Mechatronics Final Report

Ethan Liu, Hunter Sadoff, Yuki Oyama

Introduction

The objective of this report is to describe and assess the process of building our mechatronics final project robot. We will delve into the motivation behind the design and how it developed over time to how it drove the formulation of our final strategy. In addition, we will discuss the strengths and weaknesses of the intended design and strategy versus its performance in the final competition.

Motivation Behind Design

Our primary motivation was to create a physical design that minimized the necessary complexity of the robot algorithm. In our team members' past experiences with robotics, programming a robot to carry out a complex algorithm has proven to be difficult. Therefore, we opted to create a design that required minimal code but would still work effectively.

Through this line of thought, we ruled out trying to develop a robot that would interact heavily with the environment—for example, a robot that would collect blocks and then move back to its own side by the end of the game. We figured it was risky and difficult to ensure the robot would be able to successfully navigate the environment and do so within the 60s time limit. For example, with the collection robot example, the robot might not be able to return to its own color within the 60 seconds, and it would end up handing points to the opponent. In addition, navigating the board is made even more difficult due to the presence of another robot.

This led us to believe it would be most practical to carry out a set path algorithm. However, this strategy comes with a caveat. The problem with a set path algorithm is the presence of the opponent robot. Ultimately, there is no sure-fire way of ensuring success through a set-path algorithm since it doesn't account for the opponent robot's strategy. Still, given the time allotted for the competition, we opted to program a set path algorithm that would work against most robots in the competition.

To increase the likelihood of our set path algorithm succeeding, we designed our robot that would be able to collect blocks as quickly as possible before it interacted with the opponent robot where uncertainties would arise. We decided to do this by maximizing the range of our robot. To do this, we designed two arms 90 degrees to each other, that when deployed fit in the 18in cylinder size constraint and when raised, fit within the 8inx8in square size constraint. We hoped this large collection range would allow us to reach blocks before other teams did.

Initially, we planned to use spring loaded arms (shown in *Figure 1* below) to give us an edge over other teams. The idea was that the springs would allow us to flick blocks onto our side quickly before the other robot had a chance to reach them. In addition, we had also misunderstood the rules, thinking that blocks that were off the board but on our side would count towards our team's points in non-tiebreaker situations. We believed the flicking spring mechanism would not only propel blocks onto our side, but also prevent other teams from reaching them by sending them off the board. Below is a picture of our spring loaded design. We planned to use one servo to allow the arms to drop down (by gravity) to the floor, and once this action completed a second servo would release torsional springs that would flick two arms outward.

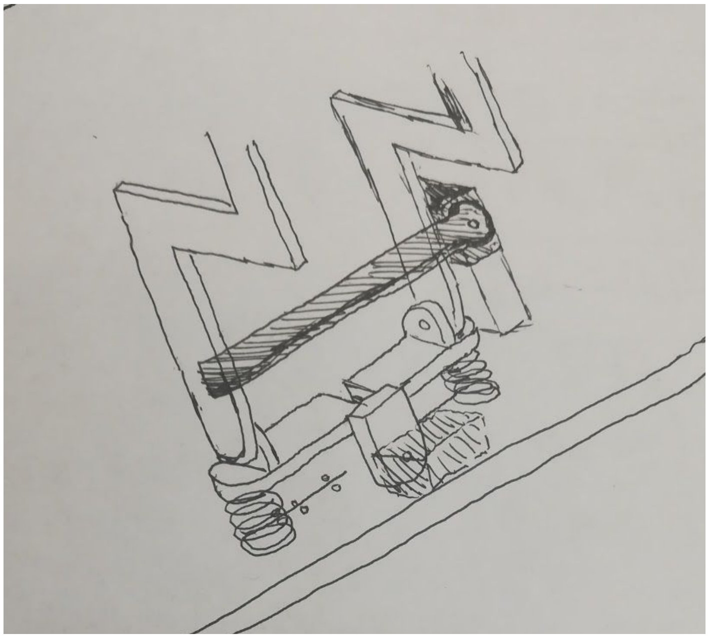


Figure 1: Initial Design for Arm Deployment

Ultimately, we decided against this strategy for two reasons. One, as aforementioned, we had misinterpreted the rules, and we realized there was little benefit in sending blocks off the board. In addition, we realized it would be difficult to try to constrain the torsional springs and ensure they didn't dislodge themselves. Overall, we saw small returns for this high effort design.

Instead, we opted to use the two servos to actuate two arms directly (shown in *Figure 2* below). The main advantage of this design was that, even though deployment was slightly slower than deployment of the spring loaded arms, it would be possible to move the arms back up. This was a key feature in the primary action in our algorithm.

