

Team 32 - Our Robot's Name is Chomp

Kathryn Ng - kln46
Charlie Wright - caw352
Sonia Talarek - skt56

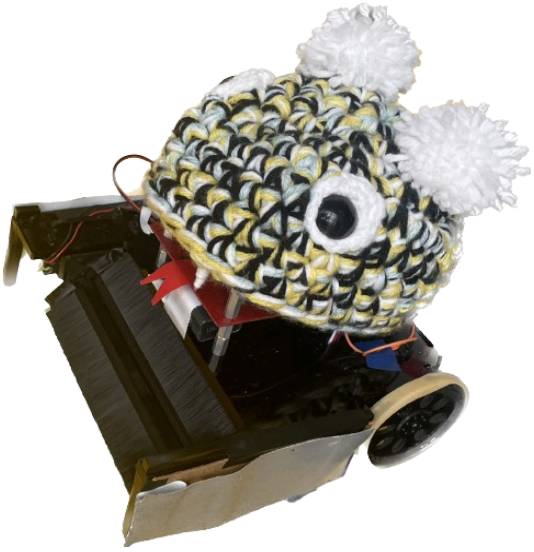

Figure 1: Chomp

Robot Design and Strategy Overview

Chomp uses a horizontal spinning brush to collect cubes and store them under his chassis. A custom laser-cut chassis allows for a wider base and a larger brush. His taped outer walls assist in grabbing any blocks his brush may have missed. His wheels and brush are driven by continuous rotation servos powered by a 6-volt battery. A 9-volt battery supplies power to an Arduino, the brains of this operation. Two QTI sensors located under the front corners of the chassis tell Chomp when he has hit the border of the field, and a color sensor located in the back lets him know whether he is home or in enemy territory.

At the beginning of the match, Chomp stores the initial color sensor reading as an RGB code so he always remembers his home color. Then he drives to the opposite side of the board and turns 90 degrees. He drives in S-shaped curves across the board until 40 seconds have passed since the beginning of the match (timed using the Arduino timer). Then, he follows the border until he finds his home color again and stops [App. D and E].

His hat serves multiple purposes. The soft and textured surface absorbs and redirects sound waves to confuse the sonar sensors of his opponents. It also creates a protective barrier between the outside world and his fragile inner wiring. Finally, it has the added effect of being fabulously stylish!

Design Process Reflection

We began our design process with a brainstorming session. We had a few ideas, and we discussed the tradeoffs and feasibility of each design. At the end of this session, we knew we wanted to do either a collection mechanism using a rotating brush or a collect-as-we-drive method with a plow-like device and a fence. We decided to use a rotating brush because we thought it was an effective and interesting idea that would challenge us and give us creative freedom. We chose to orient the brush horizontally as it would give us the maximum collection area with only one servo.

After deciding on a general strategy, we were not sure whether we should use the metal chassis given to us or design and laser-cut our own. We chose a custom laser-cut acrylic chassis because the metal chassis would make it difficult to collect blocks since they wouldn't fit under the robot. The custom chassis would allow us to use a rotating brush to sweep cubes underneath the chassis and keep them there. It would also give us more space to collect the cubes [App. C]. We worked on our mechanical design and coding in parallel, using the metal chassis to complete Milestones and test code while the custom acrylic chassis design was in progress.



After a trip to the RPL, some super glue, screws, and zip ties, Chomp was born. Throughout our testing process, we noticed several issues. The first issue was that he had a tendency to drift to the left. To counteract this, we shifted the battery packs to the right side of the robot to weigh that side down more. Another issue was that Chomp would get stuck in corners. For example, he would read a black border on his right QTI, then turn left, only to read a black border on his left QTI, then turn right, then left. He would bounce haplessly around in the corner until we picked him up and brought him to safety. To mitigate this, we implemented code that would tell Chomp to back up after reading borders on alternate QTIs seven times [App. E]. After more testing, we updated this to include the case where he would alternately read one QTI and both QTIs. Thus, Chomp no longer got stuck in corners. However, as we reached the cube collection milestone, we found a new issue. Sometimes, when Chomp would collect a cube, he would lift himself up slightly due to the stiffness of the brush and would receive false positives on both QTIs. He would then back up, losing the cube! We tried to mechanically mitigate this issue by lengthening our pegs and adding spacers between the wheel servos and the chassis so that the brush would be higher off the ground. We also increased the alternate QTI reading count to thirty before backing up giving him more time to bounce over a block. This did mean that Chomp might get stuck in a corner, but only temporarily.

We took certain measures to prevent failure modes that we anticipated before they could happen. For example, we soldered our wires permanently to avoid any wires unplugging in the middle of a match. Another failure mode we considered while preparing for Milestone 3 was an inability to read the colors due to the lighting differences between the lab and Duffield atrium. To prevent issues, we added physical cardboard blockers to isolate the color sensors and QTIs from the ambient lighting as much as possible [App. F, Fig.12]. We also implemented an auto-calibration function where Chomp would read the color when he was placed on the board and use that as his home color. When he read a color outside of our color tolerances, he would determine that he was on the away color [App. E]. We had no issues with color detection after implementing this method.

We knew that because our robot design collected cubes inside the robot instead of just pushing them to our side of the board, one important failure mode to consider was Chomp getting stuck on our opponent's side with all his hard-earned cubes. Our strategy was to follow the border until we reached our color after 40 seconds [App. D]. One issue with this was the case where Chomp approached the blue/yellow border parallel to it. In this orientation, Chomp could detect our home color without going all the way home, thus leaving some blocks on the other side. We added a delay of half a second to make sure that Chomp made it home. We were worried that he might go off the board since he could not read QTI during the delay, but we thought that since it was such a short time, it would be okay. During the competition, we found that it was not okay.

