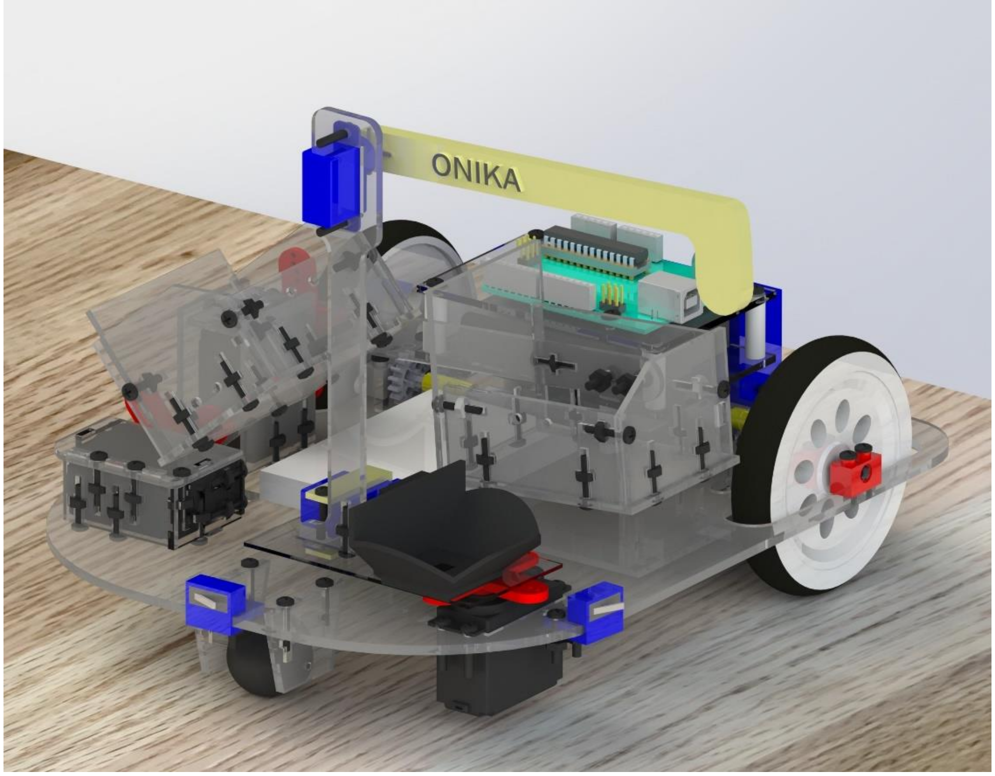
MECHATRONICS COMPETITION

Michaela Curcio & Michael Sherman



FALL 2020

Professor: Dr. Matthew Stein

Abstract

Mechatronics is a combination between Mechanics and Electronics. The class project was to complete the final task to build and program an autonomous mobile device to score as many points as possible in a 10-minute period by navigating a playfield while collecting and depositing objects into designated bases. The final task was a competition between all the teams in the class. There were 9 teams of 2-3 people in the class this semester. There were milestones throughout the semester that would help us to complete the final task. The robot was first designed on SolidWorks. Then it was made from scratch out of sheets of acrylic cut from a laser cutter, 3D printed structures, Lego pieces, and electrical components such as motors, servos, buttons, and ultrasonic. The robot was powered by an Arduino using C++ coding language to navigate around the playfield and to handle blocks. There were two days at the end of the semester to display the robot to complete the final task. The robot scored a perfect score of 64 points on the first day, putting with a lead for first place. The perfect score was not able to be replicated on the second day, but the robot scored 56 points, giving a total score of 116/128 points, landing in first place overall out of all 9 groups with the only group to score a perfect score on one of the demonstration days.

Table of Contents

Milestone 1

For our first milestone, we were instructed to make the Arduino play two easily distinguishable tunes of our choosing on the buzzer by pushing either of two buttons. The tune had to be at least 14 notes. The first tune that we chose was “Pound the Alarm” and the second tune was “Starships,” both by Nicki Minaj. Each button played the same tune and only that tune. After the tune was played, the Arduino stopped and waited until either button was pressed again. If either button was pressed while the tune was playing, the Arduino waited two seconds and began playing the pending tune after completing the current tune. The Arduino correctly responded to the instructor’s button pushes without rest.

The purpose of this milestone was to practice the use of buttons that would later be used for our later milestones when navigating the robot around the playfield. This milestone was done just with the use of a breadboard, Arduino, and computer. Other electronic components used were a buzzer, and two switches.

Circuit Diagram

Below is the circuit diagram for Milestone 1.

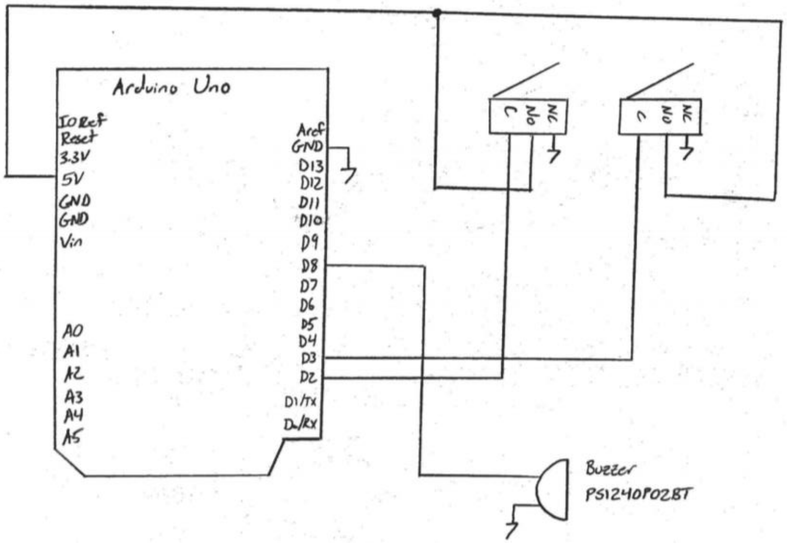


Figure 1: Circuit Diagram for Milestone 1

Pictures

Below is a photograph of the breadboard and Arduino used in this milestone.

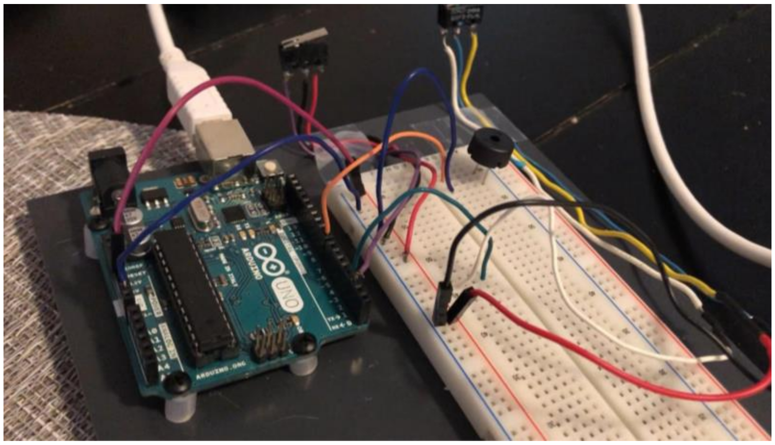


Figure 2: Photograph of Milestone 1 Setup

Milestone 2

For Milestone 2, we were told to navigate the playing field. The milestone directed to start the robot in the starting cube, play a note/tune when touching the white side base rail, and a different note/tune when touching the black side base rail (or vice versa), and then come back into the starting cube and play a third note/tune.

We began by piecing the robot together by using the materials given in our kit and by looking at the design already laid out for us. Legos had to be drilled, and wires had to be soldered together. Once the structure of the robot was completed using the given model, we added three buttons (switches). One in the front middle, one on the right side closer to the front, and another in the back closer to the left side. The breadboard was also completed based off of the given model. We adjusted the breadboard to add the buttons. Button holders were also created with the 3D printer and attached to the buttons and the acrylic base of the robot.

Our robot started in the starting cube at a diagonal facing the top left corner. We programmed our Arduino in order of states. State 1 caused the robot to move forward and once it touched the black side base rail it played a tune and switched to state 2. State 2 sent the robot backwards until it hit the back button against the back wall and switched to state 3. State 3 caused the robot to turn right until it hit the side button against the back wall and switched to state 4. State 4 caused the robot to “hug” the back wall. When the side button was pressed, the robot would move forward. If the button was not pressed, it shimmied right to “hug” the wall. Once it hit the right wall with the front button it was sent to State 5. State 5 caused the robot to turn left for a specific amount of time. Once the time was up, it was sent to state 6. State 6 caused the robot to “hug” the right wall until the front button was pressed on the white side base rail. This was similar to State 4. When the side button was not pressed, it shimmied right. If the side button was pressed, it went forward. Once the button was pressed, a second tune was played twice, it was then switched to state 7. State 7 caused the robot to be sent backwards until the back button was pushed and sent to state 8. State 8 caused the robot to turn left for a specific amount of time, then move forward for a specific amount of time to then play the third tune three times while the back two wheels landed inside the starting cube.

When we were testing our code, we found that our biggest issue was with the batteries on the robot. We saw that, when the robot slammed against the walls, it would pop the batteries out of place and cause the Arduino to reset. To fix this, we changed the orientation of the batteries and we changed our turning time to cause it not to slam sideways.

Our robot performed just as we had planned for both our unofficial trial and our first official trial. The robot did just as our code told it to do.

Circuit Diagram

Below is the circuit diagram for Milestone 2. The breadboard was initiated using a model given by the professor. After the wire design was copied over, we added the front, back, and right switches.

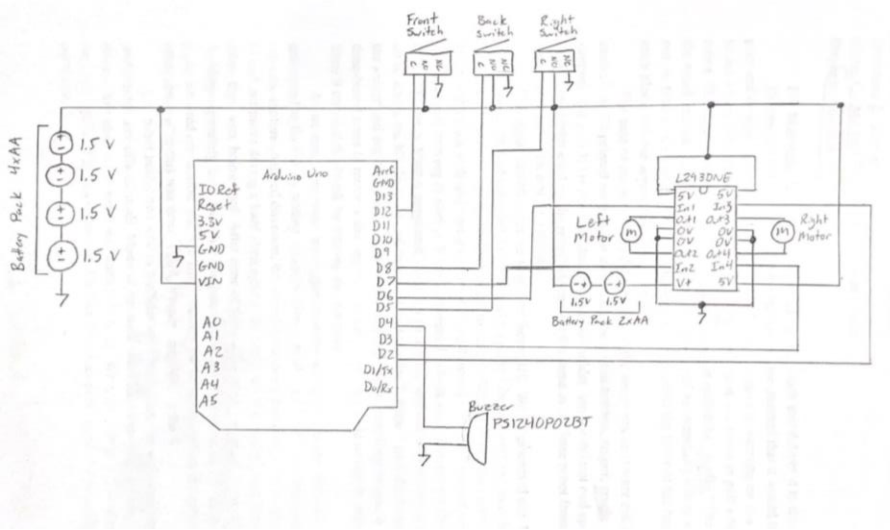


Figure 3: Circuit Diagram for Milestone 2

Pictures

Below is a photograph of our first robot used for Milestone 2. This robot is a replica of the model robot given by our professor. The photograph was taken in the starting position on the playfield. No SolidWorks assembly was completed for this milestone because every group in the class had the same design.

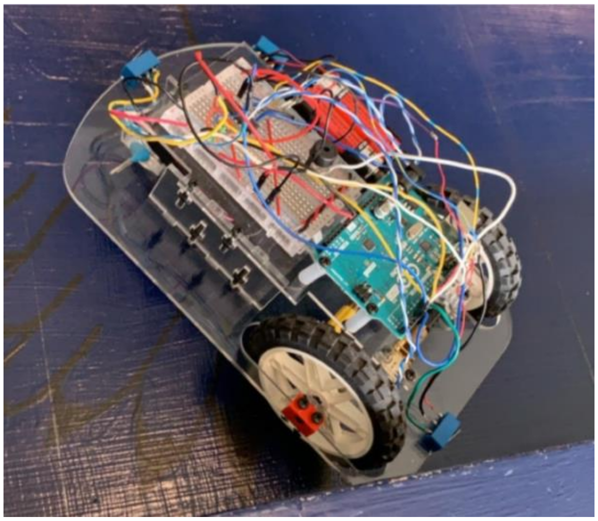


Figure 4: Photograph of Milestone 2 robot

Milestone 3

For Milestone 3, our task was to pick up one block and deliver it to the opposite side bin. During our brainstorming for this milestone, we decided that it would be a good idea to plan and design for future milestones as well, rather than only focusing on this one. Our idea was to use a long 3D printed part called the “sniper,” attached to a servo to pull a block onto the robot. The block would then fall into a 3D printed part called the “cradle.” The cradle then drops the block into the “dumpster” which holds the block before eventually lifting it and dumping it out. In future milestones, our goal is for the cradle to detect the color of the blocks and then put each block into the appropriate dumpster.

The base of our robot, and the majority of the structures on it were cut out of acrylic. We used some 3D printed parts in our design for the button holders, sniper, cradle, and Arduino support. We also 3D printed an ultrasonic sensor holder, but we did not end up using that in our code. The circuit used on the robot was mostly the same as the base robot from Milestone 2 with only a few additions and pin changes.

Our robot started in the starting cube facing left. We programmed our Arduino in order of states. State 1 moved the robot forward until it reached the left wall and switched to state 2. State 2 turned the robot to the right and switched to state 3. State 3 moved the robot forward until it reached the back wall and sent it to state 4. State 4 turned it to the right and moved it into the corner before moving to state 5. State 5 attempted to back the robot up into the other corner and move to state 6. State 6 positioned the robot so that the left side was touching the wall and drove up to where the blocks were and then switched to state 7. State 7 took the block off the wall with the sniper and emptied the cradle into the dumpster before switching to state 8. State 8 used time-based turns to move to the opposite corner of the play field so that it could empty the block. State 9 emptied the block by rotating the dumpster.

As we tested our code, the biggest problem we faced was the Arduino resetting often. We attempted to fix this by adding “platformStop()” with a delay of 20 milliseconds between each time the motors changed direction, we turned the battery pack sideways so that the batteries would not move during a hard slam against the wall, and we attached and detached servos only when they were being used. After none of these adjustments worked, we decided to power the Arduino separately from the other components. Without much room for another large battery pack, we used one double and two single battery packs spread out on the robot. There were no other resets after this was done.

Our robot performed well in multiple unofficial trials, but unfortunately it could not perform for any official trials. Many of our turns were based on time, and the robot did not always turn exactly the same way each time it ran. We were able to consistently get to the blocks and pick one up.

Circuit Diagram

Below is the circuit diagram for Milestone 3. This circuit diagram had the most drastic change from the previous milestone because we had to completely dismantle the previous robot and come up with our own design. As you can see below, we added three servos to retrieve and deposit the blocks. We the two small servos were added to move the sniper and the cradle. The large servo was added to move the back dumpster. We originally had problems with our first circuit diagram (not pictured), because of the lack of batteries that we used. Once we added more batteries, the circuit diagram had to be adjusted as seen below.

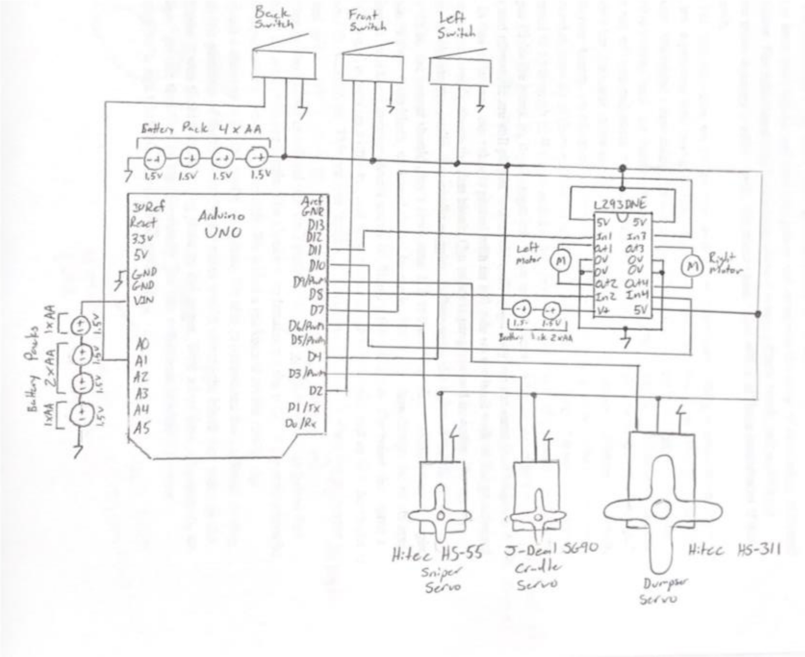


Figure 5: Circuit Diagram for Milestone 3

SolidWorks & Pictures

Below is a SolidWorks drawing of a layout of all the acrylic pieces that were cut using the laser cutter. The top picture shows the pieces that were cut with 1/8 inch thick acrylic, and the bottom picture shows the pieces that were cut with 3/32 inch thick acrylic. For the first milestone we did not use an access amount of acrylic because we wanted to make sure that we had extra for our following milestones, and for any pieces that could have broken in the process of assembling the robot together.

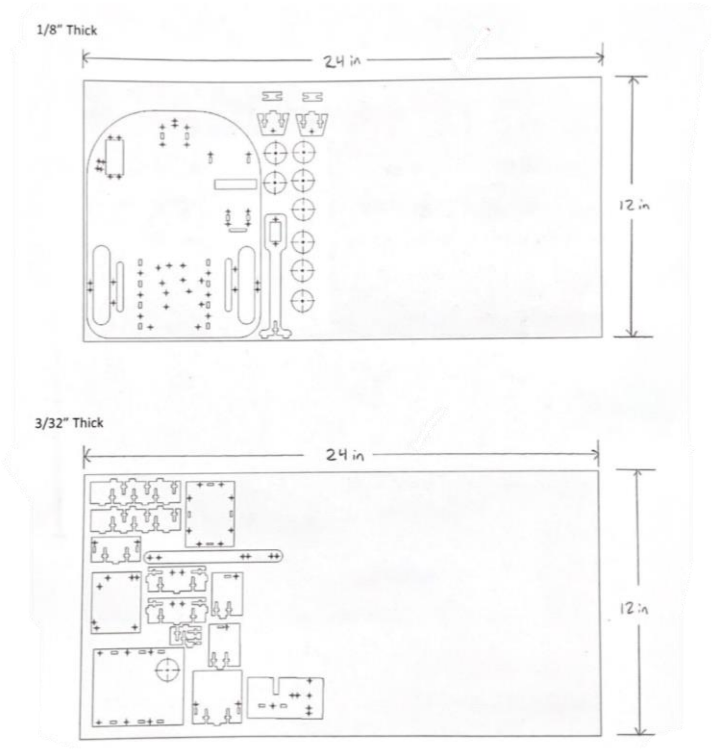


Figure 6: SolidWorks Layout of All Laser Cut pieces for Milestone 3

Below is a photograph of a top view layout and the robot created for Milestone 3. The robot was 8 inches wide, and 10.5 inches long. This is because we wanted to make the structure as large as possible to fit all of our components on it while still pushing most of our weight to the back closer the back wheels. We also had a restraint of keeping the full robot inside a 12 inch by 12 inch starting position square. The restraint meant that all components and wires not just the base itself had to be inside the starting square. We also knew that if we made the base too large, it would be too hard to maneuver around the playfield. We established these dimensions so that the robot could run as efficiently as possible.

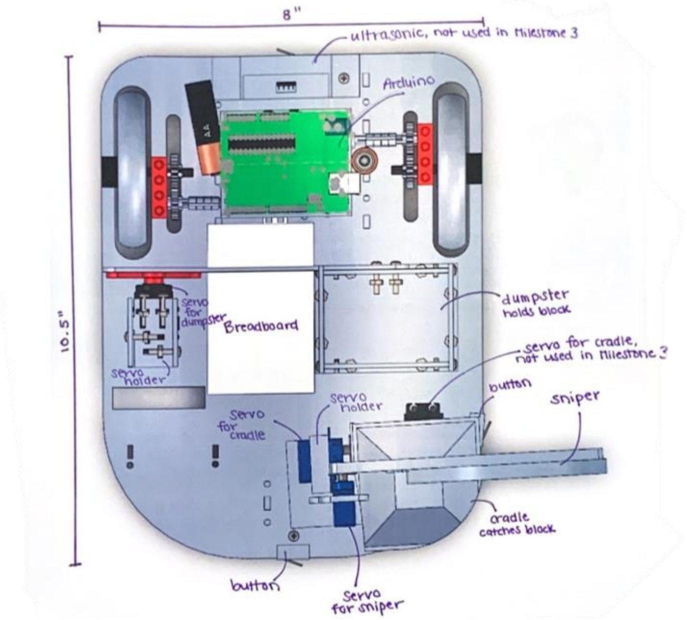
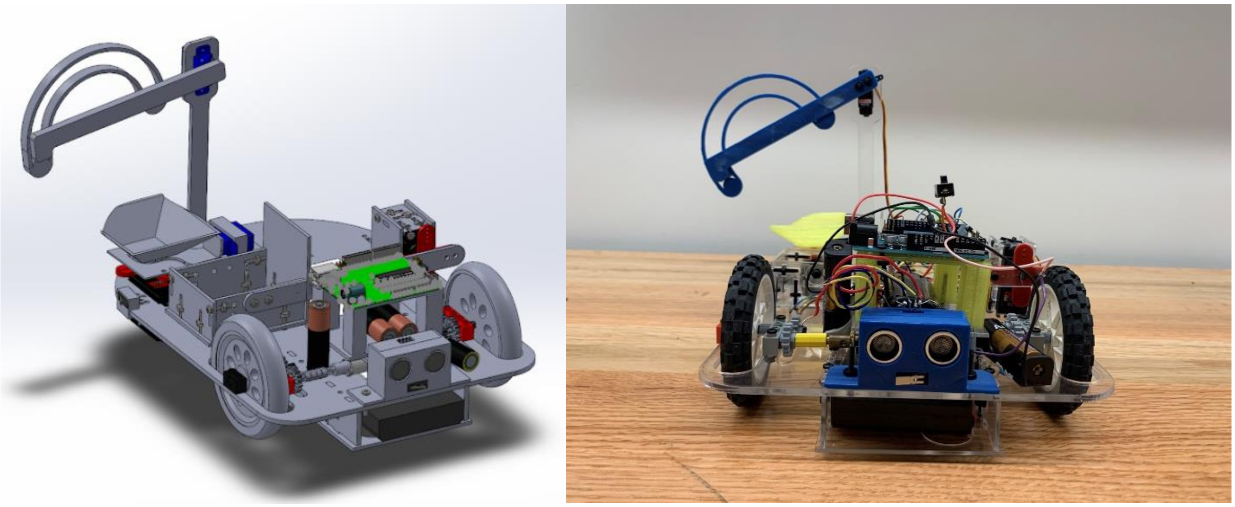


Figure 7: SolidWorks screenshot of Top View of robot for Milestone 3. This figure shows dimensions and labels to all components.

Below are SolidWorks screenshots of the robot paired with photographs of the robot for Milestone 3.



Figures 8 & 8: SolidWorks Model of Back Left View and Photograph of Back View of robot for Milestone 3.

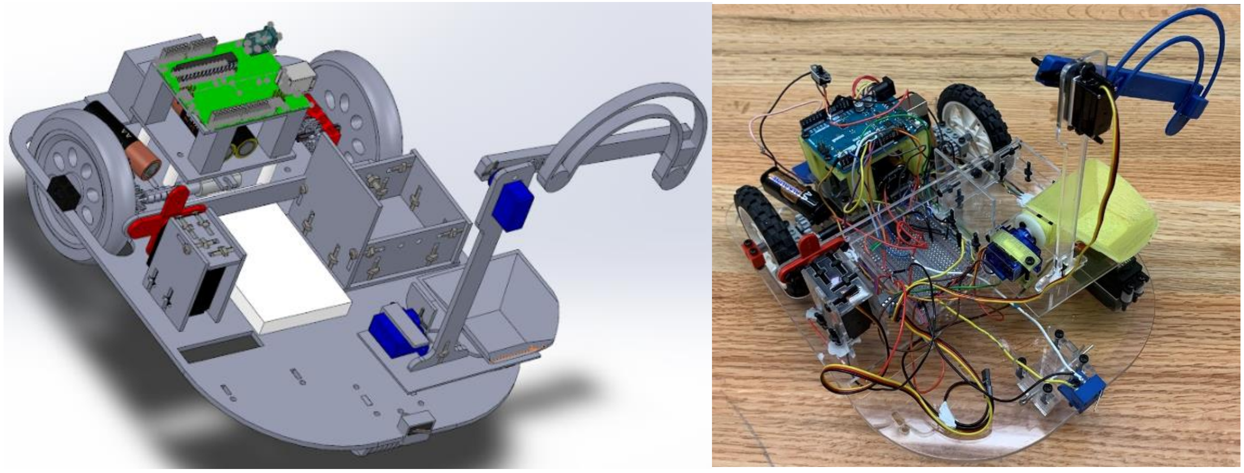


Figure 9 & 10: SolidWorks Model and Photograph of Front Right View of robot for Milestone 3.

Milestone 4

For Milestone 4, our robot could start anywhere on the gameboard. The robot must acquire at least three blocks under its own power and recognize the color of each when delivered to the sensor. The robot must play a note/tune when it senses a black block, and a different note/tune when it senses a white block. The robot cannot play either of those notes/tunes if there is no block.

For this milestone, we did not have to do much more laser cutting or assembling. Since we ran into a problem with having to add extra batteries for Milestone 3, we decided to modify our design. We added a new holder for our Arduino. Instead of having two single batteries and one double battery back, we decided to switch it to a battery pack holding four batteries and put it under our Arduino and attach it to the Arduino holder. We also decided to modify our “sniper.” The sniper has the same shape as the one used in Milestone 3, but instead of it being two pieces, it is only one. Lastly, we adjusted our “cradle.” Instead of just having some of the sides curved, we decided to have all of them curved. We also added a hole in the bottom of it just large enough for a sensor to fit through it. We also added a “backboard” to the inside of the cradle so that when the sniper flicks the block in, it goes right into place in the cradle on top of the sensor. Our buttons and ultrasonic are still placed in this robot, although they are not used for Milestone 4.

In this milestone, our robot is placed with its left side on the back wall of the gameboard with the sniper directly above the first block. Our robot is programmed in states. The sniper flicks the block into the cradle. The cradle wiggles, to then sense the color of the block. If the block is white, the buzzer should play a siren tune. If it is black, the buzzer should play a single deep note. If there is no block, no sound should be made. The cradle then dumps the block into the dumpster, and the dumpster should empty by flinging the block out. The robot then turns a little to the right, moves up a little bit, and turns a little more to the right, just so that the sniper is right above the next block. This process repeats until we manually stop it when the milestone is completed.

Our group got our unofficial trial on Tuesday, October 13. We realized the issues that were appearing were with our cradle. The blocks were landing on the sides of the cradle instead of on the sensor at the bottom of the cradle. We added a backboard on the cradle and programmed a shimmy to fix the cradle into place. We also programmed the dumpster to drop back onto the platform of the robot so that the robot would vibrate the block into place on the cradle in case it was flicked in the wrong place by the sniper. With all of these adjustments, we were able to get our first Official trial flawlessly.

Circuit Diagram

Below is the circuit diagram for Milestone 4. This circuit diagram is very similar to Milestone 3. The only difference is that the display of batteries was rearranged, and the buttons were taken out.

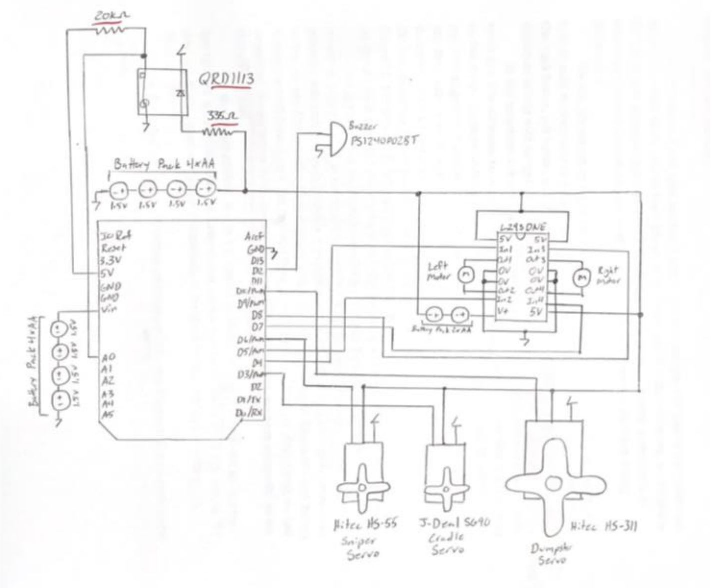


Figure 11: Circuit Diagram for Milestone 4

SolidWorks & Pictures

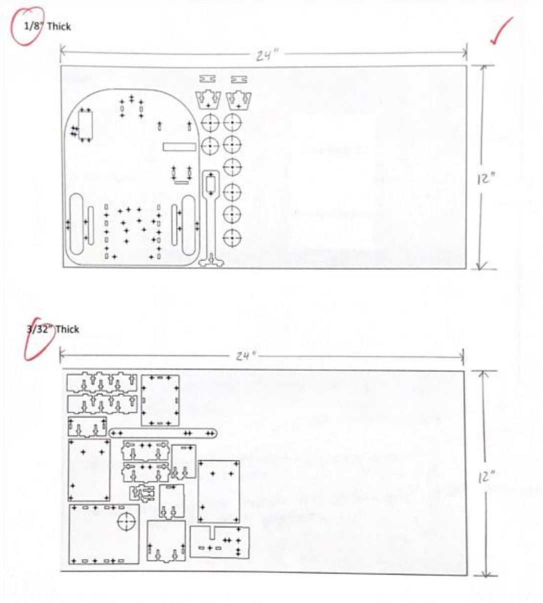


Figure 12: SolidWorks Layout of All Laser Cut pieces for Milestone 4

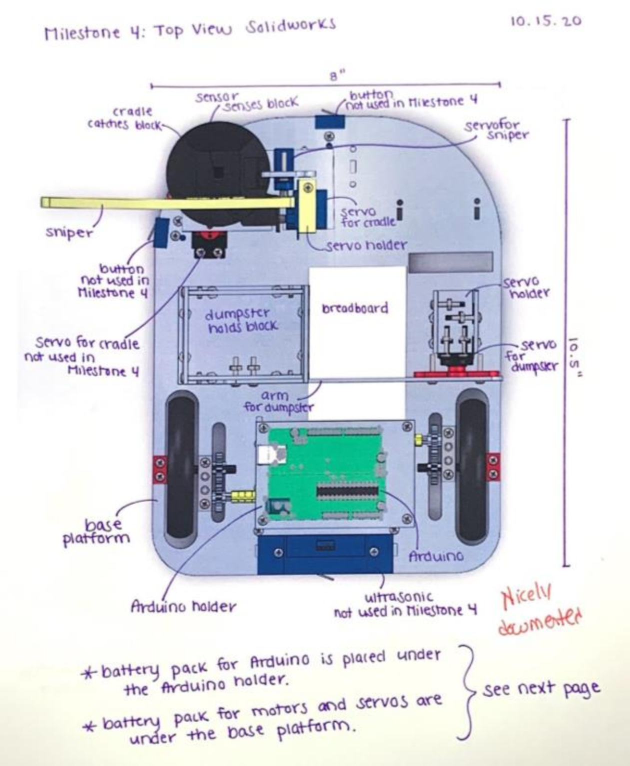

Milestone 4: Top View Solidworks
10.15.20
8"
sensor senses block
cradle catches block
button not used in Milestone 4
servofor sniper
sniper
servo for cradle
servo holder
button not used in Milestone 4
dumpster holds block
breadboard
servo holder
Servo for cradle not used in Milestone 4
servo for dumpsier
10.5"
arm for dumpster
base platform
Arduino
Arduino holder
ultrasonic not used in Milestone 4
Nicely documented
* battery pack for Arduino is placed under the Arduino holder.
* battery pack for motors and servos are under the base platform.
see next page

Figure 9: SolidWorks screenshot of Top View of robot for Milestone 4. This figure shows dimensions and labels to all components.

