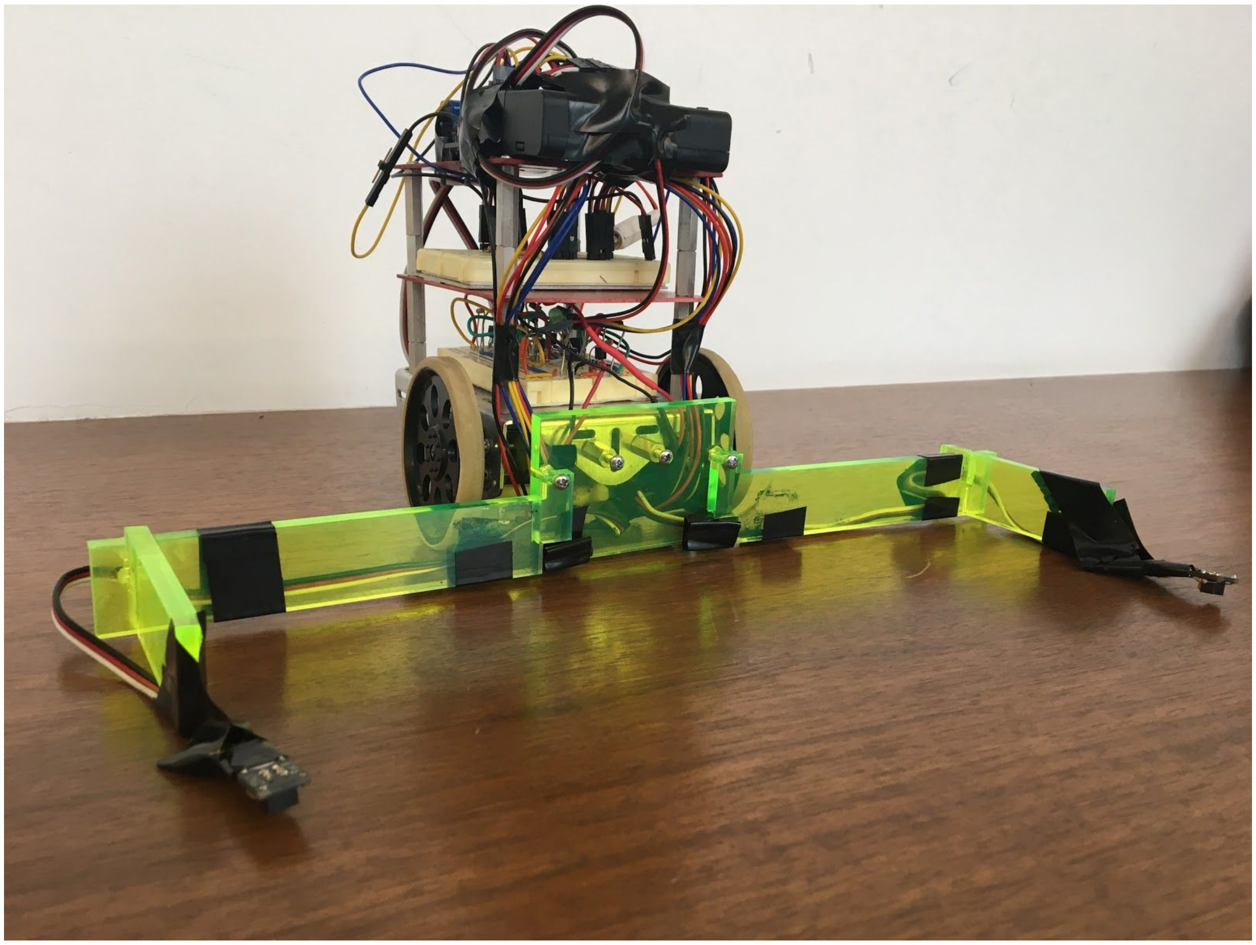
MAE 3780
Mechatronics

Fall 2019

Final Project Report



Produced by:

Jacquelyn Peng, Kimberly Yap, Bryan Zhong

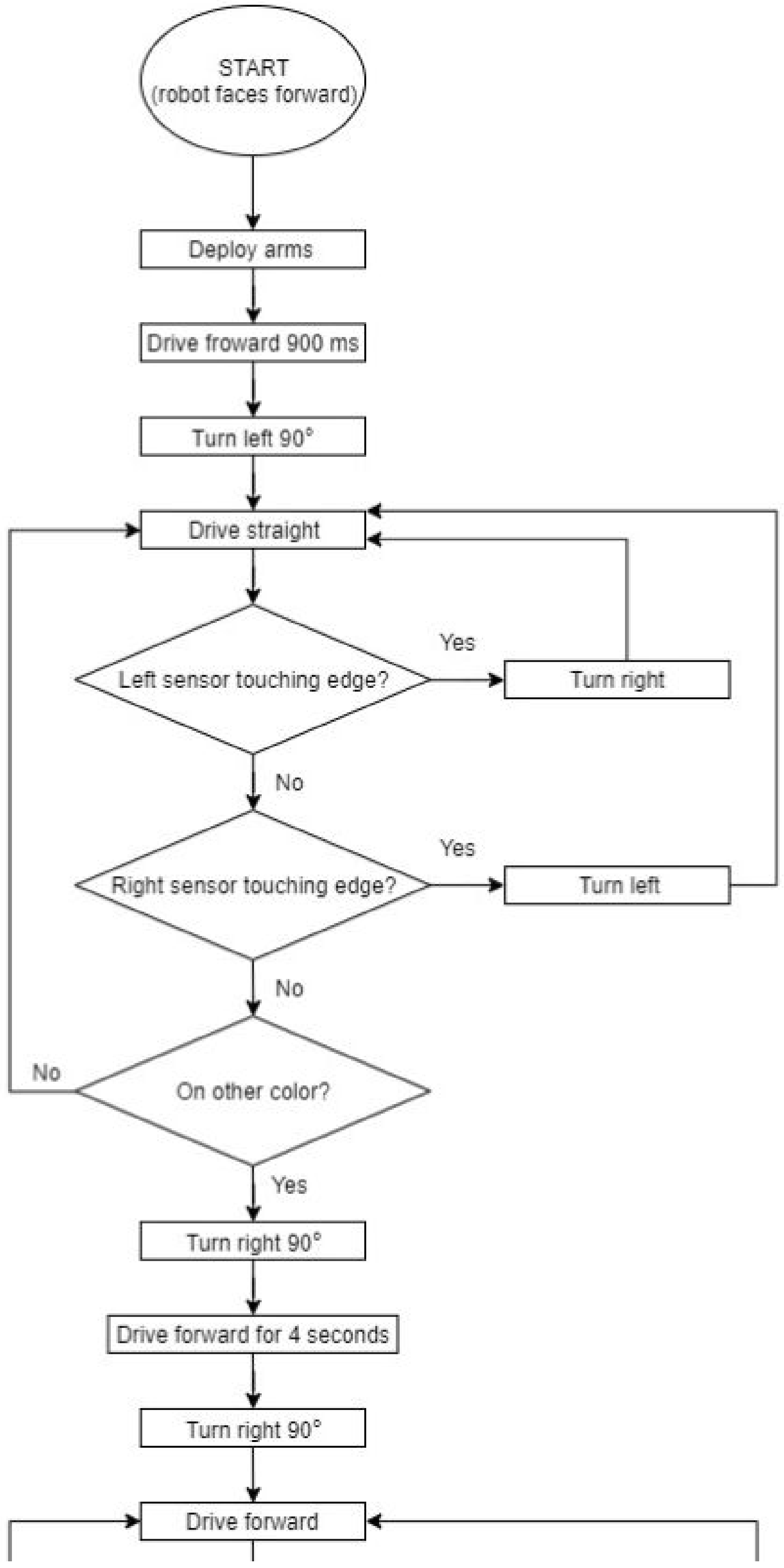
Introduction
The objective of the Cube Craze project is to design and construct a robot that moves cubes in a competition against other classmates. The goal of each one-on-one match is to have more cubes on your starting side than on your opponent's side at the end of one minute. In order to construct our robot, we utilized an Arduino Uno to control our robot, Atmel Studio to write code for our robot's behavior, and tools supplied in our bin to physically build the robot. The following report outlines the motivation behind our design, a flowchart of our gameplay strategy, our robot's strengths and weaknesses, a review of how we performed in the competition, as well as future modifications that can be done to our robot.