Competition Analysis

In the competition we made it into the brackets, having 5 wins, 1 loss, and 1 tie (our opponents forfeited out of fear) in our round-robin matches. We lost the first round of the elimination bracket and ended up in the top 16, but we are happy we made it that far to begin with.



In the first match we lost; we placed Chomp angled towards the right side of the board when we had previously always angled him to the left or center. The issue of drifting to the left caused him to not hit the right border before crossing to the opponent's side, and he followed the borders around the board instead of going into the COLLECT mode [App. E].

The final feature we added was a 0.5-second delay to drive forward before stopping at the end of the match, to ensure Chomp was safely on our side of the board. In this window, our QTI or color sensors were not being read. In one round, we were on our home color and heading towards the opposite side when this timer triggered, causing us to stop on the opposite color and have a tied game. In another round, we were on our color and the timer triggered when Chomp was reaching a border which caused him to drive off the board.

In our final match of the competition, Chomp was on the opponent's color, and he did not turn when he hit the border, eliminating us from the competition. This happened on the opponent's color and well before 40 seconds, so our best guess for the cause of this issue was the lighting affected the QTI's reading and triggered a false negative. Chomp's motivation for retirement remains a mystery.

Aside from these challenges, our S-path strategy was consistently successful when Chomp started angled to the left. Opponents would sometimes grab more blocks than us but we were able to steal some back, and also had enough power to push other robots of the board. Some robots seemed to have color sensing and QTI issues which caused them to go off course, but our code had accounted for most changes in lighting and issues when running into other robots such as lifting up. Overall, we had a pretty robust strategy and algorithm that allowed us to prevent a lot of issues that other teams seemed to run into.

Conclusions

In conclusion, we are very proud of our son. Over 4 weeks we tweaked our code and fully altered our chassis to create a robot that performed extremely consistently. In our testing, Chomp consistently cleared boards against no opponent, and when there was an opponent it was a matter of who rammed the other first.

If we were to revise our strategy, we would remove the 0.5-second delay, which was intended to be a safety net but ultimately worked against us. Another feature we frequently discussed was increased movement speed. We discussed and did calculations for using an op-amp to increase the voltage across our servo motors while still being able to supply enough voltage to the Arduino, but we were concerned about being able to draw enough current to run all of the components. Given more time we would have experimented with op-amps and other possible solutions to increase voltage to the servos. We also discussed implementing PWM to account for his leftward drift, but decided it would not make too big of a difference in our performance. Our strategy was effective and performed consistently in the competition, and we are afraid that these additional features would make Chomp too powerful and too perfect.

Our advice to next year's class would be to do something unique and have fun. We settled on the idea of the brush collector because it was effective but also different. It had its own challenges such as how to store the blocks, but gave us an interesting design challenge that we had a lot of fun with. The project is very open-ended and gives groups a lot of room to be creative, and it is up to us to take advantage of that opportunity. A bootleg Roomba turned into a robot with a mouth, which became our beloved child.



Appendices:

Appendix A: Bill of Materials (BOM)

	Item Name	Unit Price	Number of Units	Total Cost	Link
McMaster	Easy-Cut Strip Brush, 7/32" Wide x 3/16" High Backing, 1" Overall Height	$3.20	3	$9.60	McMaster
	Item Name	**Perimeter [in]**	**Cost per part**	**Total Cost**	
RPL Laser Cutting	Chassis	76.339	$4.32	$13.06	
	Brush Axel	14.992	$1.25		
	Brush Axel Cap	2.805	$0.64		
	Peg 1	10.845	$1.04		
	Peg 2	6.216	$0.81		
	Item Name			**Total Cost**	
Extra Parts	Continuous Rotation Servo			$3	
	Item Name	**Cost Per Unit**	**Number of Units**	**Total Cost**	
Misc Supplies	Super Glue [tubes]	$1.08	1	$1.08	Amazon
	Zip Ties [ties]	$0.01	9	$0.13	Amazon
	Electrical Tape [ft]	$0.14	2	$0.27	Amazon
	Yarn [oz]	$0.57	7	$3.99	Michaels
	Cardboard [boxes]	$1.28	1	$1.28	Home Depot
	Party Poppers [poppers]	$0.15	8	$1.16	Walmart
			Total Price	$32.42	

