
      state4 = S4_TURN_TO_FACE_STORAGE_LEFT;
    }
  }
  break;

// Sub-state 5: Direct the robot to turn right 90 degrees towards the storage tube
case S4_TURN_TO_FACE_STORAGE_RIGHT:
  lcd.setCursor(0,1);
  lcd.print("SubState: 5");
  turnRight();
  lineCount();
  if (lines > 2) {
    stopRobot();
    lines = 0;
    state4 = S4_DRIVE_TO_STORAGE;
  }
  break;

// Sub-state 6: Direct the robot to turn left 90 degrees towards the storage tube
case S4_TURN_TO_FACE_STORAGE_LEFT:
  lcd.setCursor(0,1);
  lcd.print("SubState: 6");
  turnLeft();
  lines = 0;
  lineCount();
  if (lines > 0) {
    stopRobot();
    lines = 0;
    state4 = S4_DRIVE_TO_STORAGE;
  }
  break;
```

```
    // Sub-state 7: Line follow to the storage tube until the front limit switch is
triggered
    case S4_DRIVE_TO_STORAGE:
      lcd.setCursor(0,1);
      lcd.print("SubState: 7");
      lineFollow();
      if (switchVal) {
        stopRobot();
        state = S_SLURP_NEW_FUEL_ROD;
      }
      break;
  }
}
// ************ End Drive to New Fuel Rod Storage ************



// ************ Slurp New Fuel Rod ************
// slurpNewFuelRod: void -> void
// This function directs the collector to retrieve a new fuel rod
void slurpNewFuelRod() {
  lcd.setCursor(0,1);
  //lcd.print("Substate: 1");
  lcd.print("High Radiation");
  slurp();
  delay(10000);
  stopCollector();
  alarmOn = true;
  if (!secondReactor) {
    state = S_RETURN_TO_REACTOR_NEW;
  } else {
    state = S_RETURN_TO_SECOND_REACTOR_NEW;
  }
}
// ************ End Slurp New Fuel Rod ************



// ************ Return to Reactor with New Fuel Rod ************
// returnToReactor: void -> void
// This function directs the robot back to Reactor A from the new fuel rod
// storage tubes
void returnToReactor() {
  /* Sub-states:
   * Sub-state 1: Back the robot away from the storage tube
   * Sub-state 2: Turn the robot right to face Reactor A
   * Sub-state 3: Line follow back to Reactor A until front limit switch is triggered
   */
  const char S5_BACKUP = 1;
  const char S5_TURN_RIGHT = 2;
  const char S5_DRIVE_TO_REACTOR = 3;

  // Initial state
  static char state5 = S5_BACKUP;
```

```
  switch (state5) {
    // Sub-state 1: Back the robot away from the storage tube
    case S5_BACKUP:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 1");
      lcd.print("Radiation Alert");
      driveBackward();
      delay(4000);
      stopRobot();
      state5 = S5_TURN_RIGHT;
      break;

    // Sub-state 2: Turn the robot right to face Reactor A
    case S5_TURN_RIGHT:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 2");
      lcd.print("Radiation Alert");
      turnRight();
      lineCount();
      if (lines > 2) {
        stopRobot();
        lines = 0;
        state5 = S5_DRIVE_TO_REACTOR;
      }
      break;

    // Sub-state 3: Line follow back to Reactor A until front limit switch is
triggered
    case S5_DRIVE_TO_REACTOR:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 3");
      lcd.print("Radiation Alert");
      lineFollow();
      if (switchVal) {
        stopRobot();
        state = S_LOWER_FOURBAR_AGAIN;
      }
      break;
  }
}
// ************ End Return to Reactor with New Fuel Rod ************



// ************ Lower Fourbar for New Fuel Rod ************
// lowerFourbarAgain: void -> void
// This function moves the fourbar from its current positon to its minimum position
void lowerFourbarAgain() {
  lcd.setCursor(0,1);
  //lcd.print("Substate: 1");
  lcd.print("Radiation Alert");
  moveFourbarBottom();
  if (!bottomLimitSwitchVal) {
    stopFourbar();
    state = S_UNSLURP_NEW_FUEL_ROD;
  }
```