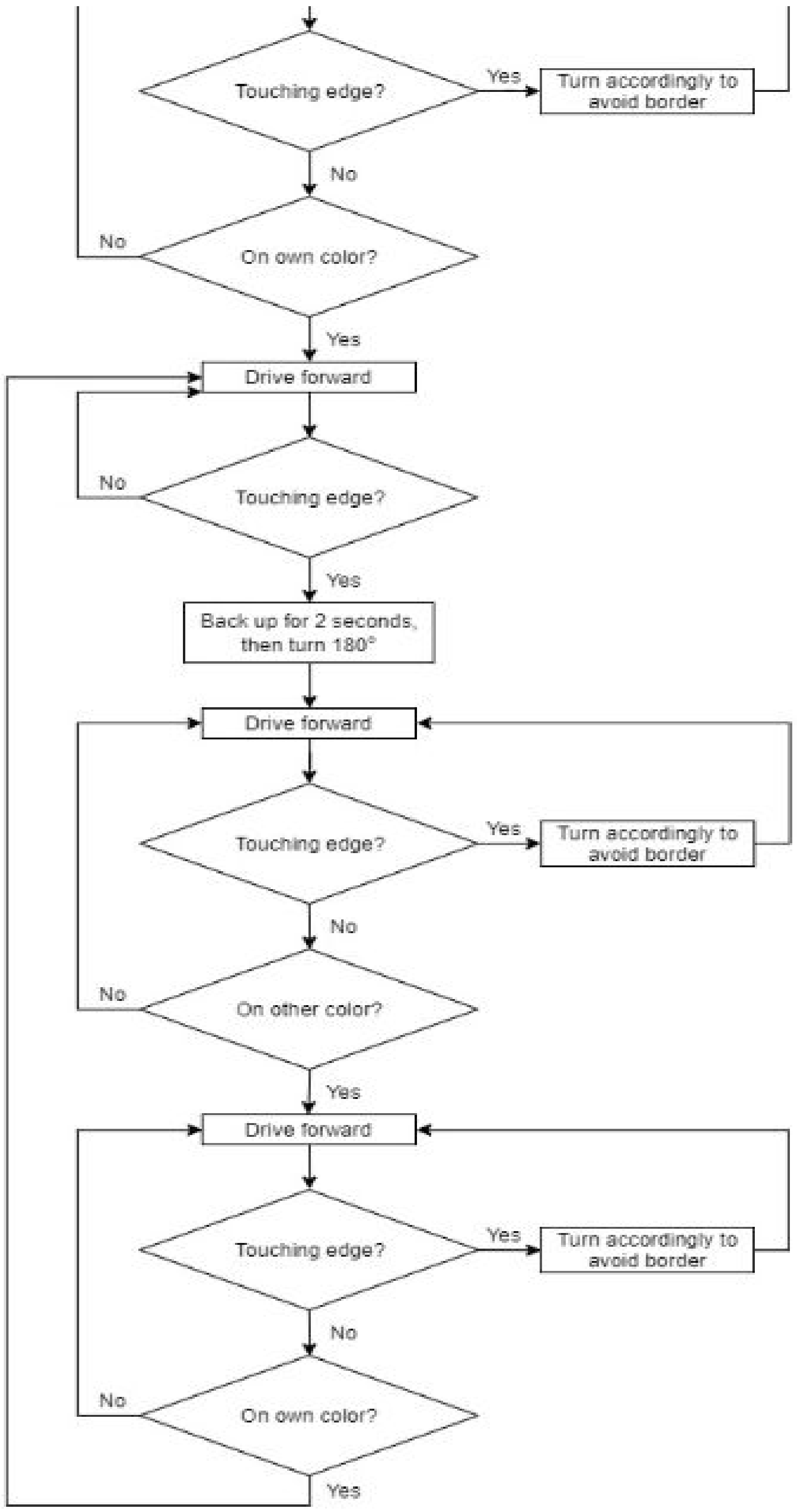
Motivation Behind the Design
Our mechanical design for our robot consisted of a color sensor, QTI sensors, and laser-cut arms. We used one color sensor placed underneath the body of our robot to detect the color of the side it started on and also continuously check what color it was on during the match. At the start of every match, we used our color sensor to detect when our robot crossed from our side to our opponent's side to indicate when it should turn in the direction of the blocks to sweep most of them during our first pass. We also used the color sensor in order to help the robot keep track of what side it is on at all times, so that as long as all the sensors work as they should, no matter what happens upon potential contact with the other robot, our robot would never lose track of which side it is on and should not accidentally do things like dropping off, and losing possession of, the blocks on the opponent's color. In addition, we used two QTI sensors, each placed on the end of each arm, to detect the surrounding black border to make sure that the robot stayed within the playing field. Each time a QTI sensor detected the black border, the robot would turn just enough in order to avoid the border, allowing it to align itself with the edge of the playing field. The final component of our robot that is important to discuss is its set of gravity-deployed arms assembled from laser-cut acrylic parts. The arms are attached by screws to an acrylic backboard that is attached to the chassis. At the start of each match, the arms are 90 degrees off the ground, held up only by the friction provided by the screw attachment to the backboard. The screw attachment is just tight enough so that it holds the arms up when the robot is still, but once the robot begins moving back and forth to deploy the arms, the net moment generated by the center of gravity of each arm overcomes the friction from the screws, which causes the arms to fall 90 degrees until they touch the ground.

Cost Analysis:

Item	Cost
Clear Scratch and UV-Resistant Cast Acrylic Sheet (12" x 12" x 1/4") from McMaster	$14.16
¼ of a roll of electrical tape	$0.42
Total	$14.58

Flowchart of Strategy





NOTE: The part of our robot's code near the end is in one while loop for the duration of the match, so there is no end statement.

Robot's Strengths and Weaknesses

Strengths:

- Our arms had a wingspan of about 13 inches when deployed. This allowed us to collect a majority of the cubes at the start of the competition during our first sweep, and effectively sweep much of the area of the other team's side during subsequent trips there.
- We had a good strategy that was well thought out that would allow us to get the maximum amount of blocks possible in the short amount of time we had. The robot first focuses on covering the middle section of the board at the start of the competition because we know that most of the blocks will be there at the start. After it returns the first set of blocks back to its own side, it travels to the other side to sweep the other team's area to try to get more cubes to bring back to its own side. Theoretically, by looking at our code, our robot would have worked really well if all our sensors and mechanical parts worked as expected, and would probably have a good chance at winning most matches.
- Our robot doesn't lose track of which side it is on at any point in the match. It will only back up to drop off its blocks when it is on its own side, no matter what happens upon potential contact with the other robot.

Weaknesses:

- We secured the QTI sensors to the robot's arms by taping the cables to which they were connected to onto the ends of the arms. Doing this did not secure the sensors at a fixed height from the ground, which added a lot of variability to whether or not the sensors worked at any point in the match.
- Our motors were not powerful enough. We usually got pushed off the board whenever we got in a head-on collision with another robot.
- Our tires did not stay on the wheels very well. They easily slipped off of our wheels during the match, especially upon contact with another robot.
- The screws from the connection of the arms to the backboard kept coming loose, so we had to tighten them after each match, and each time we had to make sure they were still loose enough for the arms to successfully deploy. This created more room for error each time because we couldn't test our robot multiple times each time we tightened the screws to make sure the arms will deploy without fail.

