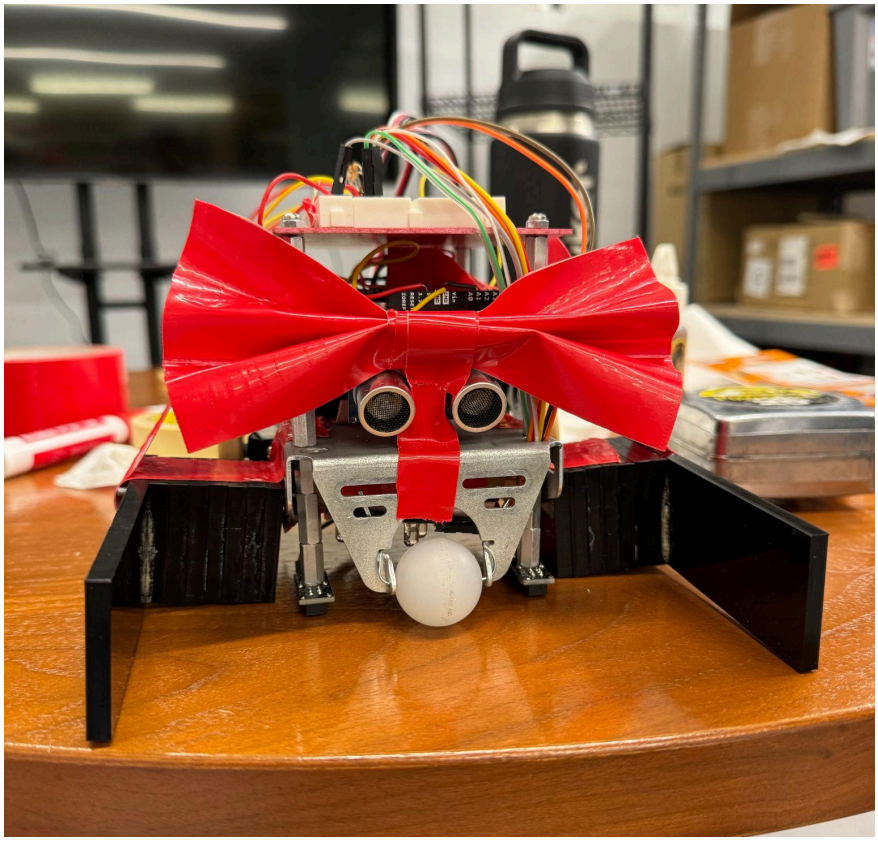
MAE 3780 : Final Report

Team #2 : RAK-Attack

5/16/2024



Team Members: Aleira Sanchez (ans222),
Krista Roessle (kqr3) and
Roberto Garcia (rfg74)

Section I: Robot Design and Strategy Overview

RAK-Attack's design was very simplistic yet efficient. It contained two QTI sensors, one on each front-edge of the robot, one color sensor attached to the bottom of the chassis, and the arms which served as our cube-collecting mechanism. The robot navigated through the field primarily dependent on its sensors. The ideal strategy was to go to the opponent's side of the field, sweep most of the cubes on their side after a few loops, and then return to its starting color (see Appendix D, Figure 5, for more in-depth detail of board strategy). RAK-Attack would read when it was no longer in its starting color and start carrying out our sweep strategy which was hard-coded into the software using the bounds of the field. However, if the QTI sensors detected the robot to be on top of a black edge, then it would override the main code and turn around immediately, this came to be a problem whenever the opposing robot attacked and RAK-Attack's course would be unfortunately altered. For our cube-collecting mechanism, we decided to go with a U-shaped arm design made out of acrylic and attached each *arm* to the side of the robot using pre-made slots in the chassis. The arms were made of a combination of 3D-printed supports and laser-cut acrylic sheets. While we had a lot of ideas for this mechanism, we decided that the simplest option would be the best to carry out this task.

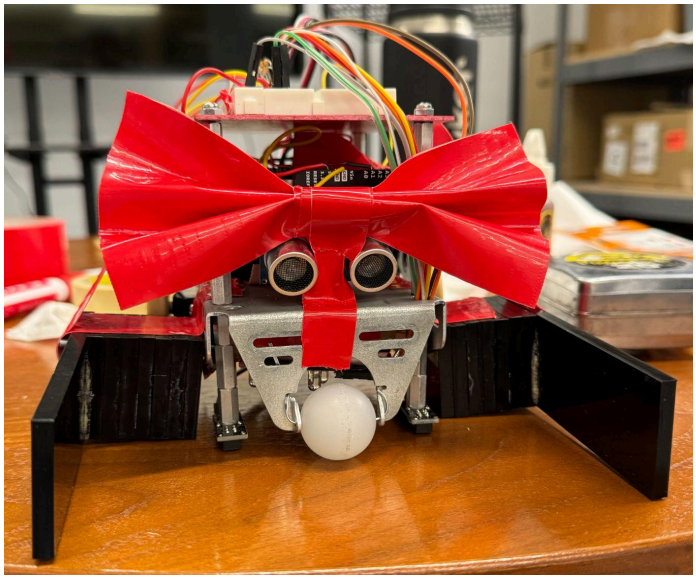


Figure 1: RAK-ATTACK Front View
The sonar sensor pictured was not used

Section II: Design Process Reflection

Our initial design process started with watching videos from prior years' competitions and speaking to last year's students to get a sense of how their robots were successful. From this, we concluded that the most successful robots had a simple cube-collecting mechanism and software configuration. Subsequently, we observed that the risk of contact with the opposing robot increased with the area traveled by the robot, and this could result in the robot failing to perform its intended task. This drove us to design a robot that would sweep the cubes into its U-shaped arms by making a few loops and traversing the board only once. See Appendix D, Figure 5, for more in-depth detail of board strategy.