```
}
// ************ End Lower Fourbar for New Fuel Rod ************



// ************ Deposit New Fuel Rod ************
// unslurpNewFuelRod: void -> void
// This function turns on the collector motor and expels the fuel rod into the Reactor
void unslurpNewFuelRod() {
  lcd.setCursor(0,1);
  lcd.clear();
  lcd.print("SubState: 1");
  unslurp();
  delay(10000);
  stopCollector();
  alarmOn = false;
  state = S_RAISE_FOURBAR_FINAL;
}
// ************ End Deposit New Fuel Rod ************



// ************ Raise Fourbar ************
// raiseFourbarFinal: void -> void
// Raise the fourbar from the current position to the maximum position
// when the top limit switch is triggered
void raiseFourbarFinal() {
  lcd.setCursor(0,1);
  lcd.print("Substate: 1");
  moveFourbarTop();
  if (!topLimitSwitchVal) {
    stopFourbar();
    secondReactor = true;
    if (!secondReactor) {
      state = S_DRIVE_TO_SECOND_REACTOR;
    } else {
      state = S_STOP_ROBOT;
    }
  }
}
// ************ End Raise Fourbar ************



// ************ Drive to Second Reactor from First Reactor ************
// driveToSecondReactor: void -> void
// Direct robot from Reactor A to Reactor B
void driveToSecondReactor() {
  /* Sub-states:
   * Sub-state 1: Back up from Reactor A
   * Sub-state 2: Turn the robot 180 degrees
   * Sub-state 3: Line follow to Reactor B until the front limit switch is triggered
   */
  const char S14_BACKUP = 1;
  const char S14_TURN_AROUND = 2;
  const char S14_DRIVE_TO_REACTOR = 3;
```

```
  // Initial state
  static char state14 = S14_BACKUP;

  switch (state14) {
    // Sub-state 1: Back up from Reactor A
    case S14_BACKUP:
      lcd.setCursor(0,1);
      lcd.print("SubState: 1");
      driveBackward();
      delay(3000);
      state14 = S14_TURN_AROUND;
      break;

    // Sub-state 2: Turn the robot 180 degrees
    case S14_TURN_AROUND:
      lcd.setCursor(0,1);
      lcd.print("SubState: 2");
      turnLeft();
      lines = 0;
      lineCount();
      if (lines > 0) {
        stopRobot();
        lines = 0;
        turnLeft();
        delay(500);
        state14 = S14_DRIVE_TO_REACTOR;
      }
      break;

    // Sub-state 3: Line follow to Reactor B until the front limit switch is triggered
    case S14_DRIVE_TO_REACTOR:
      lcd.setCursor(0,1);
      lcd.print("SubState: 3");
      lineFollow();
      if (switchVal) {
        stopRobot();
        lcd.clear();
        state = S_LOWER_FOURBAR;
      }
      break;
  }
}
// ************ End Drive to Second Reactor from First Reactor ************



// ************ Drive to Spent Fuel Rod Storage ************
// driveToSpentStorageReactorTwo: void -> void
// This function directs the robot from Reactor B to an empty storage tube
void driveToSpentStorageReactorTwo() {
  /* Sub-states:
   * Sub-state 1: Backup from the reactor
   * Sub-state 2: Determine the goal storage tube from bluetooth data
   * Sub-state 3: Turn the robot 180 degrees
   * Sub-state 4: Line follow forward until the robot has encountered the goal line
```

```
 * Sub-state 5: Turn to face the storage tube
 * Sub-state 6: Line follow until the front limit switch is triggered
 */

const char S15_BACKUP = 1;
const char S15_DETERMINE_GOAL = 2;
const char S15_TURN_AROUND = 3;
const char S15_DRIVE_TO_LINE = 4;
const char S15_TURN_RIGHT = 5;
const char S15_DRIVE_TO_STORAGE = 6;