Discussion of Competition Performance

What worked well:

Something that worked well during the competition was that most of our more complicated circuits stayed together well, like our H-bridge. The night before, during testing, one of our motors somehow got plugged into the wrong place on the breadboard, and luckily this kind of situation did not happen on the day of the competition. Our robot's arms deployed successfully within the first few seconds of the match most of the time when we put more effort into making sure that the screws connecting them were not too tight. The color sensor worked well also, allowing the robot to properly detect which side it is on at all times during the match, and as desired, the robot only backed up to drop off the blocks at its own side, and never on the opponent's side. When the QTI sensors were at the right height from the board and were working properly at the beginning of each match, the robot behaved as expected according to our code, but this usually didn't last long because the QTIs would go out of place upon contact with the other robot. For some matches, the other team's robot would stop moving for some reason, so even if our robot's QTI sensors began to malfunction in the middle of the match, the blocks we collected during the beginning of the match still gave us a majority of the blocks on our side, even if both robots stayed stationary for the rest of the match.

What did not work well:

Our QTI sensors were not secured to the ends of our arms well. We attached our QTI sensors by taping the cables connected to them to their respective arms, which allowed for flexibility in terms of how far the QTI sensors were from the ground. At the beginning of the match, at random times during the match, and especially upon contact with the other robot, one of the QTI sensors sometimes came too far off the ground, making it constantly read high. This continuously triggered the interrupt that caused the robot to turn until the border was no longer detected, making it spin in circles for the rest of the match. During our first two matches, one of the QTIs was in a bad position at the start, which made the robot spin in circles for the entire match. During the competition, we tried to add more tape to keep the QTI sensors at a fixed height from the ground, but they were still not very secure, even if we kept checking in between matches to make sure the QTI sensors started off in the right position. When our robot hit the other robot during a match, it was very easy for our QTI sensors to get out of position, which made the robot unable to do anything else meaningful for the rest of the match.

We also faced some challenges with our arms during the competition. Since our strategy was to deploy the arms using gravity, we had to keep adjusting the tightness of the screws that held our arms to the backboard piece that was fixed to the chassis. We had to make sure that the screws were tight enough to hold the arms vertically before the match started to fit within the 8" x 8" constraint. However, the screws also had to be loose enough so that the arms would deploy via gravity when our robot drove back and forth rapidly at the beginning of the match. In order for the arms to deploy correctly, the center of gravity of each arm while it was up in the starting position had to be in the correct location relative to the screw attaching the arm to the backboard, so that it would produce a net moment in the right direction once the friction in the screws is overcome when the robot begins trying to deploy its arms. If the center of gravity is in the wrong location, then the arm may deploy in the wrong direction. In one of our matches, the center of gravity of one arm was in the wrong direction relative to the screw attachment. When the robot tried to deploy its arms, this arm tended to rotate in the opposite direction than what we wanted. It is really important that our arms fall in the right direction at the beginning of each match, because our QTI sensors are attached to the arms, and the robot would not perform properly if the QTI sensors are not on the ground.

When we got into a head-on collision with the other robot during the match, the other robot tended to push ours off the board, either because our motors were not powerful enough, or because the rubber band tire from our wheels slipped off, causing the wheel to lose traction with the ground.

<u>Future Modifications to our Robot</u>

- Create arms with holes on the bottom to screw in the QTI sensors so that they are secure and at a fixed height from the ground.
- Buy stronger motors so that our robot would not be pushed around by the opponent.
- Buy wheels with built-in tires so that we would not have to worry about the tires slipping off upon collision with another robot.
- Use motors to deploy the arms so that we would not have to rely on gravity to make sure that the arms dropped in the proper orientation at the beginning of every match. Alternatively, we could still use the gravity-deployed arms, but be more mindful of their exact starting position and the tightness of the screw attachments so that the arms will always deploy in the right direction.

<u>Conclusion</u>

Unfortunately, our robot did not really perform as expected at the competition. Seemingly small problems as simple as a QTI sensor being a few millimeters further off the ground than it should or the arms starting off in the wrong angle prior to deploying caused us a lot of problems during each match. There is room for improvement in terms of our mechanical design, specifically in securing our QTI sensors to our arms. Otherwise, most of the aspects of our robot were well thought out and if we were to make the necessary improvements to fix these issues, our robot would have a lot more potential to perform much better in the future.