Appendix B: Circuit Diagram

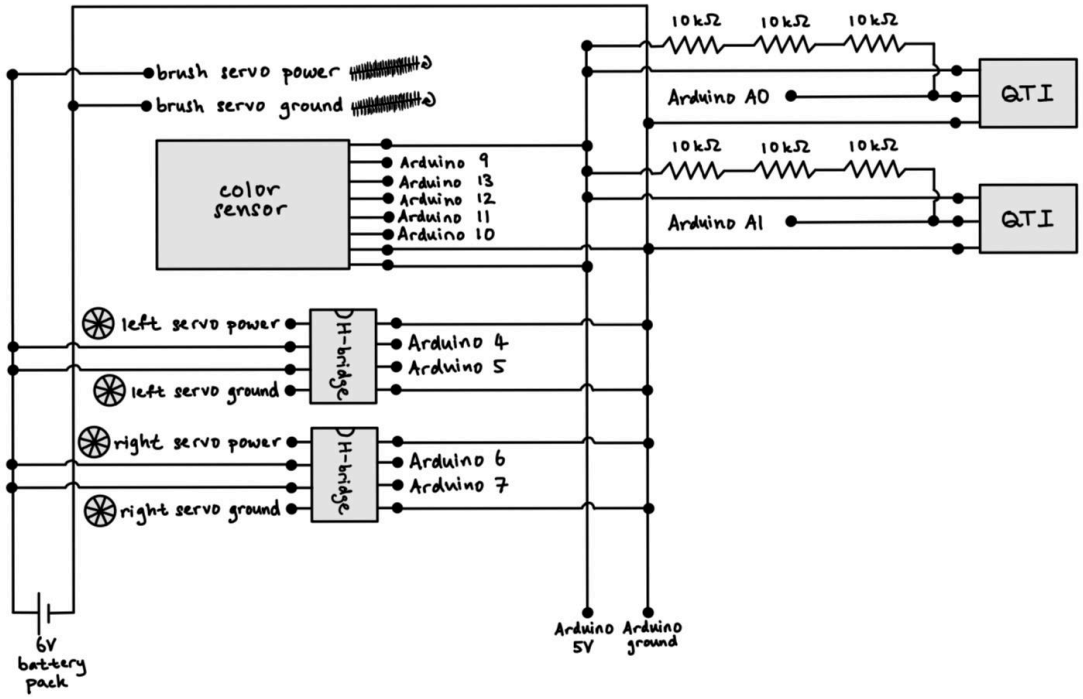

Figure 2a: Complete circuit diagram of electronics included in Chomp

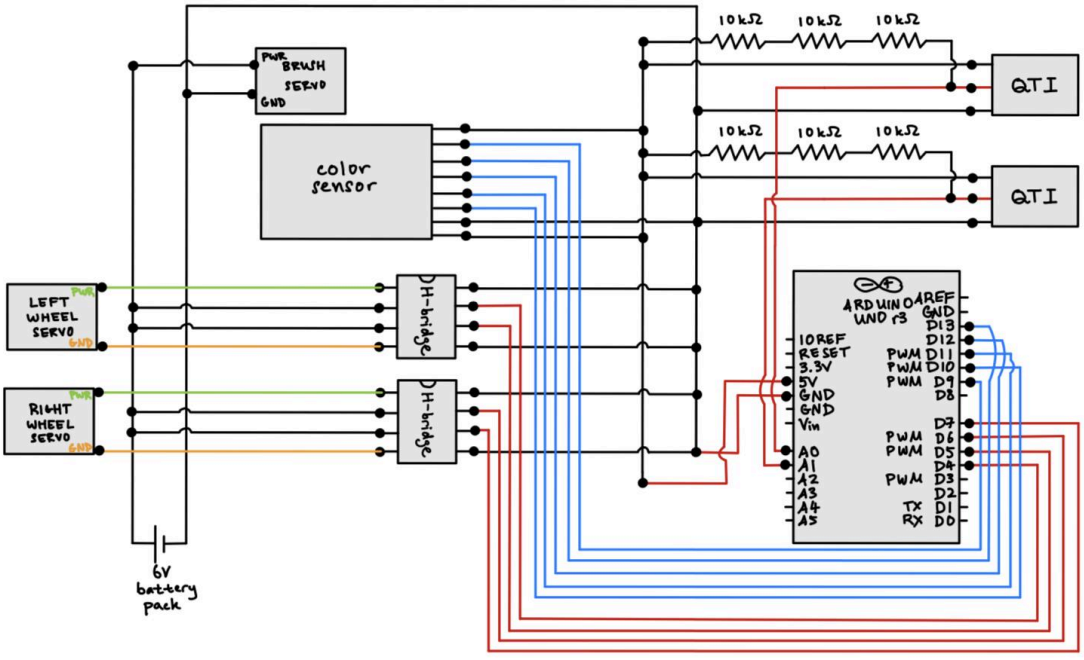

Figure 2b: Complete circuit diagram of electronics with Arduino and servos drawn



Appendix C: CAD files and drawings

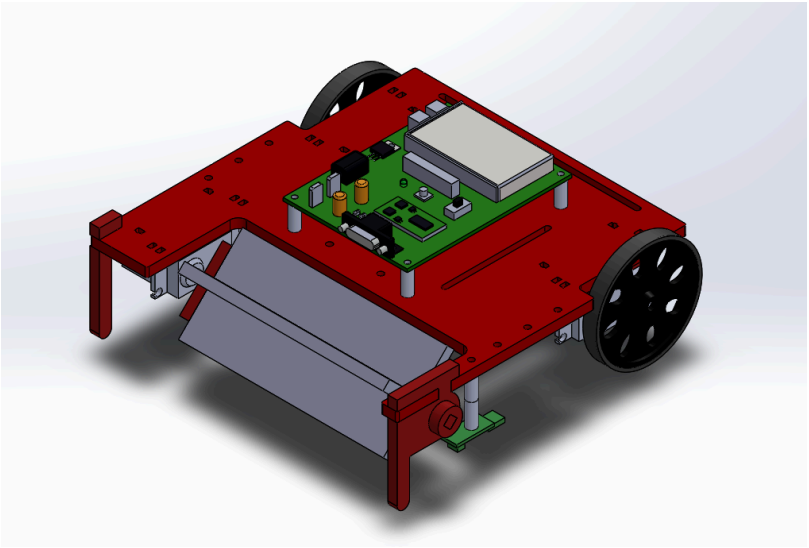

Figure 3: Isometric view of Chomp CAD

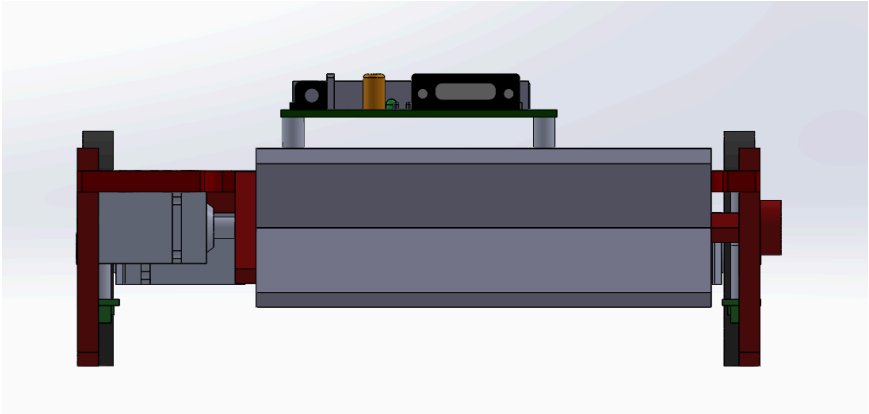

Figure 4: Front view of Chomp CAD (POV: You're about to be collected.)