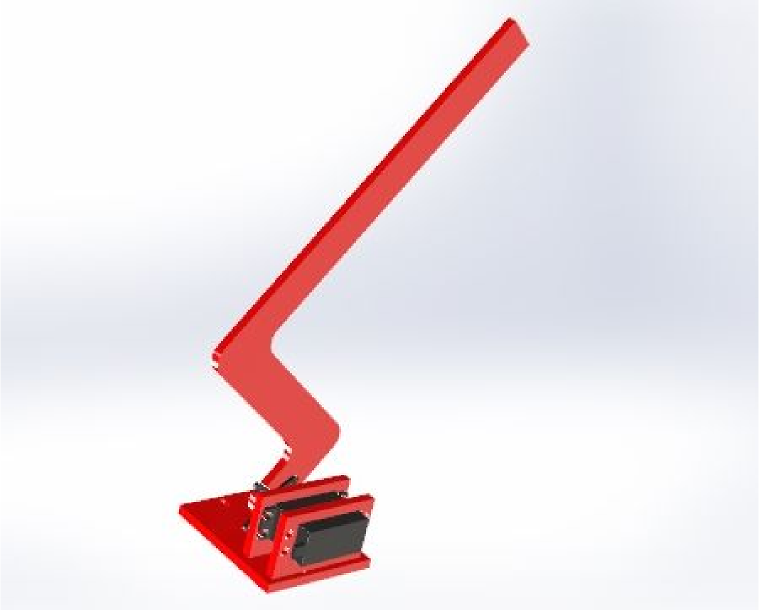


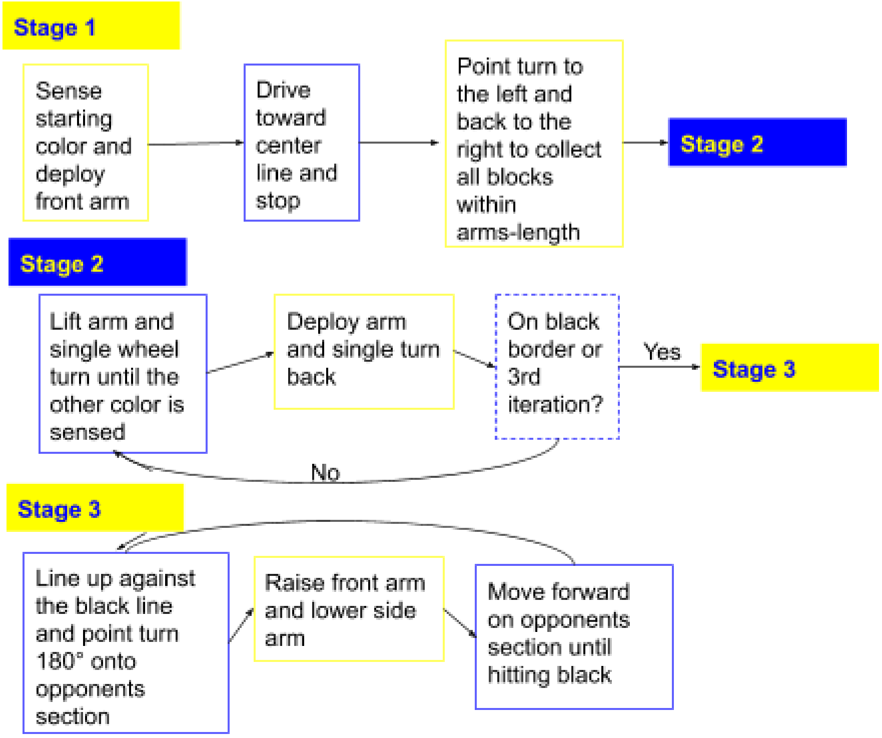
Figure 2: Design of deployable arm

We decided to have the robot drive directly towards the middle of the board, then sweep in both the left and right directions. We believed this initial sweep would already be enough to win many games if successful & consistent. After this initial sweep, the robot would follow the centerline to the right until reaching the black border, while sweeping blocks from the opponents' side of the board. By using a line follow that compared the current color the robot was on to its start color, the program would lift the arm when the robot was on the starting color side, it would single wheel turn until it was on the opposite color side, and then place the arm back down and sweep until it sensed the other color again.

Lastly, we also decided to add border pieces around the chassis of our robot so that collisions with other robots would not cause damage to the electronics and wheels mounted on the metal frame chassis provided to us.