Appendix

Competition Code:

```
/*
 * FinalCompetition.c
 *
 *
 * This is the code that runs at the final competition.

 */


#include <avr/io.h>


#define F_CPU 16000000UL
#include <avr/io.h>
#include "serial.h"
#include <util/delay.h>
#include <avr/interrupt.h>

int color_1=0;
int starting_color=0;
int i=0;     // for loop counter
int back_up = 0;
int leftQTI = 0;
int rightQTI = 0;

// gets period of output wave signal from color sensor

ISR(PCINT0_vect)
{

        if((PINB & 0b00010000) == 0b00010000) // check if pin 4 (PB4) is high
        {
                TCNT1 = 0b00000000; // reset the timer
                } else {
                color_1 = TCNT1; // set period to current timer value
        }
}

// detectBorder executes when external interrupt for either QTI is triggered (border is detected)
// this function gets the robot to turn accordingly to avoid the border.
// or, if the robot is supposed to drop off cubes, this gets the robot to back up upon detecting the border, allowing it to drop off
the cubes there.

void detectBorder()
{
        if(back_up == 1)      {               // this is the code for dropping off blocks on own side once QTI detects the border after
reaching own side
                // go backwards
                PORTD &= 0b00000000; // reset motors
                PORTB &= 0b00000000;
                PORTB |= 0b00001000; // set right motor reverse
                PORTD |= 0b00100000; // set left motor reverse
                _delay_ms(2000);   // just enough time for arms to be clear of cubes
                PORTD &= 0b00000000; // reset motors
                PORTB &= 0b00000000;
                // turn 180 degrees
                PORTD |= 0b01000000; // set left motor forward
                PORTB |= 0b00001000; // right motor reverse
                _delay_ms(2900);
                PORTD &= 0b00000000; // reset motors
                PORTB &= 0b00000000;
                back_up = 0;          // finished dropping off cubes; now detectBorder() operates as usual, ready for next
round of cube collection
                // drive forward
                PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
                PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
        }
        else {                  // regular response; if robot is NOT dropping off cubes and just wants to avoid the border
                while((PIND &= 0b00000100) == 0b00000100)     {              // while left QTI reads high
                        // turn right a bit
                        PORTD &= 0b00000000; // reset motors
                        PORTB &= 0b00000000;
                        PORTD |= 0b01000000; // set left motor forward
                        PORTB |= 0b00001000; // right motor reverse
                        _delay_ms(50);
                        PORTD &= 0b00000000; // reset motors
                        PORTB &= 0b00000000;
                }
                while((PIND &= 0b00001000) == 0b00001000)     {              // while right QTI reads high
                        // turn left a bit
                        PORTD &= 0b00000000; // reset motors
                        PORTB &= 0b00000000;
                        PORTD |= 0b00100000; // set left motor reverse
                        PORTB |= 0b00000100; // right motor forward
                        _delay_ms(50);
                        PORTD &= 0b00000000; // reset motors
                        PORTB &= 0b00000000;
                }
        }

        // drive forward
                PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
                PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive

}
```

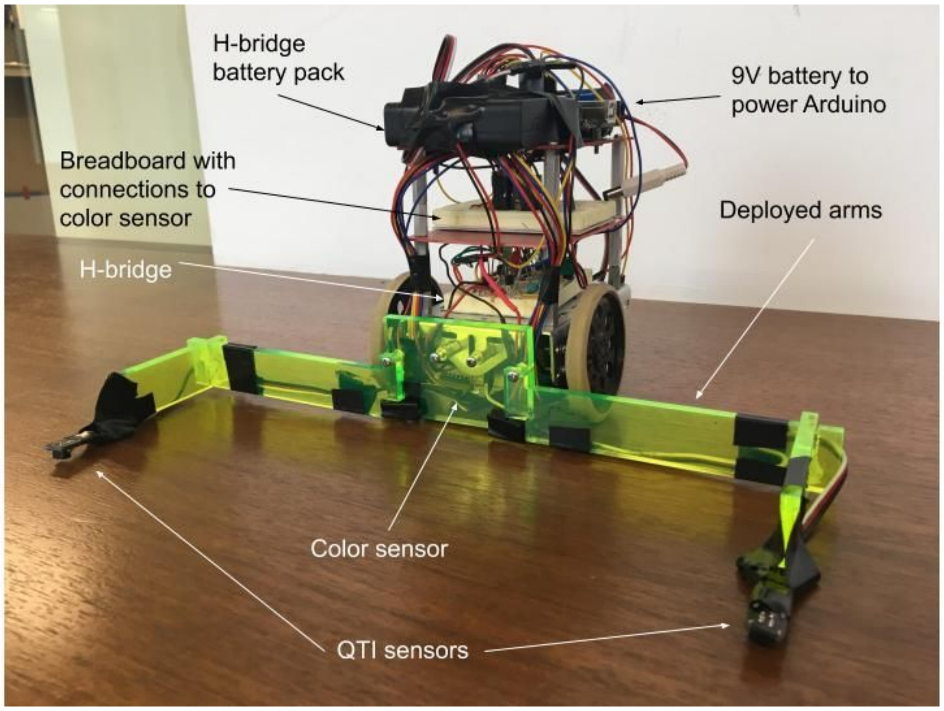
```
ISR(INT0_vect) { // interrupt vector for left QTI
        detectBorder();
}

ISR(INT1_vect) { // interrupt vector for right QTI
        detectBorder();
}


// initColor initializes all the pins necessary for color detection, border detection and any resulting actions
void initColor()
{
        sei(); // enable interrupts globally

        // INITIALIZE I/O PINS
        DDRB &= 0b11001111; // sets pin 12 (PB4) and pin 13 (PB5) to be input pins for color sensors 1 and 2
        DDRD &= 0b11100011; // sets pin 2 (PD2) and pin 3 (PD3) as input pins for QTI

        // INITIALIZE MOTOR PINS
        // H-Bridge 1: P0 on pin 5, P1 on pin 6
        DDRD |= 0b01100000; //set PWM pins 5 and 6 (PD5 and PD6) as outputs

        // H-Bridge 2: P0 on pin 11, P1 on pin 10
        DDRB |= 0b00001100; //set PWM pins 10 and 11 (PB2 and PB3) to be outputs

        // initialize pin change interrupt
        PCICR = 0b00000101; // initialize register B for PC interrupts

        // initialize timer 1, normal mode
        // (WGM10, WGM11, WGM12, WGM13 = 0), prescaler to 1 (CS12, CS11 to 0, CS 10 = 1)
        TCCR1A = 0b00000000;
        TCCR1B = 0b00000001;

        // initialize external interrupts
        EICRA |= 0b00001111;        // set external interrupts at pins 2 and 3 to trigger at rising edge
        EIMSK |= 0b00000011;        // enable external interrupts 0 and 1

}

// getColor1 returns the period of timer 1 in microseconds, which is related to the input from color sensor 1

int getColor1()
{
        PCMSK0 |= 0b00010000; // enable pin 12 (PCINT4) as a PC interrupt
        _delay_ms(10); // Give the interrupt a sec
        return color_1*.0625*2; // Change to units of microseconds
        PCMSK0 &= 0b11101111;
}
```