One of the first problems we faced was programming the code for Milestone 2. We realized that one of our motors was running at a noticeably slower rate than the other. This was fixed by getting a new motor in lab hours and modifying our delays accordingly,allowing us to complete the milestone on time.

When it came to integrating the sensors, programming them and getting them to work consistently was our biggest challenge. For the color sensor, we hard-wired two different filters, on separate trials, and recorded their thresholds for the field's colors. The Red filter had significantly more difference between threshold values leading us to choose it out of the two. The placement for the color sensor was an ongoing struggle throughout the design process. Since it had to be very close to the board for an accurate reading, we cut a piece of cardboard to make a platform for the sensor so it would be able to distinguish the colors more efficiently. However, this set-up would prove rather inconvenient when it came to the attachment of the arms.

With the QTI sensors, at first we only had one to detect the black borders and we placed it in the frontmost part of the robot. This resulted in inaccurate border detection, especially when the robot approached the borders at an angle or at the corners. Therefore, we decided to attach an additional QTI sensor. By placing them in the front corners of the robot, our previous issue was effectively resolved.

However, issues with the sensors still remained. What we thought was initially a software problem ended up being a hardware problem, and we realized that the QTI sensors could not be placed in close proximity to the color sensor because the light from the color sensor interfered with the QTI's reading. The robot would get near one of the borders, be unable to detect it, and run off the board. We fixed this by increasing their QTi's proximity to the board, leaving as small of a gap as possible.

After finishing Milestone 3, our QTI sensors stopped outputting properly. They were working intermittently even after adjusting the position of the color sensor. This problem was due to not using a high enough resistance. After changing the original 10 kOhm resistors to a 50 kOhm equivalent resistance (acquired by putting two 100 kOhm resistors in parallel), the QTI sensors read the correct values consistently.

Our initial board strategy relied on the use of a sonar sensor to detect the opposing robot and move away from the robot if it interfered with our path. Unfortunately, we were unable to get the sensor programmed in time, causing us to modify our strategy. Our updated strategy was based on predicting where the least likely areas of interference would occur and what other teams' possible strategy would look like. Our in-depth board strategy can be found in Appendix D.

After Milestone 4, one of our QTIs stopped recognizing the light colors and would only read black, causing the robot to turn in circles. We found that this was because the sensor got scratched and damaged from repeated handling. We changed out both QTIs, just to be safe, and were significantly more careful with how we handled and set down the robot from that point on to prevent this issue from occurring again before competition.

Section III: Competition Analysis

RAK-Attack at the competition performed as expected, but there were some bumps throughout the tournament. When we were about to commence our first match, the color sensor would not turn on, but this was an issue we had encountered previously. We immediately thought about the 9V battery the TA had just given us and that it could be a dead battery, so we used our back-up battery and it turned on. The robot was able to win a few competitions while fully operational. Since the strategy was majorly hard-coded, there were one or two matches where the robot went out of bounds and wasn't able to fully carry-out the job. However, that was not our main problem on the day of the competition.

Three to four matches passed, and then the right wheel of our robot stopped working. At first, we thought that it was the sensors acting up so we let it run another match to see if it would perform the same, and it did. Once we lost two matches in a row because the robot was not carrying-out the code, we went into checking the wiring and hardware. That is where we realized that the wire that connects the right wheel's motor to the Arduino came off, even though we had added tape on top of all those wires because the same issue had happened in lab hours as well. The likely cause of this issue happening again is due to the constant handling and moving of the robot, which could have disconnected the wire. When we realized, it was too late and we were too low in ranking to get back up again. Nonetheless, RAK-attack was able to win another match before retiring.

Section IV: Conclusions

For the most part, RAK-ATTACK performed according to our preconceived thoughts and standards. If we were to conduct this project again, we would adjust the mechanical design for the arms of our robot. While our design was secure, it was also heavy which slowed down our mobility. Furthermore, we would add a protective casing to the chassis to ensure our wires stayed in place. To optimize performance, we would implement features such as larger wheels and an op-amp in our circuitry to increase speed and ensure we attacked first if we ran into another robot. Ideally, we would also master using the SONAR sensors to prevent collisions and potentially detect blocks to make our robot's path more informed. Our advice to students next year would be to implement new components one-by-one and not make excessive modifications at once. This will help to simplify the debugging process which is crucial when working with deadlines.