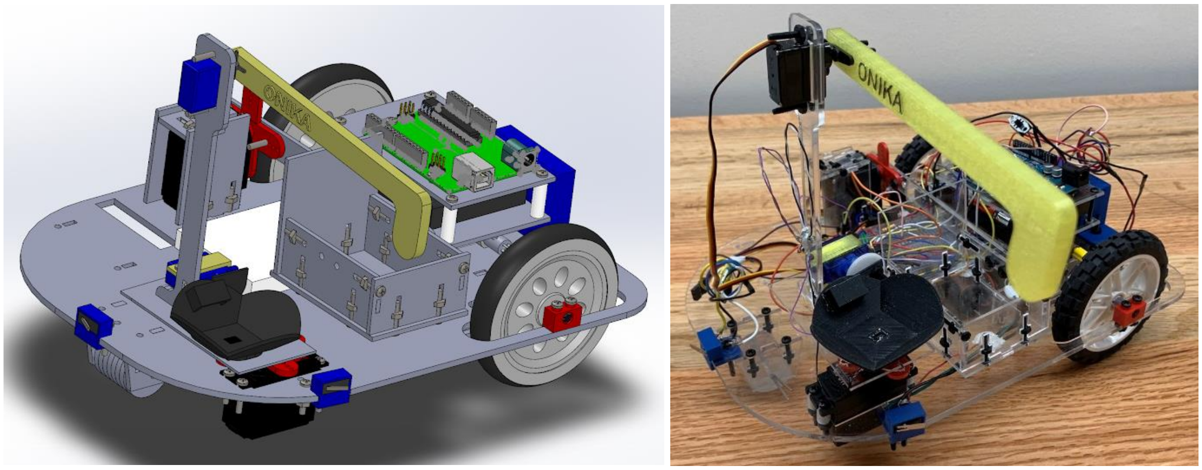


Figure 10: Front Left View of SolidWorks Model and Photograph of Milestone 3 Robot

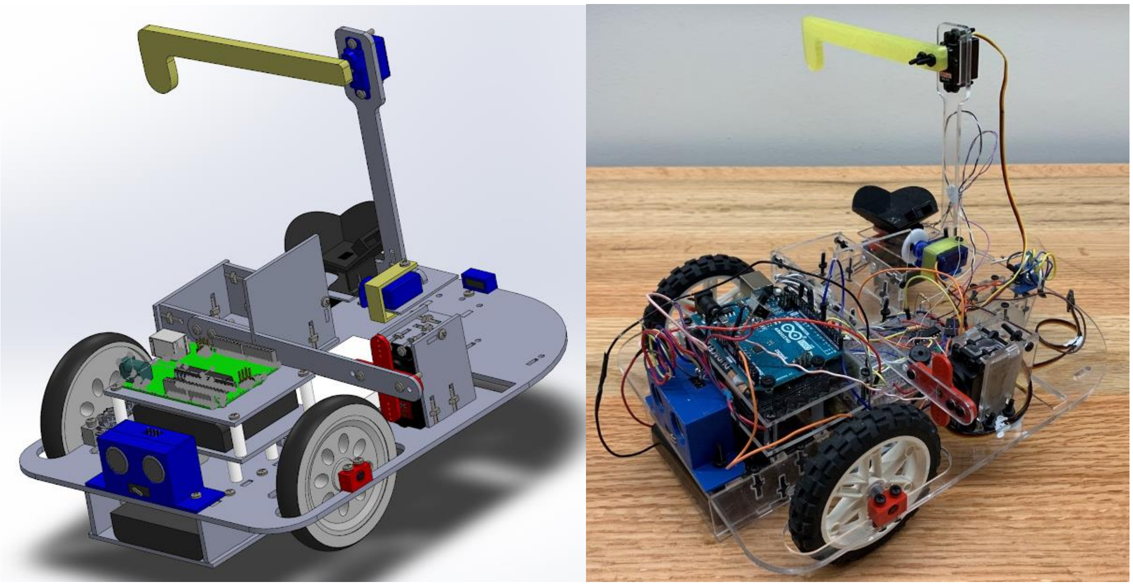


Figure 11: Back Right View of SolidWorks Model and Photograph of Milestone 3 robot

Milestone 5

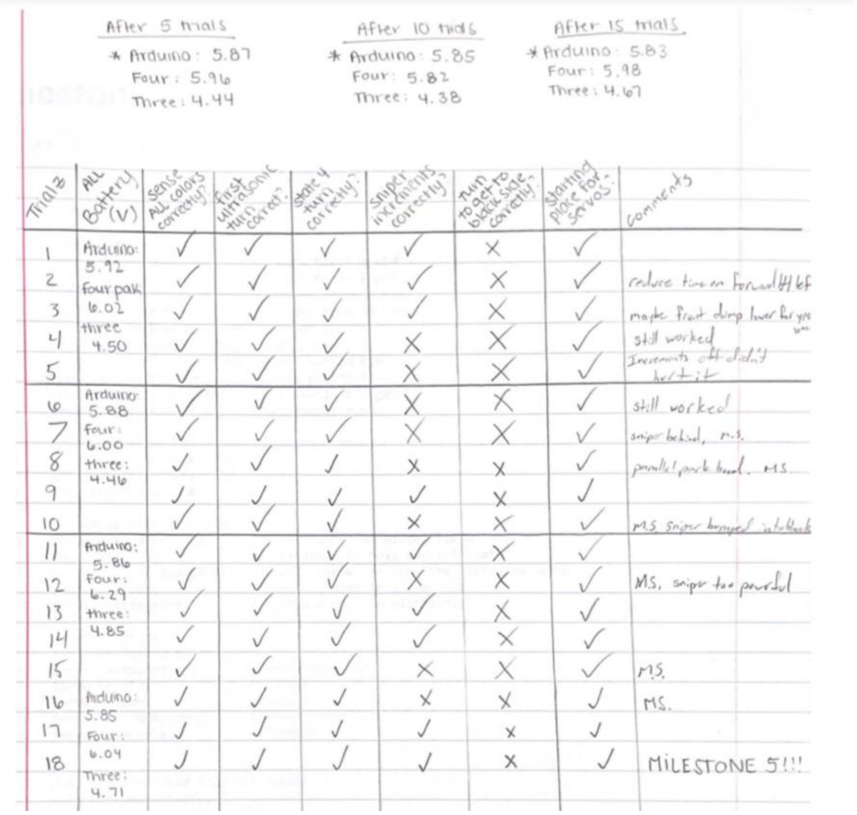
For Milestone 5, our robot needed to start in the starting cube, navigate to the blocks, and deliver two white blocks and no black blocks to the correct bin. For this milestone, we needed to add our second dumpster and connect our “turntable” servo. These additions were simple to implement, because we had planned for them when we designed most of the robot during Milestone 3.

We also 3D printed an improved “cradle” design. By adjusting the angles and adding a vertical side, we were able to position blocks on top of the color sensor much more effectively. Our back button, left button, and ultrasonic sensor were connected to the Arduino for this milestone to help with navigation.

In this milestone, our robot begins by reversing out of the starting cube towards the white block bin. The ultrasonic sensor reads when the robot is under 8 cm away and tells it to turn right so that it can reverse towards the blocks. After some timed turns and reversing, the robot is perfectly aligned with the first block. Just as it did in Milestone 4, the “sniper” pulls the block into the cradle where the color is sensed. If it is a black block, the cradle empties into the back dumpster. If it is a white block, the robot turns away from the wall, rotates the turntable, lowers the front dumpster, and empties the cradle. It then moves back to the way it was and moves forward to the next block. After all of the blocks have been collected and sorted, the robot backs up into the wall, turns and empties the front dumpster filled with white blocks to the correct bin.

Our group got our unofficial trial on Friday, October 30. Because this was the first milestone requiring navigation since Milestone 3, we did not want to take any chances with potential malfunctions. To combat this, we created a chart with all of the problematic areas listed so that we could keep track of how frequently each one occurred. We also frequently took battery readings at each battery pack and recorded those as well. Prior to calling an official trial, we had completed 18 runs with a 100% success rate. Due to this preparation, we were successful on our first official trial.

Our next major addition will be the implementation of a magnetometer. We are hoping that this will allow us to have more control over our turns and rely less on unreliable timing. We plan on improving our navigation and delivery to the black bin and we hope to attempt The Doubler.



After 5 trials	After 10 trials	After 15 trials
* Arduino: 5.87	* Arduino: 5.85	* Arduino: 5.83
Four: 5.96	Four: 5.82	Four: 5.98
Three: 4.44	Three: 4.38	Three: 4.67

Trial #	ALL Battery (V)	sense ALL colors correctly?	first ultrasonic turn correct?	state 4 turn correctly?	sniper increments correctly?	turn to get to black side correctly?	starting place for servos?	comments
1	Arduino: 5.92 four pak 6.02 three 4.50	✓	✓	✓	✓	✗	✓	
2		✓	✓	✓	✓	✗	✓	reduce time on forward lift left
3		✓	✓	✓	✓	✗	✓	maybe front clamp lower for yes
4		✓	✓	✓	✗	✗	✓	still worked
5		✓	✓	✓	✗	✗	✓	Increments off didn't hurt it
6	Arduino: 5.88 four: 6.00 three: 4.46	✓	✓	✓	✗	✗	✓	still worked
7		✓	✓	✓	✗	✗	✓	sniper behind, M.S.
8		✓	✓	✓	✗	✗	✓	parallel park bad. MS.
9		✓	✓	✓	✓	✗	✓	
10		✓	✓	✓	✗	✗	✓	M.S. sniper bumped into black
11	Arduino: 5.86 Four: 6.29 three: 4.85	✓	✓	✓	✓	✗	✓	
12		✓	✓	✓	✗	✗	✓	M.S. sniper too powerful
13		✓	✓	✓	✓	✗	✓	
14		✓	✓	✓	✓	✗	✓	
15		✓	✓	✓	✗	✗	✓	M.S.
16	Arduino: 5.85 Four: 6.04 Three: 4.71	✓	✓	✓	✗	✗	✓	MS.
17		✓	✓	✓	✓	✗	✓	
18		✓	✓	✓	✓	✗	✓	MILESTONE 5!!!

Circuit Diagram

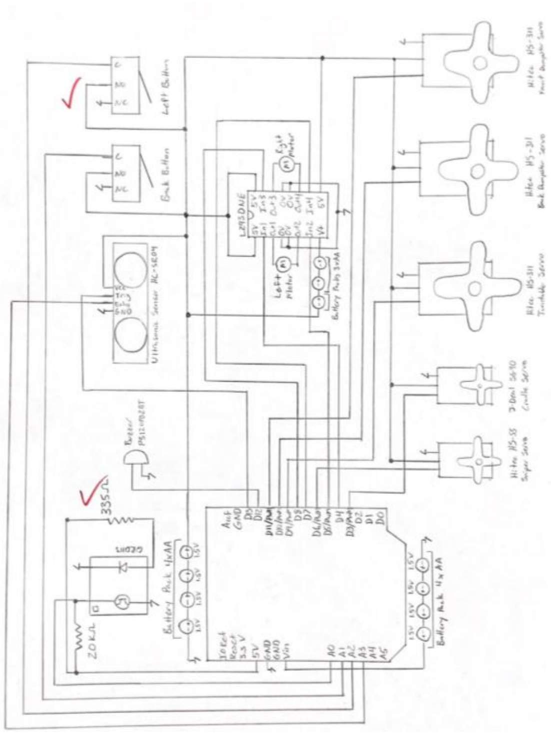


Figure 12: Circuit Diagram for Milestone 5

SolidWorks & Pictures

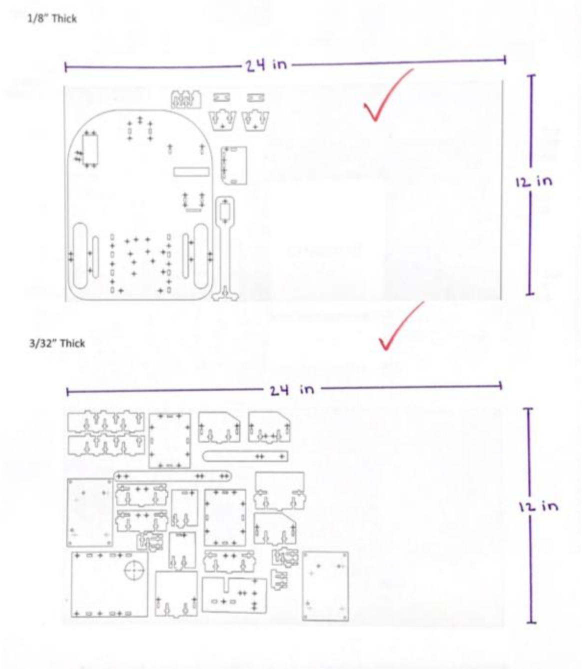


Figure 13: SolidWorks Layout of All Laser Cut pieces for Milestone 5

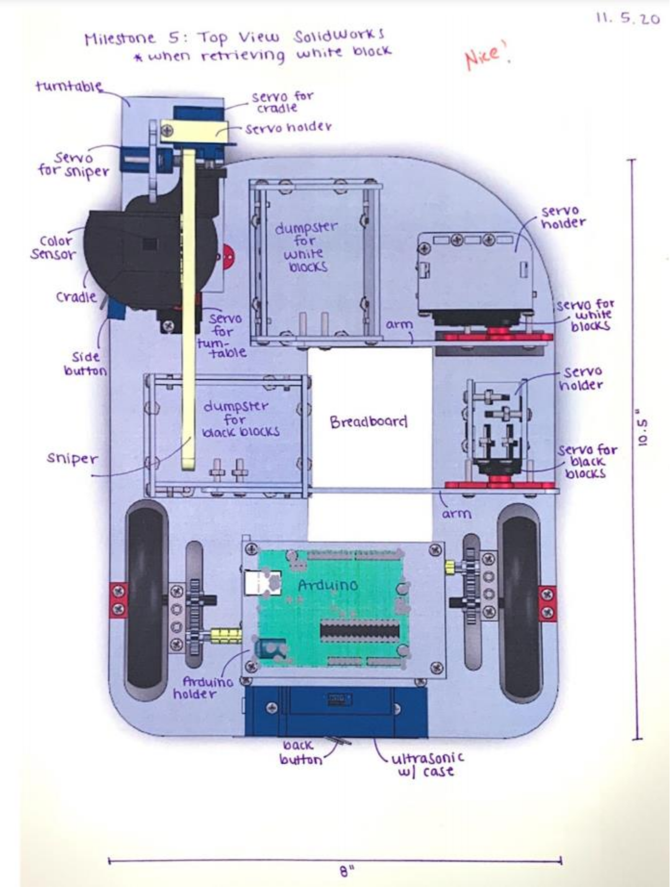


Figure 14: SolidWorks screenshot of Top View of robot for Milestone 5. This figure shows dimensions and labels to all components.

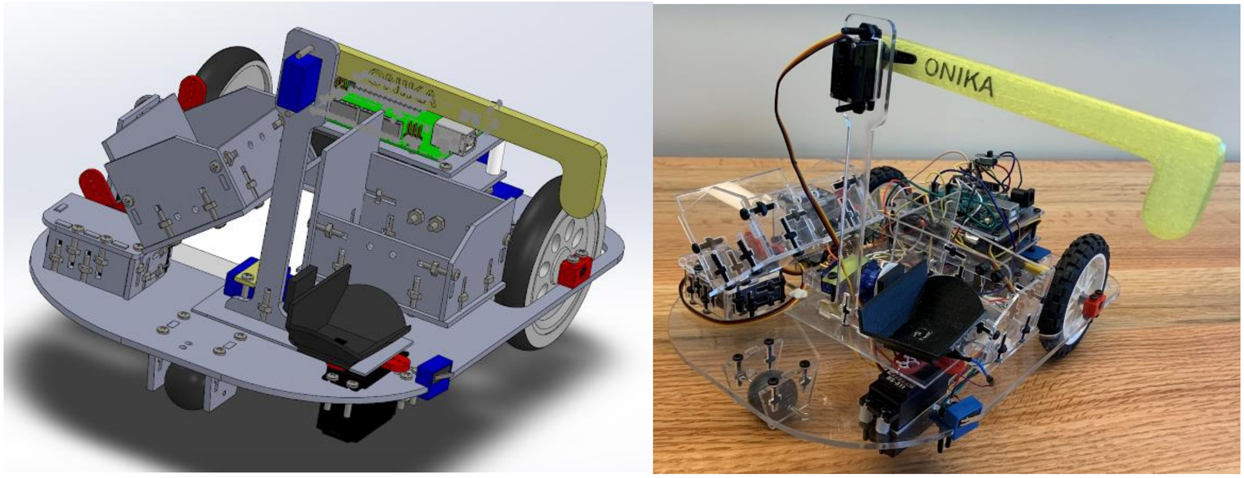


Figure 15: Front Left View of SolidWorks Model and Photograph of Milestone 5 Robot

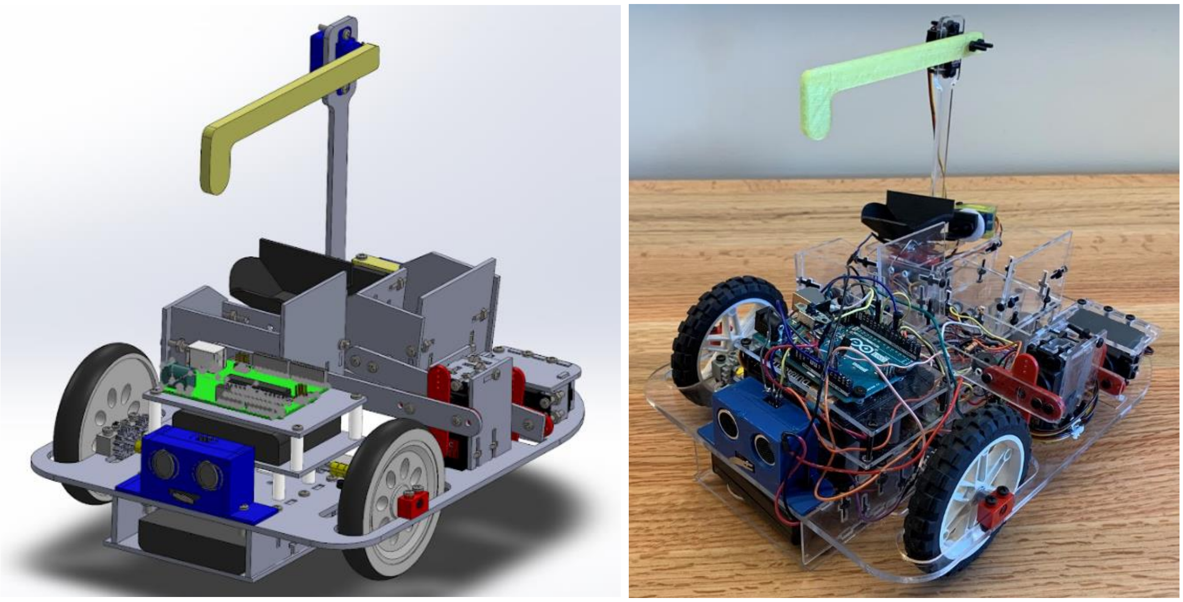


Figure 16: Back Right View of SolidWorks Model and Photograph of Milestone 5 Robot to dump white blocks

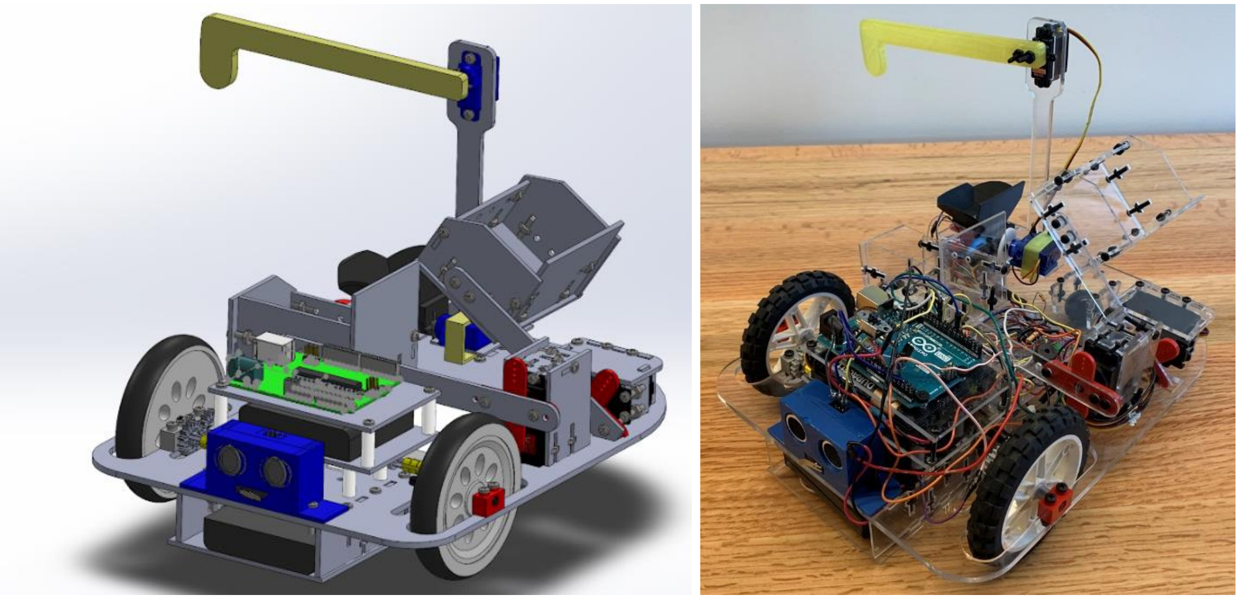


Figure 17: Back Right View of SolidWorks Model and Photograph of Milestone 5 Robot to dump black blocks

Final Product

The final mechatronics design project task is to score as many points as possible in a 10-minute period by navigating a playfield while collecting and depositing white and black blocks into their designated bases.

The final design for our robot was the same design we had constructed in Milestone 5. We did not change any physical mechanical components. Because our robot was physically ready after Milestone 5, we had a lot of time to experiment with our code and the electrical components. Our Milestone 5 code was a good starting place for our final project, we kept it mostly the same. We added delivering to the black bin, and then navigating to the second set of blocks. We copied over the majority of the code from the first set of blocks to the second. We also added the depositing of the blocks. Lastly, we added code to retrieve the doubler. This code was attached to the beginning so that it could be the first thing that the robot does. We decided to get the doubler first because we felt confident that our robot could get the majority of the blocks. We did not want to see the robot retrieve all the blocks, then miss the doubler at the end. We could easily restart the robot if it had missed the doubler in the beginning, saving time.

One of the hardest parts of this project was figuring out how to navigate the robot around the playfield. This was because it depended a lot on the battery voltage that our robot had. Because our robot was a larger robot, it burned out its batteries fast. The robot consisted of 11 batteries. Four of the batteries connected to the Arduino, and the other 7 connected to the motor and servos. If the robot had fresh new batteries, the 7 batteries usually added up to about 11 V. When the robot was under 10.60 V, she was unable to turn and ride against a wall. The navigation was also difficult because if the batteries varied, the degree of the turns varied. This is because we base our turns off of time. We saw that this was an issue, so we tested out different ideas. We decided to rule out the idea of a light sensor and tape because we did not want to waste time taping down the playfield if we could not get it in the exact position every time. Different electrical components we tried were connecting both a magnetometer and a gyroscope. We decided not to go through with these because the magnetometer was not accurate enough and the gyroscope only measured angular velocity, which could not reliably be related to position.

Some of the strengths that our design had was that it was capable of holding all of the blocks from each rack. Because we had different dumpsters for each block color, we were able to retrieve all the blocks from a rack at a time, and then deliver them all at the same time. This reduced our time because we did not have to return to the same rack more than once, and we did not have to navigate as much. Although we had a larger robot, with many components, we did a good job of keeping the heaviest components near the back wheels of the robot. This caused the robot to be able to turn easier. Another big advantage of our design was our “cradle.” Our cradle allows us to retrieve the blocks before we sense what color they are. This is an advantage because when other designs sense the block before retrieving it, they sometimes retrieve the wrong color, or they get multiple blocks at once because it is not perfectly aligned. Our biggest advantage was that we had the same design since Milestone 3. This was a huge benefit to us because it allowed us to really get to know our robot. We knew exactly how it was able to navigate, what the problem was if there was one, and what battery voltages it would perform the best on. Having our design done early allowed us to spend less time on designing, and more time on perfecting our navigation. We were able to run multiple tests, and we were able to see all of the problems that could occur.

Weaknesses that we had in our design was our ability to get to the blocks the same way every time. If the voltage was low, the increments between each block were lessened, but if the voltage was high, she would go too far. To solve this issue, we decided to have shorter scooches, and more sniping of the “sniper.” We did this because it would be better if the sniper hit the front corner of the block rather than the back corner of the block. If the sniper hit the back corner of the block, there was a chance that the block could just skip over the cradle and fall directly into the black dumpster. If the sniper hit the front corner, the block would still shoot right into place on the cradle because of the following block sitting next to it. We added extra snipes in case the sniper missed a block. Another weakness in our design was relying on time for turns, but we still thought that that would be the best option for our design. Another weakness in our design was how large our robot was causing it to wear down batteries fast, and causing it to be harder to maneuver around the playfield.

Lessons learned from this competition were how to design and program a robot from scratch. We learned how to fabricate our 3D models using the laser cutter and 3D printer. We also learned that it is much easier to have a mechanical error to solve than it is to find a bug in the code.

Before we were called up to the board for our final demonstrations, we went over a checklist of things we needed to do. This checklist included changing the batteries; check all wires to make sure they are attached correctly, especially the wires connected to the sensor on the cradle; check to make sure the Lego pieces were attached correctly to the motors because they tend to fall out from time to time; check to make sure the sensor was adhered to the cradle correctly, because we found that if the sensor was sticking out a little bit, the block would be too close to it to sense the color correctly, and it would also misalign the block from sitting in the cradle the correct way; and check the entire robot for any loose screws. This checklist was created because these were the most common mistakes that would occur to cause our robot to fail. We are extremely happy with the result of our first performance on Thursday, November 19. After just a second run through, our robot scored a perfect score of 64 points, putting us in the lead for first place. The results of our second performance on Tuesday, November 24 was a score of 56 points, giving us a total score of 116 points. This landed us in first place overall!!

Some of the things that our group would have done differently if we were able to start fresh would be spending more time on the geometry of the cradle so that it catches the blocks better and positions them on the color sensor better. We would also look into using tape for our block increments, so we do not need to rely on time for those anymore.