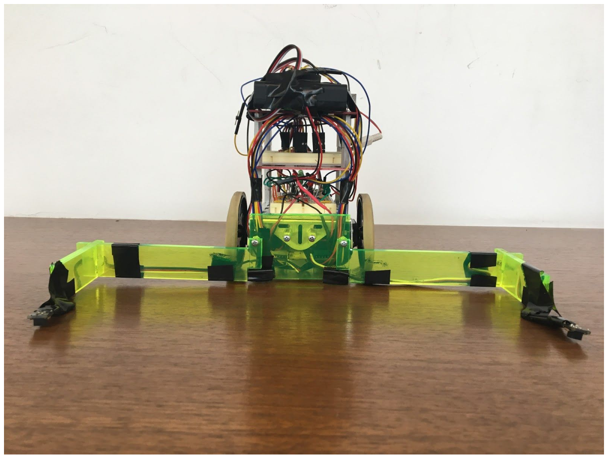
```
// main function

int main(void)
{
        init_uart(); // initialize serial
        initColor(); // initialize everything else
        starting_color = getColor1(); // assigns the starting color period to the variable starting_color


        // ROBOT STARTS FACING FRONT


        // deploy arms by driving back and forth 3 times
        for(i=0; i<3; i++)      {               // executes 3 times
                PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
                PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
                _delay_ms(300);
                PORTD &= 0b00000000; // reset motors
                PORTB &= 0b00000000;
                PORTD |= 0b00100000; // set left motor reverse
                PORTB |= 0b00001000; // right motor reverse
                _delay_ms(300);
                PORTD &= 0b00000000; // reset motors
                PORTB &= 0b00000000;
        }

        // drive forward a bit, then turn left 90 degrees
        // (this is to make sure the QTIs on the arms will not be off the border after deploying)

        PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
        PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
        _delay_ms(900);
        PORTD &= 0b00000000; // reset motors
        PORTB &= 0b00000000;
        // turn left 90 degrees
        PORTD |= 0b00100000; // set left motor reverse
        PORTB |= 0b00000100; // right motor forward
        _delay_ms(600);
        PORTD &= 0b00000000; // reset motors
        PORTB &= 0b00000000;


        // drive forward (if not detecting border), or follow border on left hand side (if detected) until other color is detected
        while ((getColor1() >= (starting_color-100)) && (getColor1() <= (starting_color+100)))      {
        // while color is same as starting color
                // drive forward
                PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
                PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
        }

        // we are out of the while loop, which means that the other color has now been detected
        EIMSK = 0b00000011;          // enable external interrupts 0 and 1

        // GO ACROSS BOARD TO GET CUBES

        PORTD &= 0b00000000; // reset motors
        PORTB &= 0b00000000;

        // drive forward a bit
        PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
        PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
        _delay_ms(200);

        // turn 90 degrees right
        PORTD &= 0b00000000; // reset motors
        PORTB &= 0b00000000;
        PORTD |= 0b01000000; // set left motor forward
        PORTB |= 0b00001000; // right motor reverse
        _delay_ms(800);    // modify this time until it gives a clean 90 degree turn
        PORTD &= 0b00000000; // reset motors
        PORTB &= 0b00000000;

        // drive forward across
        PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
        PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
        _delay_ms(4000);


        // begin return to own side
        // turn a little more than 90 degrees right
        PORTD &= 0b00000000; // reset motors
        PORTB &= 0b00000000;
        PORTD |= 0b01000000; // set left motor forward
        PORTB |= 0b00001000; // right motor reverse
        _delay_ms(1000);


        // start driving back to its own side. drive straight (utilizing QTI interrupts if necessary) until reaching own side again
        PORTD &= 0b00000000; // reset motors
        PORTB &= 0b00000000;
        PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
        PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive


        while(!((getColor1() >= (starting_color-100)) && (getColor1() <= (starting_color+100)))) { // if not detecting own color
                // continue driving straight (QTI interrupts will trigger as necessary)
        }
```