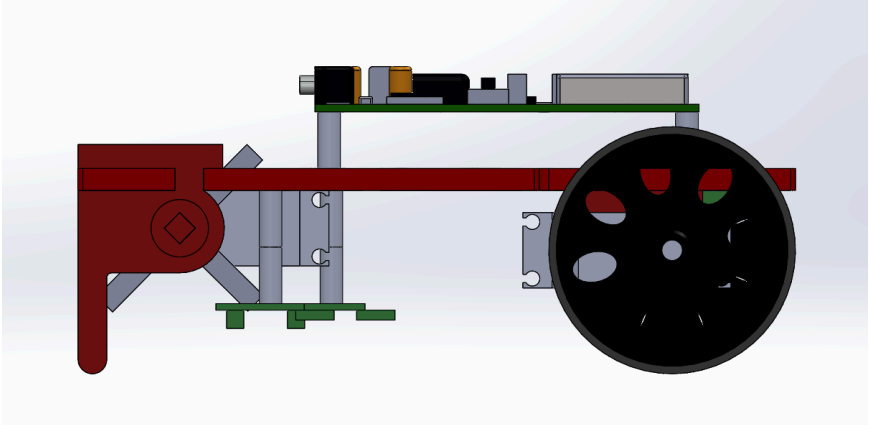

Figure 5: Side view of Chomp CAD

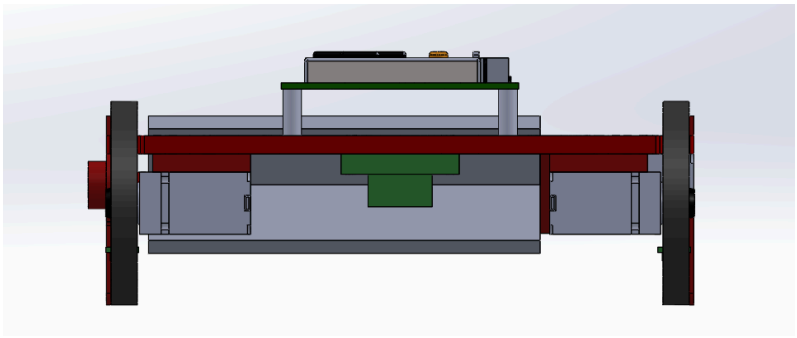

Figure 6: Back view of Chomp CAD

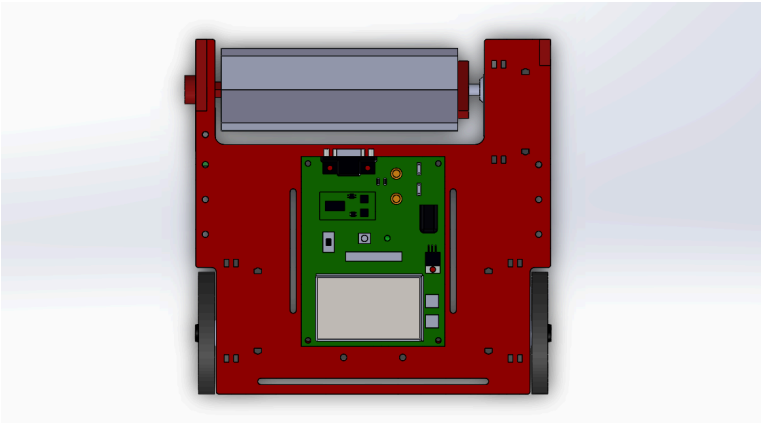

Figure 7: Top view of Chomp CAD

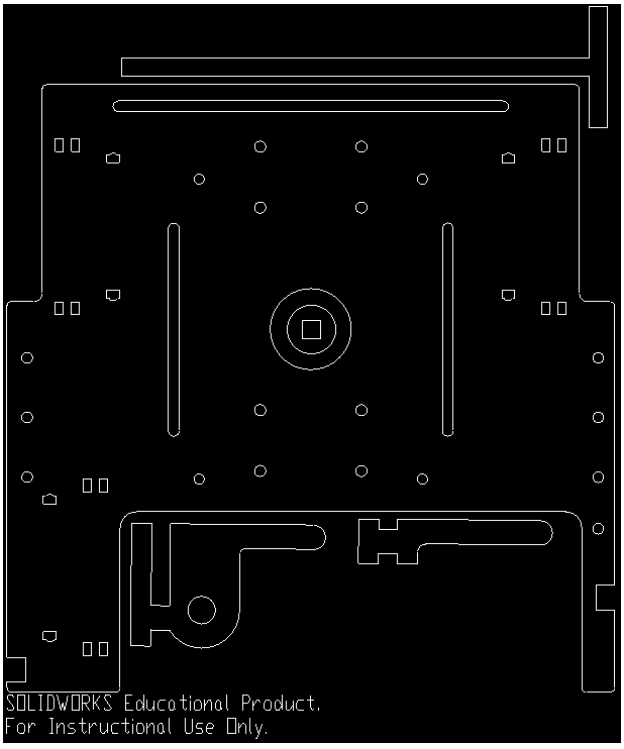

Figure 8: DXF File for Chomp's laser-cut acrylic parts

Appendix D: Flowchart

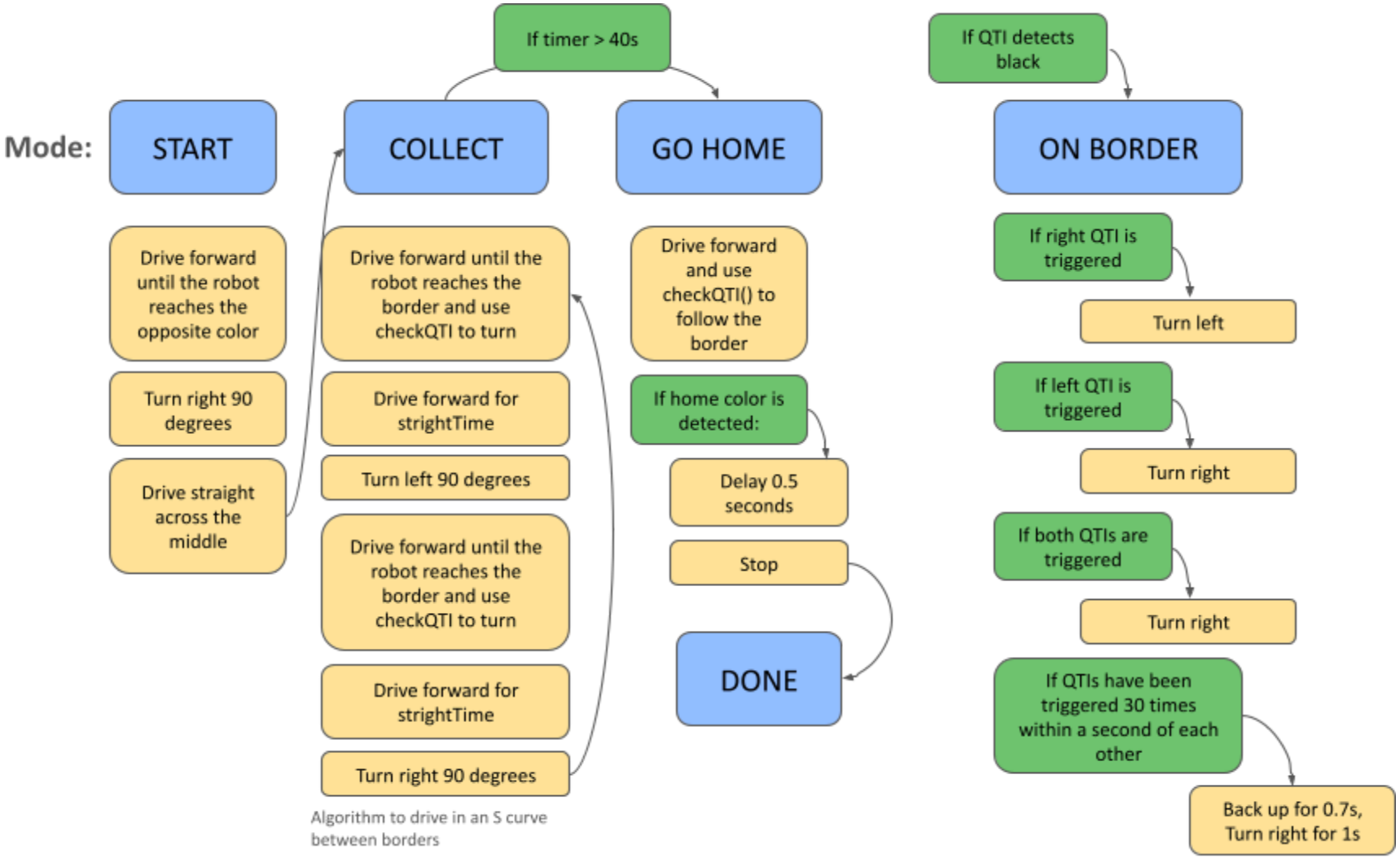

Figure 9: Flow chart of the competition algorithm



Appendix E: Code

```
//DEFINE VARIABLES

// Drive times
int turn90 = 93; //0.93 seconds
int straightTime = 60; //0.6 seconds
int driveTime = 600; //6 seconds

//Defining movement mode
enum Mode{
  START,
  COLLECT,
  ONBORDER,
  GOHOME,
  DONE

};
enum Mode mode;
enum Mode storeMode;

//Defining color sensor
volatile float period; //in ms
volatile int ctimer;
//Define QTI pins
int PIN_QTI_LEFT = 0b00000010; // pin A0
int PIN_QTI_RIGHT = 0b00000001; // pin A1

int borderHit = 0;
bool edge_left = PINC & PIN_QTI_LEFT;
bool edge_right = PINC & PIN_QTI_RIGHT;

struct colorRGB{
  int red;
  int green;
  int blue;
};

struct colorRGB homeRGB = {87, 142, 200}; //default is blue
struct colorRGB awayRGB = {250, 234, 179}; //default is yellow
int awaynotset = 1;

int checkingCorner = 0;
int cornerCheckStartTime = 0;
int cornerCount = 30;
int lastTouchedLeft = 9;

volatile int time = 0; //in hundredths of a second

int step = 1;
int startTime = 0;



// MAIN METHOD

int main(void){

  Serial.begin(9600);
  sei();
```



```
  DDRD = 0b11110000; // D4, 5, 6, 7 are outputs
  // D7 is left wheel forward
  // D6 is left wheel backward
  // D5 is right wheel forward
  // D4 is right wheel backward

  initColor();
  initQTI();
  initTimer();
  goForward();

  homeRGB = readRGB();
  mode = START;

  while(1){
  readQTI();
  struct colorRGB rgb = readRGB();

  /*
  Mode: START
  step 1 - drive forward until the robot reaches the opposite color
  step 2 - turn right 90 degrees
  step 3 - drive straight across the middle
  */
  if(mode == START){
    if(!isHome(rgb) && awaynotset){ //setting away color
      awayRGB = rgb;
      awaynotset = 0;
      Serial.println("step 1");
    }
    else if(!isHome(rgb) && step==1){
      turnRight();
      startTime = time;
      step = 2;
      Serial.println("step 2");
    }
    else if (step==2 && time-startTime>=turn90){
      goForward();
      step = 3;
      Serial.println("step 3");
    }
    else if (step==3){
      mode = COLLECT;
      step = 1;
      startTime = time;
      Serial.println("finished START");
      Serial.println("start COLLECT");
      Serial.println("step 1");
    }
  }

  /*
  Mode: COLLECT
  step 1 - drive forward until the robot reaches the border and use checkQTI to
turn
  step 2 - drive forward
  step 3 - turn left 90 degrees
  setp 4 - drive forward until the robot reaches the border and use checkQTI to