Circuit Diagram

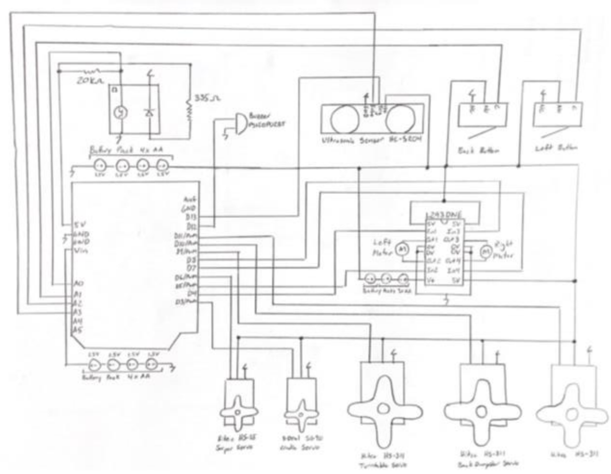


Figure 18: Circuit Diagram for Final Product

SolidWorks & Pictures

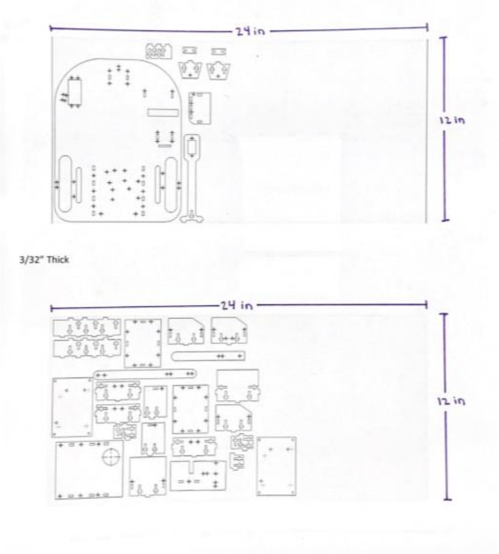


Figure 19: SolidWorks Layout of All Laser Cut pieces for Final Product

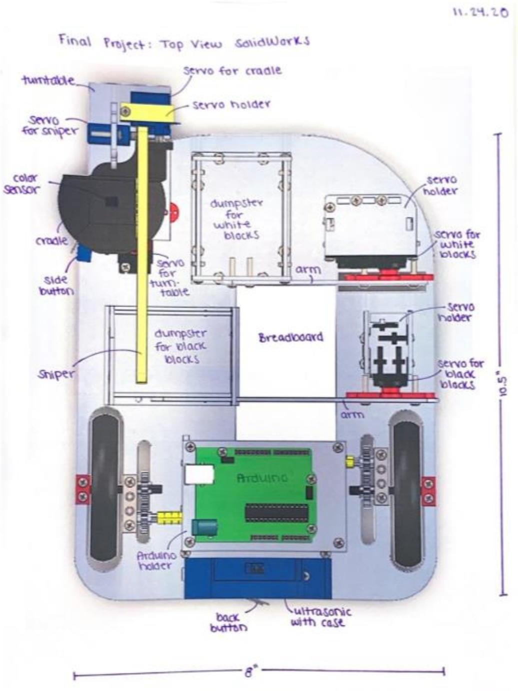
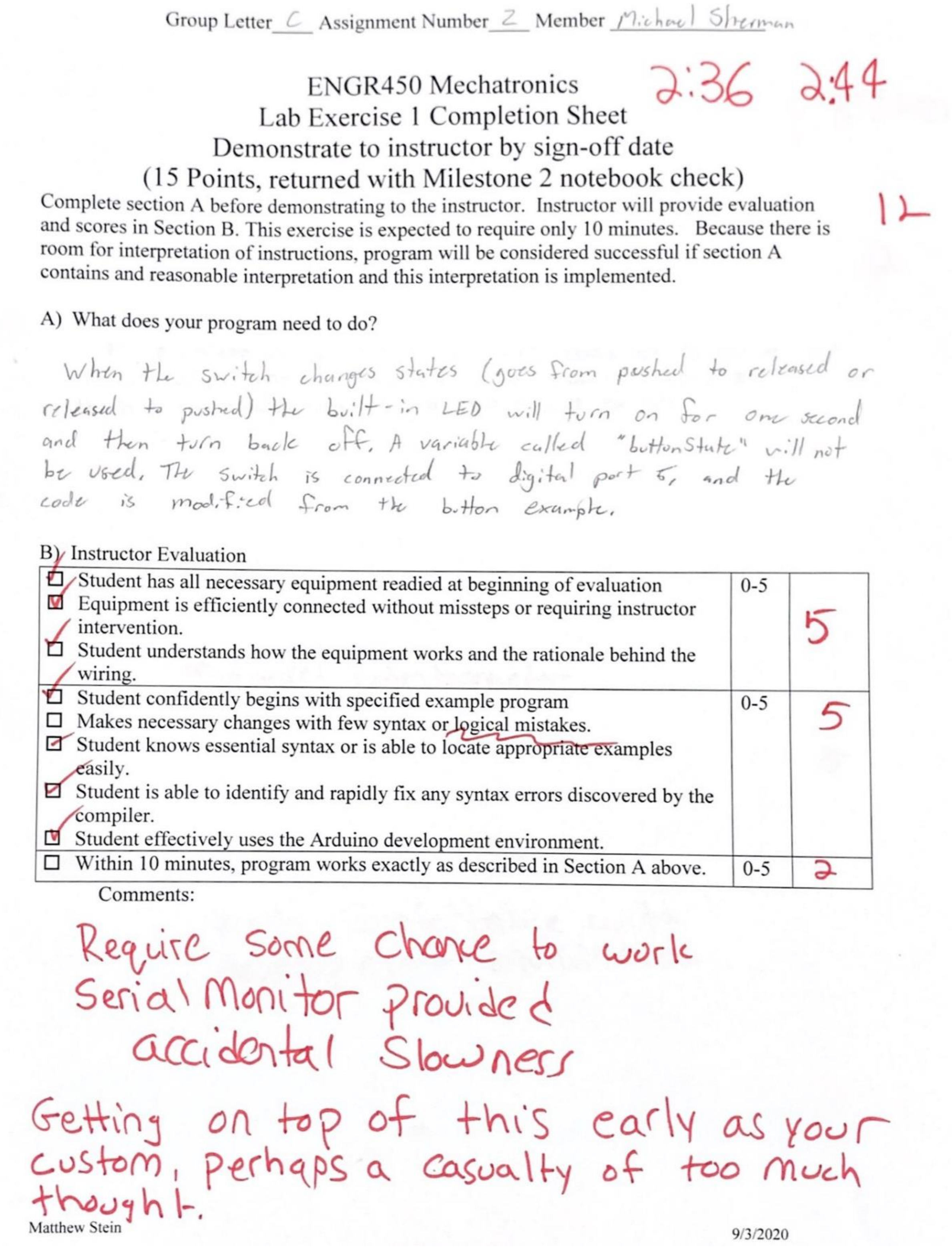


Figure 20: SolidWorks screenshot of Top View of robot for Final Product. This figure shows dimensions and labels to all components.

Appendix A

Independent Labs



Group Letter C Assignment Number 2 Member Michael Sherman

2:36 2:44

ENGR450 Mechatronics
Lab Exercise 1 Completion Sheet
Demonstrate to instructor by sign-off date
(15 Points, returned with Milestone 2 notebook check)

12

Complete section A before demonstrating to the instructor. Instructor will provide evaluation and scores in Section B. This exercise is expected to require only 10 minutes. Because there is room for interpretation of instructions, program will be considered successful if section A contains and reasonable interpretation and this interpretation is implemented.

A) What does your program need to do?

When the switch changes states (goes from pushed to released or released to pushed) the built-in LED will turn on for one second and then turn back off. A variable called "buttonState" will not be used. The switch is connected to digital port 5, and the code is modified from the button example.

B) Instructor Evaluation

☑ Student has all necessary equipment readied at beginning of evaluation ☑ Equipment is efficiently connected without missteps or requiring instructor intervention. ☑ Student understands how the equipment works and the rationale behind the wiring.	0-5	5
☑ Student confidently begins with specified example program ☐ Makes necessary changes with few syntax or logical mistakes. ☑ Student knows essential syntax or is able to locate appropriate examples easily. ☑ Student is able to identify and rapidly fix any syntax errors discovered by the compiler. ☑ Student effectively uses the Arduino development environment.	0-5	5
☐ Within 10 minutes, program works exactly as described in Section A above.	0-5	2

Comments:

Require some Chance to work
Serial Monitor provided
accidental Slowness

Getting on top of this early as your custom, perhaps a casualty of too much thought.

Matthew Stein 9/3/2020

Group Letter C Assignment Number 1 Member Michaela Curcio

ENGR450 Mechatronics
Lab Exercise 1 Completion Sheet
Demonstrate to instructor by sign-off date
(15 Points, returned with Milestone 2 notebook check)

1:57-2:00

Complete section A before demonstrating to the instructor. Instructor will provide evaluation and scores in Section B. This exercise is expected to require only 10 minutes. Because there is room for interpretation of instructions, program will be considered successful if section A contains and reasonable interpretation and this interpretation is implemented.

12

A) What does your program need to do?

My program makes the Built-In LED blink for 1/2 sec on and 1/2 off only if the potentiometer knob is being turned. If the knob is not being turned, the LED should be off.

B) Instructor Evaluation

□ Student has all necessary equipment readied at beginning of evaluation ☒ Equipment is efficiently connected without missteps or requiring instructor intervention. □ Student understands how the equipment works and the rationale behind the wiring. ? Miswired Potentiometer	0-5	2
☑ Student confidently begins with specified example program ☑ Makes necessary changes with few syntax or logical mistakes. ☑ Student knows essential syntax or is able to locate appropriate examples easily. ☑ Student is able to identify and rapidly fix any syntax errors discovered by the compiler. ☑ Student effectively uses the Arduino development environment.	0-5	5
☑ Within 10 minutes, program works exactly as described in Section A above.	0-5	5

Comments:

Pretty Comfortable with develop ment environment

Group C Number 2 Name Michael Sherman 25

ENGR450 Mechatronics
Lab Exercise 2 Completion Sheet
Demonstrate to instructor before the sign-off date
(25 Points, returned with Milestone 3 notebook)

Complete section A before demonstrating to the instructor. The instructor will initial section B on successful demonstration of the program. Instructor will ask you questions about your program and evaluate your comprehension of, and ability to precisely describe your program. Section C contains feedback from the interview.

A) In the space below, describe your program. How does it work?

The program takes a user input between 10 and 80 characters and converts it to title case. If the user enters less than 10 characters, they are scolded and prompted again. If they enter more than 80, the characters beyond 80 are ignored. After the user is prompted and enters their sentence, it is read until 80 bytes. The length is stored in variable inputLen and the character array is stored in variable userArray. If the input is greater than 10, the first character is converted to uppercase by adding 32 to its ASCII Decimal value if it is lowercase. Then, a for loop checks each remaining character. If it is a space (ASCII decimal of 20), then the next letter in the array is converted to uppercare if it is a lowercase letter.

B) Program is demonstrated works:

✓ Exactly as specified	______ Most of the time
______ Some of the time	______ Occasionally

Specifics:

C) Instructor comments on the interview. Has student successfully demonstrated full comprehension of the program he or she has submitted for completion of this assignment?

Seemed to be able to fix appacl- Problem with the -32

You Continue to Amaze all of us with your Systematic approach to Course assignments. We would have a lot More like you.

Group C Number 1 Name Michaela Curcio

25

ENGR450 Mechatronics
Lab Exercise 2 Completion Sheet
Demonstrate to instructor before the sign-off date
(25 Points, returned with Milestone 3 notebook)

Complete section A before demonstrating to the instructor. The instructor will initial section B on successful demonstration of the program. Instructor will ask you questions about your program and evaluate your comprehension of, and ability to precisely describe your program. Section C contains feedback from the interview.

A) In the space below, describe your program. How does it work?

The serial monitor should promt the user to wait for a reading. Once the button is pressed, it should read the potentiometer. This should happen 3 times. It then should average the 3 numbers, and start again. There should be a minimum of 0.25 s between the readings.

B) Program is demonstrated works:

✓ Exactly as specified	______ Most of the time
______ Some of the time	______ Occasionally

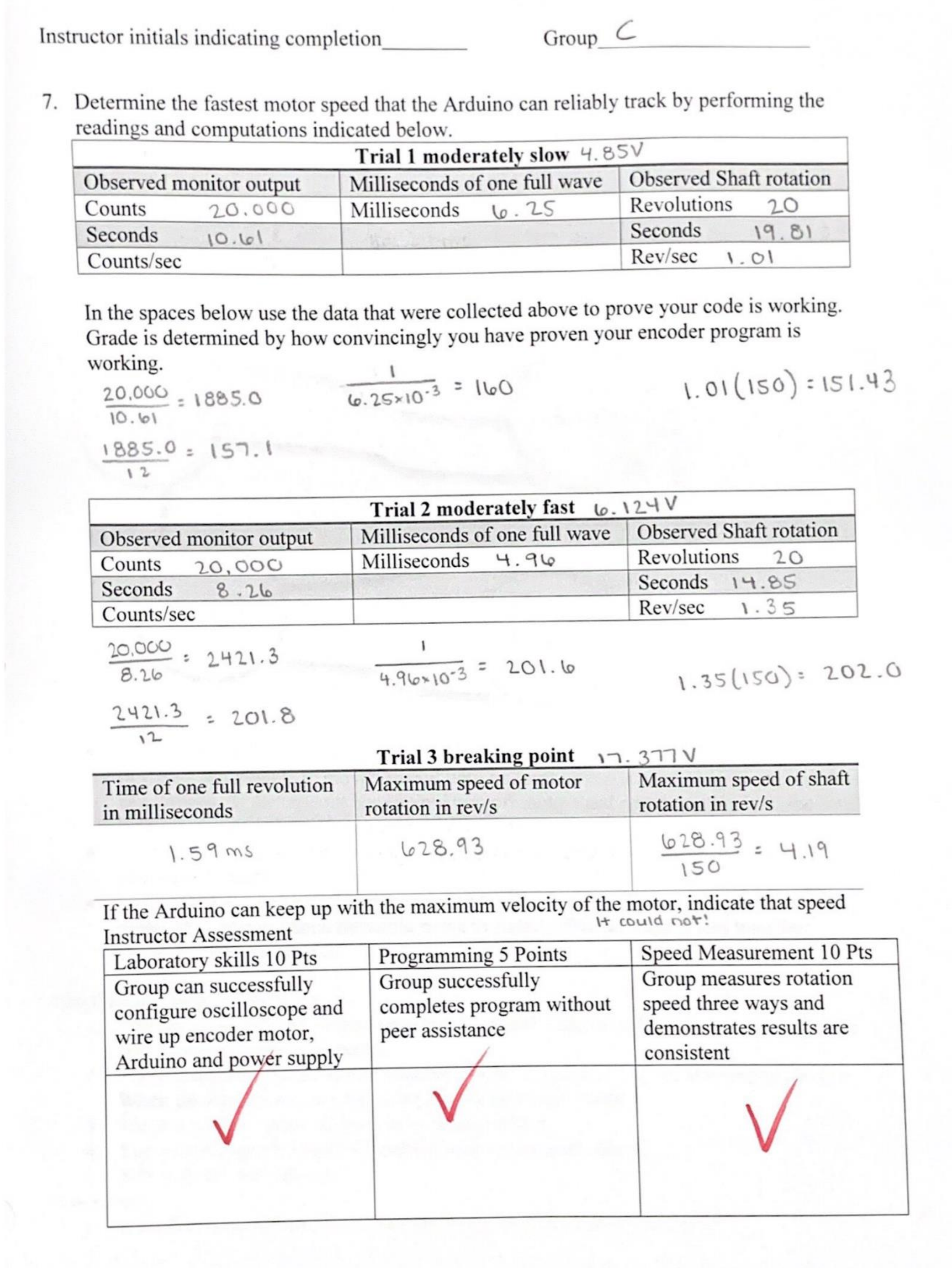
Specifics:

Program is fine, delay(250) takes care of multiple presses

C) Instructor comments on the interview. Has student successfully demonstrated full comprehension of the program he or she has submitted for completion of this assignment?

A truly Excellent job, impressive how you have picked this up.

Partner Labs



Instructor initials indicating completion________ Group C________________

7. Determine the fastest motor speed that the Arduino can reliably track by performing the readings and computations indicated below.

Trial 1 moderately slow 4.85V		
Observed monitor output	Milliseconds of one full wave	Observed Shaft rotation
Counts 20,000	Milliseconds 6.25	Revolutions 20
Seconds 10.61		Seconds 19.81
Counts/sec		Rev/sec 1.01

In the spaces below use the data that were collected above to prove your code is working. Grade is determined by how convincingly you have proven your encoder program is working.

$\frac{20{,}000}{10.61} = 1885.0$ $\frac{1}{6.25\times10^{-3}} = 160$ $1.01(150) = 151.43$

$\frac{1885.0}{12} = 157.1$

Trial 2 moderately fast 6.124V		
Observed monitor output	Milliseconds of one full wave	Observed Shaft rotation
Counts 20,000	Milliseconds 4.96	Revolutions 20
Seconds 8.26		Seconds 14.85
Counts/sec		Rev/sec 1.35

$\frac{20{,}000}{8.26} = 2421.3$ $\frac{1}{4.96\times10^{-3}} = 201.6$ $1.35(150) = 202.0$

$\frac{2421.3}{12} = 201.8$

Trial 3 breaking point 17.377V

Time of one full revolution in milliseconds	Maximum speed of motor rotation in rev/s	Maximum speed of shaft rotation in rev/s
1.59 ms	628.93	$\frac{628.93}{150} = 4.19$

If the Arduino can keep up with the maximum velocity of the motor, indicate that speed It could not!

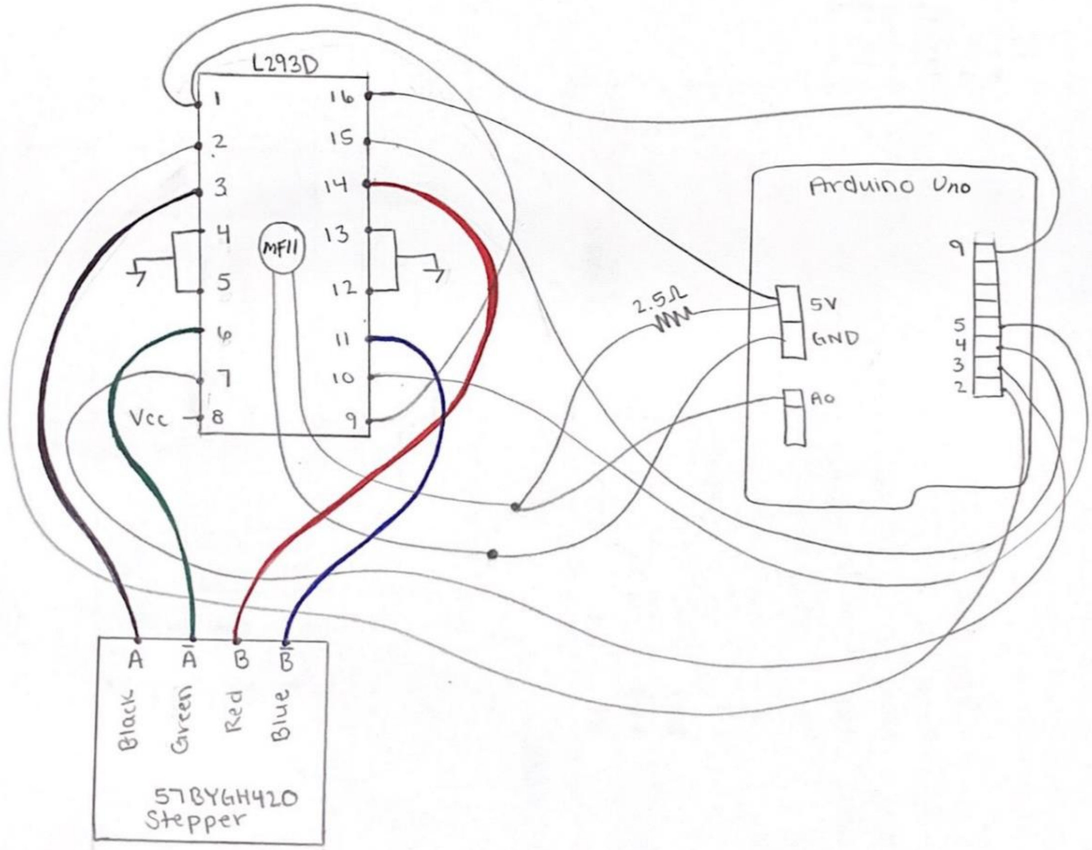
Instructor Assessment

Laboratory skills 10 Pts	Programming 5 Points	Speed Measurement 10 Pts
Group can successfully configure oscilloscope and wire up encoder motor, Arduino and power supply	Group successfully completes program without peer assistance	Group measures rotation speed three ways and demonstrates results are consistent
✓	✓	✓

istructor initials (before Nov 28, 5PM)________ Group C__________________

8. Demonstrate operation to the instructor by reading a number of degrees from the user by serial monitor and rotating the stepper motor that number of **degrees** within the resolution of the stepper. Accept large positive values to allow for multiple CW revolutions and large negative integers to produce CCW rotation. Prompt the user for input on the serial monitor.
9. Answer the following questions through experimentation with the stepper motor:
   a. What is the smallest delay between steps that the motor can tolerate before it either does not move or skips steps when the motor is loaded with bolts closest to the center __7 ms__, loaded with bolts halfway__9 ms__ and loaded with bolts at the end of the slat__9 ms__.
   b. With bolts removed, how fast can the motor move in degrees/s __188.7__? Determine this by timing a number of turns with a stopwatch.
   c. Without actively controlling the temperature, how many seconds does it take for the chip to overheat and how many to cool back to "chip warm" from an overheated temperature? Time to overheat__36.4__s_ Cooldown time.__37.7__s

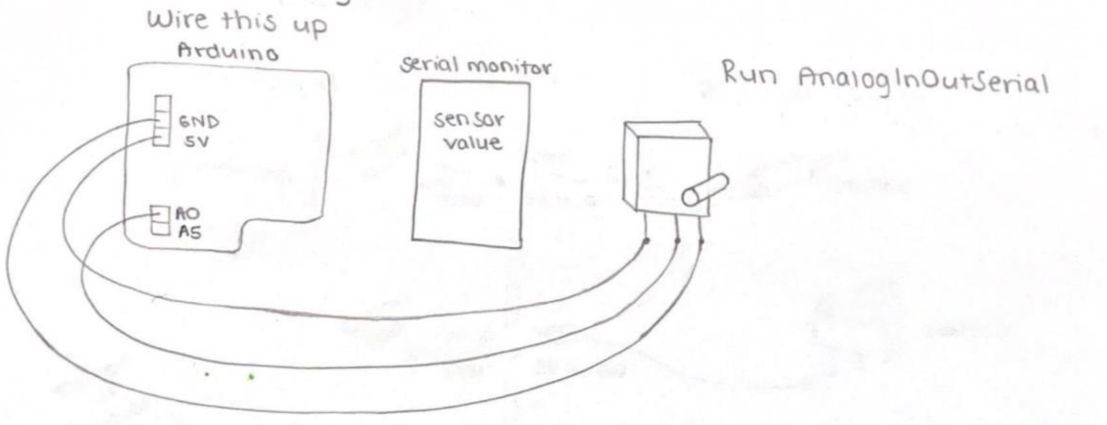
Circuit Diagram (10 Points)



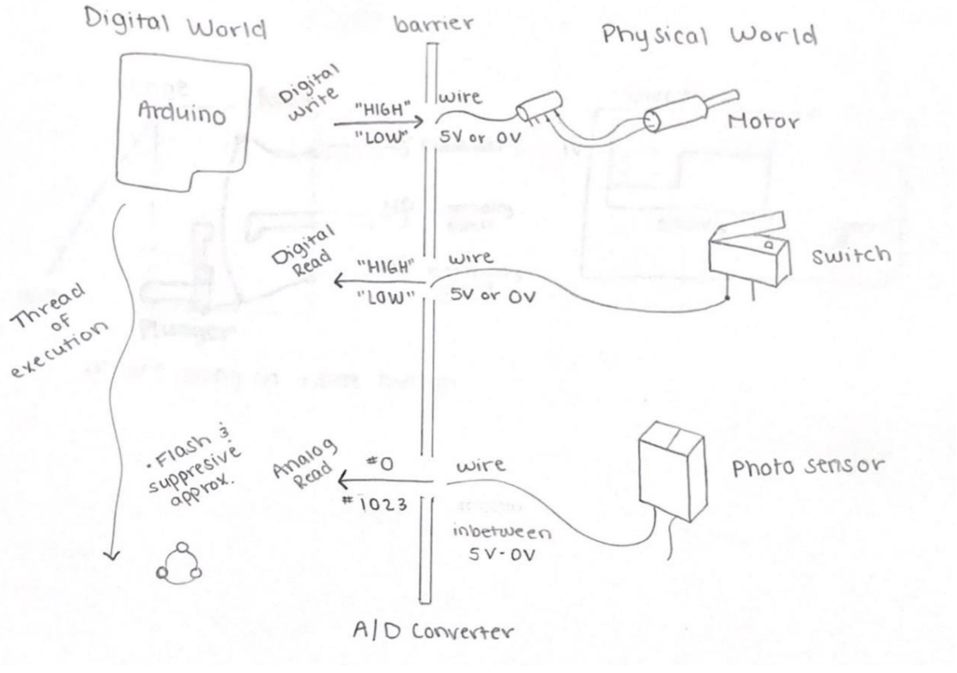
Appendix B

Semester In-Class Notes

9.8.20 Drawing

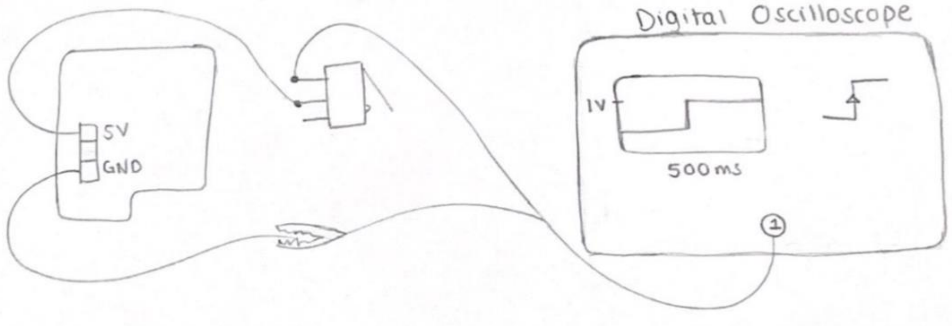


9.10.20 Drawings

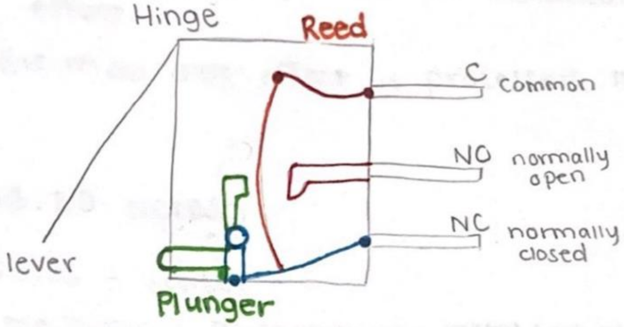


Measuring Contact

Wire this up

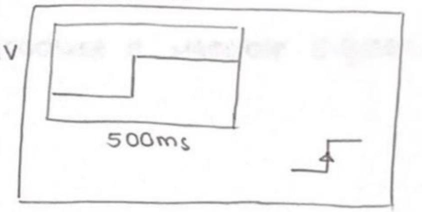


Switch internals



what's going on inside button

Give this



9.10.20 Notes

Sensors

- Physical signal is transduced into an electrical signal.
- Mechanical change has an incidental/accidental electrical side effect.
- Electrical side effect is processed to produce a useable signal.