To ensure our robot was under the 40 dollar budget limit, we relied heavily on laser cutting acrylic to cut down our cost. In the end, we used 2 sheets of acrylic and bought 1 extra servo, and so we were well under budget.

Flow Chart of Strategy


Stage 1
Sense starting color and deploy front arm
Drive toward center line and stop
Point turn to the left and back to the right to collect all blocks within arms-length
Stage 2
Stage 2
Lift arm and single wheel turn until the other color is sensed
Deploy arm and single turn back
On black border or 3rd iteration?
Yes
Stage 3
No
Stage 3
Line up against the black line and point turn 180° onto opponents section
Raise front arm and lower side arm
Move forward on opponents section until hitting black

Robot's Strengths and Weaknesses

Our robot's primary weakness was a lack of adaptability. While our robot performed well in competition, this was mostly due to the fact that we happened to match up well against opponent strategies (as we planned). Our success was based around the fact that we were able to quickly retrieve a few blocks more quickly than our opponents, and we were then able to engage with the opponent robot and stall for the rest of the game, preventing the opponent robot from moving back onto their side of the board. Had we played more robots that did not directly move towards the center of the board and engage in a pushing contest with our robot, we likely would not have won as many matches. Moreover, our robot completes 80% of its block collection within the first 30 seconds of the game, and it would likely fail against robots that are able to avoid our robot and collect blocks for the full minute by navigating the board well.

In addition, our robot was not as heavy as other robots, and while we were strong enough for most matches, the match we lost we lost due to getting outpowered. The opponent robot had used the same servos as our own robot (with no gearing), but since the opponent robot had more mass it was able to push its way back to its own side just enough that it was able to win by 1 block. For our robot specifically, the fact that it was not as heavy as it could have been was a detriment.

We also often found our robot would often push blocks on our own side off of the board, which was our initial strategy when we believed those blocks would still count for us. Upon discovering this was not the case, we adapted our strategy to push our opponents blocks off the board and to engage other robots on their side more directly so that they wouldn't be able to find their way to our side to collect blocks we had already won. This plan was good in theory and often wound up winning us matches, but when we encountered larger robot that pushed us back onto our side, we didn't have much of a strategy to deal with it and often our robot would simply push our own blocks off the board. This, however, did not wind up being a significant issue in the majority of our matches, as we anticipated.

The general strength of our design was our ability to collect blocks quickly and traverse along the border to prevent those blocks from being recollected by our opponent. Our border collection algorithm (Stage 2 in the flowchart) wound up working better than anticipated at collecting blocks in their starting positions. Additionally, our acrylic arm wound up being much stronger than anticipated, and would often engage our opponent and delay whatever strategy they had. This, coupled with our general strategy to traverse the middle of the board for the first half of the game, often allowed us to meet with and prevent the opposing design from achieving their early game goals as they would become entangled with our robot, often for a substantial number of seconds. Another strength of our robot was consistency. Our initial sweep algorithm worked 90% of the time during test runs in open-lab. This would guarantee us roughly 5 blocks onto our side.

From a physical design perspective, our robot was efficient because it made use of vertical space. At the start of each round, both arms are raised to fit within the 8x8in square and are lowered after the round begins to expand to the full 18in circle. This range was what allowed us to carry out our initial sweep algorithm without intervention by the opponent robot.

Discussion of Competition Performance

Overall, we were satisfied with our competition performance, and our robot performed as planned for most matches. As we correctly anticipated, many robots actually tried to move to the other side of the board (unlike our own) in order to collect blocks. As a result, our set path algorithm proved to be a good match-up against most teams. Moreover, since we line followed the center-line and swept our deployable arm from opponents' side of the board to our own, when we collided with other robots we ended up pushing the opponent robot towards' our own side. This was effective because during some rounds we would end in a collision standstill where our robot arm was preventing the other opponent robot from returning to its own side of the board with the blocks it collected. While this was not the primary intended purpose of our algorithm, we did program the algorithm with the hope of this occurring. Since we had already collected a few blocks with our initial, preemptive sweep, we would win most matches that ended in this sort of standstill. During our first 5 matches, we lost only a single game by a 1 block since we ended in standstill but the other robot had managed to barely make it over the centerline. Still, we were able to complete the initial sweep in all of these 5 matches, showing that our consistency in open lab practice translated well into actual competition. As we planned, the initial sweep never collided with the opponent robot since we were able to utilize our robot's reach in order to perform this action before other teams even reached the blocks. The consistency of this preemptive sweep was a huge contributor to our overall win-loss ratio.