// Initial state
static char state15 = S15_BACKUP;

switch (state15) {
  case S15_BACKUP:
    lcd.setCursor(0,1);
    //lcd.print("SubState: 1");
    lcd.print("Radiation Alert");
    driveBackward();
    delay(3000);
    state15 = S15_DETERMINE_GOAL;
    break;

  case S15_DETERMINE_GOAL:
    lcd.setCursor(0,1);
    //lcd.print("SubState: 2");
    lcd.print("Radiation Alert");
    openTube = 2;
    place = openTube;
    state15 = S15_TURN_AROUND;
    break;

  case S15_TURN_AROUND:
    lcd.setCursor(0,1);
    //lcd.print("SubState: 3");
    lcd.print("Radiation Alert");
    turnLeft();
    lines = 0;
    lineCount();
    if (lines > 0) {
      stopRobot();
      lines = 0;
      turnLeft();
      delay(500);
      state15 = S15_DRIVE_TO_LINE;
    }
    break;

  case S15_DRIVE_TO_LINE:
    lcd.setCursor(0,1);
    //lcd.print("SubState: 4");
    lcd.print("Radiation Alert");
    lineFollow();
    lineCount();
    if (lines >= openTube) {
```

```
        stopRobot();
        driveForward();
        delay(900);
        stopRobot();
        lines = 0;
        state15 = S15_TURN_RIGHT;
      }
      break;

    case S15_TURN_RIGHT:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 5");
      lcd.print("Radiation Alert");
      turnRight();
      lineCount();
      if (lines > 1) {
        delay(1000);
        driveForward();
        delay(1000);
        stopRobot();
        lines = 0;
        state15 = S15_DRIVE_TO_STORAGE;
      }
      break;

    case S15_DRIVE_TO_STORAGE:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 6");
      lcd.print("Low Radiation");
      lineFollow();
      if (switchVal) {
        stopRobot();
        lcd.clear();
        state = S_UNSLURP_FUEL_ROD;
      }
      break;
  }
}
// ************ End Drive to Spent Fuel Rod Storage ************



// ************ Return to Second Reactor with New Fuel Rod ************
// returnToSecondReactor: void -> void
// This function directs the robot back to Reactor B from the new fuel rod storage
tube
void returnToSecondReactor() {
  /* Sub-states:
   * Sub-state 1: Back the robot away from the storage tube
   * Sub-state 2: Turn the robot right to face Reactor A
   * Sub-state 3: Line follow back to Reactor A until front limit switch is triggered
   */

  const char S16_BACKUP = 1;
  const char S16_TURN_LEFT = 2;
  const char S16_DRIVE_TO_REACTOR = 3;
```

```
  // Initial state
  static char state16 = S16_BACKUP;

  switch (state16) {
    case S16_BACKUP:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 1");
      lcd.print("Radiation Alert");
      driveBackward();
      delay(4000);
      stopRobot();
      state16 = S16_TURN_LEFT;
      break;

    case S16_TURN_LEFT:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 2");
      lcd.print("Radiation Alert");
      turnLeft();
      lineCount();
      if (lines > 0) {
        stopRobot();
        lines = 0;
        state16 = S16_DRIVE_TO_REACTOR;
      }
      break;

    case S16_DRIVE_TO_REACTOR:
      lcd.setCursor(0,1);
      //lcd.print("SubState: 3");
      lcd.print("Radiation Alert");
      lineFollow();
      if (switchVal) {
        stopRobot();
        state = S_LOWER_FOURBAR_AGAIN;
      }
      break;
  }
}
// ************ End Return to Second Reactor with New Fuel Rod ************


// ********************************** Sub-functions
***********************************
// ************ Line Count ************
// This function increments a global variable 'lines' whenever it encounters a line
void lineCount() {
  // Fetch sensor values returned in the counters array
  lineCounter.read(counterValues);
  // If either of the sensors reads a low reflectance and the line is a new line,
  // increment the line counter
  if ((counterValues[0] > 2100) && canCountLine) {
    lines++;
    canCountLine = false;
  } else if ((counterValues[0] < 2100) && !canCountLine) {
```