9.15.20 Notes

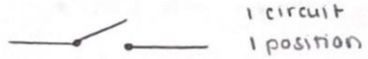
Switches - sensor

- Mechanical Phenomenon: metal bodies touching
- Electrical Side Effect: allows electron flow
- Problem: rough contact

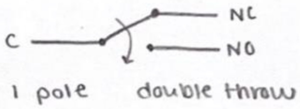
Funny Things about Switches

- 10,000's of them
- Poles: # of circuits switch controls
- Throws: # of Positions Poles Contact

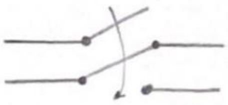
Switches Schematic



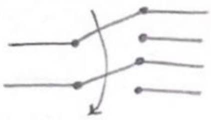
SPST - single pole, single throw switch



SPDT - single pole, double throw



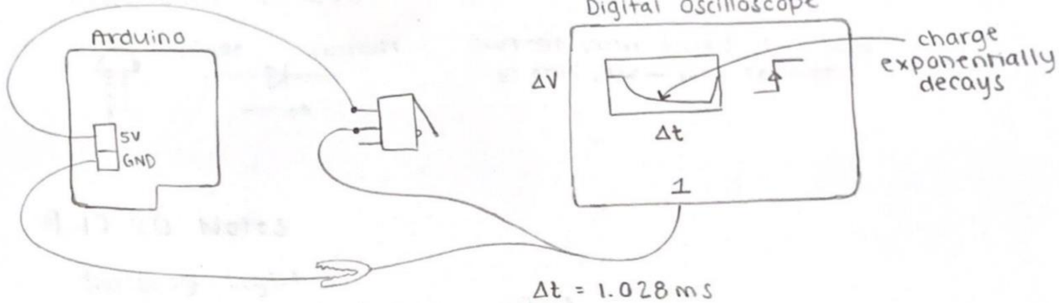
DPST



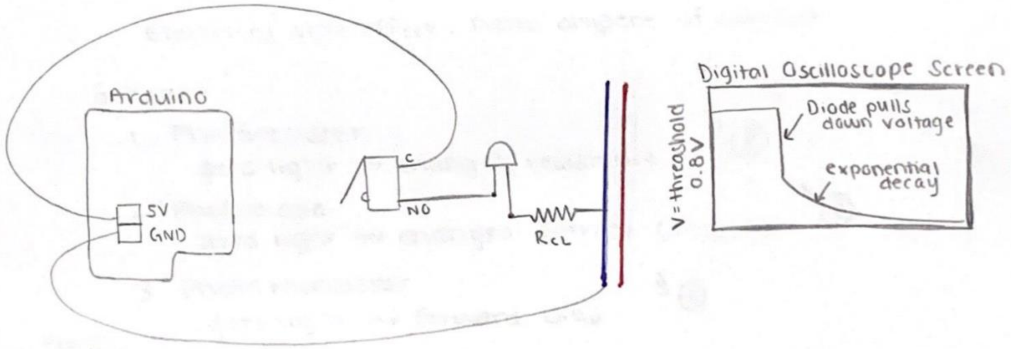
DPDT

9.15.20 Drawings

wire this up



$\Delta t = 1.028\,ms$
$\Delta V = 5.03\,V$

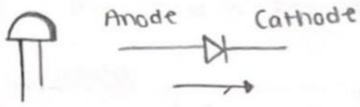


* limit current through diode

* need R_{CL} → ohm's law → $R = \frac{V}{I} = \frac{5V}{0.015A} = 333\,\Omega$

↑from spreadsheet

Fun with Diodes



current only flows one way in this case — left to right

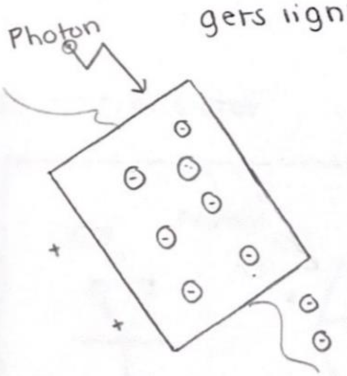
9.17.20 Notes

Sensing Light

- All work on photoelectric effect
- Mechanical phenomenon: momentum transfer from photon to electron.
- Electrical side effect: nano ampere of current

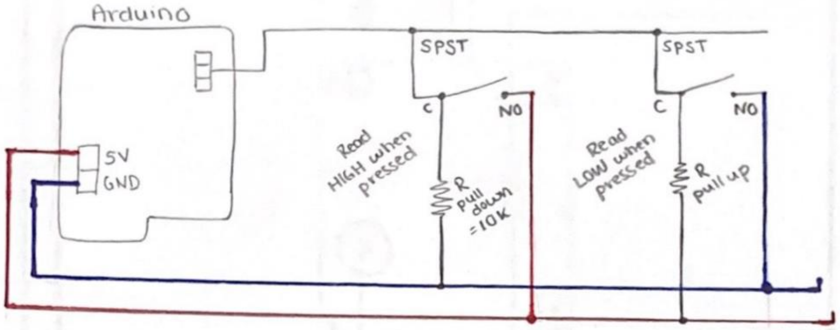
3 types

1. Photoresistor
   gets light → changes resistance
2. Photodiode
   gets light → changes current (on/off)
3. Phototransistor
   gets light → forward bias



9.17.20 Drawings

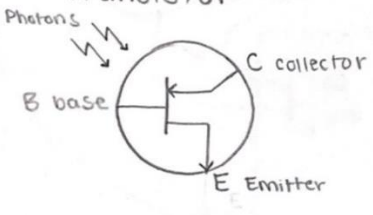
Pull Up & Pull Down Resistors



*if hold button down, there will be loss

*avoid reading nothing

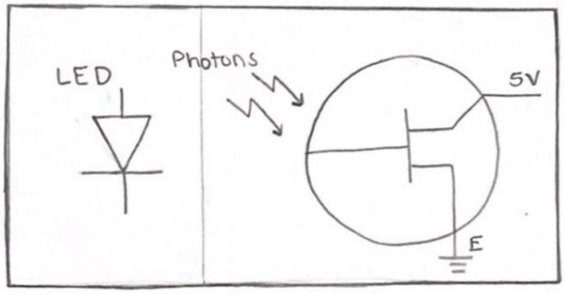
Transistor



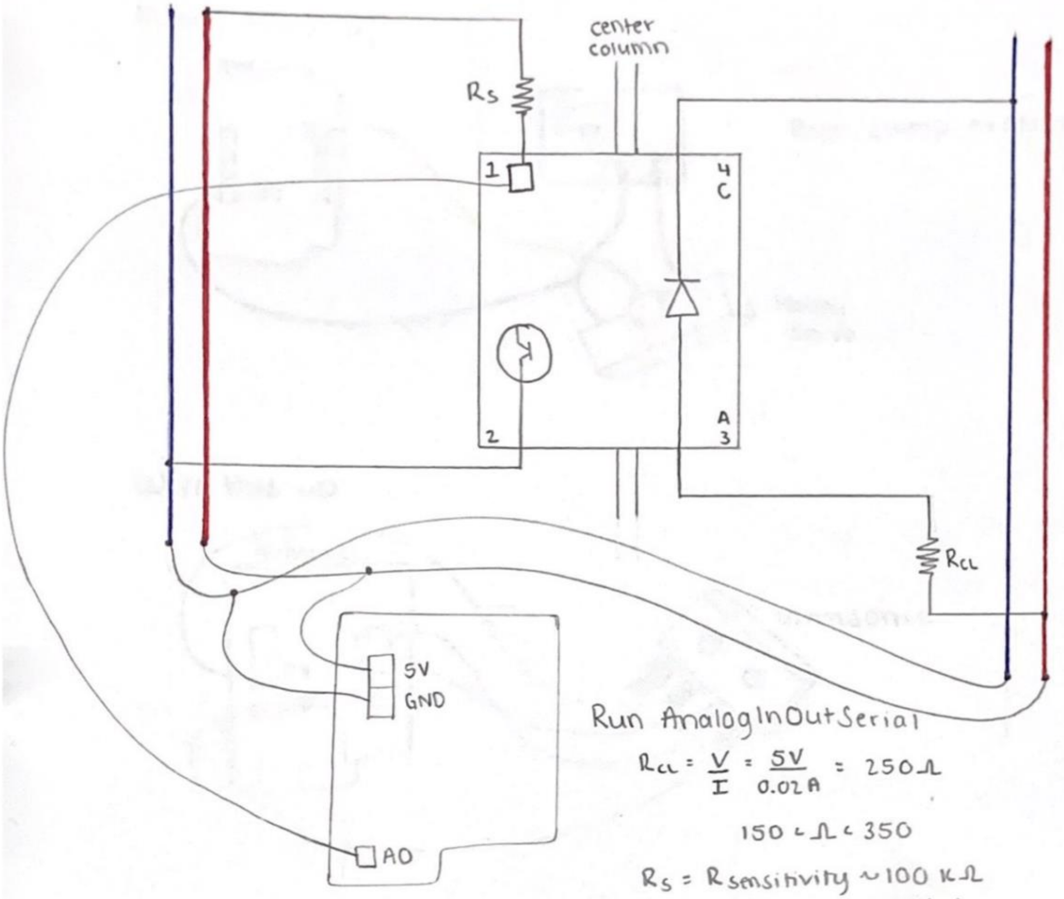
$i_{CE} = h \, i_{BE}$

Gain 50

Phototransistor



Wire this Up



Run AnalogInOutSerial

$$R_{cl} = \frac{V}{I} = \frac{5V}{0.02A} = 250\Omega$$

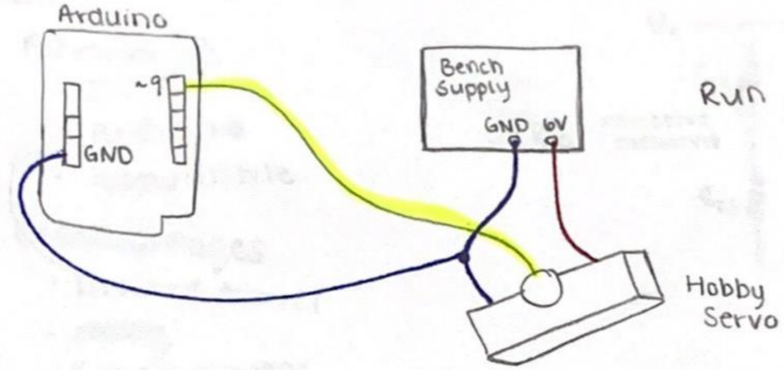
$$150 < \Omega < 350$$

$$R_s = R_{sensitivity} \sim 100\,k\Omega$$

*dependent on robot

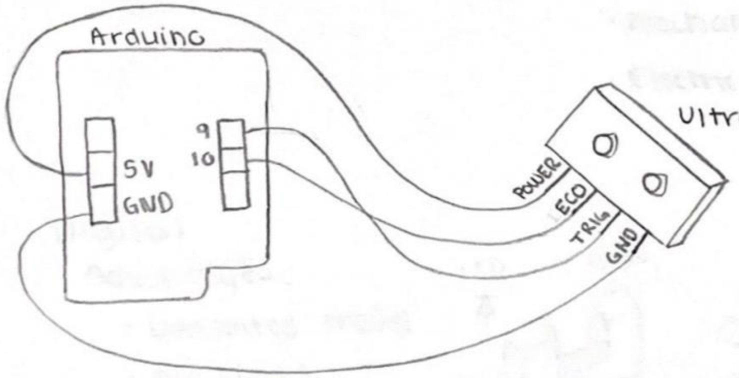
9.22.20 Drawings

Wire this up



Run sweep example

Wire this up



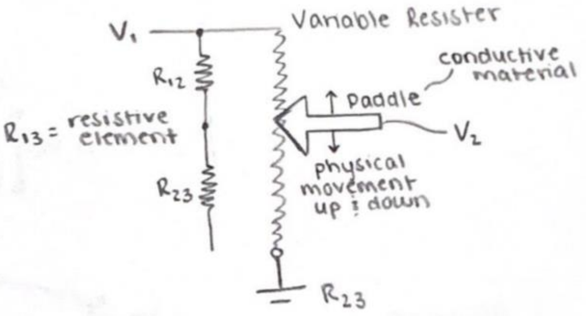
9.24.20 Notes

Sensing Rotation

Analog

Advantages
- Easy
- Absolute
- Nonvolitile

Disadvantages
- limited travel
- Noisy
- Subject to wear



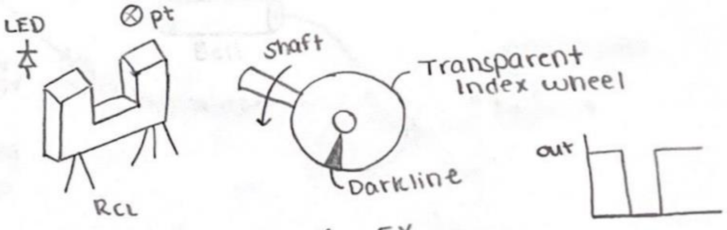
$$V_2 = V_1 - \frac{R_{23}}{R_{13}}$$

- Mechanical change: position of paddle
- Electrical side effect: change of resistance: $R_{23} \rightarrow V_2$ changes

Digital

Advantages
- Unlimited travel
- No Noise
- High Resolution

Disadvantages
- Volitile
- Relative position?
  - fix w/ Gray or index
- Complex
- Can't tell CW from CCW
  - Don't care
  - Quadrature

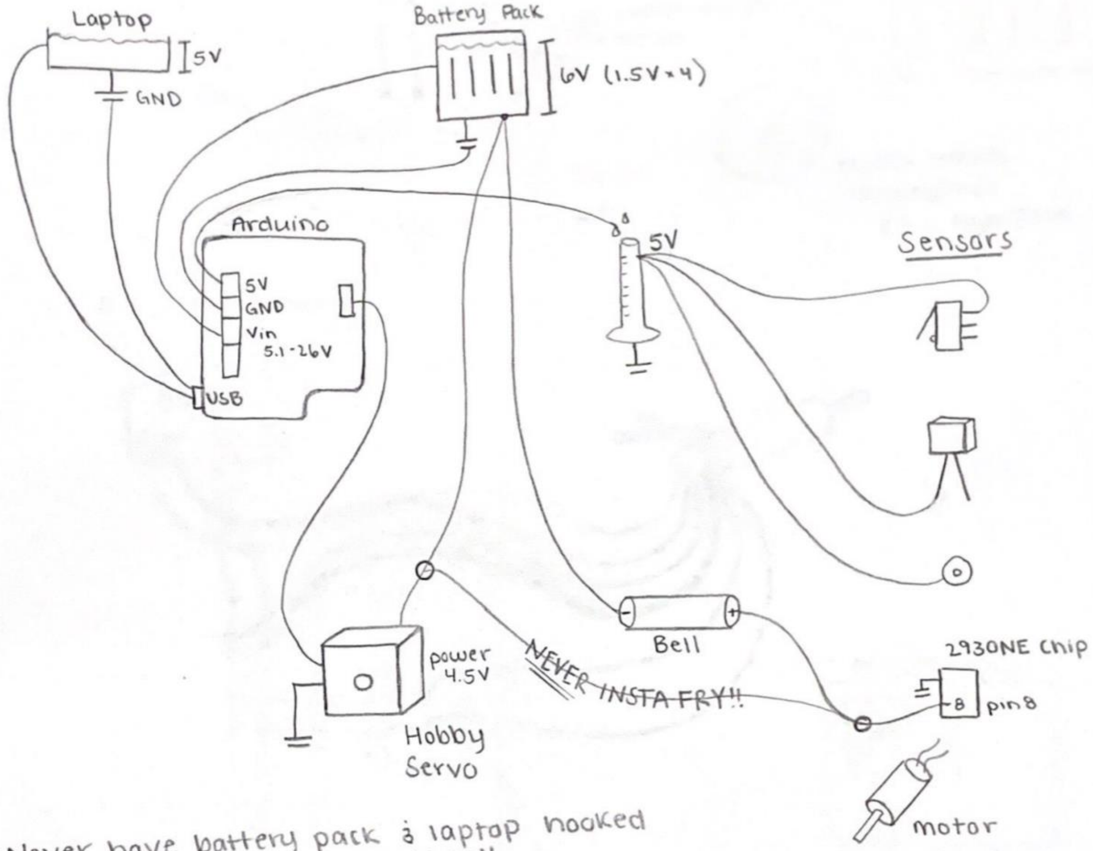


$R_{CL} = \frac{V}{I} = \frac{5V}{0.005A}$

$= 1 k\Omega$

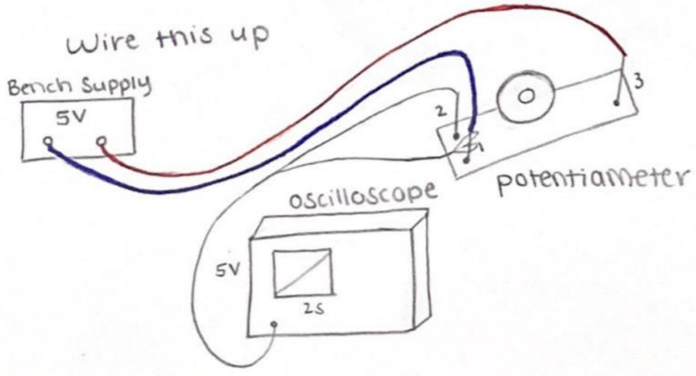
9.24.20 Drawings

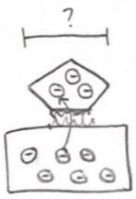
#1 most important thing – segregate power supplies!!
3 power supplies



- Never have battery pack & laptop hooked up at the same time!!
- Servo power needs to go directly to battery pack
- Never connect Hobby Servo to battery bell

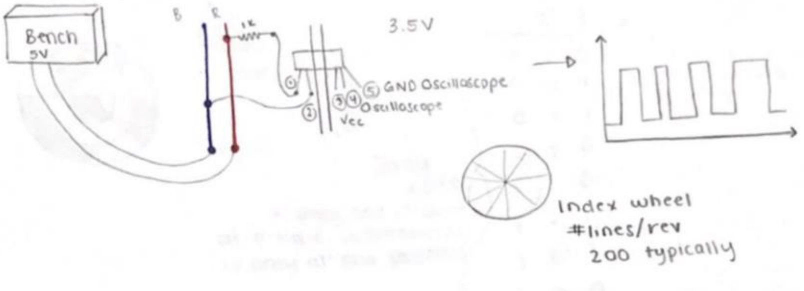
Wire this up



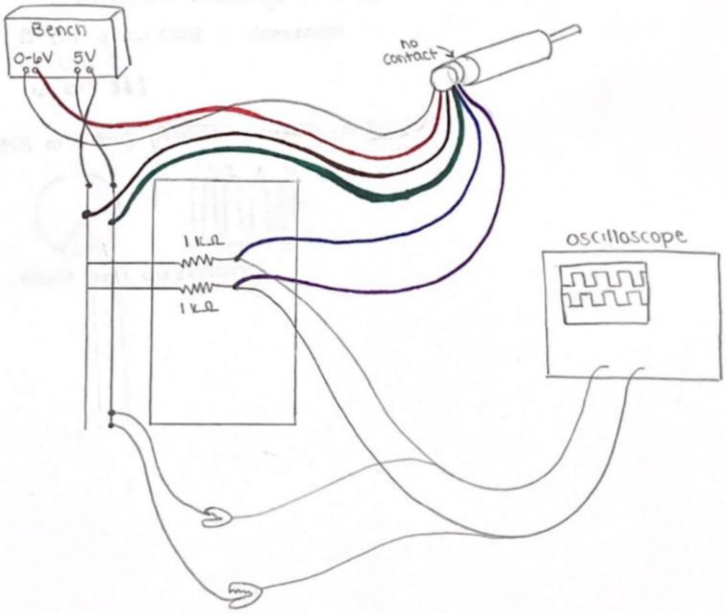


9/29/20 Drawings

Wire this up



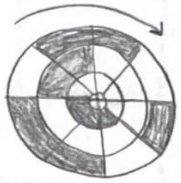
10/1/20 Drawings



10.1.20 Notes

Absolute encoder

ex:



white = 0
black = 1

1	2	3
0	0	0
0	0	1
0	1	1
0	1	0
1	1	0
1	1	1
1	0	1
1	0	0

list of values that sensors pick up

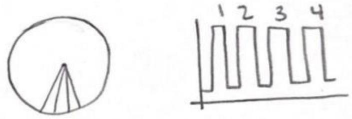
Gray code

* only one changes at a time, uncertainty is only at one position

Resolution: For 360°, there are 8 different readings → 45°

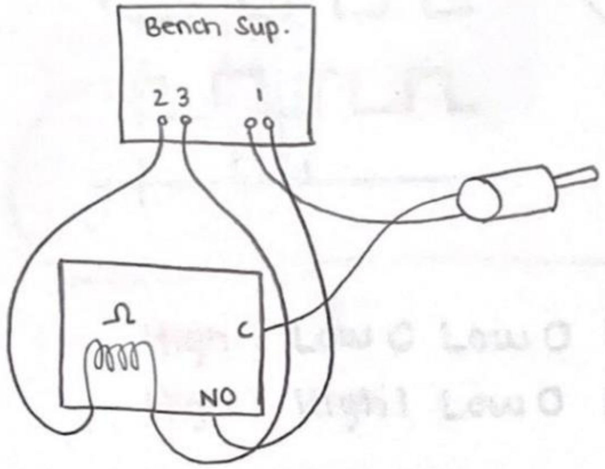
8-bit gray code is common
12-bit $
16-bit $$$

* Still one big problem with digital

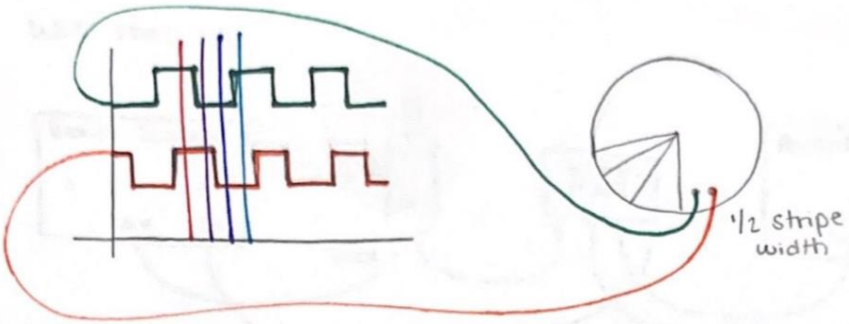


can't tell direction

9/6/20 Drawings:

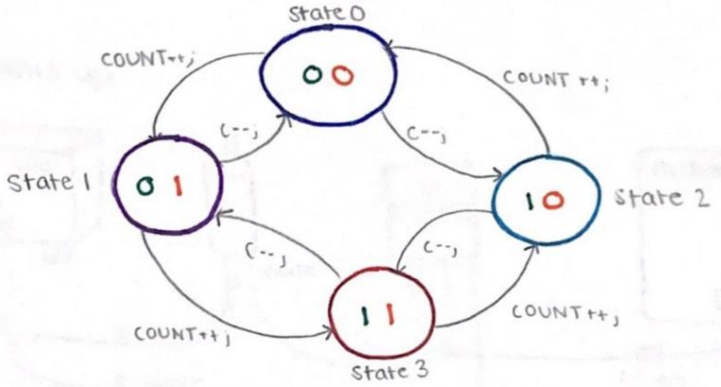


9/6/20 Notes:



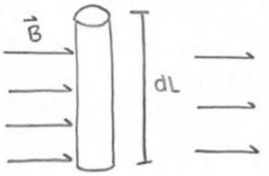
High 1 Low 0 Low 0 High 1

High 1 High 1 Low 0 Low 0



* can't go from 0-3 or 1-2 If it does, it's BROKEN!!

Actuators Basics



what happens?

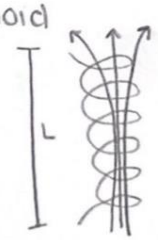
gets force

Lorenz Force Law

$\vec{F} = i\, dL \times \vec{B}$

*direction matters

Solenoid



magnetic fountain

$$\vec{B} = \frac{\mu i N}{L}$$

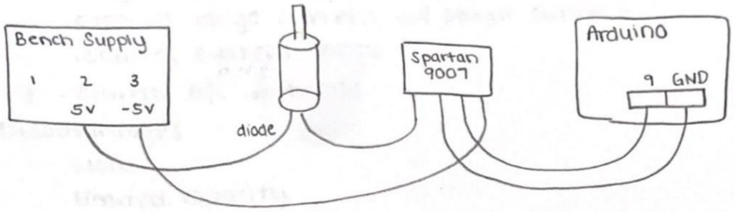
N = # of coils

magnetic permeability

$\mu \sim 1$ vacuum $\mu \sim 20{,}000$ Iron

10|8|20 Drawings:

Wire this up

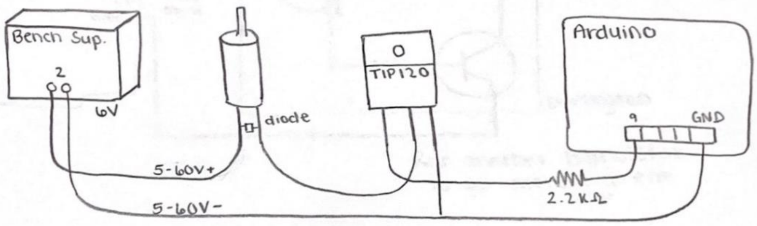


Run blink example

$V = L \frac{di}{dt}$

clipping diode

Wire this up



10|8|20 Notes:

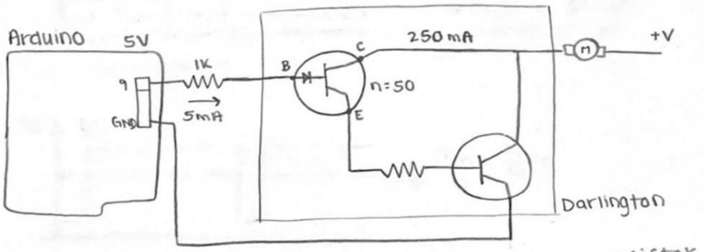
Relays:

Advantages

- controls large current w/ small current
- isolates current loops
- switch A/C with D/C

Disadvantages

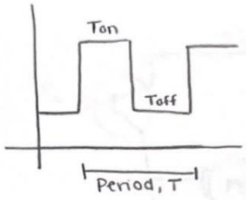
- Slow
- limited capacity
- Subject to wear
- Noisy



Run another transistor to get more current.

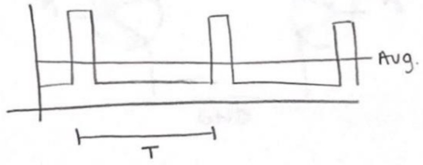
10/13/20 Notes:

Pulse With Modulation (PWM)



$\frac{T_{on}}{T}$ = Duty Cycle (DC)

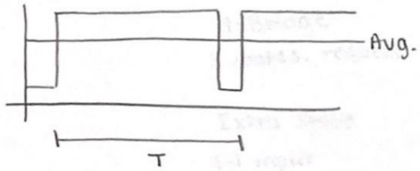
10% DC



$T_{on} = \frac{1}{10} T$

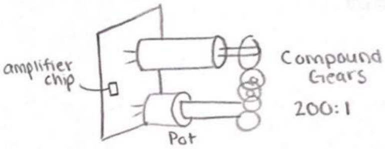
* Physical devices average out

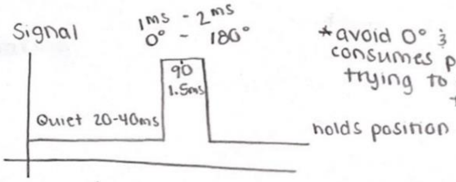
90% DC



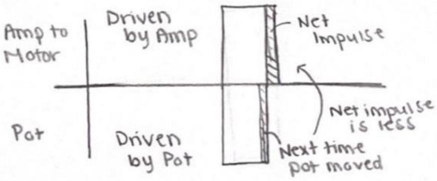
$T_{on} = \frac{9}{10} T$

Hobby Servo? Kinda PWM



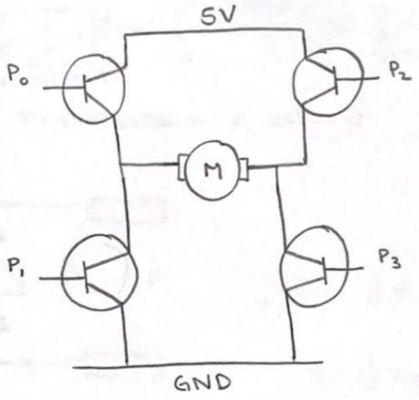


*avoid 0° & 180°, consumes power trying to get there.



* or motor moves the other way until they are equal.

Try this



P_3	P_2	P_1	P_0	Result
0	0	0	0	Stop
0	0	0	1	Stop
0	0	1	0	Stop
0	0	1	1	Smoke
0	1	0	0	Stop
0	1	0	1	??
0	1	1	0	CW
0	1	1	1	Smoke
1	0	0	0	Stop
1	0	0	1	CCW
1	0	1	0	Really Stop
1	0	1	1	Smoke
1	1	0	0	Smoke
1	1	0	1	Smoke
1	1	1	0	Smoke
1	1	1	1	Smoke

H-Bridge

- 3 states, requres 2 digital lines

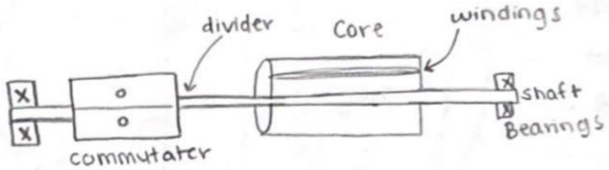
Extra State

- 1-1 input
  - Brake
  - Dynamic braking
  - Coast

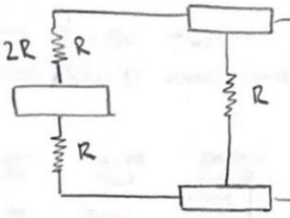
Notes / Drawings 10.22.20

Motors::

Background Info



Find Resistance of winding

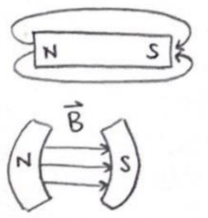


$R_T = \frac{2}{3}R$

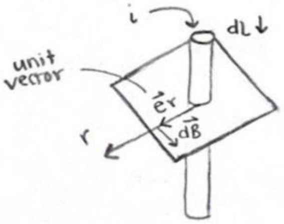
$R = \frac{3}{2}R_T$

$\frac{1}{R_T} = \frac{1}{R} + \frac{1}{2}R = \frac{3}{2}R$

What is inside?



Background Magnetic Principles



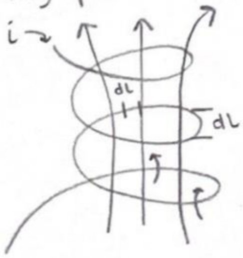
$\vec{dB} = \frac{i\mu\vec{dL}\times\vec{er}}{4\pi r^2}$

μ = mag permetivity

Iron ~ 22,000

Vacuum ~ 1

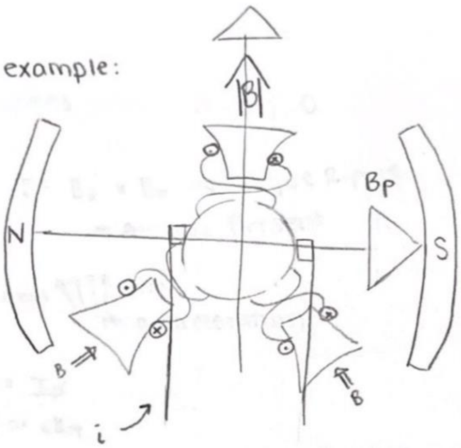
Every part of the wire contributes the same direction



$B = \frac{\mu N i}{L}$

- Moving away from you on right, coming towards you on left
- Criss-crossed, then goes out making a circuit
- B goes up & toward center
- right hand rule

example:



- Little B's are smaller because less i went through it
- Net magnetic field = up
- Honzontal components cancel out
- Both B_T & B_p create a torque, spins CW.

What is an event?

The brush switching from one commentator ring to the next

	1	2	3
I	small out	small out	Big in
II	Big out	small in	small in
III	small out	big in	small out
IV	small in	small in	big out

after each event, a new drawing was made.
The resultant was still in the same direction, as well as the magnetic field, therefore it continued to turn CW.

1) $\tau = B_R \times B_P \Rightarrow$ Torque Ripple

* Always Present

min ↑↑↑↑ max

Ang acceleration

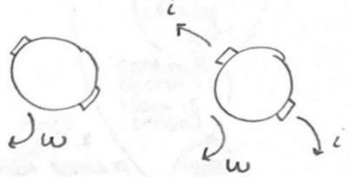
2) $\tau = I\alpha$

$\tau \sim i k_T$

$\alpha \sim \frac{i k_T}{I_{mom}}$? NO!

* spins at a constant speed

* Kind of "Terminal Velocity" after <u>very</u> short time

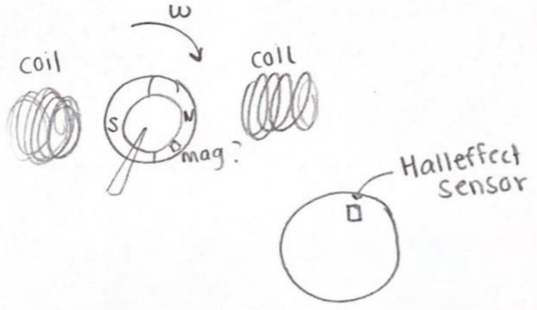


Back EMF
Opposes Applied Voltage

1. Brushed DC PM Motor
   a. advantages
      i. simple
   b. disadvantages
      i. brushes wear out
      ii. Noisy
         - sharply reversing mag. field
         - conducting all power through moving surface
2. Brushless DC PM Motor
   a. Move the magnet not the brushes?