Section V: Appendix A

Part Number	Part Name	Vendor/Source	Quantity	Unit Cost	Total Cost
1	QTI Sensor	Lab	1	$2.00	$2.00
2	Left Support	RPL (3D Print)	1	$5.11	$5.11
3	Right Support	RPL (3D Print)	1	$5.11	$5.11
4	Short Arm Piece	RPL (Laser Cut)	14	$0.97	$13.52
5	Long Arm Piece	RPL (Laser Cut)	2	$1.27	$2.54
N/A	Acrylic Sheet	RPL	1	$5.00	$5.00
N/A	Duct Tape	Scavenged	1	$3.00	$3.00
				Total	$36.27

Figure 2: Bill of Materials

Section VI: Appendix B

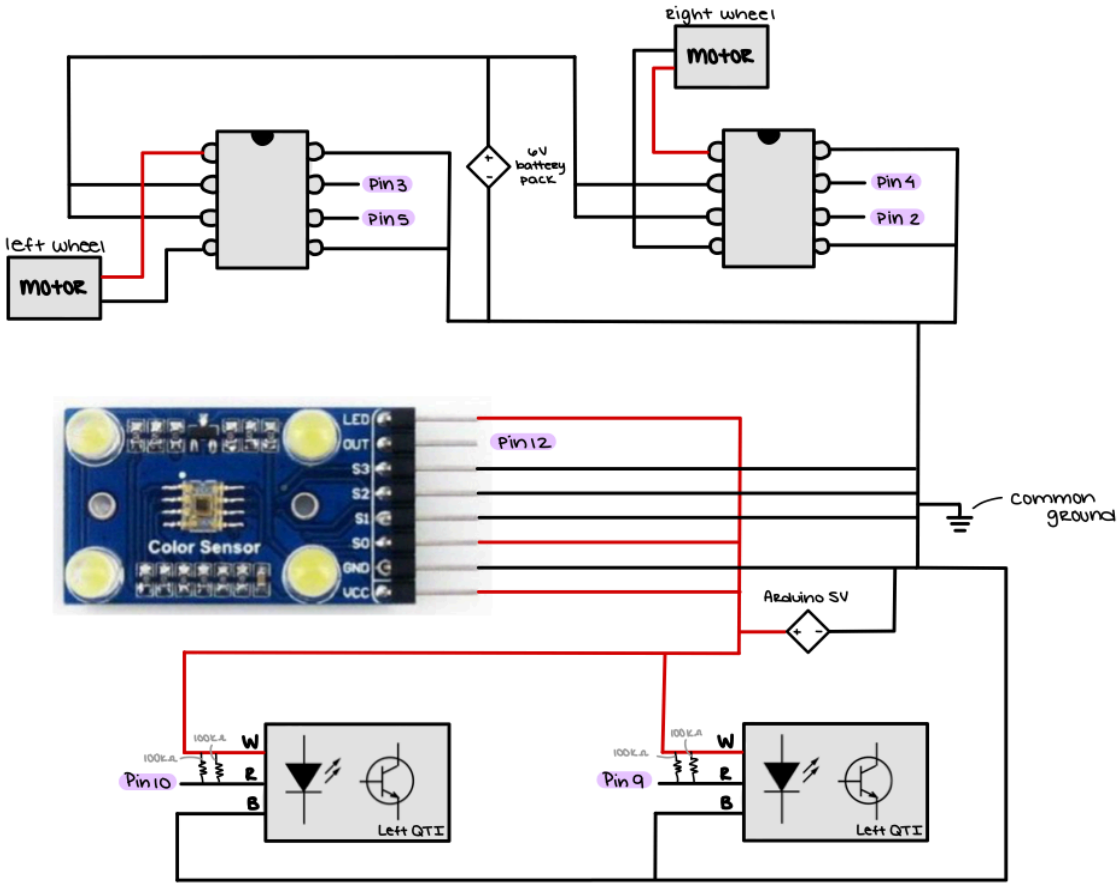


Figure 3: Circuit Diagram

* Each resistor used for the QTIs were 100 kilo-ohms (diagram may be too small to see their values)*