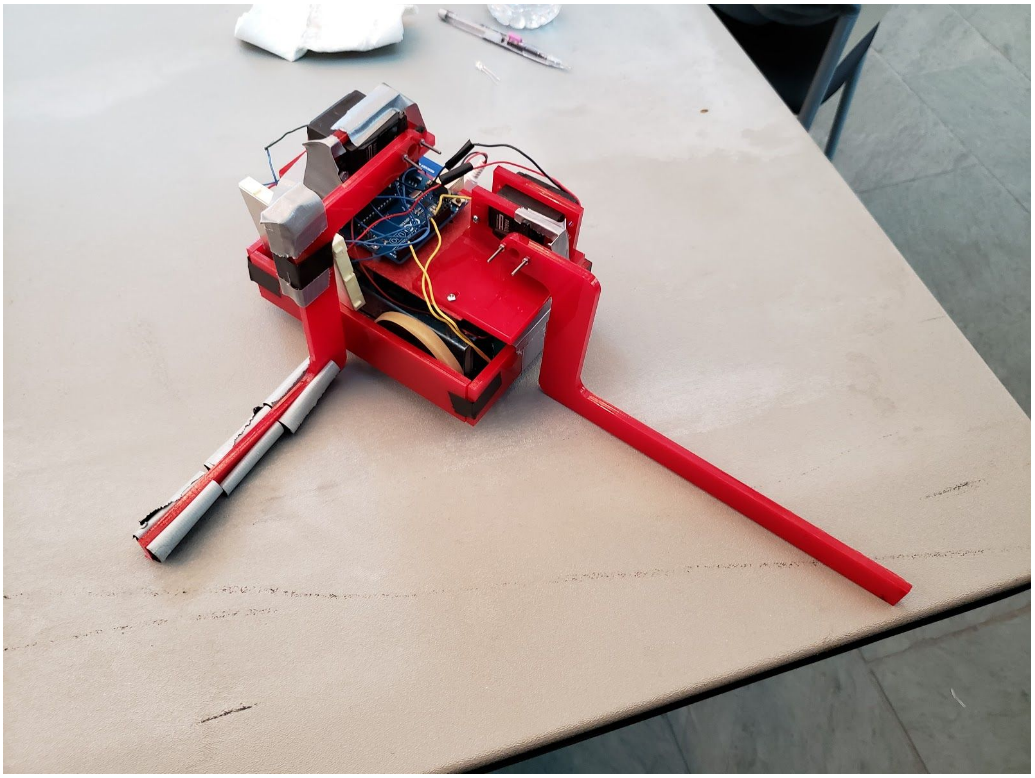
Still, we did encounter various problems in competition, and there were many areas of potential improvement. Related to our ability to push against other robots, our robot had problems with our wheel design during the competition. The rubber bands did not provide adequate traction for pushing against heavier opponents, which resulted in our one loss during the round robin. In addition, during the elimination round, one of the rubber bands slipped off its wheel and was caught in between the wheel and a side panel, causing the wheel to jam. Our robot was unable to drive and consequently lost that round. Learning from other robot designs, if we used rubber tires that were part of the same kit as the given wheels, the traction would be significantly increased and we would not have to worry about the rubber slipping off the wheel. On the other hand, we could also have purchased different wheels with more robust construction and better traction. Many other teams purchased wheels that better fit their overall strategy, which was not something we took advantage of.

Another aspect of our strategy that did not work according to plan was the deployment and use of our secondary arm mounted on the side of the robot. The original strategy was to drive forwards and use the side arm to push blocks on the opponent's half off the board to prevent those blocks from earning points. However, because this function triggers after sensing the black line, our plan did not work consistently during the competition, as collisions with the opponent's robot would throw off our robot's sensors and decision flow. The most plausible conjecture is that collisions would cause our robot's QTI sensors to briefly sense "black," but we have no way of verifying that this indeed was the issue. As a result of collisions, the side arm would deploy while still on our half of the board and inadvertently push our own blocks around. One possible solution is to implement bump sensors, so the robot can modify its behavior when collisions inevitably occur. We could also add a gear train to the servos to increase the wheel torque so that we would be able to execute our set path algorithm forcefully even in the presence of other robots.