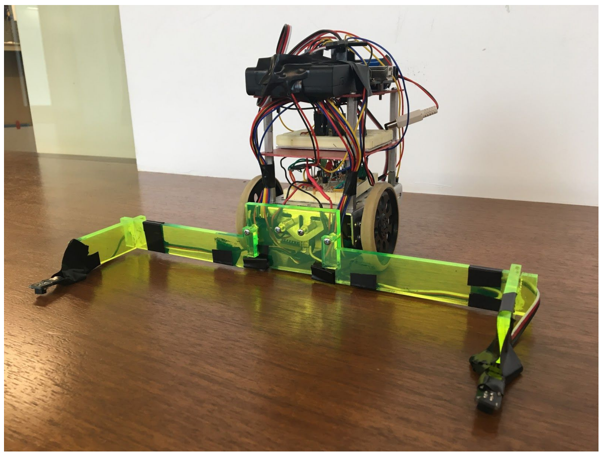
```
	// exited while loop. it is now back on its own color!

	// keep going straight until a QTI is triggered (until it reaches a border on its own side)
		back_up = 1;			// if back_up=1, it will get the robot to back up the next time the QTI triggers
		while(back_up == 1) {		// blocks have not yet been finished dropping off
			while(!((getColor1() >= (starting_color-100)) && (getColor1() <= (starting_color+100)))) { // while
not on own color
				back_up = 0; // don't back up to drop off cubes; if QTI triggers, execute avoiding border,
not dropoff
				// drive forward
				PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward
drive
				PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward
drive
			}
			// now it is on its own color
			back_up = 1;		// if QTI triggers, execute dropoff
			// drive forward
			PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
			PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
		}

		// back_up = 0 now, which means that the blocks have finished being dropped off; robot has backed up and
turned around 180 degrees



	// FIRST TRIP TO COLLECT BLOCKS COMPLETE! now, we will continue to scavenge for blocks. they can be anywhere on
the other side.
	// basically, randomly drive around on the other side's color and hope we can grab some blocks. bring the blocks over
to our own side. repeat forever.

	while(1)	{
		// drive forward (with QTI interrupts as necessary) until it reaches the other side
		while ((getColor1() >= (starting_color-100)) && (getColor1() <= (starting_color+100)))	{ // while color is
same as starting color
			// drive forward
			PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
			PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
		}

		// we have reached the other side! keep driving forward and hopefully get some blocks, until we reach our
own side again

		while(!((getColor1() >= (starting_color-100)) && (getColor1() <= (starting_color+100)))) { // if not detecting
own color
			// continue driving straight (QTI interrupts will trigger as necessary)
			PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
			PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
		}
```

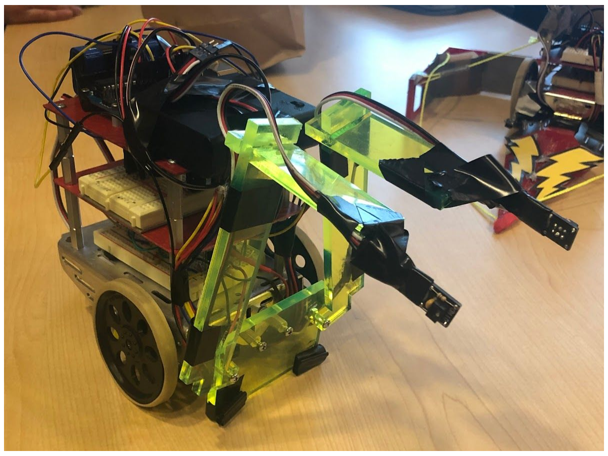
```
                    // we have returned to our own side!
                    // keep going straight until a QTI is triggered (until it reaches a border on its own side)

                    back_up = 1;                    // get robot to back up the next time the QTI triggers
                    while(back_up == 1) {           // blocks have not yet been finished dropping off
                              while(!((getColor1() >= (starting_color-100)) && (getColor1() <= (starting_color+100)))) { // while
not on own color
                                        back_up = 0; // don't back up to drop off cubes; if QTI triggers, execute avoiding border,
not dropoff
                                        // drive forward
                                        // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
                                        PORTD |= 0b01000000;
                                        // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive
                                        PORTB |= 0b00000100;
                              }

                              // it is on own color
                              back_up = 1;          // if QTI triggers, execute dropoff
                              // drive forward
                              PORTD |= 0b01000000; // set P0 (PD5, pin 5) to 0 and P1 (PD6, pin 6) to 1 for forward drive
                              PORTB |= 0b00000100; // set P0 (PB3, pin 11) to 0 and P1 (PB2, pin 10) to 1 for forward drive

                    }

                    // blocks finished dropping off; robot has backed up and turned around 180 degrees

          }

}
```