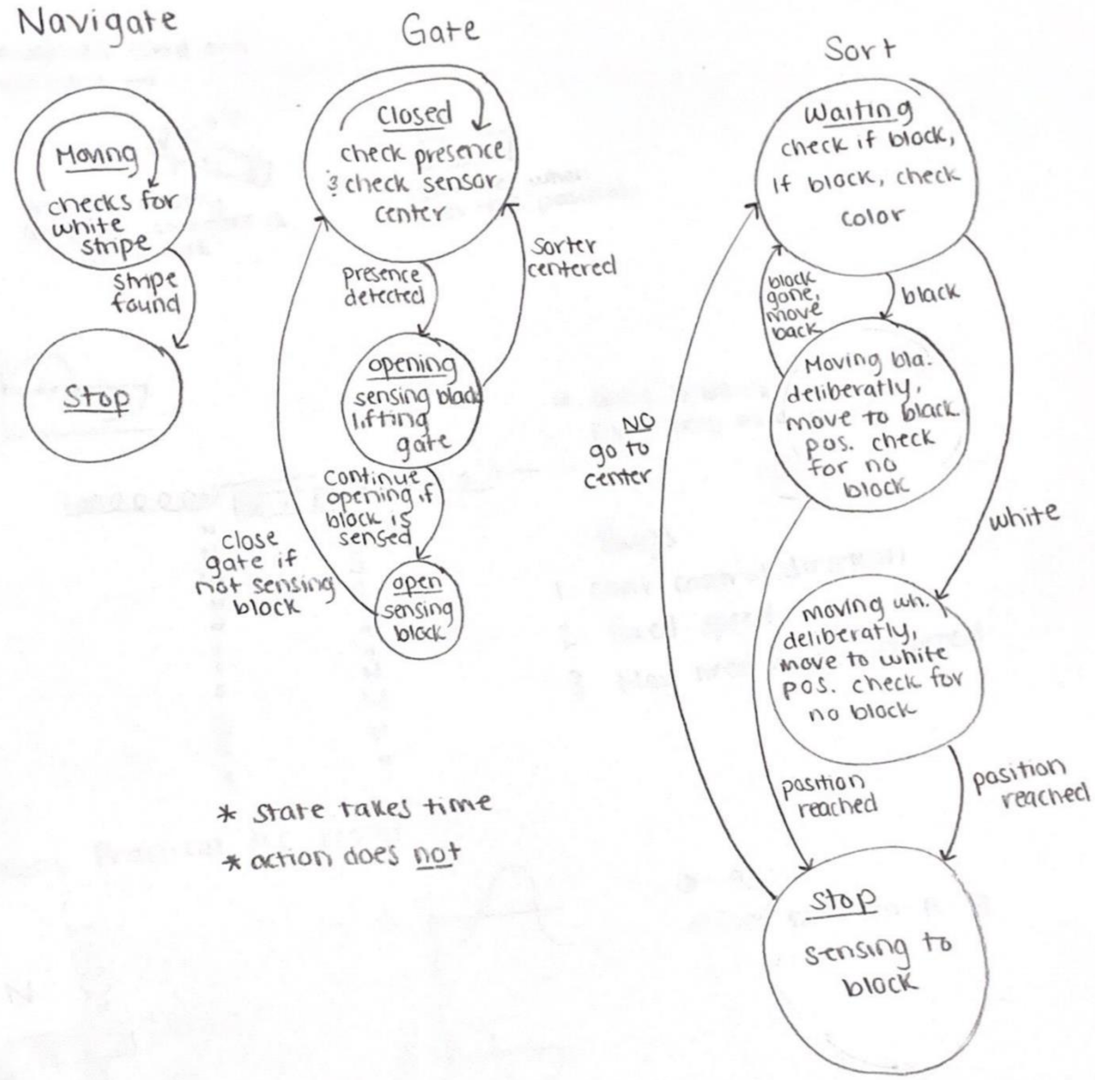
Brushless DC PM Motor

* Need Communication Control



Notes: 10/29/20

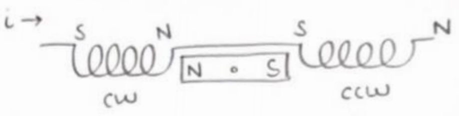
In Class Coding Lab 2


Navigate
Moving
checks for white stripe
stripe found
Stop
Gate
Closed
check presence & check sensor center
presence detected
Sorter centered
opening
sensing block lifting gate
continue opening if block is sensed
open
sensing block
close gate if not sensing block
Sort
Waiting
check if block, if block, check color
black
block gone, move back
Moving bla. deliberatly, move to black pos. check for no block
white
Moving wh. deliberatly, move to white pos. check for no block
position reached
position reached
Stop
sensing to block
NO go to center

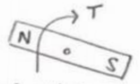
* state takes time

* action does not

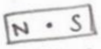
Notes: 11.5.20



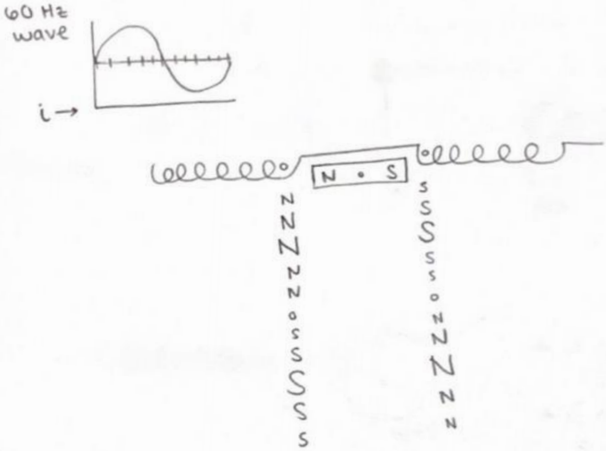
magnetic field ←
current →



* not perfectly alligned, so there is a Torque



*switches when in this position

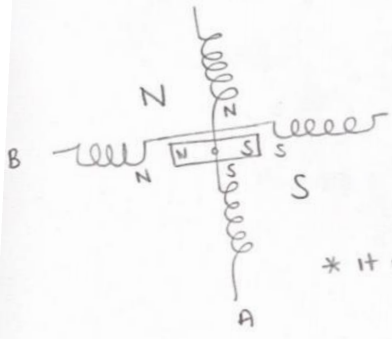


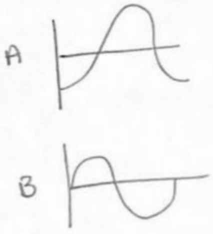
* spins @ 60 Hz same freq. as current

Bugs

1. can't control direction
2. fixed speed
3. May need to get started

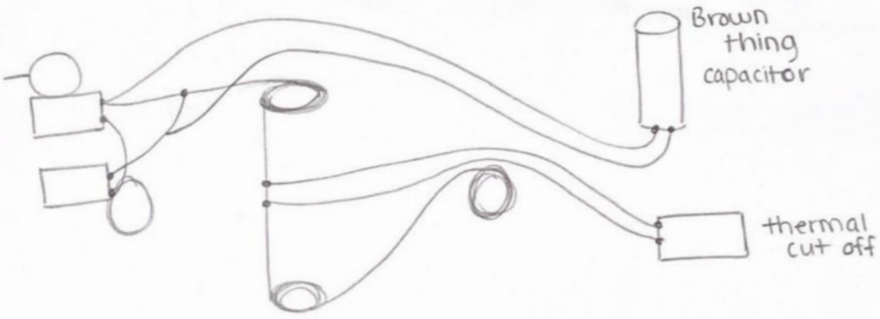
Minimum Practical AC Motor



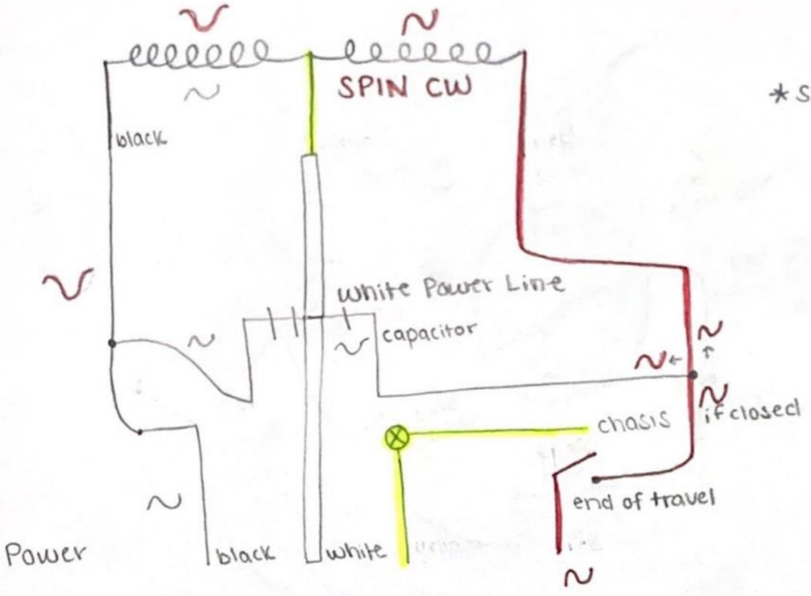


Φ = 90°
offset between A & B

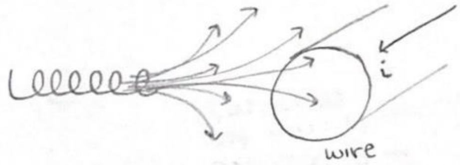
* it will want to move ccw



Notes/Drawings 11/10/20

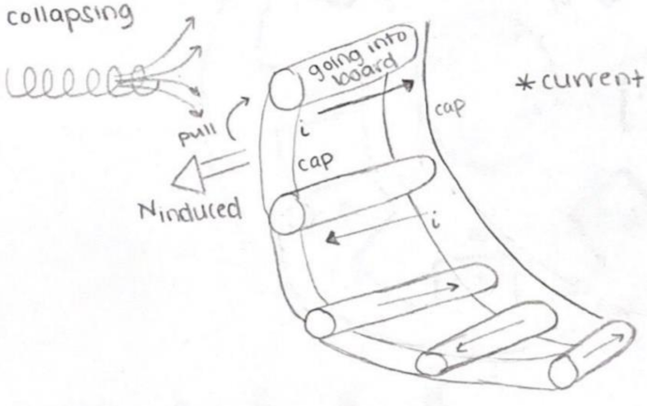


* spins cw until switch opens, then send to black switch, then spins ccw.



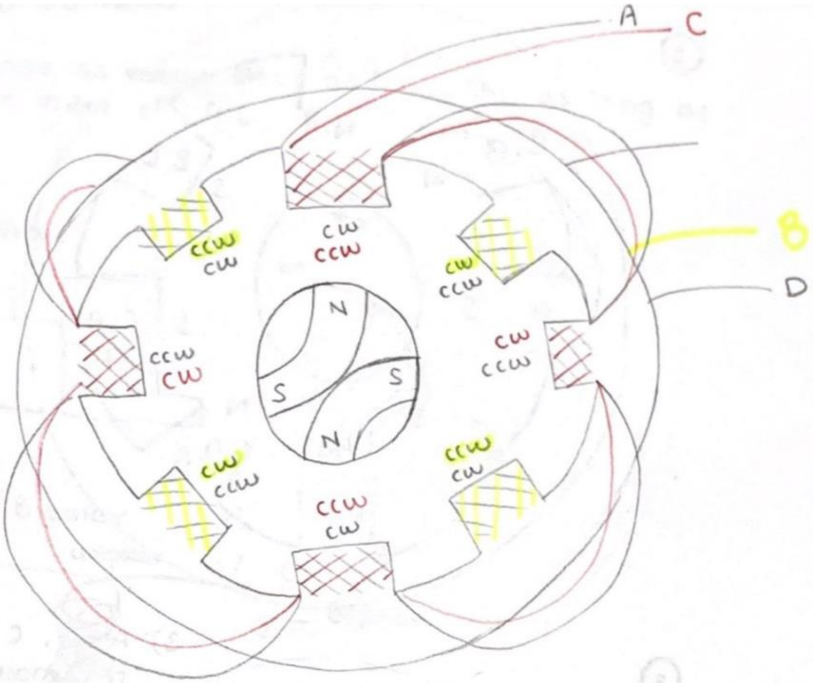
* if getting bigger, current goes out of board
* if getting smaller, current goes into board (RHR)

Induction Motor

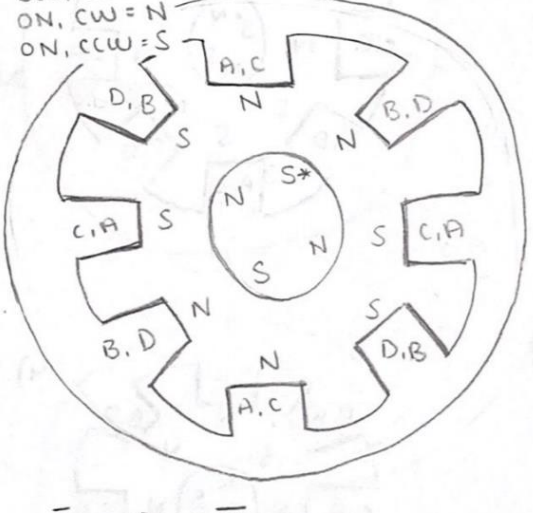


* current has to be a loop

* No perm. magnet
* Only works because field is decaying
  - Field passes Rotor Cage
    - Slip
* Self-controlling



①



CW, CCW
ON, CW = N
ON, CCW = S

1) Activate A, B

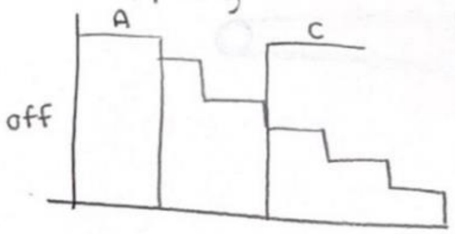
*happy in this position, & only shifted 180°

Sequence	A	$\bar{A}$	B	$\bar{B}$
φ1	1	0	1	0
φ2	1	0	0	1
φ3	0	1	0	1
φ4	0	1	1	0

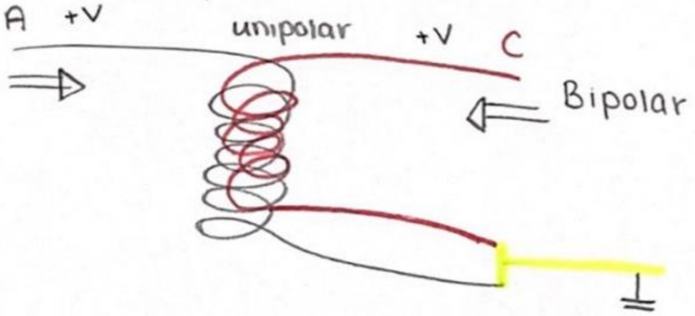
Notes: 11/12/20

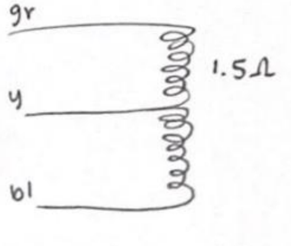
1. * Can have as many steps as it wants, as long as it's in order of A,C ; B,D ; C,A ; D,B

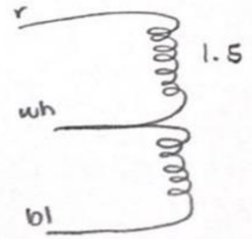
2. Microstepping



3. Unipolar / Bipolar

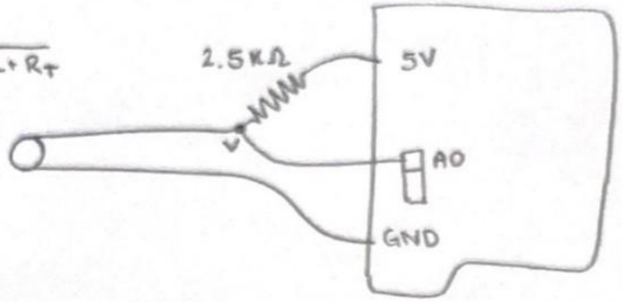






Stepper Motor

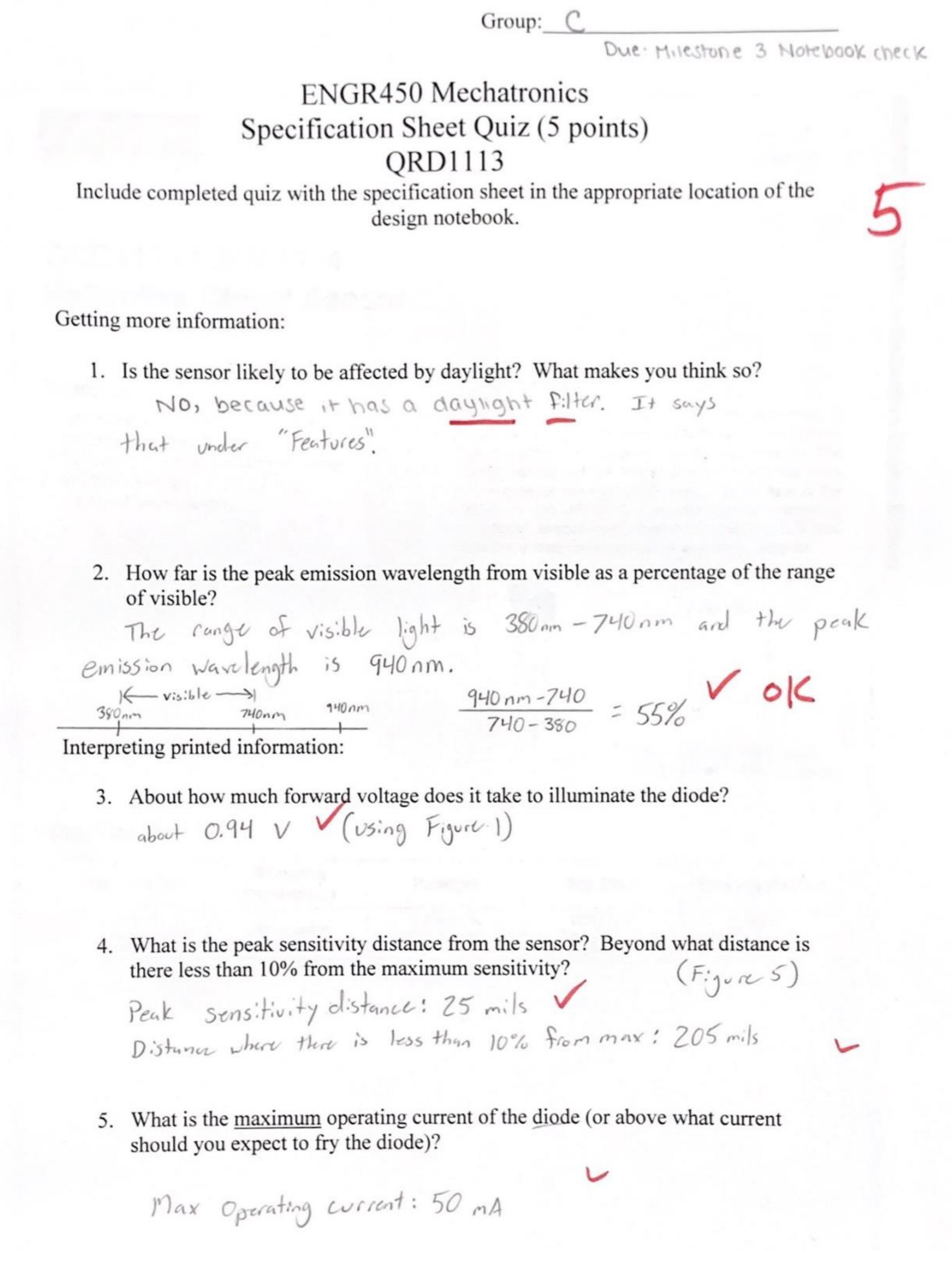
$$V = \frac{R_T}{2.5K\Omega + R_T}$$



$$P = IV = \frac{VV}{R} = \frac{V^2}{R} = \frac{(5V)^2}{275} = \sim 0.1W$$

Appendix C

Quizzes



Group: C

Due: Milestone 3 Notebook check

ENGR450 Mechatronics
Specification Sheet Quiz (5 points)
QRD1113

Include completed quiz with the specification sheet in the appropriate location of the design notebook.

5

Getting more information:

1. Is the sensor likely to be affected by daylight? What makes you think so?

   No, because it has a daylight filter. It says that under "Features".

2. How far is the peak emission wavelength from visible as a percentage of the range of visible?

   The range of visible light is 380nm – 740nm and the peak emission wavelength is 940nm.

   |← visible →|
   380nm 740nm 940nm

   $$\frac{940\,\text{nm} - 740}{740 - 380} = 55\%$$ ✓ OK

Interpreting printed information:

3. About how much forward voltage does it take to illuminate the diode?

   about 0.94 V ✓ (using Figure 1)

4. What is the peak sensitivity distance from the sensor? Beyond what distance is there less than 10% from the maximum sensitivity?

   (Figure 5)

   Peak sensitivity distance: 25 mils ✓

   Distance where there is less than 10% from max: 205 mils ✓

5. What is the maximum operating current of the diode (or above what current should you expect to fry the diode)?

   ✓

   Max Operating current: 50 mA

Group: C

due w/ Milestone 4 Report

ENGR450 Mechatronics
Specification Sheet Quiz (5 points)
9000 Spartan

Include completed quiz with the specification sheet in the appropriate location of the design notebook.

Getting more information:

1. What is meant by "SIP Relay" on the specification sheet (the "P" stands for "Package")?

   Single Inline Package ✓

2. Note 4 states that this option is equipped with a "56V Zener diode"; what is a Zener diode and what is its typical function in a circuit?

   Zener diode is a special type of diode designed to allow current to flow "backwards" when a certain set reverse voltage (Zener voltage) is reached. It is used for voltage regulation, as reference elements, surge suppressors, and in switching applications and clipper circuits. ✓

Interpreting printed information:

3. What would I be getting if I were to order a 9007-12-11?

   Model 9007, 12 means that the coil voltage is 12, The second to last numbers (1) shows that it has an external magnetic sheild, and the last number (1) shows that there is a diode that is connected to pins 2 and 3. ✓

4. How recent is the specification sheet, i.e. when was its last revision?

   Revised on January 2010 ✓

5. Using terms from the in-class discussion of switches, what is the primary difference between the 9081 and the 9081C?

   The 9081 is SPST
   The 9081C is SPDT ✓

Group: C

due w/ Milestone 4 Report

ENGR450 Mechatronics
Specification Sheet Quiz (5 points)
TIP 120

Include completed quiz with the specification sheet in the appropriate location of the design notebook.

5

Getting more information:

1. What is meant by the TIP120 being "complimentary" to the TIP125?

   They have near identical characteristics, and have matched pair silicon power transistors. The TIP120 is NPN, and the TIP125 is PNP. ✓ ✓

2. What does the Ⓜ (circled M) mean on the mechanical drawing? ✓

   It is the maximum matenal condition.

Interpreting printed information:

3. At a DC current of .12 A, what is the largest safe collector-emitter voltage? ✓

   60V → from figure 4

4. At V_{CE}=4V, what is the expected DC current gain when the collector current is 0.8A? ✓

   about 2900, from figure 1

5. Is a base current of 150mA within the safe operating range of the device? If not, how much above? If yes, how much below?

   It is not within the safe operating range, it is 30 mA above ✓

Group: C

due w/ Milestone 5

ENGR450 Mechatronics
Specification Sheet Quiz (5 points)
L293D

5

Include completed quiz with the specification sheet in the appropriate location of the design notebook.

Getting more information:

1. What is meant by referring to the unit as a "complete totem-pole drive circuit"?

   A type of output structure used w/ integrated circuits in which one transistor drives the output high while another transistor connected below it pulls the output low. ✓

   * Brackets stacked on eachother

2. What does having the outputs "in the high-impedance state" mean?

   Outputs is not driven by inputs, output is neither high (1) nor low (0). ✓

Interpreting printed information:

3. What are the minimum and maximum recommended voltages on pin 16? ✓

   min = 4.5 V

   max = 7 V

4. What is the typical high to low propagation time of the L293DNE?

   400 ns

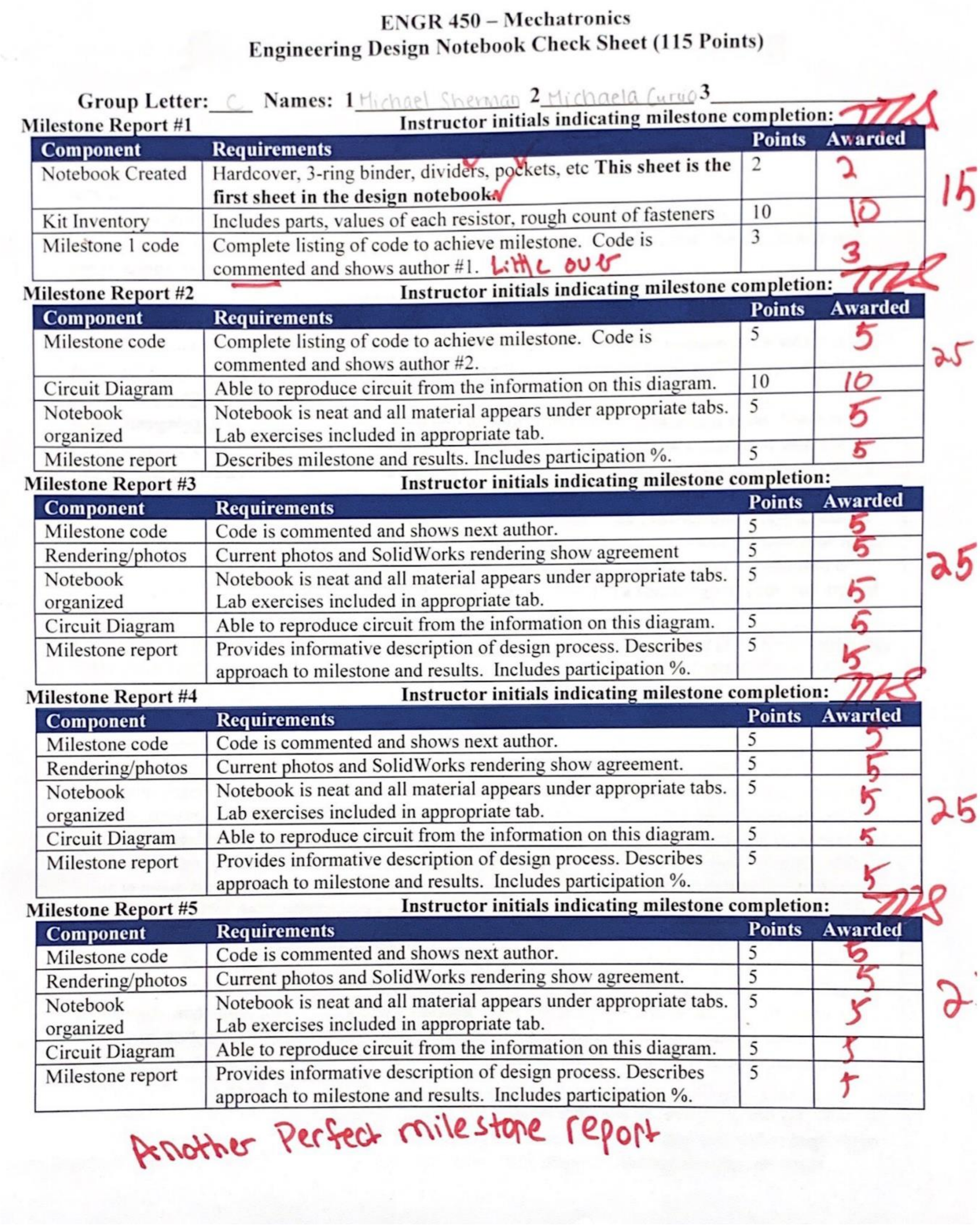
   ✓

5. At 70°C in the free air, how much power can the L293D safely dissipate?

   ~1 W

Appendix D

Milestone Notebook Check Sheet



ENGR 450 – Mechatronics

Engineering Design Notebook Check Sheet (115 Points)

Group Letter: C Names: 1 Michael Sherman 2 Michaela Curuo 3 ____

Milestone Report #1 Instructor initials indicating milestone completion:

Component	Requirements	Points	Awarded
Notebook Created	Hardcover, 3-ring binder, dividers, pockets, etc **This sheet is the first sheet in the design notebook.**	2	2
Kit Inventory	Includes parts, values of each resistor, rough count of fasteners	10	10
Milestone 1 code	Complete listing of code to achieve milestone. Code is commented and shows author #1. Little out	3	3

15

Milestone Report #2 Instructor initials indicating milestone completion:

Component	Requirements	Points	Awarded
Milestone code	Complete listing of code to achieve milestone. Code is commented and shows author #2.	5	5
Circuit Diagram	Able to reproduce circuit from the information on this diagram.	10	10
Notebook organized	Notebook is neat and all material appears under appropriate tabs. Lab exercises included in appropriate tab.	5	5
Milestone report	Describes milestone and results. Includes participation %.	5	5

25

Milestone Report #3 Instructor initials indicating milestone completion:

Component	Requirements	Points	Awarded
Milestone code	Code is commented and shows next author.	5	5
Rendering/photos	Current photos and SolidWorks rendering show agreement	5	5
Notebook organized	Notebook is neat and all material appears under appropriate tabs. Lab exercises included in appropriate tab.	5	5
Circuit Diagram	Able to reproduce circuit from the information on this diagram.	5	5
Milestone report	Provides informative description of design process. Describes approach to milestone and results. Includes participation %.	5	5

25

Milestone Report #4 Instructor initials indicating milestone completion:

Component	Requirements	Points	Awarded
Milestone code	Code is commented and shows next author.	5	5
Rendering/photos	Current photos and SolidWorks rendering show agreement.	5	5
Notebook organized	Notebook is neat and all material appears under appropriate tabs. Lab exercises included in appropriate tab.	5	5
Circuit Diagram	Able to reproduce circuit from the information on this diagram.	5	5
Milestone report	Provides informative description of design process. Describes approach to milestone and results. Includes participation %.	5	5

25

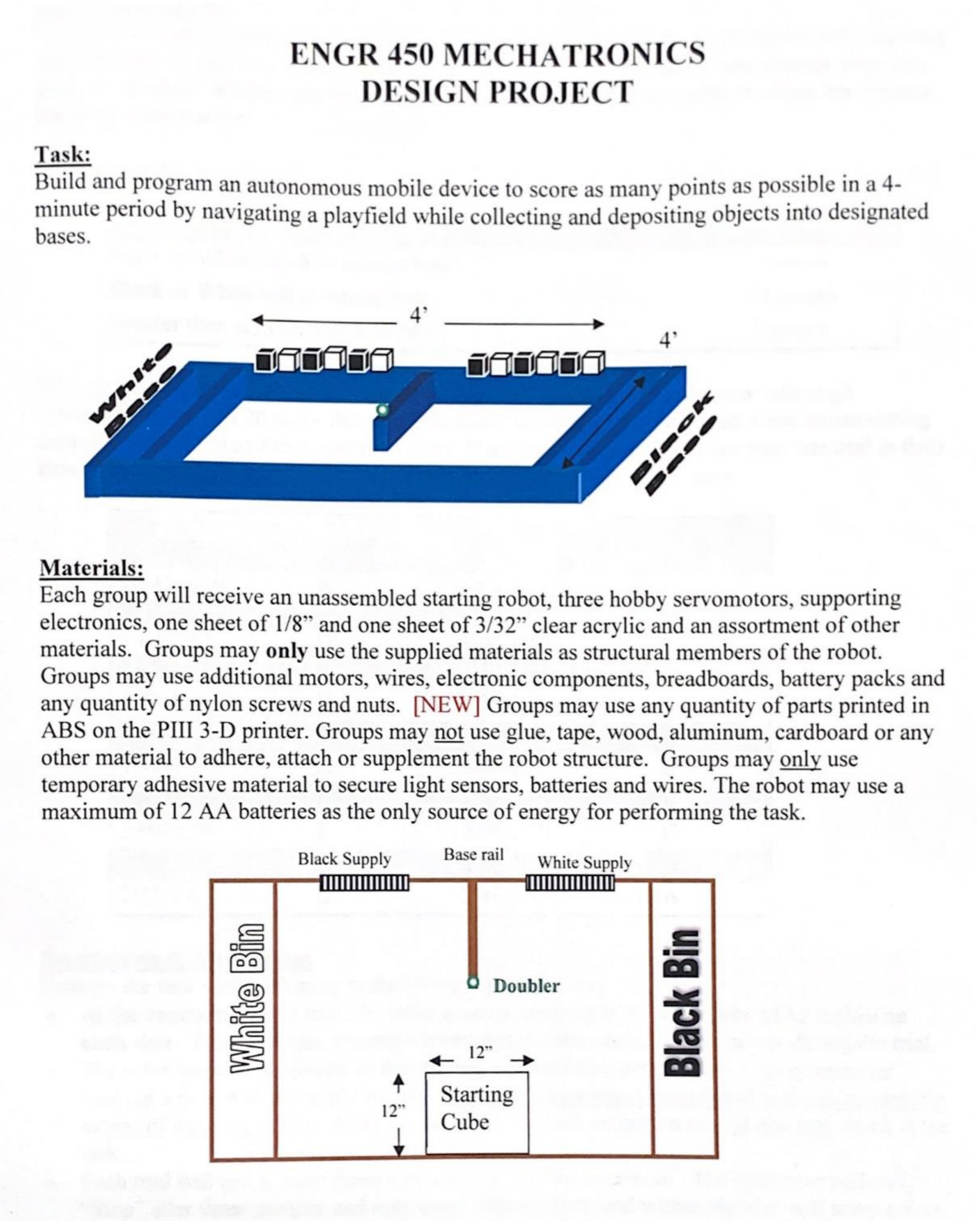
Milestone Report #5 Instructor initials indicating milestone completion:

Component	Requirements	Points	Awarded
Milestone code	Code is commented and shows next author.	5	5
Rendering/photos	Current photos and SolidWorks rendering show agreement.	5	5
Notebook organized	Notebook is neat and all material appears under appropriate tabs. Lab exercises included in appropriate tab.	5	5
Circuit Diagram	Able to reproduce circuit from the information on this diagram.	5	5
Milestone report	Provides informative description of design process. Describes approach to milestone and results. Includes participation %.	5	5

2

Another Perfect milestone report

Task



ENGR 450 MECHATRONICS DESIGN PROJECT

Task:

Build and program an autonomous mobile device to score as many points as possible in a 4-minute period by navigating a playfield while collecting and depositing objects into designated bases.