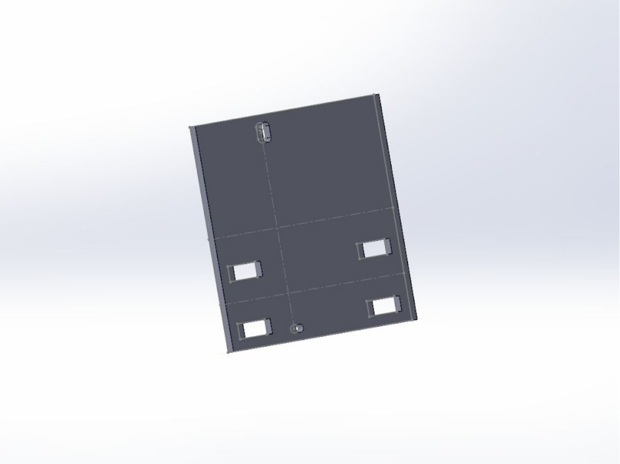
Conclusion:

Overall we were happy with the performance of our design and our strategy. It also seemed like a lot of the results of the competition came down to luck. Our robot only lost in the playoffs because we didn't make sure the rubber band was on the wheel properly, and there was also a match where the opponent's robot glitched and drove directly off the board allowing us to execute our strategy without collisions. But our strategy and algorithm and mechanical design were all generally conducive to winning during a chaotic game. With that said there were some improvements we could have made in order to be more consistent, like making our robot slightly heavier or using a gear train to make it faster despite a heavier frame.

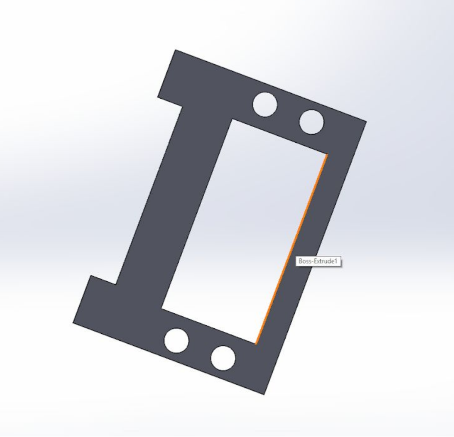
Appendix



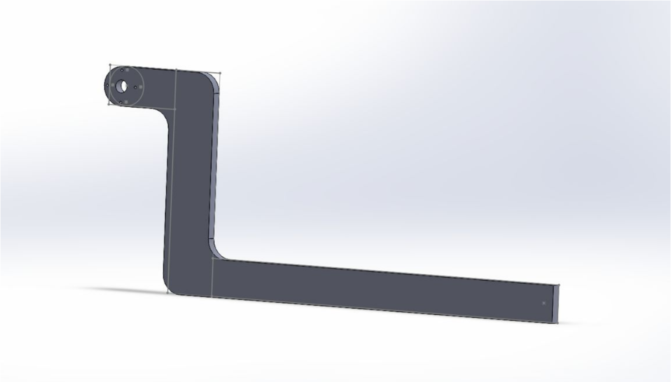
Picture of final design used at competition



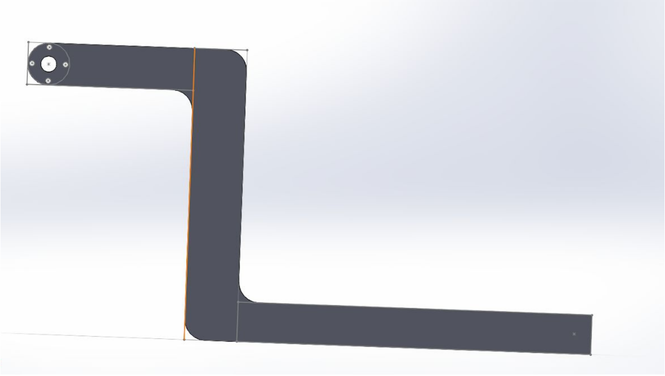
Board1.SLDPRT: Part file used for interfacing between servo mount for the front arm and main chassis



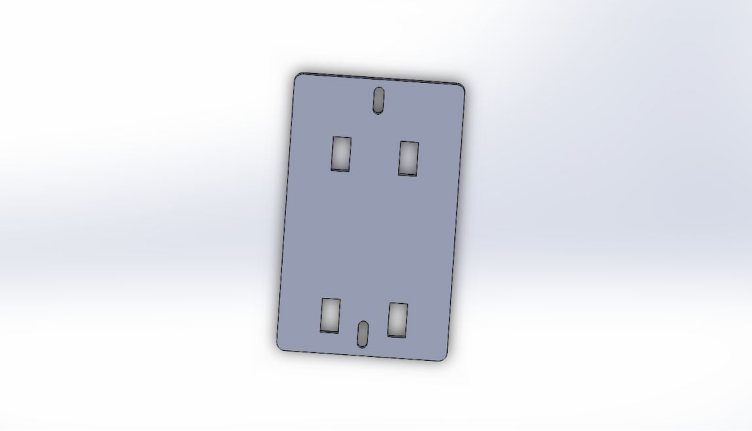
ServoMount.SLDPRT: Part file used for mounting servo