```
    canCountLine = true;
  }

//  Serial.println(counterValues[0]);
//  // Uncomment to debug
//  // Print the line count
//  lcd.setCursor(0,1);
//  char lineBuffer[9];
//  sprintf(lineBuffer, "Lines: %2d", lines);
//  lcd.print(lineBuffer);
}
// ************ End Line Count ************



// ************ Line Follow ************
// This function updates a variable 'position' that is used to make line following
decisions
void lineFollow() {
  // Read calibrated sensor values and obtain a measure of the line position from 0 to
5000
  unsigned int position = lineFollower.readLine(sensorValues);

  if (position <= 1500) {           // If the line is right of the center of the
sensor
    driveForwardLeft();
  } else if (position >= 3500) {    // If the line is left of the center of the sensor
    driveForwardRight();
  } else {                          // If the line is centered under the robot
    driveForward();
  }

//  // Uncomment to debug
//  // ************ Print Current Position ************
//  // Print the position
//  lcd.setCursor(0,1);
//  char positionBuffer[14];
//  sprintf(positionBuffer, "Position: %4d", position);
//  lcd.print(positionBuffer);
//  // ************ End Print ************
}
// ************ End Line Follow ************



// ************ Robot Motor Functions ************
// Stop the drivetrain
void stopRobot() {
  rightMotor.write(90);
  leftMotor.write(90);
}

// Drive forward at full speed (approx 2in/sec)
void driveForward() {
  rightMotor.write(180);
  leftMotor.write(0);
```

```
}

// Drive backward at full speed (approx 2in/sec)
void driveBackward() {
  rightMotor.write(0);
  leftMotor.write(160);
}

// Turn left around the center of the robot (approx NUMdeg/sec)
void turnLeft() {
  rightMotor.write(180);
  leftMotor.write(180);
}

// Turn right around the center of the robot (approx NUMdeg/sec)
void turnRight() {
  rightMotor.write(0);
  leftMotor.write(0);
}

// Used for line following, turns slightly left as it goes forward
void driveForwardLeft() {
  rightMotor.write(180);
  leftMotor.write(70);
}

// Used for line following, turns slightly right as it goes forward
void driveForwardRight() {
  rightMotor.write(110);
  leftMotor.write(0);
}
// ************ End Robot Motor Functions ************



// ************ Fourbar Motor Functions ************
// Turn collector motor on so that a fuel rod is drawn in
void slurp() {
  collectorMotor.write(180);
}

// Turn collector motor on so that a fuel rod is pushed out
void unslurp() {
  collectorMotor.write(0);
}

// Turn off the collector motor
void stopCollector() {
  collectorMotor.write(90);
}

// Rotate the fourbar towards its maximum position
void moveFourbarTop() {
  fourbarMotor.write(65);
}
```

```
// Rotate the fourbar towards its minimum position
void moveFourbarBottom() {
  fourbarMotor.write(130);
}

// Turn off the fourbar motor
void stopFourbar() {
  fourbarMotor.write(90);
}
// ************ End Fourbar Motor Functions ************



// ************ Radiation Alarm Function ************
// This function sounds an alarm on a piezo-electric speaker when there
// is an exposed fuel rod
void soundAlarm() {
  if (!timeTaken) {
    now = millis();
    timeTaken = true;
  }
  if (millis() >= (now + 10)) {
    tone(speakerPin, currentPitch, 10);
    currentPitch += pitchStep;
    if(currentPitch >= pitchHigh) {
      pitchStep = -pitchStep;
    }
    else if(currentPitch <= pitchLow) {
      pitchStep = -pitchStep;
    }
    timeTaken = false;
  }
}

// This function turns off the alarm
void stopAlarm() {
  noTone(speakerPin);
}
// ************ End Radiation Alarm Function ************
// ********************************** End Sub-functions
**********************************
```

Appendix I

Lab Performance:

Max Luu- 33%

Nick Benoit - 33%

Sierra Palmer - 33%

Final Project Performance:

Max Luu - 33%

Nick Benoit - 33%
Sierra Palmer- 33%

Appendix J

to see BOM, please open the SWDWG attached in the submissions