turn
  step 5 - drive forward
```



```
  setp 6 - turn right 90 degrees
  */
  if(mode == COLLECT){
    if((step == 1) && ((borderHit == 1) && (time-startTime >= driveTime))){
      startTime = time;
      Serial.println("step 2");
      step = 2;
    }
    else if((step == 2) && (time-startTime >= straightTime)){
      turnLeft();
      startTime = time;
      Serial.println("step 3");
      step = 3;
    }
    else if((step == 3) && (time-startTime >= turn90)){
      goForward();
      Serial.println("step 4");
      step = 4;
    }
    else if((step == 4) && ((borderHit == 1) )){ //&& (time-startTime >=
driveTime)
      startTime = time;
      Serial.println("step 5");
      step = 5;
    }
    else if((step == 5) && (time-startTime >= straightTime)){
      turnRight();
      startTime = time;
      Serial.println("step 6");
      step = 6;
    }
    else if((step == 6) && (time-startTime >= turn90)){
      goForward();
      Serial.println("start COLLECT");
      Serial.println("step 1");
      step = 1;
    }
  }


  /*
  Mode: GOHOME
  goes forward and reads QTI until the color sensor finds the home color - then
it stops
  */
  if((mode != GOHOME && mode != DONE) && (time >= 4000)){
    mode = GOHOME;
    goForward();
    Serial.println("start GOHOME");
  }
  if(mode == GOHOME){
    if(isHome(rgb)){
      goForward();
      _delay_ms(500);
      stop();
      mode = DONE;
    }
  }
}
```



```
// MOVEMENT FUNCTIONS

void goForward(){
  // right wheel forward, left wheel backward
  PORTD = 0b10010000;
}
void goBackward(){
  // right wheel forward, left wheel backward
  //constexpr int time = 0;
  PORTD = 0b01100000;
}
void turnLeft(){
  // right wheel forward, left wheel forward
  PORTD = 0b10100000;
}
void turnRight(){
  // right wheel backward, left wheel backward
  PORTD = 0b01010000;
}
void stop(){
  // both wheels stop
  PORTD = 0b00000000;
}

//COLOR SENSOR FUNCTIONS

void calibrateColors(){
    struct colorRGB rgb = readRGB();
    period = getColor();
}

//Color sensor interrupt function
ISR(PCINT0_vect) {
    if (PINB & 0b00000010) { //CHANGE BASED ON PIN
        TCNT1 = 0;
    } else {
        ctimer = TCNT1;
    }
}

void initColor()
{
  //USE TIMER0 FOR COLOR INTERRUPT
  // s0 - pin10, s1 - pin11, s2 - pin12, s3 - pin13
  // output - pin9

  // a. Initialize your I/O pins
  DDRB |= 0b00111100; //set pin2-5 to output
  DDRB &= 0b11111101; //set pin9 to input
  // b. Initialize my pin change interrupt
  PCICR |= 0b00000001; //enable PCIE0 which includes pin9(PCINT1)
  PCMSK0 = 0b0000000;
  // c. Initialize your timer
  TCCR1A = 0b00000000;
  TCCR1B = 0b00000001;
  TIMSK1 = 0b00000000;

  TCNT1 = 0;
```



```
  ctimer = 0;

  //initialize color filters
  //high low for 20%, low low for red