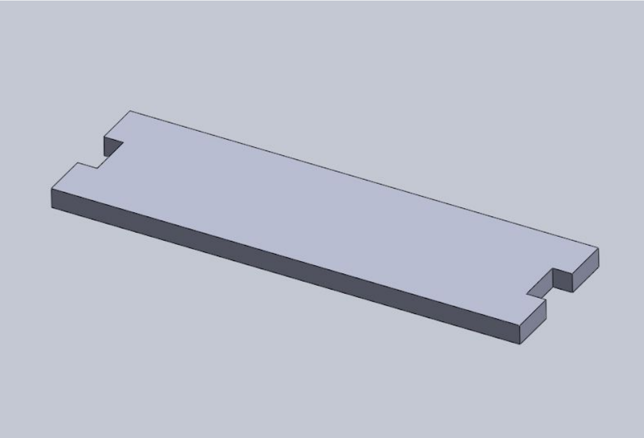
Arm.SLDPRT: Part file used for front arm that attaches to servo



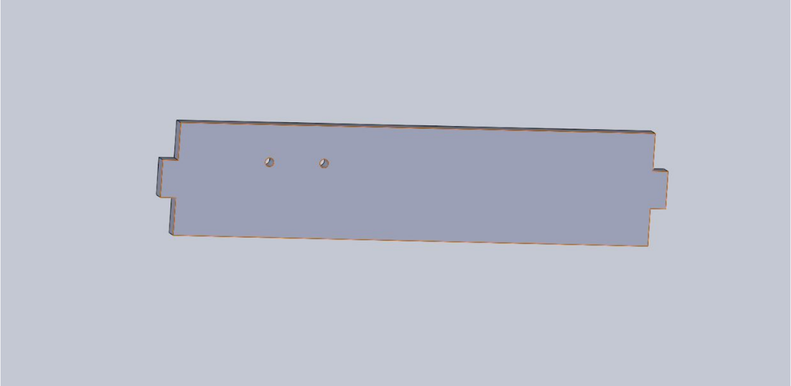
Arm2.SLDPRT:Part file for side arm that attaches to servo (note how appendage lengths are different from front arm since it is mounted at a different height and position on chassis)



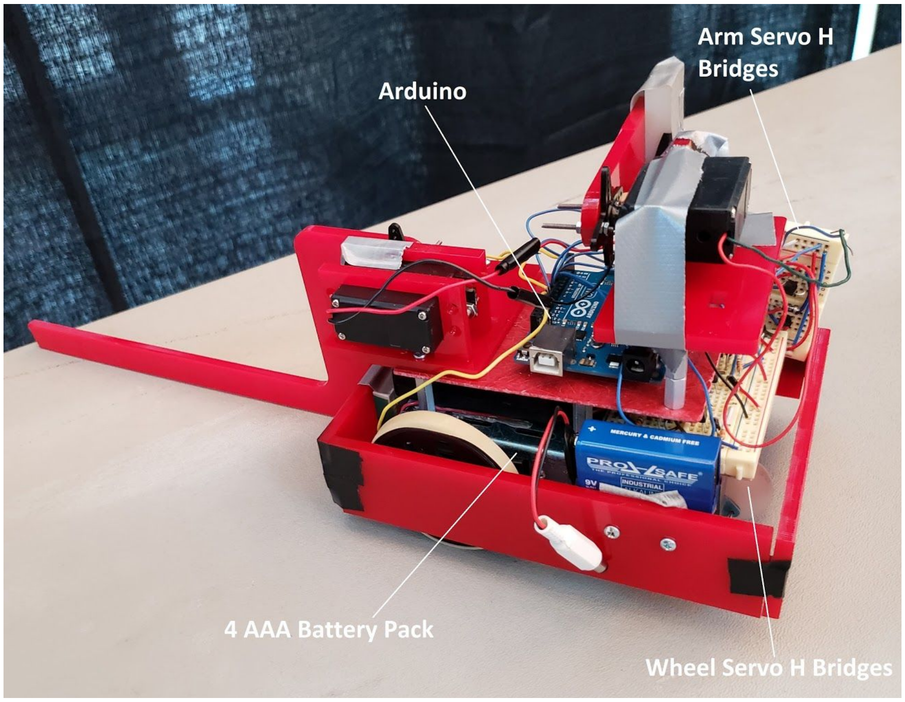
Mount.SLDPRT: Part file for the plate that interfaces between the servo mount for the side arm and the main chassis of the robot.