Section VII: Appendix C

Robot CAD

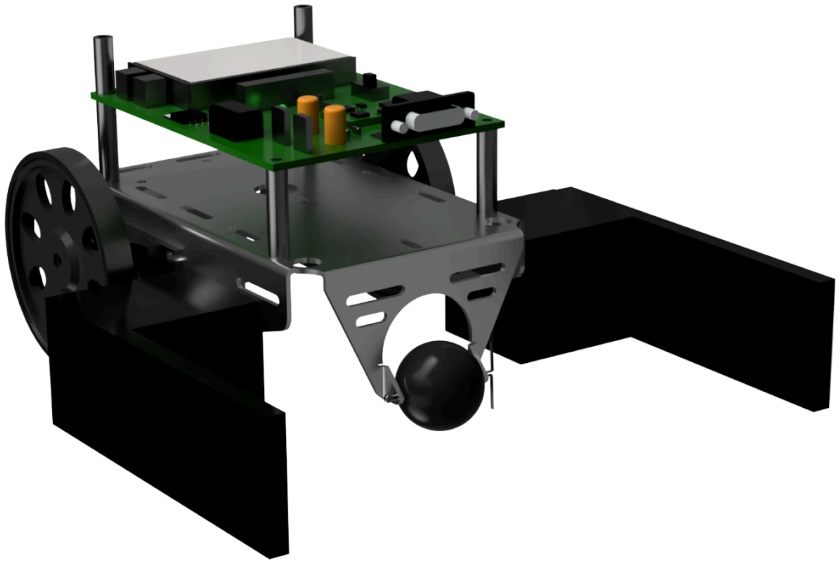


Figure 4: CAD Render of Robot Chassis and U-Shaped Arms

3D-Printed Supports

Figure 5: CAD Renders of Right and Left 3D-Printed Supports

Laser Cut File

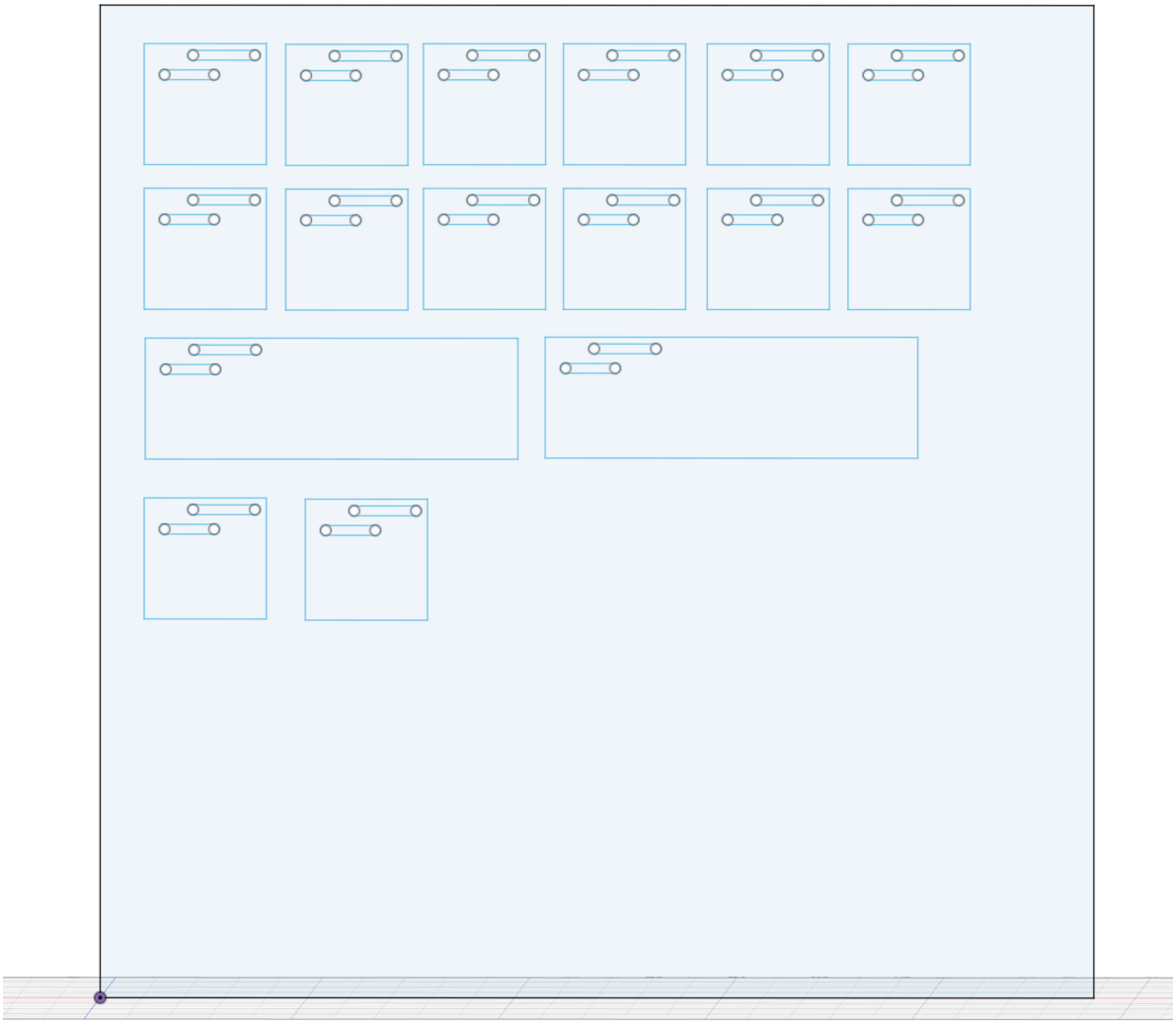


Figure 6: Screenshot of Laser Cut File Submitted to RPL

Part Volumes and Weight Calculations for BOM

Part 2: Left Support