4'
4'
White Base
Black Base

Materials:

Each group will receive an unassembled starting robot, three hobby servomotors, supporting electronics, one sheet of 1/8" and one sheet of 3/32" clear acrylic and an assortment of other materials. Groups may **only** use the supplied materials as structural members of the robot. Groups may use additional motors, wires, electronic components, breadboards, battery packs and any quantity of nylon screws and nuts. [NEW] Groups may use any quantity of parts printed in ABS on the PIII 3-D printer. Groups may not use glue, tape, wood, aluminum, cardboard or any other material to adhere, attach or supplement the robot structure. Groups may only use temporary adhesive material to secure light sensors, batteries and wires. The robot may use a maximum of 12 AA batteries as the only source of energy for performing the task.

Task performance:

Two supply stations contain three white and three black blocks each. Each supply will contain a random order of white and black blocks. Robots must collect the blocks and deposit them into bins. A "doubler" marble rests on the center support. Placing this marble in either bin doubles the score from that bin.

Point values for the objects:

Any block in any base	5 points
Black or White block in correct base	3 points
Black or White ball in wrong base	-3 points
Greater than zero total in both bases	5 points

Each group will have trials at the scheduled time and date as indicated below (although circumstances in Fall 20 make this schedule likely to change). Groups must score points during their designated slot and may not trade slots. If a group can complete more than one trial in their time slot, the highest score will count. Total score is the total of both days.

Nov 19th		Nov 24th	
12:30-12:40	A	12:30-12:40	L
12:41-12:51	B	12:41-12:51	K
12:52-1:02	C	12:52-1:02	J
1:03-1:13	D	1:03-1:13	I
1:14-1:24	E	1:14-1:24	H
1:25-1:35	F	1:25-1:35	G
1:36-1:46	G	1:36-1:46	F
1:47-1:57	H	1:47-1:57	E
1:48-2:08	I	1:48-2:08	D
2:09-2:19	J	2:09-2:19	C
2:20-2:30	K	2:20-2:30	B
2:31-2:41	L	2:31-2:41	A

Restrictions & Allowances:

Perform the task while adhering to the following restrictions:

- At the beginning of the trial, the robot must fit completely **inside a cube of 12 inches on each side**. The robot may expand beyond that volume under its own power during the trial.
- The robot must be stationary in the starting base and then activated by a group member pushing a button or moving a switch. Plugging a wire into a breadboard is not an acceptable means of initiating a trial. Once the button is pressed, no person may in any way assist in the task.
- Each trial will last at most three minutes timed by the instructor. The instructor will call "Stop" after three minutes and only those objects contained within the base will score points.

- At any point in the trial, the group may declare the trial over. Before the robot is touched, score will be determined and the group credited with the total. The group may then run again within its allotted time at no risk to the points already scored.
- For the purposes of this competition, "in the base" means the entirety of the object is beneath a horizontal plane flush with the surface top surface of the base and within the containing walls of the base.
- Groups may use up to three feet of colored electrical tape to create visible landmarks on the playfield. The tape may be applied anywhere on the playfield but may serve only as visible marking. The tape may not be used as a mechanical aid or as a structural member. No other modifications to the playfield are permitted.
- If necessary to amend or revise the rules or restrictions, the instructor will post final rules and/or restrictions by October 30th. Instructor retains final discretion regarding rule violations or disqualification. Instructor also retains the authority to disqualify designs that violate the spirit of the competition, intentionally circumvent or exploit loopholes or omissions in the rules, or present a safety hazard to personnel or property.

Design Notebook:

Each group must keep a Design Notebook to record and document the design process. It must be an 8½x11 inch three-ring binder with a hard cover. This book should be the repository for all sketches, data sheets, sources of information, notes and all significant thinking about the project. Keep an index and use side-tabbed dividers between sections. Required headings of sections:

1. Design Problem
2. Milestone Reports
3. Sketches/Photos
4. Laboratory Exercises
5. Data Sheets
6. Course Notes
7. Summary

Notebook Check:

Instructor will collect notebooks for evaluation after each milestone. Notebooks must be delivered to the instructor's office by 4PM the next day after milestone due dates. Instructor will return all graded course material to students in the appropriate tabs of the design notebook.

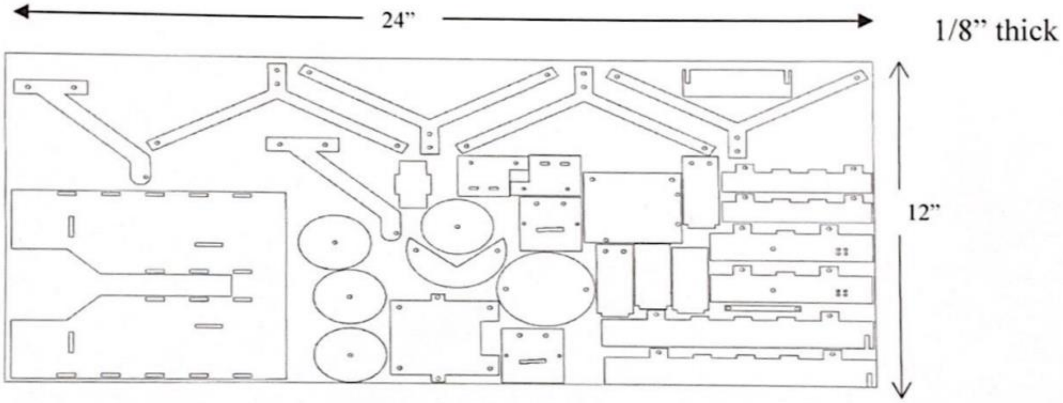
Notebook Content:

Milestone Reports - Responsibility for preparing the Milestone Report will cycle through all group members. Milestones 2-5 must include a printed summary, **in one page or less**, of the approach to the milestone, the robot's performance in the milestone and **planned modifications for the next milestone.** The Date, milestone #, group name and/or number and preparer's name must appear on the top on the report. Report must include a participation percentage for each group member since the last milestone report. Milestone reports are due the Friday after each milestone.

All Milestone Reports must include a complete listing of the Arduino code used to complete the milestone. The code must be fully commented and indicate author and date of creation. Groups of two members must alternate responsibility for authoring the code each milestone; groups with three members must cycle responsibility. For example, if Luigi and Mario are a group, then Luigi must develop the code for milestones 1, 3 & 5 and Mario the code for milestones 2, 4.

Milestone Reports 2-5 must include a complete circuit diagram showing the Arduino wired to accomplish the milestone. The diagram should contain all information to reconstruct the circuit using only this diagram. Include Arduino pin numbers and appropriate labels for all components. If an integrated circuit chip is present, indicate the part number. If showing a resistor, indicate its value in ohms. Please complete this diagram by hand on a single sheet of paper. I understand there are programs and online widgets, but it is a valuable experience to attempt to do this yourself.

Milestone reports 3-5 must contain a rendering of the current SolidWorks model and photos of the robot demonstrating significant agreement with the model. The rendering must show relevant information such as date, overall dimensions, labels and short description of the primary function of components. Rendering must demonstrate that the group is adhering to material restrictions by mapping all components fabricated from each thickness of material onto a sheet of that material. Example:



Laboratory Exercises – Notebook must document completion of each laboratory exercise. All deliverables listed for each exercise must be present in the appropriate tab in the notebook.

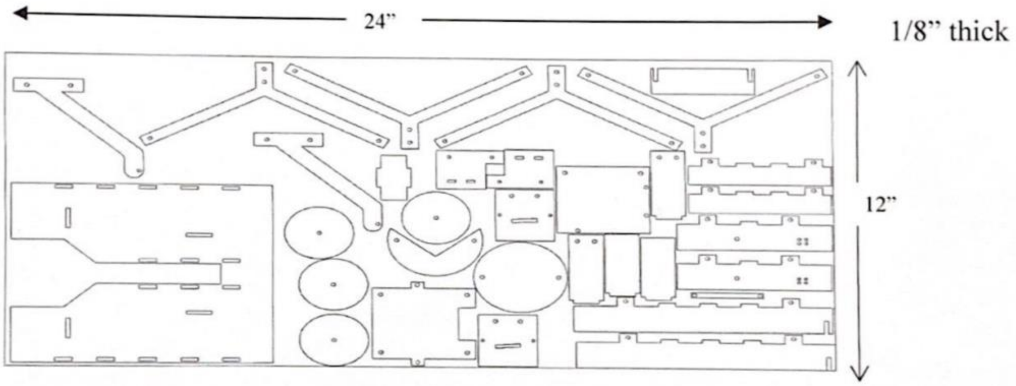
Summary - The Design Notebook must include a summary of the design effort and the goals achieved. The summary must present the robot and indicate its performance. The summary should also describe the limitations of the robot discovered during testing.

Final Notebook check - The remaining 25 points of the notebook grade are for Final Notebook Check. Groups are required to assign participation percentages on the top of the notebook check. Awarded points will be scaled by these percentages. Rubric for the final notebook check is included at the end of this document.

three members must cycle responsibility. For example, if Luigi and Mario are a group, then Luigi must develop the code for milestones 1, 3 & 5 and Mario the code for milestones 2, 4.

Milestone Reports 2-5 must include a complete circuit diagram showing the Arduino wired to accomplish the milestone. The diagram should contain all information to reconstruct the circuit using only this diagram. Include Arduino pin numbers and appropriate labels for all components. If an integrated circuit chip is present, indicate the part number. If showing a resistor, indicate its value in ohms. Please complete this diagram by hand on a single sheet of paper. I understand there are programs and online widgets, but it is a valuable experience to attempt to do this yourself.

Milestone reports 3-5 must contain a rendering of the current SolidWorks model and photos of the robot demonstrating significant agreement with the model. The rendering must show relevant information such as date, overall dimensions, labels and short description of the primary function of components. Rendering must demonstrate that the group is adhering to material restrictions by mapping all components fabricated from each thickness of material onto a sheet of that material. Example:



Laboratory Exercises – Notebook must document completion of each laboratory exercise. All deliverables listed for each exercise must be present in the appropriate tab in the notebook.

Summary - The Design Notebook must include a summary of the design effort and the goals achieved. The summary must present the robot and indicate its performance. The summary should also describe the limitations of the robot discovered during testing.

Final Notebook check - The remaining 25 points of the notebook grade are for Final Notebook Check. Groups are required to assign participation percentages on the top of the notebook check. Awarded points will be scaled by these percentages. Rubric for the final notebook check is included at the end of this document.

Milestones must be demonstrated by 10:00 PM on the due date shown

Milestone	Title	Due	Points
1	*Press a button to play a tune* Unofficial demonstration - 0 Points.	9/3	10
2	*Navigate the playfield. 20 points first trial, 15 points second or third.* Unofficial demonstration 5 Points.	9/17	20
3	*One block to base. 30 points first trial, 25 second, and 20 third.* Unofficial demonstration 10 Points.	10/1	30
4	*Determine block color. 30 points first trial, 25 second, and 20 third.* Unofficial demonstration 10 Points	10/15	30
5	*Two white and no black. 30 points first trial, 25 second, and 20 third.* Unofficial demonstration 10 Points. Official but with robot/model mismatch 20 Points.	11/5	30

Milestone 1 Press a button to play a tune: Make the Arduino play two easily distinguishable tunes (of your choosing) on the buzzer by pushing either of two buttons. A tune is at least 14 notes. Each button must always play the same tune and only that tune (not both). After the tune is played the Arduino must stop and wait until either button is pressed again. If either button is pressed while the tune is playing, the Arduino must wait two seconds and begin playing the pending tune after completing the current tune. Arduino must correctly respond to instructor's button pushes without reset.

Milestone 2 Navigate the playfield: Robot must start in the starting cube. The robot must play a distinct note or tune when touching the white side base rail and a different note or tune when touching the black side base rail (reverse order also acceptable) and then come to rest with either rubber wheel in contact with the floor inside the starting cube, playing a third note or tune. The robot must not play either note or tune when it is not in contact with the rail or stopped. For this milestone "in contact" will be loosely interpreted. For Milestone 2 only, buttons may be secured with temporary adhesive material. Groups have three official trials before the deadline.

Milestone 3 One block to opposite base: Robot must start in the starting cube. Robot must acquire a block from either dispenser and then deliver it to the **opposite side** bin. Groups have three official trials before the deadline.

Milestone 4 Determine block color: With the playfield in starting state, robot must acquire at least three blocks and recognize the color of each when delivered to a sensor. Robot may be placed anywhere on the field but must acquire the objects under its own power. Robot must play one distinct note or tune when it senses a white block, a different note or tune when it senses a black block and must not play either note or tune when is there is no block. Groups have three official trials before the deadline.

Milestone 5 Two white and no black: Robot must acquire two white blocks and place these in the white base without placing any black (reverse also acceptable). Robot must start in the starting cube. Groups have three official trials before the deadline.

ENGR 450 Design Project
Final Notebook Check (25 points)
Due Friday after last project demonstration at 5:00 PM

Group C

Member: Michael Sherman % Participation 50 % Total 100%

Member: Michaela Curcio % Participation 50 %

Member: % Participation

Component	Maximum	Awarded/comment
Condition: Notebook is clean and well-maintained and turned in on time. All project materials are contained in the notebook. Course notes are up to date. This sheet and final report are in the Summary tab.	5	
Model: SolidWorks model of final design is loaded into the appropriate location in Bridges.	5	
Rendering: Documents progress of design through a progression of sketches, SolidWorks renderings and/or photos. Photos are clear enough to show relevant design details and the photos show that the robot construction matches SolidWorks design. Final circuit diagram allows reconstruction of the circuit using only the information on the diagram. Final layout shows all parts within material restriction.	5	
Final Summary: A 1-2 page summary is included that presents the final design and includes the results of the robot's performance. Summary briefly discusses the strengths and weaknesses of the design and discusses lessons learned from the competition. Summary suggests what the group would do differently if the project were started afresh.	5	
Teamwork: Group has completed the participation percentage table above. Group has resisted over-specialization ensuring each member has had an opportunity to gain practice coding, wiring, designing, etc. Should be consistent with milestone reports.	5	

Appendix E

Code for all Milestones and Final

Milestone 1

Michael Sherman & Michaela Curcio

```
//Milestone 1 Code
//Authored by Michael Sherman

#include "pitches.h"                      // include pitches library

//millis stuff
unsigned long previousMillis = 0 ;       // will store last time the note changed
int interval = 0;                        // time between button and first note

// constants to set pin numbers
const int buttonPin2 = 2;               // the number of the pushbutton pin
const int buttonPin3 = 3;               // the number of the pushbutton pin


// variables will change:
int buttonState2 = 0;                   // variable for reading the pushbutton status
int buttonState3 = 0;                   // variable for reading the pushbutton status

//boolean stuff to check if song added to queue
boolean press2 = false;                 // starts pin 2 as false
boolean press3 = false;                 // starts pin 3 as false

// Song Pin2 - POUND THE ALARM:

//array storing the notes of song
int melody2[] = {
  NOTE_A4, NOTE_F4, NOTE_D4, NOTE_C4, 0, NOTE_B4, NOTE_B4, NOTE_B4, NOTE_B4, NOTE_A4, NOTE_G4, NOTE_F4, NOTE_A4, NOTE_F4, NOTE_D4, NOTE_C4
};

//array storing note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations2[] = {
  4, 4, 3, 2, 8, 6, 6, 6, 6, 6, 6, 6, 4, 4, 3, 2
};


//Song Pin3 - STARSHIPS:

//array storing the notes of song
int melody3[] = {
  NOTE_FS4, NOTE_FS4, 0, NOTE_FS4, NOTE_A4, NOTE_A4, NOTE_B4, NOTE_FS4, NOTE_E4, NOTE_D4, 0, NOTE_FS4, NOTE_FS4, 0, NOTE_D4, NOTE_E4, NOTE_E4, NOTE_FS4, NOTE_E4, NOTE_D4
};

//array storing note durations: 4 = quarter note, 8 = eighth note, etc.:
int noteDurations3[] = {
  3, 3, 9, 7, 5, 6, 3, 6, 5, 6, 8, 3, 3, 8, 8, 5, 5, 4, 7, 7
};


void setup() {
  Serial.begin(9600);                     //initialize serial monitor
  pinMode(buttonPin2, INPUT);             //initialize pushbutton pin 2 as input
  pinMode(buttonPin3, INPUT);             //initialize pushbutton pin 3 as input
}

void loop()
```

```
{
  //read state of pushbutton values
  buttonState2 = digitalRead(buttonPin2);
  buttonState3 = digitalRead(buttonPin3);

  //change boolean if button was pressed
  if (buttonState2 == HIGH) {
    Serial.println("Switch 2 Pressed");
    press2 = true;
  }
  else if (buttonState3 == HIGH) {
    Serial.println("Switch 3 Pressed");
    press3 = true;
  }

  //Playing Song Pin2 - POUND THE ALARM:
  else if (press2 == true) {                  //check to see if boolean is true
    press2 = false;                   //change it back to false before starting the song
    Serial.print("Playing Song Pin2");

    for (int thisNote = 0; thisNote < 16;) {    //for loop for each note
      unsigned long currentMillis = millis(); //set variable equal to the number of milliseconds passed from beginning

      //Check other button continuosly
      Serial.println("Checking Buttons");
      buttonState3 = digitalRead(buttonPin3); //read pin 3
      if (buttonState3 == HIGH) {              //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press3 = true;
      }
      buttonState2 = digitalRead(buttonPin2); //read pin 2
      if (buttonState2 == HIGH) {              //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press2 = true;
      }

      if (currentMillis - previousMillis >= interval) {       //check if time passed during loop is greater than pause between notes
        previousMillis = currentMillis;                       //set new previousMillis to equal currentMillis
        int noteDuration2 = 1000 / noteDurations2[thisNote]; //calculate note duration from array, one second divided by note type
        tone(8, melody2[thisNote], noteDuration2);           //play tone from array on pin 8 for duration calculated above

        // to distinguish the notes, set a minimum time between them.
        int pauseBetweenNotes = noteDuration2 * 1.30;       //calculate time between notes time of note + 30%
        interval = pauseBetweenNotes;                       //set variable "interval" equal to this value
        thisNote++;                                         //add one to the count for the for loop so it can play next note
      }
    }

    // Adding two second delay before next song, but still checking for button presses during delay
    int difference;
    unsigned long currentMillis = millis();
    unsigned long previousMillis = millis();
    difference = currentMillis - previousMillis;
    while (difference < 2000) {
      Serial.println("waiting...");
      buttonState3 = digitalRead(buttonPin3);                 //read pin 3
      if (buttonState3 == HIGH) {                             //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press3 = true;
      }
      buttonState2 = digitalRead(buttonPin2);               //read pin 2
      if (buttonState2 == HIGH) {                            //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press2 = true;
      }
      unsigned long currentMillis = millis();                   //update currentMillis
      difference = currentMillis - previousMillis;              //update difference
      Serial.println(difference);
    }
  }

  //Playing Song Pin3 - STARSHIPS:
  else if (press3 == true) {                                    //check to see if boolean is true
    press3 = false;                                            //change it back to false before starting the song
    Serial.print("Playing Song Pin2");

    for (int thisNote3 = 0; thisNote3 < 20;) {                  //for loop for each note
      unsigned long currentMillis = millis();                  //set variable equal to the number of milliseconds passed from beginning

      //Check buttons continuosly
      buttonState3 = digitalRead(buttonPin3);                  //read pin 3
      if (buttonState3 == HIGH) {                             //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press3 = true;
      }
      buttonState2 = digitalRead(buttonPin2);                 //read pin 2
      if (buttonState2 == HIGH) {                            //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press2 = true;
      }

      if (currentMillis - previousMillis >= interval) {       //check if time passed during loop is greater than pause between notes
        previousMillis = currentMillis;                       //set new previousMillis to equal currentMillis
        int noteDuration3 = 1000 / noteDurations3[thisNote3]; //calculate note duration from array, one second divided by note
        tone(8, melody3[thisNote3], noteDuration3);           //play tone from array on pin 8 for duration calculated above

        int pauseBetweenNotes3 = noteDuration3 * 1.30;       //calculate time between notes time of note + 30% *taken from toneMelody*
        interval = pauseBetweenNotes3;                       //set variable "interval" equal to this value
        thisNote3++;                                         //add one to the count for the for loop so it can play next note
      }
    }

    // Adding two second delay before next song, but still checking for button presses during delay
    int difference;
    unsigned long currentMillis = millis();
    unsigned long previousMillis = millis();
    difference = currentMillis - previousMillis;
    while (difference < 2000) {
      Serial.println("waiting...");
      buttonState3 = digitalRead(buttonPin3);                 //read pin 3
      if (buttonState3 == HIGH) {                             //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press3 = true;
      }
      buttonState2 = digitalRead(buttonPin2);               //read pin 2
      if (buttonState2 == HIGH) {                            //change boolean to true if it is pressed
        Serial.println("Song Added to Queue");
        press2 = true;
      }
      unsigned long currentMillis = millis();                   //update currentMillis
      difference = currentMillis - previousMillis;              //update difference
      Serial.println(difference);
    }
  }
}
```

Milestone 1

pitches.h

```
/*************************************************
 * Public Constants
 *************************************************/

#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
```

```
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
```

Milestone 2

Michael Sherman & Michaela Curcio

```
//Milestone 2
//Author 2: Michaela Curcio

// constants won't change. Used here to set pin numbers:
const int leftA = 2; // Left Motor A pin
const int leftB = 3; // Left Motor B pin
const int rightA = 6; // Right Motor A pin
const int rightB = 7; // Right Motor B pin
const int frontButton = 12; //Front Button Pin
const int rightButton = 8; //Right Button Pin
const int backButton = 5; //Back Button Pin
const int BuzzerPin = 4; //Buzzer Pin

//Initialize Button States
int frontButtonState = 0;
int backButtonState = 0;
int rightButtonState = 0;

// Variables will change:
int state = 0; // variable to hold current state
unsigned long startTime; // will store the time the state was setup

// the following variable is a long because the time, measured in miliseconds,
// will quickly become a bigger number than can be stored in an int.
long interval = 2000; // interval at which to change

void setup() {
  // set the digital pins as outputs and inputs:
  pinMode(leftA, OUTPUT);
  pinMode(leftB, OUTPUT);
  pinMode(rightA, OUTPUT);
  pinMode(rightB, OUTPUT);
  pinMode(frontButton, INPUT);
  pinMode(rightButton, INPUT);
  pinMode(backButton, INPUT);
  pinMode(BuzzerPin, OUTPUT);
  //state1Setup();
  state = 1;
}
void loop() {
  // This loop simply calls the state function for the current State

  switch (state) {
    case 1:
      state1();
      break;
    case 2:
      state2();
      break;
    case 3:
      state3();
      break;
    case 4:
      state4();
      break;
    case 5:
      state5();
      break;
    case 6:
      state6();
      break;
    case 7:
      state7();
      break;
    case 8:
      state8();
      break;
  }
}
```

Milestone 2

motorFunctions

```
// Never change these functions
// If they have the reversed outcome rewire the platform
// Do not re-write these functions
void platformForward()
{
  leftForward();
  rightForward();
}
void platformBackward()
{
  leftBackward();
  rightBackward();
}
void platformStop()
{
  leftStop();
  rightStop();
}
void platformSpinLeft()
{
  leftBackward();
  rightForward();
}
void platformSpinRight()
{
  rightBackward();
  leftForward();
}
//left
void leftForward()
{
  digitalWrite(leftA, HIGH);
  digitalWrite(leftB, LOW);
}
void leftBackward()
{
  digitalWrite(leftA, LOW);
  digitalWrite(leftB, HIGH);
}
void leftStop()
{
  digitalWrite(leftA, LOW);
  digitalWrite(leftB, LOW);

}
//right
void rightForward()
{
  digitalWrite(rightA, HIGH);
  digitalWrite(rightB, LOW);
}
void rightBackward()
{
  digitalWrite(rightA, LOW);
  digitalWrite(rightB, HIGH);
}
void rightStop()
```

```
{
  digitalWrite(rightA, LOW);
  digitalWrite(rightB, LOW);
}
```

Milestone 2

State 1

```
void state1Setup() {
  startTime = millis();
  state = 1;
}

void state1() {
  // put your main code here, to run repeatedly:
  unsigned long currentTime;

  //Go Forward
  platformForward();
  //Check if front button is pressed
  frontButtonState = digitalRead(frontButton);

  if (frontButtonState == HIGH)
  {
    platformStop();
    tone(BuzzerPin, 400, 1000); //plays 1st tone once button is pressed
    delay(1000);
    state2Setup(); //move to next state
  }

  //USE AS FAILSAFE
  //Move to next state if it has been driving forward for 15 sec. w/o pushing button
  currentTime = millis();
  if ((currentTime - startTime) > 10000) {
    platformStop();
    tone(BuzzerPin, 400, 1000); //plays 1st tune incase it does not hit wall/button
    delay(1000);
    //Next State
    state2Setup(); //move to next state
  }
}
```

Milestone 2

State 2

```
void state2Setup() {
  startTime = millis();
  state = 2;
}

void state2() {
  startTime = millis();
  unsigned long currentTime;

  //Go Backward
  platformBackward();
  //Check if back button is pressed
  backButtonState = digitalRead(backButton);
  if (backButtonState == HIGH)
  {
    platformStop();
    delay(500);
    state3Setup(); //moves to next state
  }

  //USE AS FAILSAFE
  //Move to next state if it has been driving forward for 10 sec. w/o pushing button
  currentTime = millis();
  if ((currentTime - startTime) > 10000) {
    platformStop();
    delay(500);
    //Next State
    state3Setup(); //moves to next state
  }
}
```

Milestone 2

State 3

```
void state3Setup() {
  startTime = millis();
  state = 3;
}

void state3() {
  // put your main code here, to run repeatedly:
  startTime = millis();
  unsigned long currentTime;

  //Turn Right until button or for 3 sec
  platformSpinRight();

  //Check if button is pressed
  rightButtonState = digitalRead(rightButton); //once right button is pressed, stop

  if (rightButtonState == HIGH)
  {
    platformStop();
    delay(500);
    state4Setup(); //move to next state
  }

  //Stop this state after a timeout
  currentTime = millis();
  if ((currentTime - startTime) > 3000) {
    platformStop();
    delay(1000);
    //Next State
    state4Setup(); //move to next state
  }
}
```