  PORTB = 0b00001000;
  //green
  // PORTB = 0b00111000;
  //blue
  // PORTB = 0b00011000;
}

int getColor()
{
  // a. Enable the specific bit for your Pin Change Interrupt
  PCMSK0 = PCMSK0 | 0b00000010;

  // b. Add a short delay (5 ~ 10 milliseconds)
  _delay_ms(5);
  // c. Disable the specific bit for your pin change interrupt
  // (to prevent further interrupts until you call getColor again)
  PCMSK0 = PCMSK0 & 0b11111101; //maybe only turn 1 off
  // d. Return the period in units of time
  //return (2*ctimer)/16; //converts timer ticks to period in ms
  return ctimer/8;
}

//Uses color sensor to read an RGB value for the current color
struct colorRGB readRGB(){
  // Set 20% scaling, red filter
        PORTB = 0b00000100;

        // Read red color and map to 0-255 range
        int red = getColor();
        int R = limit(range(red, 44.0, 415.0, 255.0, 0), 0, 255);

        _delay_ms(5);

        // Set green filter
        PORTB = 0b00110100;

        // Read green color and map to 0-255 range
        int green = getColor();
        int G = limit(range(green, 47.0, 475.0, 255.0, 0), 0, 255);

        _delay_ms(5);

        // Set blue filter
        PORTB = 0b00100100;

        // Read blue color and map to 0-255 range
        int blue = getColor();
        int B = limit(range(blue, 33.0, 349.0, 255.0, 0), 0, 255);

        //Reset to red filter (or whatever we use)
        PORTB = 0b00001000;

        // struct colorRGB v = {R,G,B};
        // Serial.print("red: ");
        // Serial.print(red);
        // Serial.print(", green: ");
```



```
        // Serial.print(green);
        // Serial.print(", blue: ");
        // Serial.println(blue);
        return {R,G,B};
}

// Used to troubleshoot the readRGB function and calibrate our color sensors
void printRGB(struct colorRGB color){
  Serial.print("(");
  Serial.print(color.red);
  Serial.print(", ");
  Serial.print(color.green);
  Serial.print(", ");
  Serial.print(color.blue);
  Serial.println(")");
}

// Boolean functions to check certain colors
bool isBlack(struct colorRGB color){
  return compareRGB(color, {0,0,0});
}
bool isBlue(struct colorRGB color){
  return compareRGB(color, {87, 142, 200}); //FIND EXPERIMENTALLY
}
bool isYellow(struct colorRGB color){
  return compareRGB(color, {250, 234, 179}); //FIND EXPERIMENTALLY
}
bool isHome(struct colorRGB color){
  return compareRGB(color, homeRGB);
}
bool isAway(struct colorRGB color){
  return compareRGB(color, awayRGB);
}

bool compareRGB(struct colorRGB c1, struct colorRGB c2){
  //range can be more than 5
  if ((c1.red > (c2.red + 20)) || (c1.red < (c2.red - 20))){
    return false;
  }
  if ((c1.green > (c2.green + 20)) || (c1.green < (c2.green - 20))){
    return false;
  }
  if ((c1.blue > (c2.blue + 20)) || (c1.blue < (c2.blue - 20))){