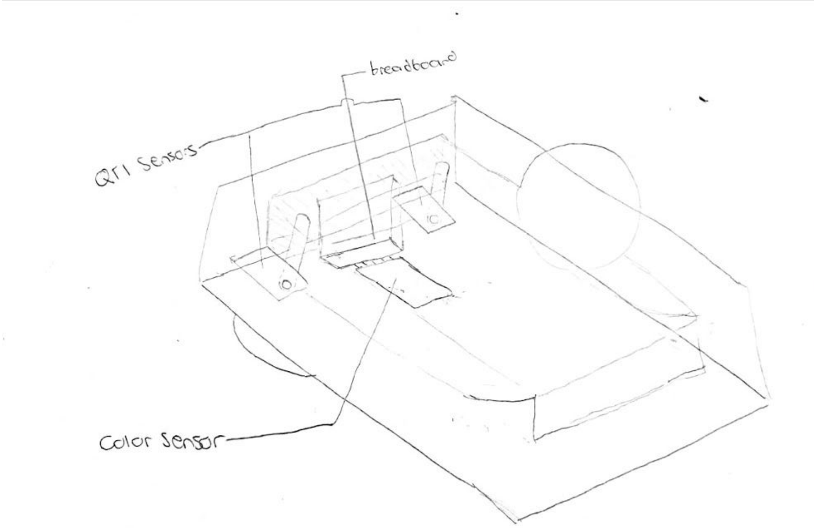
FrontBackPanel.SLDPART: Part file for front and back panels protecting internals of robot.



SidePanel.SLDPART: Part file for side panels protecting internals of robot.



Picture of Final Design with Major Circuit Components Labeled



Bottom up view of robot, showing positions of QTI sensors, Color sensor, and associated breadboard