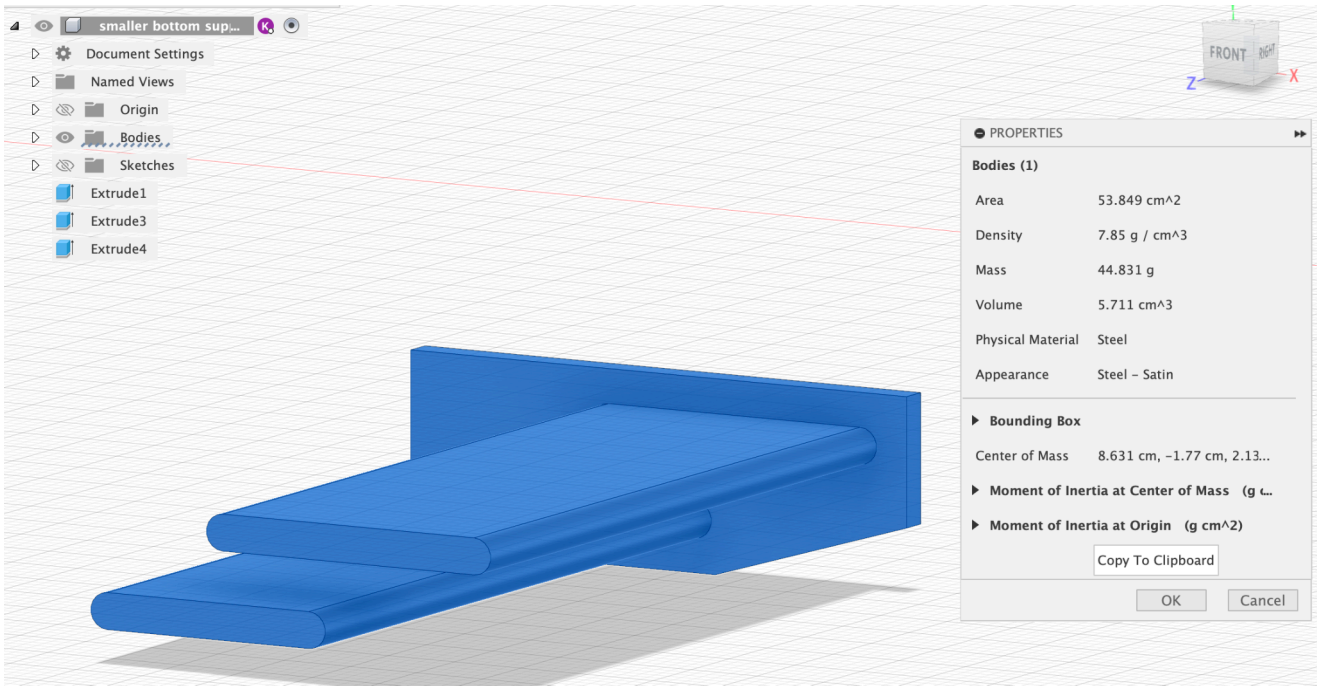


Figure 7: Screenshot of Left Support Body Volume

$$Unit\ Cost\ =\ \$1.50\ +\ \$\,0.50\ \times\ (5.771\ cm^3\ \times\ 1.25\frac{g}{cm^3})\ \times\ 1^*$$

$$Unit\ Cost\ =\ \$5.11$$

** Please note that weight and unit cost calculation is done with a conservative estimate of 100% infill as we didn't specify infill with the RPL when we printed our part.*