    return false;
  }
  return true;
}

//Range and limit functions - used in readRGB
float range(float x, float in_min, float in_max, float out_min, float out_max)
{
    return (x - in_min) / (in_max - in_min) * (out_max - out_min) + out_min;
}

int limit(int x, int min, int max) {
    if (x < min) {
        return min;
    }
    if (x > max) {
        return max;
```



```
    }
    return x;
}

//QTI FUNCTIONS

void initQTI() {
  DDRC = DDRC & 0b11111100; //sets pin A0 and A1 as inputs
}

void readQTI() {
  edge_left = PINC & PIN_QTI_LEFT;
  edge_right = PINC & PIN_QTI_RIGHT;
  if((edge_left || edge_right) && !checkingCorner){
    checkingCorner = 1;
    cornerCheckStartTime = time;

  }

  if(edge_left & edge_right){ //if both sensors read black
    if(mode != ONBORDER){
      storeMode = mode;
      mode = ONBORDER;
    }

    maxCornerHits(2);
    turnRight();
  }
  else if (edge_left){ //if left side reads black
    if(mode != ONBORDER){
      storeMode = mode;
      mode = ONBORDER;
    }
    turnRight();
    maxCornerHits(0);
  }
  else if (edge_right) { //if right side reads black
    if(mode != ONBORDER){
      storeMode = mode;
      mode = ONBORDER;
    }
    turnLeft();
    maxCornerHits(1);
  }
  else {
    if(mode == ONBORDER) {
      Serial.println("on border");
      mode = storeMode;
      borderHit = 1;
      goForward();
    }
    else{
      borderHit = 0;
    }
  }
}

// Counts each time the QTIs read the black border to see if the bot is stuck
in a corner
void maxCornerHits(int edge){
```



```
  //0 = left edge, 1 = right edge
  if(edge == 2){
    cornerCount -=1;
    lastTouchedLeft = !lastTouchedLeft;
  }
  else if ((lastTouchedLeft==1) && (edge == 1)){
    //if it was left and is now right
    cornerCount -= 1;
    lastTouchedLeft = 0;

  } else if ((lastTouchedLeft==0) && (edge == 0)){
    //if it was right and is now left
    cornerCount -= 1;
    lastTouchedLeft = 1;

  } else if (lastTouchedLeft == 9){
    cornerCount -= 1;
    lastTouchedLeft = !edge;
  }
  if(cornerCount <= 0 && (time - cornerCheckStartTime <= 100)){
    //1 sec
    stop();
    goBackward();
    _delay_ms(700);
    turnRight();
    _delay_ms(1000);
    time += 170;
    cornerCount = 30;
    checkingCorner = 0;
    lastTouchedLeft = 9;

  }
  else if((time-cornerCheckStartTime) > 100){
    cornerCount = 30;
    checkingCorner = 0;
    lastTouchedLeft = 9;
  }
}

//TIMER

//Timer interrupt function - increments the timer value
ISR(TIMER0_COMPA_vect){
  time = time + 1;
}

void initTimer(){
  //Using timer 0 to count hundreths of a second
  TCCR0A = 0b00000010;
  TCCR0B = 0b00000101; //sets prescaler of 1024 (in data sheet)

  OCR0A = 156; //count corresponding to 1 hundredth of a second

  TIMSK0 = 0b00000010; //enable CTC A interrupt

  time = 0;
}
```