Commented code:

```
/*
 * GccApplication1.c
 *
 * Created: 11/21/2019 1:50:21 PM
 * Author : MAE Lab
 */

 #include <avr/io.h> // This includes the C library for I/O devices
 #define F_CPU 16000000 // This sets the clock to the crystal on the UNO
 #include <util/delay.h> // This includes the C library for the _ms_delay() method
 #include <avr/interrupt.h> // This includes the C library for interrupts

 //Initialize TCS Library
 //InitTCS3200();


 int period = 0;

 ISR(PCINT2_vect) {
 if(PIND & 0b00001000){
 TCNT1 = 0;
 }
 else{
 period = TCNT1;
 }
 }

//Initialize all color sensors and timers
 void initColor()
 {
 DDRD &= 0b11110111; //set pin 3 to input with masking
 DDRB |= 0b00100000; //set pin 13 to output with masking
 PCICR = 0b00000100; //pg 92 PCINT2
 TCCR1A = 0b00000000; //timer1 pg 170
 TCCR1B = 0b00000001; //timer1 pg 173 prescaler = 1
 sei();//enable global interrupt
 }

//Method to gather the color at any point for the rest of the program
 int getColor()
 {
 PCMSK2 |= 0b00001000; //pg94 enable PCINT19
 _delay_ms(10);
 PCMSK2 &= 0b11110111; //disable PCINT19
 return period*2*0.0625; //return full period in microseconds
 }

//Method to gather the QTI data at any point for the rest of the program
//Returns 1 in the tens place if left QTI is triggered, 1 in the 1s place if
//right QTI is triggered, 0 in either place otherwise
 int getQTI(){
 int l_QTI = (PINB & 0b00010000) && 0b00010000;
 int r_QTI = (PINB & 0b00001000) && 0b00001000;
 return 10*l_QTI + r_QTI;
 }

//Moves right wheel backward and left wheel forward to point turn right
 void point_turn_right(){
 PORTD = 0b01010000;
 }

//Moves left wheel backward and right wheel forward to point turn left
 void point_turn_left(){
 PORTD = 0b10100000;
 }

//Moves right wheel forward and keeps left still to single wheel turn left
 void single_turn_left(){
 PORTD = 0b00100000;
 }


//Moves left wheel forward and keeps right still to single wheel turn right
 void single_turn_right(){
 PORTD = 0b01000000;
 }

//takes an input that determines if side arm moves up or down
//Delay depends on whether raising or lowering
void arm2Move(int up_or_down){
PORTB = 0;
if(up_or_down){
PORTB |= 0b00000010;
_delay_ms(700);}
else{
PORTB |= 0b00000001;
_delay_ms(400);}
}

//takes an input that determines if side arm moves up or down
//and another to determine the length of delay if raising
//Delay depends on whether raising or lowering
 void arm1Move(int up_or_down, int SorL){
PORTB = 0;
if(up_or_down){
PORTD |= 0b00000100;
_delay_ms(400);

}
else{
PORTB |= 0b00000100;
if (SorL){
PORTD = 0;
_delay_ms(400);}              //short delay for stage 2
else
_delay_ms(1000);              //long delay to raise permanently
```

```
}
}

//Implements stage 3 with the direction of the turn as a bool input
 void sweep(int right_or_left) {
 PORTD = 0b00000000;
//move until hitting the black border
 while(getQTI()==0){
PORTD = 0b01100000;
 }
 if (right_or_left){
 while(getQTI()!=0){
PORTD = 0b10010000;

}
//square up to the black border
while(getQTI()==10){
single_turn_left();
}
while(getQTI()==1){
single_turn_right();
}

point_turn_right();
 }
else{
while(getQTI()!=0){
PORTD = 0b10010000;                    //back up from black line slightly

}
while(getQTI()==10){
single_turn_left();
}
while(getQTI()==1){
single_turn_right();
}
point_turn_left();

 _delay_ms(1000);
 }

 }

//Main helper of linefollow, moves along interface of blue and yellow
//lifts arm while on start color and lowers when on opposite color
 void lineFollowHelp(int left, int start_color) {

 PORTD = 0b00000000;
 arm1Move(0, 1);
 while(getColor()>start_color-80 && getColor()<start_color+80){

single_turn_left();
```

```
 }
 //drive forward

 arm1Move(1, 0);
 while(getColor()<start_color-80 || getColor()>start_color+80){
single_turn_right();
 }

 }

//calls line follow help while not on black or for 3 iterations (whatever's first)
 void lineFollow(int start_color) {
 //while (getQTI()==0){
 //_delay_ms(1000);
 int num_lf = 3;
 while (num_lf>0 && getQTI()==0){
 lineFollowHelp(1, start_color);
 num_lf--;
 }
 while(getQTI()!=0){
PORTD = 0b10010000;
 }

 while(getQTI()==10){
single_turn_left();
 }
 while(getQTI()==1){
single_turn_right();
 }

 point_turn_left();
 _delay_ms(800);
 //}
 /*PORTD = 0b10010000;
 _delay_ms(300);
 PORTD = 0b00000000;
 point_turn_right();
 _delay_ms(1000);*/

 }




 //yellow period: 33 black period: 600 blue period: 280

 int main(void)
 {
//pin and variable initializations
 DDRB = 0b11111111;
 DDRB &= 0b11100111; // set pins 11, 12 as QTI inputs
 DDRD |= 0b11110100; // set pins 4-7 as motor outputs
 PORTB = 0;
 initColor();
 int QTI_status = 0;
 int QTI_r = 0;
 int QTI_l = 0;
 int num = 0;

 while (1)
 {
int start_color = getColor();
arm1Move(1, 0); //lower arm
 while(start_color-60<getColor()&& start_color+60>getColor()){
 PORTD = 0b01100000; //drive forward
 }

//gather QTI data
 QTI_status = getQTI();
 QTI_r = QTI_status%10;
 QTI_l = QTI_status/10;

  PORTD = 0b00000000;
  point_turn_left();          //initial sweep left
  _delay_ms(950);
  PORTD = 0b00000000;
  point_turn_right();         //initial sweep right
  _delay_ms(1850);
  PORTD = 0b00000000;
  _delay_ms(1000);

  lineFollow(start_color);              //calls linefollow with start color info

 PORTD = 0b01100000;                    //secondary sweep
 PORTD = 0b00000000;
 point_turn_left();
 _delay_ms(950);
 PORTD = 0b00000000;
 point_turn_right();
 _delay_ms(1850);
 PORTD = 0b00000000;
 _delay_ms(1000);

 int count = 0;
 arm1Move(0, 0);             //raise front arm
 PORTB = 0;
 int iters = 3;
 while(iters>0){
 arm2Move(0);                           //lower side arm
 int mod_count = count%2;
 sweep(mod_count);                      //calls sweep and alternates which way to turn
 count++;
 iters--;
 arm2Move(1);                           //raises side arm
 }
 PORTD = 0b10010000;
 _delay_ms(500);



 arm2Move(1);                                   //raises side arm before while loop starts again
 }
 }
```