Milestone 2

State 4

```
void state4Setup() {
  startTime = millis();
  state = 4;
}

void state4() {
  startTime = millis();
  // put your main code here, to run repeatedly:
  unsigned long currentTime;

  //check if front button is pressed
  frontButtonState = digitalRead(frontButton);
  currentTime = millis();

  //if the time is less than 10 sec, and the front button is not pressed, begin "hugging"
  while ((currentTime - startTime < 10000) and frontButtonState == LOW)
  {
    frontButtonState = digitalRead(frontButton); //check buttons
    rightButtonState = digitalRead(rightButton);
    if (rightButtonState == LOW) {
      frontButtonState = digitalRead(frontButton);
      //"hugging" makes the robot move against the side wall
      platformForward();
      delay(200);
      platformSpinRight();
      delay(200);
    }
    else
    {
      platformForward();
    }
    currentTime = millis();
  }
  platformStop();
  delay(1000);
  state5Setup(); //move to next state
}
```

Milestone 2

State 5

```
void state5Setup() {
  startTime = millis();
  state = 5;
}

void state5() {
  startTime = millis();
  unsigned long currentTime;

  //Go back and turn left
  platformBackward();
  delay(100);
  platformStop();
  delay(50);
  platformSpinLeft();
  delay(650);
  platformStop();
  delay(500);
  state6Setup(); //move to next state
}
```

Milestone 2

State 6

```
void state6Setup() {
  startTime = millis();
  state = 6;
}

void state6() {
  startTime = millis();
  unsigned long currentTime;

  frontButtonState = digitalRead(frontButton); //check button
  currentTime = millis();

  while ((currentTime - startTime < 10000) and frontButtonState == LOW)
  {
    frontButtonState = digitalRead(frontButton); //check buttons
    rightButtonState = digitalRead(rightButton);
    if (rightButtonState == LOW) {
      frontButtonState = digitalRead(frontButton);
      platformForward();
      delay(200);
      platformSpinRight();
      delay(100);
    }
    else
    {
      platformForward();
    }
    currentTime = millis();
  }

  //plays 2nd tone twice once button is pressed
  platformStop();
  tone(BuzzerPin, 600, 1000);
  delay(2000);
  tone(BuzzerPin, 600, 1000);
  delay(1000);
  state7Setup(); //move to next state
}
```

Milestone 2

State 7

```
void state7Setup() {
  startTime = millis();
  state = 7;
}
void state7() {
  startTime = millis();
  unsigned long currentTime;

  //Go Backward
  platformBackward();

  //Check if back button is pressed
  backButtonState = digitalRead(backButton); //check back button
  if (backButtonState == HIGH)
  {
    platformStop(); //stop when back button is pressed
    delay(500);
    state8Setup(); //move to next state
  }

  //Move to next state if it has been driving forward for 10 sec. w/o pushing button
  currentTime = millis();
  if ((currentTime - startTime) > 10000) {
    platformStop();
    delay(500);
    //Next State
    state8Setup();
  }
}
```

Milestone 2

State 8

```
void state8Setup() {
  startTime = millis();
  state = 8;
}

void state8() {
  startTime = millis();
  unsigned long currentTime = millis();

  platformForward(); //move forward
  delay(250);

  while (currentTime - startTime < 2450) //while time is less than 2450, begin "shimmy"
  {
    platformSpinLeft(); //"shimmy" moves slowly so that robot doesn't slam against board
    delay(300);
    platformStop();
    delay(200);
    currentTime = millis();
  }

  platformForward();
  delay(1800);
  platformStop();
  //plays 3rd tone three times once button is pressed
  tone(BuzzerPin, 800, 1000);
  delay(2000);
  tone(BuzzerPin, 800, 1000);
  delay(2000);
  tone(BuzzerPin, 800, 1000);
  delay(1000);
  delay(30000);
}
```

Milestone 3

Michael Sherman & Michaela Curcio

```
//Milestone 3 Code
//Authored By Michael Sherman

// set pin numbers:
const int leftA =  8;      // Left Motor A pin
const int leftB =  11;      // Left Motor B pin
const int rightA =  7;     // Right Motor A pin
const int rightB =  10;     // Right Motor B pin
const int frontButton = 2; //Front Button Pin
const int leftButton = 4; //Right Button Pin
const int backButton = A1; //Back Button Pin

//Initialize Button States
int frontButtonState = 0;
int backButtonState = 0;
int leftButtonState = 0;

// Variables will change:
int state = 0;             // variable to hold current state
unsigned long startTime;   // will store the time the state was setup

//Setup Servos
#include <Servo.h>
Servo myservod; //Dumpster
Servo myservoc; //Cradle
Servo myservos; //Sniper

void setup() {
  // set the digital pins as output:
  pinMode(leftA, OUTPUT);
  pinMode(leftB, OUTPUT);
  pinMode(rightA, OUTPUT);
  pinMode(rightB, OUTPUT);
  pinMode(frontButton, INPUT);
  pinMode(leftButton, INPUT);
  pinMode(backButton, INPUT);

  //Begin Servos in the Right Spot
  myservos.attach(6);
  delay(500);
  myservos.write(180);
  delay(500);
  myservos.detach();
  delay(500);

  state1Setup();
}


void loop() {
  // This loop simply calls the state function for the current State
  switch (state) {
    case 1:
      state1();
      break;
    case 2:
      state2();
      break;
    case 3:
      state3();
      break;
    case 4:
      state4();
      break;
    case 5:
      state5();
      break;
    case 6:
      state6();
      break;
    case 7:
      state7();
      break;
    case 8:
      state8();
      break;
    case 9:
      state9();
      break;
  }
}
```

Milestone 3

Motor Functions

```
//  Never change these functions
//  If they have the reversed outcome rewire the platform
//  Do not re-write these functions
void platformForward()
{
  leftForward();
  rightForward();
}
void platformBackward()
{
  leftBackward();
  rightBackward();
}
void platformStop()
{
  leftStop();
  rightStop();
}
void platformSpinLeft()
{
  leftBackward();
  rightForward();
}
void platformSpinRight()
{
  rightBackward();
  leftForward();
}
//left
void leftForward()
{
      digitalWrite(leftA, HIGH);
      digitalWrite(leftB, LOW);
}
void leftBackward()
{
      digitalWrite(leftA, LOW);
      digitalWrite(leftB, HIGH);
}

void leftStop()
{
      digitalWrite(leftA, LOW);
      digitalWrite(leftB, LOW);
}
//right
void rightForward()
{
      digitalWrite(rightA, HIGH);
      digitalWrite(rightB, LOW);
}
void rightBackward()
{
      digitalWrite(rightA, LOW);
      digitalWrite(rightB, HIGH);
}

```

```
void rightStop()
{
	digitalWrite(rightA, LOW);
	digitalWrite(rightB, LOW);
}
```

Milestone 3

State 1

```
//Forward Until Button or Time

void state1Setup() {
  platformForward();
  startTime = millis();
  state = 1;
}

void state1() {
  unsigned long currentTime;
  currentTime = millis();

  //Go Until Front Button
  frontButtonState = digitalRead(frontButton);
  if (frontButtonState == HIGH)
  {
    state2Setup();
  }


  //Go Until Timeout
  currentTime = millis();
  if ((currentTime - startTime) > 7000) {
    state2Setup();
  }
}
```

Milestone 3

State 2

```
//Turn Right

void state2Setup() {
  platformStop();
  startTime = millis();
  state = 2;
}

void state2() {

  //Back Off Wall, Turn Right
  delay(1000);
  platformBackward();
  delay(300);
  platformStop();
  delay(20);
  platformSpinRight();
  delay(400);
  platformStop();
  delay(20);
  platformForward();
  delay(1200);
  platformStop();
  delay(20);

  state3Setup();
}
```

Milestone 3

State 3

```
//Forward Until Button or Time

void state3Setup() {
  platformForward();
  startTime = millis();
  state = 3;
}

void state3() {
  // put your main code here, to run repeatedly:
  unsigned long currentTime;

  currentTime = millis();

  //Go until front button
  frontButtonState = digitalRead(frontButton);
  if (frontButtonState == HIGH)
  {
    state4Setup();
  }

//Timeout
  if ((currentTime - startTime) > 7000)
  {
    state4Setup();
  }
}
```

Milestone 3

State 4

```
//Turn Right

void state4Setup() {
  platformStop();
  delay(20);
  startTime = millis();
  state = 4;

}

void state4() {

  //Turn Right
  platformBackward();
  delay(500);
  platformStop();
  delay(20);
  platformSpinRight();
  delay(450);
  platformStop();
  delay(20);
  state5Setup();
}
```

Milestone 3

State 5

```
//Positioning into Corner

void state5Setup() {
  platformStop();
  delay(20);
  startTime = millis();
  state = 5;
}

void state5() {
  unsigned long currentTime;
  currentTime = millis();

  //Forward
  platformForward();
  delay(1500);
  platformStop();
  delay(20);

  //Position by only moving one wheel
  rightBackward();
  leftStop();
  delay(800);

  platformStop();
  delay(20);

  state6Setup();
}
```

Milestone 3

State 6

```
//Parallel Parking

void state6Setup() {
  platformStop();
  delay(20);
  startTime = millis();
  state = 6;
}

void state6() {
  // put your main code here, to run repeatedly:
  unsigned long currentTime;
  currentTime = millis();

  //Parallel Parking into corner
  //Buttons were extremely unreliable for this, so we used time
  platformBackward();
  backButtonState = digitalRead(backButton);
  if (backButtonState == HIGH)
  {
    platformStop();
    delay(20);
    rightForward();
    delay(500);
    platformStop();
    delay(20);
    platformBackward();
    delay(1000);
    platformStop();
    delay(20);
    rightForward(); //Controlling the wheels allowed us to turn without spinning
    leftStop();
    delay(150);
    platformForward();
    delay(300);
    platformStop();
    delay(1000);
    state7Setup();
  }

  //Timeout
  currentTime = millis();
  if ((currentTime - startTime) > 2000) {
    platformStop();
    delay(20);
    rightForward();
    delay(500);
    platformStop();
    delay(20);
    platformBackward();
    delay(1000);
    platformStop();
    delay(20);
    rightForward(); //Controlling the wheels allowed us to turn without spinning
    leftStop();
    delay(150);
    platformForward();
    delay(300);
    platformStop();
    delay(1000);
    state7Setup();
  }
}
```

Milestone 3

State 7

```
//Get the block

void state7Setup() {
  platformStop();
  delay(20);
  state = 7;
}

void state7() {

  //Sniper takes block off wall
  myservos.attach(6);
  delay(100);
  myservos.write(40);
  delay(1000);
  myservos.write(180);
  delay(1000);
  myservos.detach();

  //Empty Cradle into Dumpster
  myservoc.attach(3);
  delay(100);
  myservoc.write(0);
  delay(1000);
  myservoc.write(90);
  delay(250);
  myservoc.detach();
  delay(1000);

  state8Setup();
}
```

Milestone 3

State 8

```
//Return to Base

void state8Setup() {
  platformStop();
  delay(20);
  state = 8;
}

void state8() {

  //Move around the board counter-clockwise to reach the base
  //All turns are timed because the buttons were unreliable
  //Letters were used to organize positions on the board

  //A
  platformSpinRight();
  delay(750);
  platformStop();
  delay(20);

  //B
  platformForward();
  delay(6000);
  platformStop();
  delay(20);

  //C
  platformBackward();
  delay(150);
  platformStop();
  delay(20);

  //D
  platformSpinLeft();
  delay(800);
  platformStop();
  delay(20);

  //E
  platformForward();
  delay(6000);
  platformStop();
  delay(20);

  //F
  platformBackward();
  delay(200);
  platformStop();
  delay(20);

  //G
  platformSpinLeft();
  delay(350);
  platformStop();
  delay(20);

  //H
  platformForward();
  delay(6000);
  platformStop();
  delay(20);

  state9Setup();
}
```

Milestone 3

State 9

```
//Empty the dumpster

void state9Setup() {
  platformStop();
  delay(20);
  state = 9;
}

void state9() {

  //Attach dumpster servo, dump it, return, detach
  myservod.attach(9);
  delay(1000);
  myservod.write(180);
  delay(1000);
  myservod.write(55);
  delay(50000);
}
```

Milestone 4

Michael Sherman & Michaela Curcio

```
//Milestone  4
//Author  2: Michaela  Curcio

// constants won't change. Used here to set pin numbers:
const int leftA =  5;      // Left Motor A pinbbh
const int leftB =  4;      // Left Motor B pin
const int rightA =  8;     // Right Motor A pin
const int rightB =  7;     // Right Motor B pin
const int analogInPin = A0;  // Analog input pin that the potentiometer is attached to
const int buzzerPin = 12; // Analog output pin that the LED is attached to
int sensorValue = 0;        // value read from the pot

int state = 0;             // variable to hold current state
unsigned long startTime;   // will store the time the state was setup

//Setup Servos
#include <Servo.h>
Servo myservod; //Dumpster
Servo myservoc; //Cradle
Servo myservos; //Sniper

void setup() {
  // set the digital pins as output:
  pinMode(leftA, OUTPUT);
  pinMode(leftB, OUTPUT);
  pinMode(rightA, OUTPUT);
  pinMode(rightB, OUTPUT);
  pinMode(buzzerPin, OUTPUT);

  state1Setup();
}

void loop() {
  // This loop simply calls the state function for the current State
  switch (state) {
    case 1:
      state1();
      break;
    case 2:
      state2();
      break;
    case 3:
      state3();
      break;
    case 4:
      state4();
      break;
    case 5:
      state5();
      break;
    case 6:
      state6();
      break;
  }
}
```

Milestone 4

motorFunctions

```
//  Never change these functions
//  If they have the reversed outcome rewire the platform
//  Do not re-write these functions
void platformForward()
{
  leftForward();
  rightForward();
}
void platformBackward()
{
  leftBackward();
  rightBackward();
}
void platformStop()
{
  leftStop();
  rightStop();
}
void platformSpinLeft()
{
  leftBackward();
  rightForward();
}
void platformSpinRight()
{
  rightBackward();
  leftForward();
}
//left
void leftForward()
{
  digitalWrite(leftA, HIGH);
  digitalWrite(leftB, LOW);
}
void leftBackward()
{
  digitalWrite(leftA, LOW);
  digitalWrite(leftB, HIGH);
}

void leftStop()
{
  digitalWrite(leftA, LOW);
  digitalWrite(leftB, LOW);
}
//right
void rightForward()
{
  digitalWrite(rightA, HIGH);
  digitalWrite(rightB, LOW);
}
void rightBackward()
{
  digitalWrite(rightA, LOW);
  digitalWrite(rightB, HIGH);
}

```

```
void rightStop()
{
  digitalWrite(rightA, LOW);
  digitalWrite(rightB, LOW);
}
```

Milestone 4

State 1

```
//Get the block

void state1Setup() {
  platformStop();
  delay(20);
  state = 1;
}

void state1() {

  //Sniper flicks block off wall
  myservos.attach(6); //attaches sniper to pin 6
  delay(100);
  myservos.write(70); //moves down lower so it doesn't just smack it at full speed
  delay(1000);
  myservos.write(50); //flicks block in cradle
  delay(1000);
  myservos.write(90); //starting position
  delay(1000);
  myservos.detach();

  //wiggle cradle
  myservoc.attach(3); //attaches to pin 3
  delay(100);
  myservoc.write(60); //wiggles
  delay(1000);
  myservoc.write(80); //moves back to original spot
  delay(250);
  myservoc.detach();
  delay(1000);

  state2Setup();
}
```

Milestone 4

State 2

```
//read the color of the block

void state2Setup() {
  platformStop();
  delay(500);
  state = 2;
}

void state2() {
  // initialize serial communications at 9600 bps:
  Serial.begin(9600);

  // read the analog in value:
  sensorValue = analogRead(analogInPin);

  // print the results to the Serial Monitor:
  Serial.print("sensor = ");
  Serial.println(sensorValue);

  if (sensorValue < 149) {  //for white block
    startTime = millis();
    while ((millis() - startTime) < 3000) {
      tone(buzzerPin, 750, 200);  //makes siren noise
      delay(200);
      tone(buzzerPin, 2600, 200);
      delay(200);
      state3Setup();
    }
  }
  else if ((150 < sensorValue) && (sensorValue < 800)) {  //for black block
    tone(buzzerPin, 200, 2000);  //makes single deep tune
    delay(2000);
    state3Setup();
  }
  else if (sensorValue > 801) {  //no block
    myservod.attach(10);  //attaches dumpster to pin 10
    delay(1000);
    myservod.write(175); //flings block out of dumpster
    delay(1000);
    myservod.write(55);  //goes back to staring position
    delay(1000);
    myservod.detach();
    delay(1000);
    state6Setup();  //doesn't make any noise, sends straight to state 6 to try again incase block is messed up in cradle
  }
}
```

Milestone 4

State 3

```
//Empty cradle into dumpster

void state3Setup() {
  platformStop();
  delay(20);
  state = 3;
}

void state3() {

  myservoc.attach(3); //attaches cradle to pin 3
  delay(100);
  myservoc.write(5); //throws block from cradle to dumpster
  delay(1000);
  myservoc.write(80);  //starting position
  delay(250);
  myservoc.detach();
  delay(1000);

  state4Setup();
}
```

Milestone 4

State 4

```
//Yeet block to empty dumpster

void state4Setup() {
  platformStop();
  delay(20);
  state = 4;
}

void state4() {

  myservod.attach(10);  //attaches dumpster to pin 10
  delay(1000);
  myservod.write(175); //flings block out of dumpster
  delay(1000);
  myservod.write(55);  //goes back to starting position
  delay(1000);
  myservod.detach();
  delay(1000);

  state5Setup();
}
```

Milestone 4

State 5

```
//Scooch forward

void state5Setup() {
  platformStop();
  delay(20);
  state = 5;
}

void state5() {
  leftStop();
  rightForward();  //turns right to get back against back wall
  delay(100);
  platformStop();
  delay(20);
  platformForward();  //moves forward to get to next block
  delay(115);
  platformStop();
  delay(2000);
  leftStop();
  rightForward();  //turns right to get back against back wall again just incase
  delay(100);
  platformStop();
  delay(20);

  state1Setup();  //repeats all 5 states again for next blocks
}
```

Milestone 4

State 6

```
//read the color of the block AGAIN
//this is to see if the dumpster fixed anything

void state6Setup() {
  platformStop();
  delay(500);
  state = 6;
}

void state6() {
  // read the analog in value:
  sensorValue = analogRead(analogInPin);

  // print the results to the Serial Monitor:
  Serial.print("sensor = ");
  Serial.println(sensorValue);

  if (sensorValue < 149) {  //for white block
    startTime = millis();
    while ((millis() - startTime) < 3000) {
      tone(buzzerPin, 750, 200);  //makes siren noise
      delay(200);
      tone(buzzerPin, 2600, 200);
      delay(200);
      state3Setup();
    }
  }
  else if ((150 < sensorValue) && (sensorValue < 800)) {  //for black block
    tone(buzzerPin, 200, 2000);  //makes single deep tune
    delay(2000);
    state3Setup();
  }
  else if (sensorValue > 801) {  //no block
    state5Setup(); //go straight to stae 5 because there is actually no block
  }
}
```

Milestone 5

Author 1: Michael Sherman

```
//Milestone 5 Code
//Author 1: Michael Sherman

//Set pin numbers:
const int leftA =  5;      // Left Motor A pin
const int leftB =  4;      // Left Motor B pin
const int rightA =  8;     // Right Motor A pin
const int rightB =  7;     // Right Motor B pin
const int analogInPin = A0; //Color Sensor
const int buzzerPin = 12;   //Buzzer
const int leftButton = A2; //Left Button Pin
const int backButton = A1; //Back Button Pin
const int trigPin = 13;   //ultrasonic trigger
const int echoPin = A3;   //ultrasonic echo

//Initialize Button States and Sensor Value
int backButtonState = 0;
int leftButtonState = 0;
int sensorValue = 0;

// Variables will change:
int blockCounter = 0;     //count number of blocks taken
int state = 0;             // variable to hold current state
unsigned long startTime;   // will store the time the state was setup
long duration;            //duration used in ultrasonic
float distanceCm;         //distance used in ultrasonic

//Setup Servos
#include <Servo.h>
Servo backDumpster;       // Back Dumpster
Servo cradle;             //Cradle
Servo sniper;             //Sniper
Servo frontDumpster;      //Front Dumpster
Servo turntable;          //Turntable


//Run this to read from ultrasonic
void ultrasonicRead() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distanceCm = duration * 0.034 / 2;
}

void setup() {
  // set the digital pins as output:
  pinMode(leftA, OUTPUT);
  pinMode(leftB, OUTPUT);
  pinMode(rightA, OUTPUT);
  pinMode(rightB, OUTPUT);
  pinMode(leftButton, INPUT);
  pinMode(backButton, INPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  //Begin Servos in the Right Spot
  //sniper
  sniper.attach(6);
  delay(20);
  sniper.write(165);
  delay(250);
  sniper.detach();
  delay(20);

  //back dumpster
  backDumpster.attach(10);
  delay(50);
  backDumpster.write(60);
  delay(150);
  backDumpster.detach();

  //front dumpster
  frontDumpster.attach(11);
  delay(50);
  frontDumpster.write(110);
  delay(150);
  frontDumpster.detach();

  //turntable
  turntable.attach(9);
  delay(20);
  turntable.write(110);
  delay(150);
  turntable.detach();
  delay(20);

  //cradle
  cradle.attach(3);
  delay(50);
  cradle.write(90);
  delay(100);
  cradle.detach();


  //Start with state 1
  state1Setup();
}

void loop() {
  switch (state) {
    case 1:
      state1();
      break;
    case 2:
      state2();
      break;
    case 3:
      state3();
      break;
    case 4:
      state4();
      break;
    case 5:
      state5();
      break;
    case 6:
      state6();
      break;
    case 7:
      state7();
      break;
    case 8:
      state8();
      break;
    case 9:
      state9();
      break;
    case 10:
      state10();
      break;
    case 11:
      state11();
      break;
  }
}
```

Milestone 5

Motor Functions

```
//  Never change these functions
//  If they have the reversed outcome rewire the platform
//  Do not re-write these functions
void platformForward()
{
  leftForward();
  rightForward();
}
void platformBackward()
{
  leftBackward();
  rightBackward();
}
void platformStop()
{
  leftStop();
  rightStop();
}
void platformSpinLeft()
{
  leftBackward();
  rightForward();
}
void platformSpinRight()
{
  rightBackward();
  leftForward();
}
//left
void leftForward()
{
      digitalWrite(leftA, HIGH);
      digitalWrite(leftB, LOW);
}
void leftBackward()
{
      digitalWrite(leftA, LOW);
      digitalWrite(leftB, HIGH);
}

void leftStop()
{
      digitalWrite(leftA, LOW);
      digitalWrite(leftB, LOW);
}
//right
void rightForward()
{
      digitalWrite(rightA, HIGH);
      digitalWrite(rightB, LOW);
}
void rightBackward()
{
      digitalWrite(rightA, LOW);
      digitalWrite(rightB, HIGH);
}

```

```
void rightStop()
{
      digitalWrite(rightA, LOW);
      digitalWrite(rightB, LOW);
}
```

Milestone 5

State 1

```
//Backwards Until Distance or Time

void state1Setup() {
  platformBackward();
  startTime = millis();
  delay(500);
  state = 1;
}

void state1() {
  //call ultrasonic function
  ultrasonicRead();

  //Check distance to wall
  if ((distanceCm < 8) and (distanceCm > 1)) {
    platformStop();
    delay(20);
    state2Setup();
  }

  //Timeout
  if ((millis() - startTime) > 3000) {
    platformStop();
    delay(20);
    platformForward(); //move away from wall before turning
    delay(200);
    platformStop();
    delay(20);
    state2Setup();
  }
}
```

Milestone 5

State 2

```
//Turn Right into Wall (Time)

void state2Setup() {
  platformSpinRight();
  startTime = millis();
  state = 2;
}

void state2() {

  //Timeout
  if ((millis() - startTime) > 1500) {
    platformStop();
    delay(20);
    state3Setup();
  }
}
```

Milestone 5

State 3

```
//Backward Until Button, or Time

void state3Setup() {
  platformBackward();
  delay(250);
  startTime = millis();
  state = 3;
}

void state3() {

  //Check Button
  backButtonState = digitalRead(backButton);
  if (backButtonState == HIGH)
  {
    platformStop();
    delay(20);
    state4Setup();
  }

  //Timeout
  if ((millis() - startTime) > 5000)
  {
    platformStop();
    delay(20);
    state4Setup();
  }
}
```

Milestone 5

State 4

```
//Turn Left

void state4Setup() {
  platformStop();
  delay(20);
  platformForward(); //Move away from wall before turning
  delay(80);
  platformStop();
  delay(20);
  startTime = millis();
  state = 4;
}

void state4() {

  //Turn Left
  platformSpinLeft();
  delay(200);

  //Stop if Left Button
  if ((digitalRead(leftButton)) == HIGH) {
    state5Setup();
  }

  //Stop if Time
  if ((millis() - startTime) > 3000) {
    state5Setup();
  }
}
```

Milestone 5

State 5

```
//Positioning into Corner

void state5Setup() {
  platformStop();
  delay(20);
  startTime = millis();
  state = 5;
}

void state5() {
  unsigned long currentTime = millis();

  //Forward
  platformForward();
  delay(1500);
  platformStop();
  delay(20);

  //Position by only moving one wheel
  rightBackward();
  leftStop();
  delay(200);

  platformStop();
  delay(20);

  state6Setup();
}
```

Milestone 5

State 6

```
//Parallel Parking

void state6Setup() {
  platformStop();
  delay(20);
  startTime = millis();
  state = 6;
  platformBackward();
}

void state6() {

  //Buttons were extremely unreliable for this, so we used time
  if ((millis() - startTime) > 2000) {
    platformStop();
    delay(20);
    rightForward(); //Control individual wheels
    delay(500);
    platformStop();
    delay(20);
    platformBackward(); //back all the way into corner to align with first block
    delay(1200);
    platformStop();
    delay(20);
    rightForward(); //Controlling the wheels allowed us to turn without spinning
    leftStop();
    delay(150);
    state7Setup();
  }
}
```

Milestone 5

State 7

```
//Get the block

void state7Setup() {
  platformStop();
  delay(20);
  state = 7;
}

void state7() {

  //Sniper flicks block off wall
  sniper.attach(6); //attaches sniper to pin 6
  delay(100);
  sniper.write(70); //moves down lower so it doesn't just smack it at full speed
  delay(500);
  sniper.write(50); //flicks block in cradle
  delay(500);
  sniper.write(90); //starting position
  delay(500);
  sniper.detach();


  //wiggle cradle
  cradle.attach(3); //attaches to pin 3
  delay(100);
  cradle.write(65); //wiggles
  delay(300);
  cradle.write(90); //moves back to original spot
  delay(200);
  cradle.detach();
  delay(50);

  state8Setup();
}
```

Milestone 5

State 8

```
//Sensing
void state8Setup() {
  platformStop();
  delay(20);

  //add block to counter
  blockCounter = blockCounter + 1;
  state = 8;
}

void state8() {

  delay(500);  // make sure block is settled after wiggle
  sensorValue = analogRead(analogInPin); //sense the color

  //White Block
  if (sensorValue < 149) {
    state10Setup();
  }

  //Black Block
  else if (150 < sensorValue) {
    state9Setup();
  }
}
```

Milestone 5

State 9

```
//Black Block

void state9Setup() {
  platformStop();
  delay(20);
  state = 9;
}

void state9() {

  //receive block from cradle
  cradle.attach(3); //attaches cradle to pin 3
  delay(100);
  cradle.write(5); //throws block from cradle to dumpster
  delay(250);
  cradle.write(90);  //starting position
  delay(250);
  cradle.detach();
  delay(20);

  //lift dumpster to move blocks
  backDumpster.attach(10);
  delay(20);
  backDumpster.write(95);
  delay(100);

  //put dumpster down gently
  for ( int pos = 95; pos >= 60; pos -= 1) {
    backDumpster.write(pos);
    delay(20);
  }

  //scooch forward
  leftStop();
  rightForward();  //turns left to get back against back wall
  delay(100);
  platformStop();
  delay(20);
  platformForward();  //moves forward to get to next block
  delay(55);         //if bats are low, put at 120
  platformStop();
  delay(20);
  leftStop();
  rightForward();  //turns left to get back against back wall again just incase
  delay(100);
  platformStop();
  delay(20);

  //Depending on the counter, get another block, or bring the blocks to the bin
  if (blockCounter > 6) {
    state11Setup();
  }
  else {
    state7Setup();
  }
}
```

Milestone 5

State 10

```
//White Block

void state10Setup() {
  platformStop();
  delay(20);
  state = 10;
}

void state10() {
  //move away from wall
  platformSpinRight();
  delay(250);
  platformStop();
  delay(20);

  //spin the turntable
  turntable.attach(9);
  delay(100);
  turntable.write(25);
  delay(500);
  turntable.detach();

  //bring down the front dumpster
  frontDumpster.attach(11);
  delay(100);
  frontDumpster.write(52);
  delay(500);
  frontDumpster.detach();

  //Empty the cradle
  cradle.attach(3); //attaches cradle to pin 3
  delay(100);
  cradle.write(5); //throws block from cradle to dumpster
  delay(500);
  cradle.write(90);  //starting position
  delay(250);
  cradle.detach();
  delay(50);

  //bring up dumpster gently
  frontDumpster.attach(11);
  delay(100);
  for ( int pos = 52; pos <= 110; pos += 1) {
    frontDumpster.write(pos);
    delay(10);
  }
  frontDumpster.detach();

  //swing the turntable back
  turntable.attach(9);
  delay(100);
  turntable.write(110);
  delay(500);
  turntable.detach();

  //turn back
  platformSpinLeft();
  delay(500);
  platformStop();
  delay(20);

  //scooch forward
  leftStop();
  rightForward();  //turns right to get back against back wall
  delay(100);
  platformStop();
  delay(20);
  platformForward();  //moves forward to get to next block
  delay(45);
  platformStop();
  delay(20);
  leftStop();
  rightForward();  //turns right to get back against back wall again just in case
  delay(100);
  platformStop();
  delay(20);

  //Depending on the counter, get another block, or bring the blocks to the bin
  if (blockCounter > 6) {
    state11Setup();
  }
  else {
    state7Setup();
  }
}
```