Part 3: Right Support

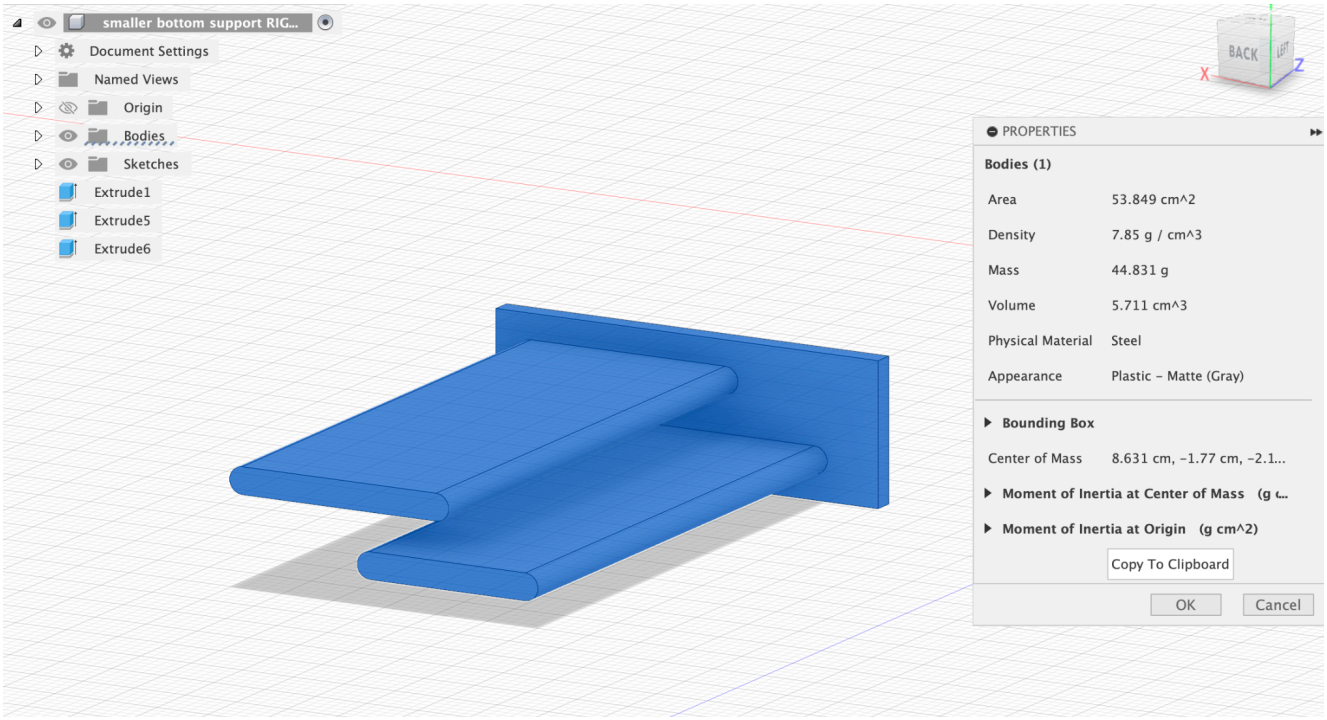


Figure 8: Screenshot of Right Support Body Volume

$$Unit\ Cost\ =\ \$1.50\ +\ \$\,0.50\ \times\ (5.771\ cm^3\ \times\ 1.25\frac{g}{cm^3})\ \times\ 1^*$$

$$Unit\ Cost\ =\ \$5.11$$

** Please note that weight and unit cost calculation is done with a conservative estimate of 100% infill as we didn't specify infill with the RPL when we printed our part.*

Part 4: Short Arm Piece

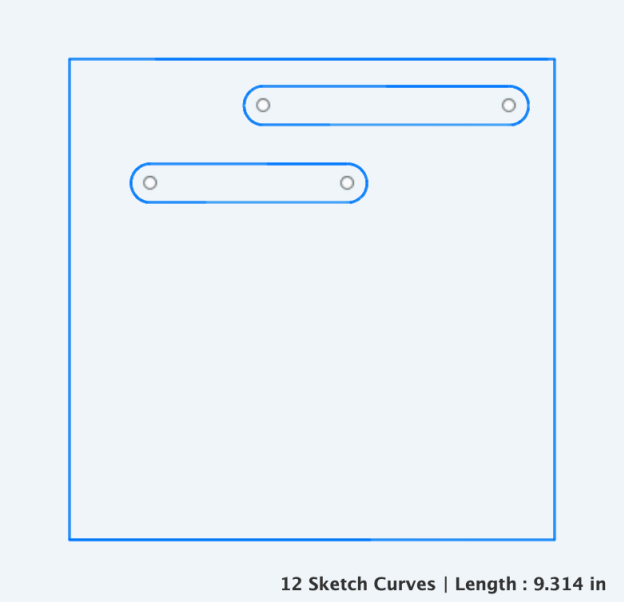


Figure 9: Screenshot of Short Arm Piece Total Sketch Perimeter

$$Unit\ Cost\ =\ \$0.50\ +\ (9.314\ in\ \times\ \$0.05)$$
$$Unit\ Cost\ =\ \$0.97$$

Part 5: Long Arm Piece

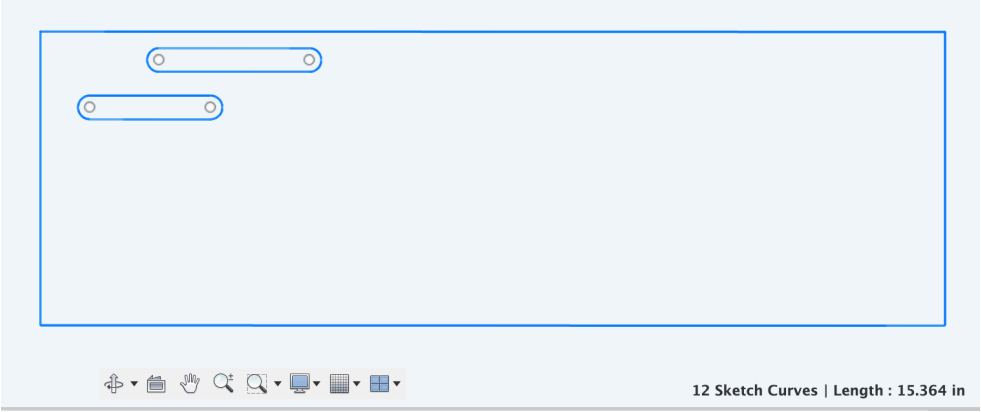


Figure 10: Screenshot of Long Arm Piece Total Sketch Perimeter

$$Unit\ Cost\ =\ \$0.50\ +\ (15.364\ in\ \times\ \$0.05)$$
$$Unit\ Cost\ =\ \$1.27$$