Appendix F: Silly Pictures

Figure 10: Chomp pit crew

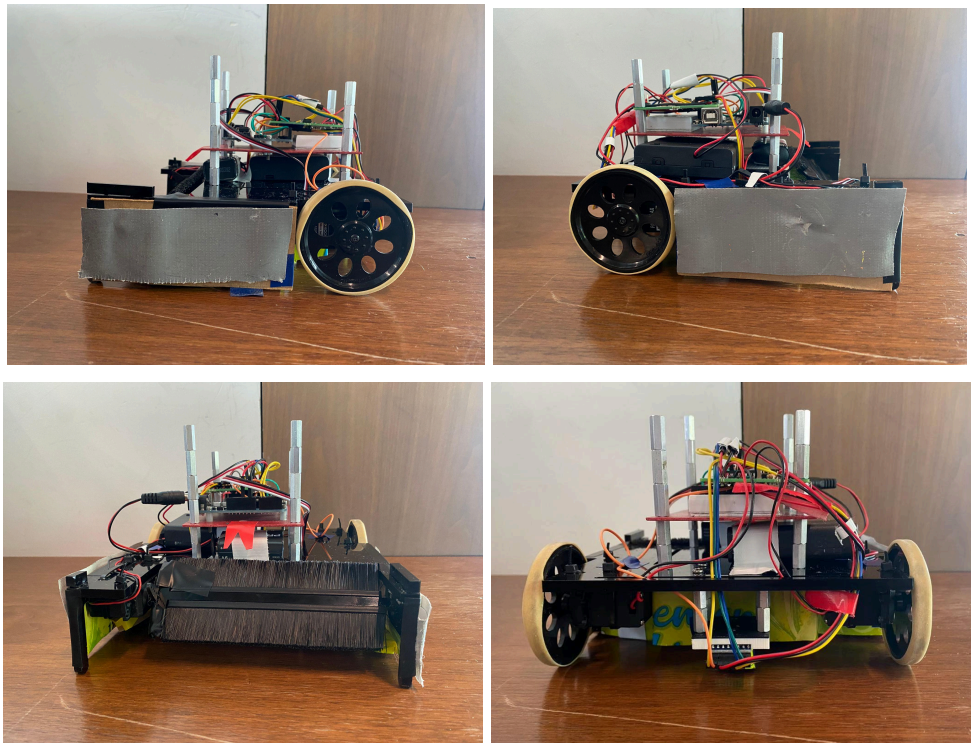

Figure 11: Chomp's Good Side (all of them)

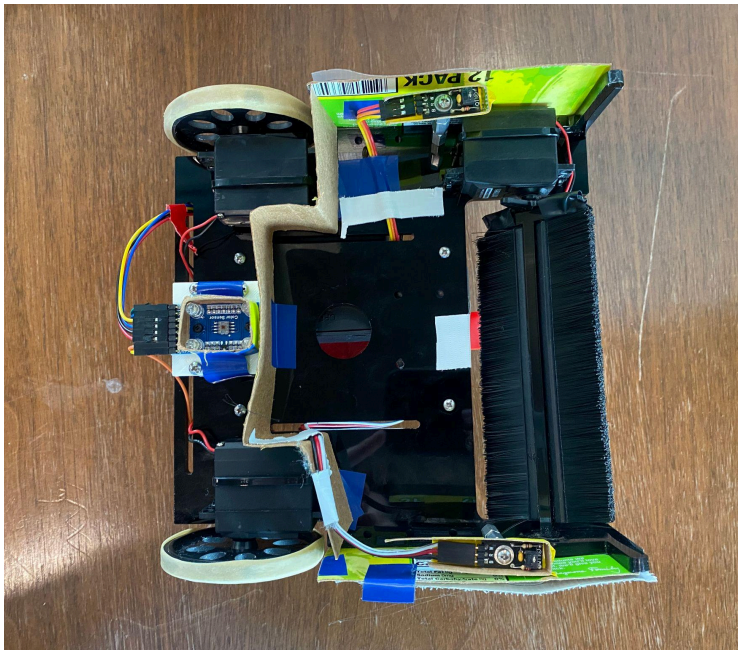

Figure 12: Undercarriage

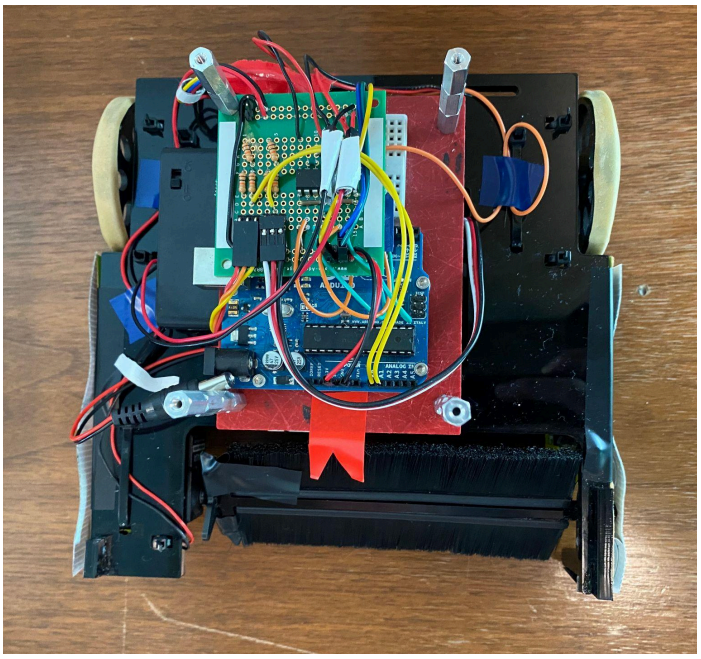

Figure 13: His Brains

Figure 14: Block POV

Figure 15: Slaughtered