Milestone 5

State 11

```
//Delivery

void state11Setup() {

  //lower the front dumpster a bit so we don't lose blocks in transport
  frontDumpster.attach(11);
  delay(100);
  frontDumpster.write(90);
  delay(500);
  frontDumpster.detach();

  //drive backwards
  platformBackward();
  startTime = millis();
  tone(buzzerPin, 2000, 250);
  Serial.print("State 11");
  state = 11;
}

void state11() {

  //Reverse into wall
  if ((digitalRead(backButton) == HIGH) or ((millis() - startTime) > 3000)) {
    platformStop();
    delay(20);
    platformForward();
    delay(100);
    platformSpinRight();
    delay(2000);
    platformForward();
    delay(5000);
    platformStop();
    delay(20);

    //Dump the blocks
    frontDumpster.attach(11);
    delay(100);
    frontDumpster.write(160);
    delay(500);
    frontDumpster.write(110);
    delay(500);
    frontDumpster.detach();
    delay(20000);
  }
}
```

Final Project Demo

Michael Sherman & Michaela Curcio

```
//Final Project Demo Code
//Author 2: Michaela Curcio

//Set pin numbers:
const int leftA =  5;        // Left Motor A pin
const int leftB =  4;        // Left Motor B pin
const int rightA =  8;       // Right Motor A pin
const int rightB =  7;       // Right Motor B pin
const int analogInPin = A0;  //Color sensor
const int buzzerPin = 12;    //buzzer
const int leftButton = A2;   //Left Button Pin
const int backButton = A1;   //Back Button Pin

//Initialize Button States and
int backButtonState = 0;
int leftButtonState = 0;
int sensorValue = 0;

// Variables will change:
int blockCounter = 0;      // count number of blocks sniped
int state = 0;             // variable to hold current state
unsigned long startTime;   // will store the time the state was setup

//Setup Servos
#include <Servo.h>
Servo backDumpster;   //Dumpster
Servo cradle;         //Cradle
Servo sniper;         //Sniper
Servo frontDumpster;  //front dumpster
Servo turntable;      //turntable

//ultrasonic
const int trigPin = 13;
const int echoPin = A3;
long duration;
float distanceCm;

//Run this to read from ultrasonic
void ultrasonicRead() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duration = pulseIn(echoPin, HIGH);
  distanceCm = duration * 0.034 / 2;
}

void setup() {
  // set the digital pins as output:
  pinMode(leftA, OUTPUT);
  pinMode(leftB, OUTPUT);
  pinMode(rightA, OUTPUT);
  pinMode(rightB, OUTPUT);
  pinMode(leftButton, INPUT);
  pinMode(backButton, INPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  //Begin Servos in the Right Spot
  //sniper
  sniper.attach(6);
  delay(20);
  sniper.write(165);
  delay(250);
  sniper.detach();
  delay(20);

  //back dumpster
  backDumpster.attach(10);
  delay(50);
  backDumpster.write(60);
  delay(150);
  backDumpster.detach();

  //front dumpster
  frontDumpster.attach(11);
  delay(50);
  frontDumpster.write(110);
  delay(150);
  frontDumpster.detach();

  //turntable
  turntable.attach(9);
  delay(20);
  turntable.write(110);
  delay(150);
  turntable.detach();
  delay(20);

  //cradle
  cradle.attach(3);
  delay(50);
  cradle.write(90);
  delay(100);
  cradle.detach();

  Serial.begin(9600);

  //Go to state 1
  Doubler();
}

void loop() {
  switch (state) {
    case 0:
      Doubler();
      break;
    case 1:
      S01Backup();
      break;
    case 2:
      S02TurnRight();
      break;
    case 3:
      S03Backup();
      break;
    case 4:
      S04TurnLeft();
      break;
    case 5:
      S05ToCorner();
      break;
    case 6:
      S06ParallelPark();
      break;
    case 7:
      S07GoToBlock();
      break;
    case 8:
      S08SenseBlock();
      break;
    case 9:
      S09Black();
      break;
    case 10:
      S10White();
      break;
    case 11:
      S11WhiteDelivery();
      break;
    case 12:
      S12BlackDelivery();
      break;
    case 13:
      S13Turning();
      break;
    case 14:
      S14Backup();
      break;
    case 15:
      S15TurnRight();
      break;
    case 16:
      S16Backup();
      break;
    case 17:
      S17TurnLeft();
      break;
    case 18:
      S18ToCorner();
      break;
    case 19:
      S19ParallelPark();
      break;
    case 20:
      S20GoToBlock();
      break;
    case 21:
      S21SenseBlock();
      break;
    case 22:
      S22Black();
      break;
    case 23:
      S23White();
      break;
    case 24:
      S24StartingPos();
      break;
    case 25:
      S25WhiteDelivery();
      break;
    case 26:
      S26BlackDelivery();
      break;
    }
}
```

Final Project Demo

motorFunctions

```
//  Never change these functions
//  If they have the reversed outcome rewire the platform
//  Do not re-write these functions
void platformForward()
{
  leftForward();
  rightForward();
}
void platformBackward()
{
  leftBackward();
  rightBackward();
}
void platformStop()
{
  leftStop();
  rightStop();
}
void platformSpinLeft()
{
  leftBackward();
  rightForward();
}
void platformSpinRight()
{
  rightBackward();
  leftForward();
}
//left
void leftForward()
{
      digitalWrite(leftA, HIGH);
      digitalWrite(leftB, LOW);
}
void leftBackward()
{
      digitalWrite(leftA, LOW);
      digitalWrite(leftB, HIGH);
}

void leftStop()
{
      digitalWrite(leftA, LOW);
      digitalWrite(leftB, LOW);
}
//right
void rightForward()
{
      digitalWrite(rightA, HIGH);
      digitalWrite(rightB, LOW);
}
void rightBackward()
{
      digitalWrite(rightA, LOW);
      digitalWrite(rightB, HIGH);
}

```

```
void rightStop()
{
      digitalWrite(rightA, LOW);
      digitalWrite(rightB, LOW);
}
```

Final Project Demo

Doubler

```
void DoublerSetup() {
  state = 0;
}

void Doubler() {

  //back up & place againt divider wall
  platformBackward();
  delay(900);
  platformStop();
  delay(20);
  platformSpinRight();
  delay(200);
  platformStop();
  delay(20);
  platformBackward();
  delay(1300);

  leftStop();
  rightForward();
  delay(400);

  platformBackward();
  delay(900);


  //position sniper into correct spot
  sniper.attach(6); //attaches sniper to pin 6
  delay(50);
  sniper.write(55); //position to get doubler
  delay(500);
  sniper.detach();

  //cradle
  cradle.attach(3);
  delay(50);
  cradle.write(85);
  delay(100);
  cradle.detach();

  //Drive by for doubler
  //give 'em the shimm
  startTime = millis();
  platformForward();
  delay(300);
  while ((millis() - startTime) < 2000) {
    platformForward();
    delay(170);
    platformSpinLeft();
    delay(100);
  }
  platformForward();
  delay(2000);
  platformStop();
  delay(20);

  //fix sniper
  sniper.attach(6);
  delay(20);
  sniper.write(165);
  delay(250);
  sniper.detach();
  delay(20);

  //cradle
  cradle.attach(3); //attaches cradle to pin 3
  delay(100);
  cradle.write(5); //throws doubler from cradle to dumpster
  delay(250);
  cradle.write(90);  //starting position
  delay(250);
  cradle.detach();
  delay(20);

  //turn to get to side wall
  platformSpinLeft();
  delay(475);
  platformStop();
  delay(20);

  platformStop();
  delay(20);
  S01BackupSetup();
}
```

Final Project Demo

S01Backup

```
//Backwards Until Distance or Time

void S01BackupSetup() {
  platformBackward();  //go backwards
  startTime = millis();
  delay(900);
  state = 1;
}

void S01Backup() {
  ultrasonicRead();
  Serial.println(distanceCm);

  //Check distance to wall
  if ((distanceCm < 8) and (distanceCm > 1)) {
    platformStop();
    delay(20);
    S02TurnRightSetup();
  }

  //Timeout
  if ((millis() - startTime) > 2500) {
    platformStop();
    delay(20);
    platformForward();
    delay(200);
    platformStop();
    delay(20);
    S02TurnRightSetup();
  }
}
```

Final Project Demo

S02TurnRight

```
//Turn Right into Wall (Time)

void S02TurnRightSetup() {
  platformSpinRight();
  startTime = millis();
  state = 2;
}

void S02TurnRight() {

  //Timeout
  if ((millis() - startTime) > 700) {
    platformStop();
    delay(20);
    S03BackupSetup();
  }
}
```

Final Project Demo

S03Backup

```
//Backward Until Button, or Time

void S03BackupSetup() {
  platformBackward();
  delay(250);
  startTime = millis();
  state = 3;
}

void S03Backup() {
  unsigned long currentTime = millis();

  //Dump that bad boy (the doubler)
  if ((currentTime - startTime) > 750)
  {
    backDumpster.attach(10);
    delay(100);
    backDumpster.write(160);
    delay(600);
    backDumpster.write(60);
    delay(600);
    backDumpster.detach();
    delay(20);
  }

  //Check Button
  backButtonState = digitalRead(backButton);
  if (backButtonState == HIGH)
  {
    platformStop();
    delay(20);
    S04TurnLeftSetup();
  }

  //Timeout
  if ((currentTime - startTime) > 3500)
  {
    platformStop();
    delay(20);
    S04TurnLeftSetup();
  }
}
```

Final Project Demo

S04TurnLeft

```
//Turn Left

void S04TurnLeftSetup() {
  platformStop();
  delay(20);
  platformForward();
  delay(80);
  platformStop();
  delay(20);
  startTime = millis();
  state = 4;
}

void S04TurnLeft() {

  platformSpinLeft();
  delay(300);
  platformStop();
  delay(50);

  //Stop if Button
  if ((digitalRead(leftButton)) == HIGH) {
    S05ToCornerSetup();
  }

  //Stop if Time
  if ((millis() - startTime) > 2500) {
    S05ToCornerSetup();
  }
}
```

Final Project Demo

S05ToCorner

```
//Positioning into Corner

void S05ToCornerSetup() {
  platformStop();
  delay(20);
  state = 5;
}

void S05ToCorner() {

  //Forward front of bot on divider
  platformForward();
  delay(1500);
  platformStop();
  delay(20);

  //Position by only moving one wheel to put left wheel aginst back wall
  rightBackward();
  leftStop();
  delay(150);

  S06ParallelParkSetup();
}
```

Final Project Demo

S06ParallelPark

```
//Parallel Parking

void S06ParallelParkSetup() {
  platformStop();
  delay(20);
  startTime = millis();
  state = 6;
}

void S06ParallelPark() {
  unsigned long currentTime;
  currentTime = millis();

  //Parallel Parking into corner
  //Buttons were extremely unreliable for this, so we used time
  platformBackward();
  backButtonState = digitalRead(backButton);
  if (backButtonState == HIGH)
  {
    platformStop();
    delay(20);
    rightForward();
    delay(500);
    platformStop();
    delay(20);
    platformBackward();
    delay(1200);
    platformStop();
    delay(20);
    rightForward(); //Controlling the wheels allowed us to turn without spinning
    leftStop();
    delay(150);
    S07GoToBlockSetup();
  }

  //Timeout
  currentTime = millis();
  if ((currentTime - startTime) > 2000) {
    platformStop();
    delay(20);
    rightForward();
    delay(500);
    platformStop();
    delay(20);
    platformBackward();
    delay(1200);
    platformStop();
    delay(20);
    rightForward(); //Controlling the wheels allowed us to turn without spinning
    leftStop();
    delay(150);
    S07GoToBlockSetup();
  }
}
```

Final Project Demo

S07GoToBlock

```
//Get the block

void S07GoToBlockSetup() {
  platformStop();
  delay(20);
  state = 7;
}

void S07GoToBlock() {

  //Sniper flicks block off wall
  sniper.attach(6); //attaches sniper to pin 6
  delay(50);
  sniper.write(70); //moves down lower so it doesn't just smack it at full speed
  delay(250);
  sniper.write(50); //flicks block in cradle
  delay(500);
  sniper.write(90); //starting position
  delay(500);
  sniper.detach();


  //wiggle cradle
  cradle.attach(3); //attaches to pin 3
  delay(50);
  cradle.write(65); //wiggles
  delay(250);
  cradle.write(90); //moves back to original spot
  delay(250);
  cradle.detach();
  delay(50);

  S08SenseBlockSetup();
}
```

Final Project Demo

S08SenseBlock

```
//Sensing
void S08SenseBlockSetup() {
  platformStop();
  delay(20);

  //add block to counter
  blockCounter = blockCounter + 1;
  Serial.println(blockCounter);
  state = 8;
}

void S08SenseBlock() {

  // wait for block to fall and read the analog in value:
  delay(300);
  sensorValue = analogRead(analogInPin);

  // print the results to the Serial Monitor:
  Serial.print("sensor = ");
  Serial.println(sensorValue);

  //White Block
  if (sensorValue < 149) {
    S10WhiteSetup();
  }

  //Black Block
  else if (150 < sensorValue) {

    S09BlackSetup();
  }
}
```

Final Project Demo

S09Black

```
//Black Block

void S09BlackSetup() {
  platformStop();
  delay(20);
  state = 9;
}

void S09Black() {

  //receive block from cradle
  cradle.attach(3); //attaches cradle to pin 3
  delay(100);
  cradle.write(5); //throws block from cradle to dumpster
  delay(250);
  cradle.write(90);  //starting position
  delay(250);
  cradle.detach();
  delay(20);

  //lift dumpster to move blocks
  backDumpster.attach(10);
  delay(20);
  backDumpster.write(95);
  delay(100);
  //put it down gently
  for ( int pos = 95; pos >= 60; pos -= 1) {
    backDumpster.write(pos);
    delay(20);
  }


  //scooch
  leftStop();
  rightForward();  //turns left to get back against back wall
  delay(100);
  platformStop();
  delay(20);
  platformForward();  //moves forward to get to next block
  delay(40);
  platformStop();
  delay(20);
  leftStop();
  rightForward();  //turns left to get back against back wall again just incase
  delay(100);
  platformStop();
  delay(20);

  if (blockCounter > 7) {
    S11WhiteDeliverySetup();
  }
  else {
    S07GoToBlockSetup();
  }
}
```

Final Project Demo

S10White

```
//White Block

void S10WhiteSetup() {
  platformStop();
  delay(20);
  state = 10;
}

void S10White() {
  //move away from wall
  platformSpinRight();
  delay(250);
  platformStop();
  delay(20);

  //spin the turntable
  turntable.attach(9);
  delay(100);
  turntable.write(25);
  delay(500);
  turntable.detach();

  //bring down the front dumpster
  frontDumpster.attach(11);
  delay(100);
  frontDumpster.write(52);
  delay(500);
  frontDumpster.detach();

  //Empty the cradle
  cradle.attach(3); //attaches cradle to pin 3
  delay(100);
  cradle.write(5); //throws block from cradle to dumpster
  delay(500);
  cradle.write(90);  //starting position
  delay(250);
  cradle.detach();
  delay(50);

  //bring up the front dumpster
  //bring up gently
  frontDumpster.attach(11);
  delay(100);
  for ( int pos = 52; pos <= 110; pos += 1) {
    frontDumpster.write(pos);
    delay(10);
  }
  frontDumpster.detach();

  //swing the turntable back
  turntable.attach(9);
  delay(100);
  turntable.write(110);
  delay(500);
  turntable.detach();

  //turn back
  platformSpinLeft();
  delay(500);
  platformStop();
  delay(20);

  //scooch
  leftStop();
  rightForward();  //turns right to get back against back wall
  delay(100);
  platformStop();
  delay(20);
  platformForward();  //moves forward to get to next block
  delay(30);
  platformStop();
  delay(20);
  leftStop();
  rightForward();  //turns right to get back against back wall again just in case
  delay(100);
  platformStop();
  delay(20);

  if (blockCounter > 7) {
    S11WhiteDeliverySetup();
  }
  else {
    S07GoToBlockSetup();
  }
}
```

Final Project Demo

S11WhiteDelivery

```
//Delivery

void S11WhiteDeliverySetup() {

//lower the front dumpster a bit so we don't lose blocks in transport
  frontDumpster.attach(11);
  delay(100);
  frontDumpster.write(90);
  delay(250);
  frontDumpster.detach();

  platformBackward();
  startTime = millis();
  state = 11;
}

void S11WhiteDelivery() {

  //Reverse for distance or time
  if ((digitalRead(backButton) == HIGH ) or ((millis() - startTime) > 2000)) {
    platformStop();
    delay(20);
    platformForward();
    delay(100);
    platformSpinRight();
    delay(2000);
    platformForward();
    delay(3000);
    platformStop();
    delay(20);

    //Dump it
    frontDumpster.attach(11);
    delay(100);
    frontDumpster.write(160);
    delay(300);
    frontDumpster.write(110);
    delay(300);
    frontDumpster.detach();
    delay(20);

    S12BlackDeliverySetup();
  }
}
```

Final Project Demo

S12BlackDelivery

```
//Delivery on other side (black)

void S12BlackDeliverySetup() {
  platformBackward();
  delay(300);
  state = 12;
  startTime = millis();
}

void S12BlackDelivery() {
  platformStop();
  delay(20);

  //Give 'em the shimm
  while ((millis() - startTime < 1600)) {

    platformSpinLeft();
    delay(200);
    platformForward();
    delay(100);
  }

  platformStop();
  delay(20);
  platformForward();
  delay(2500);
  platformStop();
  delay(20);
  platformBackward();
  delay(200);
  platformStop();
  delay(20);

  //turning after got to black side in order to ride black wall
  platformSpinLeft();
  delay(400);
  platformForward();
  delay(400);

  platformStop();
  delay(20);

  //Give 'em the shimm, but mostly forward
  startTime = millis();
  while ((millis() - startTime < 1600))
  {
    platformSpinRight();
    delay(50);
    platformForward();
    delay(200);
  }
  //stop so she doesn't ride the wall and still gets to perfect angle
    platformStop();
    delay(20);

  //Dump it
  backDumpster.attach(10);
  delay(100);
  backDumpster.write(160);
  delay(600);
  backDumpster.write(60);
  delay(600);
  backDumpster.detach();
  delay(20);

  S13TurningSetup();
}
```

Final Project Demo

S13Turning

```
//Back up to turn

void S13TurningSetup() {
  state = 13;
}

void S13Turning() {

//doesn't go back as much because it stops beforehand (and doesn't go forward)
  platformBackward();
  delay(200);

  platformSpinLeft();
  delay(800);
  platformForward();
  delay(400);


  platformSpinRight();
  delay(1600);

  S14BackupSetup();
}
```

Final Project Demo

S14Backup

```
//Backwards Until Distance or Time

void S14BackupSetup() {
  platformBackward();
  startTime = millis();
  delay(500);
  state = 14;
}

void S14Backup() {

  ultrasonicRead();
  Serial.println(distanceCm);

  //Check distance to wall
  if ((distanceCm < 8) and (distanceCm > 1)) {
    platformStop();
    delay(20);
    S15TurnRightSetup();
  }


  //Timeout
  if ((millis() - startTime) > 2000) {
    platformStop();
    delay(20);
    platformForward();
    delay(100);
    platformStop();
    delay(20);
    S15TurnRightSetup();
  }
}
```

Final Project Demo

S15TurnRight

```
//Turn Right into Wall (Time)

void S15TurnRightSetup() {
  platformSpinRight();
  startTime = millis();
  state = 15;
}

void S15TurnRight() {

  //Timeout
  if ((millis() - startTime) > 700) {
    platformStop();
    delay(20);
    S16BackupSetup();
  }
}
```

Final Project Demo

S16Backup

```
//Backward Until Button, or Time

void S16BackupSetup() {
  platformBackward();
  delay(250);
  startTime = millis();
  state = 16;
}

void S16Backup() {
  unsigned long currentTime = millis();

  //Check Button
  backButtonState = digitalRead(backButton);
  if (backButtonState == HIGH)
  {
    platformStop();
    delay(20);
    S17TurnLeftSetup();
  }

  //Timeout
  if ((currentTime - startTime) > 2000)
  {
    platformStop();
    delay(20);
    S17TurnLeftSetup();
  }
}
```

Final Project Demo

S17TurnLeft

```
//Turn Left

void S17TurnLeftSetup() {
  platformStop();
  delay(20);
  platformForward();
  delay(80);
  platformStop();
  delay(20);
  startTime = millis();
  state = 17;

}

void S17TurnLeft() {

  platformSpinLeft();

  //Stop if Button
  if ((digitalRead(leftButton)) == HIGH) {
    S18ToCornerSetup();
  }

  //Stop if Time
  if ((millis() - startTime) > 3000) {
    S18ToCornerSetup();
  }
}
```

Final Project Demo

S18ToCorner

```
//Positioning into Corner

void S18ToCornerSetup() {
  platformStop();
  delay(20);
  state = 18;
}

void S18ToCorner() {

  //Forward
  platformForward();
  delay(1500);
  platformStop();
  delay(20);

  //Position by only moving one wheel
  rightBackward();
  leftStop();
  delay(150);

  platformStop();
  delay(20);

  S19ParallelParkSetup();
}
```

Final Project Demo

S19ParallelParking

```
//Parallel Parking

void S19ParallelParkSetup() {
  platformStop();
  delay(20);
  startTime = millis();
  state = 19;
}

void S19ParallelPark() {
  unsigned long currentTime;
  blockCounter = 0;

  //Parallel Parking into corner
  platformBackward();

  //Timeout
  currentTime = millis();
  if ((currentTime - startTime) > 2000) {
    platformStop();
    delay(20);
    rightForward();
    delay(500);
    platformStop();
    delay(20);
    platformBackward();
    delay(1200);
    platformStop();
    delay(20);
    rightForward(); //Controlling the wheels allowed us to turn without spinning
    leftStop();
    delay(150);

    //gets into position right in front of first block
    platformStop();
    delay(20);
    platformForward();
    delay(155);
    platformStop();
    delay(20);

  S20GoToBlockSetup();
}
}
```

Final Project Demo

S20GoToBlock

```
//Get the block

void S20GoToBlockSetup() {
  platformStop();
  delay(20);
  state = 20;
}

void S20GoToBlock() {

  //Sniper flicks block off wall
  sniper.attach(6); //attaches sniper to pin 6
  delay(20);
  sniper.write(70); //moves down lower so it doesn't just smack it at full speed
  delay(500);
  sniper.write(50); //flicks block in cradle
  delay(300);
  sniper.write(90); //starting position
  delay(300);
  sniper.detach();
  delay(20);


  //wiggle cradle
  cradle.attach(3); //attaches to pin 3
  delay(20);
  cradle.write(65); //wiggles
  delay(250);
  cradle.write(90); //moves back to original spot
  delay(250);
  cradle.detach();
  delay(20);

  S21SenseBlockSetup();

}
```

Final Project Demo

S21SenseBlock

```
//Sensing
void S21SenseBlockSetup() {
  platformStop();
  delay(20);

  //add block to counter
  blockCounter = blockCounter + 1;
  Serial.println(blockCounter);
  state = 21;
}

void S21SenseBlock() {

  //wait for block to settle and read the analog in value:
  delay(500);
  sensorValue = analogRead(analogInPin);

  // print the results to the Serial Monitor:
  Serial.print("sensor = ");
  Serial.println(sensorValue);

  //White Block
  if (sensorValue < 149) {
    S23WhiteSetup();
  }

  //Black Block
  else if (150 < sensorValue) {

    S22BlackSetup();
  }
}
```

Final Project Demo

S22Black

```
//Black Block

void S22BlackSetup() {
  platformStop();
  delay(20);
  state = 22;
}

void S22Black() {

  //receive block from cradle
  cradle.attach(3); //attaches cradle to pin 3
  delay(100);
  cradle.write(5); //throws block from cradle to dumpster
  delay(250);
  cradle.write(90);  //starting position
  delay(250);
  cradle.detach();
  delay(20);

  //lift dumpster to move blocks
  backDumpster.attach(10);
  delay(20);
  backDumpster.write(95);
  delay(100);
  //put it down gently
  for ( int pos = 95; pos >= 60; pos -= 1) {
    backDumpster.write(pos);
    delay(20);
  }

  //scooch
  leftStop();
  rightForward();  //turns left to get back against back wall
  delay(100);
  platformStop();
  delay(20);
  platformForward();  //moves forward to get to next block
  delay(35);
  platformStop();
  delay(20);
  leftStop();
  rightForward();  //turns left to get back against back wall again just incase
  delay(100);
  platformStop();
  delay(20);

  if (blockCounter > 8) {
    S24StartingPosSetup();
  }
  else {
    S20GoToBlockSetup();
  }
}
```

Final Project Demo

S23White

```
//White Block

void S23WhiteSetup() {
  platformStop();
  delay(20);
  state = 23;
}

void S23White() {
  //move away from wall
  platformSpinRight();
  delay(250);
  platformStop();
  delay(20);

  //spin the turntable
  turntable.attach(9);
  delay(100);
  turntable.write(25);
  delay(500);
  turntable.detach();

  //bring down the front dumpster
  frontDumpster.attach(11);
  delay(100);
  frontDumpster.write(52);
  delay(500);
  frontDumpster.detach();

  //Empty the cradle
  cradle.attach(3); //attaches cradle to pin 3
  delay(100);
  cradle.write(5); //throws block from cradle to dumpster
  delay(500);
  cradle.write(90);  //starting position
  delay(250);
  cradle.detach();
  delay(50);

  //bring up the front dumpster

  //bring up gently
  frontDumpster.attach(11);
  delay(100);
  for ( int pos = 52; pos <= 110; pos += 1) {
    frontDumpster.write(pos);
    delay(10);
  }
  frontDumpster.detach();

  //swing the turntable back
  turntable.attach(9);
  delay(100);
  turntable.write(110);
  delay(500);
  turntable.detach();

  //turn back
  platformSpinLeft();
  delay(500);
  platformStop();
  delay(20);

  //scooch
  leftStop();
  rightForward();  //turns right to get back against back wall
  delay(100);
  platformStop();
  delay(20);
  platformForward();  //moves forward to get to next block
  delay(25);
  platformStop();
  delay(20);
  leftStop();
  rightForward();  //turns right to get back against back wall again just in case
  delay(100);
  platformStop();
  delay(20);

  if (blockCounter > 8) {
    S24StartingPosSetup();
  }
  else {
    S20GoToBlockSetup();
  }
}
```

Final Project Demo

S24StartingPos

```
//Third Delivery

void S24StartingPosSetup() {

  //lower the front dumpster a bit so we don't lose blocks in transport
  frontDumpster.attach(11);
  delay(100);
  frontDumpster.write(90);
  delay(500);
  frontDumpster.detach();

  platformBackward();
  startTime = millis();
  tone(buzzerPin, 2000, 250);
  Serial.print("State 24");
  state = 24;
}

void S24StartingPos() {

//turn to get to divider wall, then forward to get to starting position
  if (((millis() - startTime) > 4000) or  (digitalRead(backButton)==HIGH)){
    platformStop();
    delay(20);
    platformForward();
    delay(100);
    platformSpinRight();
    delay(2000);
    platformForward();
    delay(4500);
    platformStop();
    delay(20);

    //turn to get to white side wall
    platformSpinLeft();
    delay(600);
    platformStop();
    delay(20);

    S25WhiteDeliverySetup();
  }
}
```

Final Project Demo

S25WhiteDelivery

```
//Backwards Until Distance or Time

void S25WhiteDeliverySetup() {
  platformBackward();
  startTime = millis();
  delay(500);
  state = 25;
}

void S25WhiteDelivery() {
  unsigned long currentTime = millis();

  ultrasonicRead();
  Serial.println(distanceCm);

  //Check distance to wall
  if ((distanceCm < 8) and (distanceCm > 1)) {
    platformStop();
    delay(20);

    //spin to move against white wall
    platformSpinRight();
    delay(900);
    platformForward();
    delay(1000);

    //Dump it
    frontDumpster.attach(11);
    delay(100);
    frontDumpster.write(160);
    delay(500);
    frontDumpster.write(110);
    delay(500);
    frontDumpster.detach();
    delay(200);

    S26BlackDeliverySetup();
  }


  //Timeout
  if ((millis() - startTime) > 3000) {
    platformStop();
    delay(20);
    platformForward();
    delay(200);
    platformStop();
    delay(20);

    //spin againt white wall
    platformSpinRight();
    delay(400);
    platformForward();
    delay(1000);

    //Dump it
    frontDumpster.attach(11);
    delay(100);
    frontDumpster.write(160);
    delay(500);
    frontDumpster.write(110);
    delay(500);
    frontDumpster.detach();
    delay(200);

    S26BlackDeliverySetup();
  }
}
```

Final Project Demo

S26BlackDelivery

```
//Delivery on other side (black)

void S26BlackDeliverySetup() {
  platformBackward();
  delay(300);
  state = 26;
  startTime = millis();
}

void S26BlackDelivery() {
  platformStop();
  delay(20);

  //Give 'em the shimm
  while ((millis() - startTime < 1500))
  {
    platformSpinLeft();
    delay(200);
    platformForward();
    delay(100);
  }

  //delay(2000);
  platformStop();
  delay(20);
  platformForward();
  delay(3000);
  platformStop();
  delay(20);
  platformBackward();
  delay(200);
  platformStop();
  delay(20);

  //turn to ride black wall
  platformSpinLeft();
  delay(400);
  platformForward();
  delay(400);

  platformStop();
  delay(20);

  //Give 'em the shimm, but mostly forward

  startTime = millis();
  while ((millis() - startTime < 2000))
  {
    platformSpinRight();
    delay(50);
    platformForward();
    delay(200);
  }

  //Dump it
  backDumpster.attach(10);
  delay(100);
  backDumpster.write(160);
  delay(600);
  backDumpster.write(60);
  delay(600);
  backDumpster.detach();
  delay(200);

  platformStop();
  delay(1000);
}
```