Section VIII: Appendix D

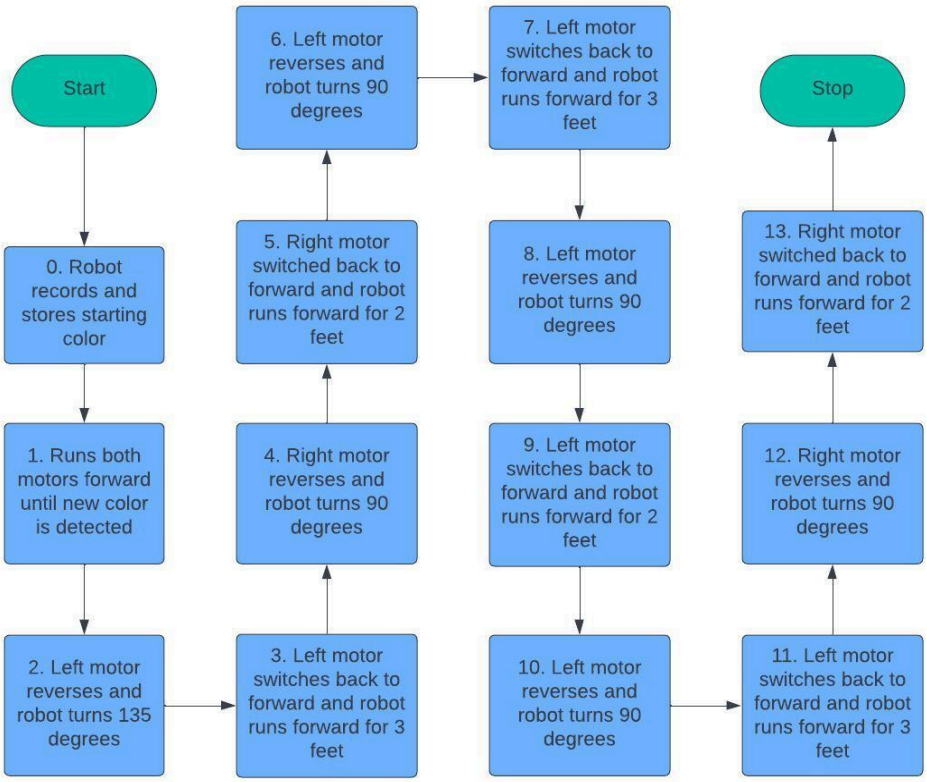


Figure 11: Strategy Flowchart

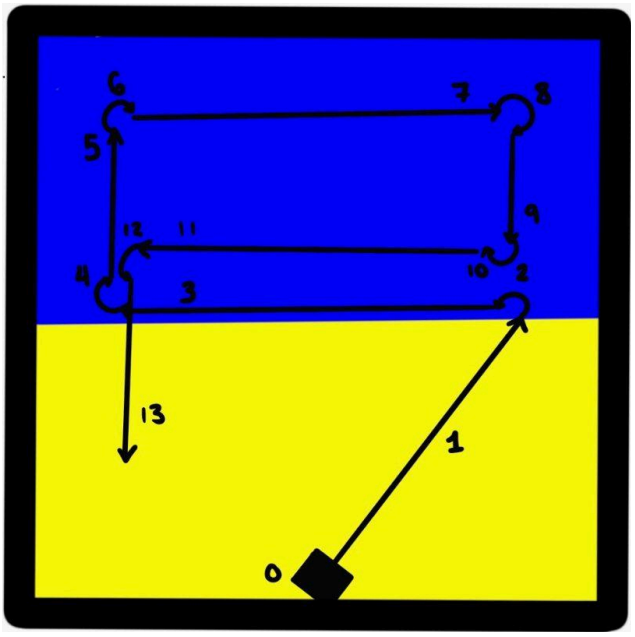


Figure 12: Drawing of Ideal Trajectory

Section IX: Appendix E

```
// --- Global Variables ---
volatile unsigned int period;  // "timer" : stores the value of TIMER1
volatile unsigned int timer;   // actual timer to track how much time has passed
int threshold = 290;           //threshold value for detecting color yellow
int upthresh = 550;           // threshold for detecting blue
int color = 0;                 // 0 if yellow and 1 if blue
int initialcolor = 0;          // 0 if yellow and 1 if blue
int starting_p = 0;              // tracks value of starting color
volatile unsigned int tracker = 0;

// --- ISR: interrupt function that resets and reads the timer value
ISR(PCINT0_vect) { // interrupt for Color sensor
 if (PINB & 0b00010000) {  // if pin 12 is high
   TCNT1 = 0;              // resets the timer to zero on rising edge
 }
 else {
   timer = TCNT1;  // stores timer value on falling edge (or vice versa)
 }
}

// ---- initColor: initializes the interrupt and timer
void initColor() {
 sei();  // enable interrupts globally
 PCICR = 0b00000001;  // initializes the interrupts

 // initializing the timer
 TCCR1A = 0b00;  // normal mode
 TCCR1B = 0b01;  // prescaler = 1
 TCNT1 = 0;      // reset timer
}

// --- getColor: calculates the period of the sensor output wave and returns it as a variable
int getColor() {
 PCMSK0 |= 0b00010000;  // enable PCINT4
 _delay_ms(5);         // delay of 5 ms
 PCMSK0 |= 0b00010000;  // disable PCINT23 to prevent further interrupts until calling
getColor again

 int p = timer * 0.0625 * 2;  // conversion faction using 1/16 and doubling for full period
```

```
 PCMSK0 &= ~0b00010000;
  return p;
}

// ---qti_sensor: reads whether the sensor detects a black edge or not
int qti_sensor() {
 int PIN_QTI_LEFT = 0b000100; // 10
 int PIN_QTI_RIGHT = 0b000010; // 9

 if ((PINB & 0b00000100) || (PINB & 0b00000010)) {  // if pin 9 or 10 (qti sensor) reads
black
   return 1;
 }
 return 0;
}

// ------- Simple command functions -------
int forward() {
 PORTD = 0b00001100;  // run wheels at full speed forward
}

int turn_left_90() {
 PORTD = 0b00100100;  // run left wheel forward to turn right
 _delay_ms(450);       // estimated time to turn 180 degrees
}

int turn_right_90() {
  PORTD = 0b00011000;  // run wheels in opposite directions to rotate 90 degrees
  _delay_ms(700);     // estimated time to turn 90 degrees
}

// main function
int main(void) {
 init();
 Serial.begin(9600);
 sei();
 initColor();

 starting_p = getColor(); //initial color period
```

```
 int PIN_QTI_LEFT = 0b000100; // pin 10
 int PIN_QTI_RIGHT = 0b000010; // pin 9

 DDRB = 0b000000; // all inputs
 DDRD = 0b00111100;  // pins 2, 3, 4, and 5 are outputs (motor)

 if (starting_p >= threshold) { //starting color is blue
   initialcolor = 1;
 }
  else {
   initialcolor = 0; // starting color is yellow
  }

//--- START OF STRATEGY CODE ----------------------------------

 if (initialcolor == 0) { // when starting color is yellow
   while(1) {
     period = getColor();

     bool border_left = PINB & PIN_QTI_LEFT;
     bool border_right = PINB & PIN_QTI_RIGHT;

     forward();

     if (border_left == 1 && border_right == 0) { //if there is a left border
       PORTD = 0b00011000; // turn right 90 degrees
       _delay_ms(400);
     }

     else if (border_left == 0 && border_right == 1) { //if there is a right border
       PORTD = 0b00100100; // turn left 90 degrees
       _delay_ms(400);
     }

     else if(border_left == 1 && border_right == 1) { // if there is a black edge right in
front
       PORTD = 0b00100100; // turn left 180 degrees
       _delay_ms(900);
     }
```

```
    if ((period < threshold) && (border_left == 0 && border_right == 0 && tracker < 1)) {
//if it's still in yellow and no border
      PORTD = 0b00001100;  // run wheels at full speed forward
    }

    // when it gets to the opposing color (in this case blue)
    else if ((period > threshold && period < upthresh) && (border_left == 0 && border_right
== 0) && tracker < 1) {

      PORTD = 0b00100100; // turn left 135 degrees
      _delay_ms(650); // position #2

      forward();
      _delay_ms(2700); // to position #3

      turn_right_90(); // position #4

      forward();
      _delay_ms(1500); // to position #5

      turn_right_90(); // position #6

      forward();
      _delay_ms(2700); // to position #7

      turn_right_90(); // position #8

      forward();
      _delay_ms(1600); // to position #9

      turn_right_90(); // position #10

      forward();
      _delay_ms(2700); // to position #11

      turn_left_90(); // position #12

      forward();
      _delay_ms(1000); // to position #13
```

```
      tracker = tracker + 1;
    }

    // when the previous strategy code runs once, it will then delay the going forward
function and stop
    else if((period < threshold) && (border_left == 0 && border_right == 0) && tracker == 1
) {
      _delay_ms(250);
      PORTD = 0b00000000;  // stop
    }
  }
 }

 else if (initialcolor == 1) { // when starting color is blue
  while(1) {

    period = getColor();

    bool border_left = PINB & PIN_QTI_LEFT;
    bool border_right = PINB & PIN_QTI_RIGHT;

    forward();

    if (border_left == 1 && border_right == 0) { //if there is a left border
      PORTD = 0b00011000; // turn right 90 degrees
      _delay_ms(400);
    }

    else if (border_left == 0 && border_right == 1) { //if there is a right border
      PORTD = 0b00100100; // turn left 90 degrees
      _delay_ms(400);
    }

    else if(border_left == 1 && border_right == 1) { // if there is a black edge right in
front
      PORTD = 0b00100100; // turn left 180 degrees
      _delay_ms(900);
    }
```

```
    if ((period > threshold && period < upthresh) && (border_left == 0 && border_right == 0
&& tracker < 1)) {
      PORTD = 0b00001100;  // run wheels at full speed forward
    }

    // when it gets to the opposing color (in this case yellow)
    else if ((period < threshold) && (border_left == 0 && border_right == 0) && tracker < 1)
{
     PORTD = 0b00100100; // turn left 135 degrees
      _delay_ms(650); // position #2

      forward();
      _delay_ms(2700); // to position #3

      turn_right_90(); // position #4

      forward();
      _delay_ms(1500); // to position #5

      turn_right_90(); // position #6

      forward();
      _delay_ms(2700); // to position #7

      turn_right_90(); // position #8

      forward();
      _delay_ms(1600); // to position #9

      turn_right_90(); // position #10

      forward();
      _delay_ms(2700); // to position #11

      turn_left_90(); // position #12

      forward();
      _delay_ms(1000); // to position #13

      tracker = tracker + 1;
```

```
     }

     // when the previous strategy code runs once, it will then delay the going forward
function and stop
     else if((period > threshold && period < upthresh) && (border_left == 0 && border_right
== 0) && tracker == 1 ) {
       PORTD = 0b00000000;  // turn left
     }
   }
 }
 PORTD = 0b00000000; //stop motors
}
//--- END OF CODE ----------------------------------
```