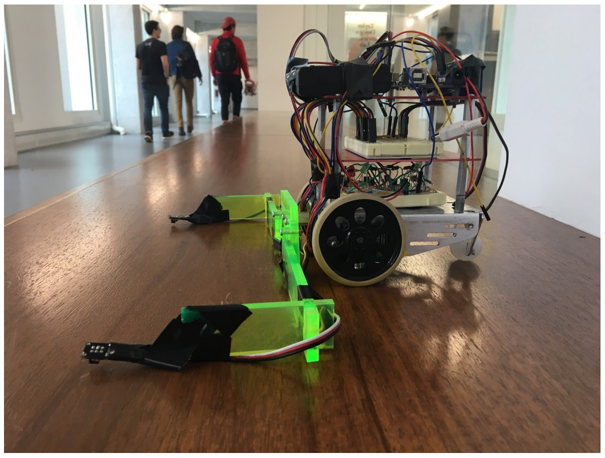
Pictures of our robot

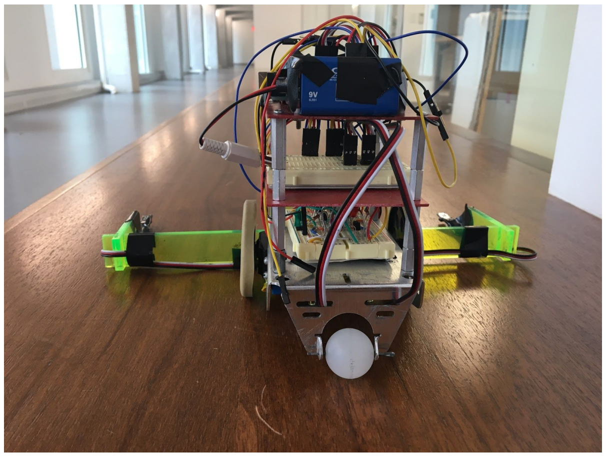

H-bridge battery pack
9V battery to power Arduino
Breadboard with connections to color sensor
Deployed arms
H-bridge
Color sensor
QTI sensors

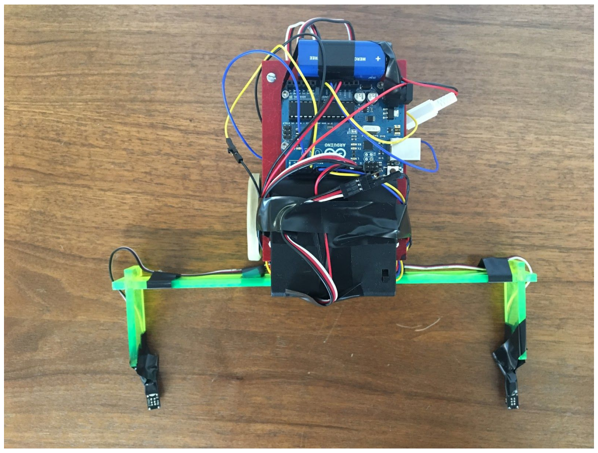




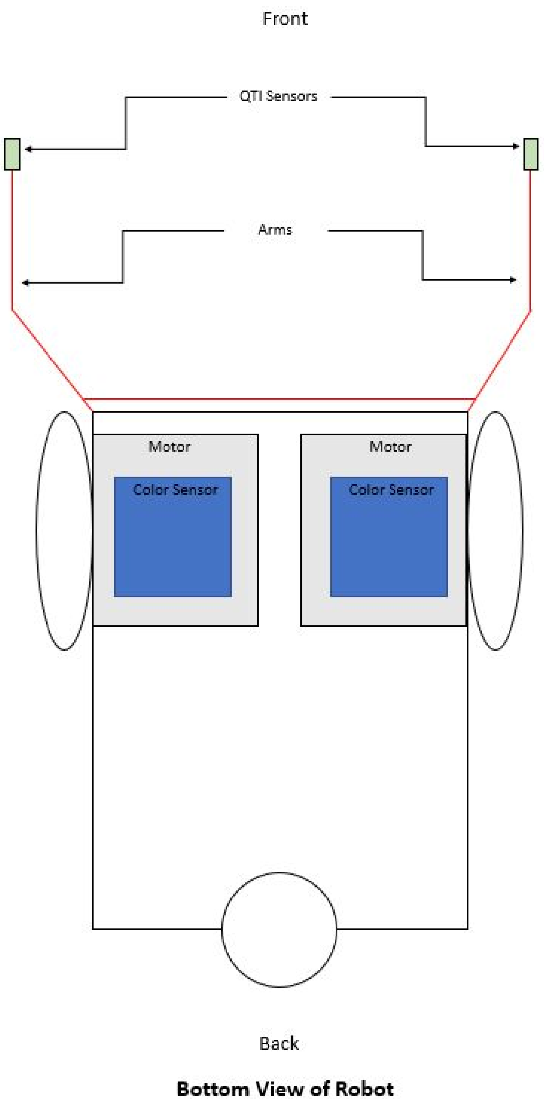








Schematic of robot:



NOTE: In the pictures and diagrams, you may see two color sensors attached to the bottom of our robot and connected through the breadboard. However, we did not include the second color sensor as an actual part of our robot because we did not end up using it at all (we did not write up any code that uses any of its information).

<u>Future Modifications to rules/constraints</u>

- During competition, designate an area for teams to place their robots down in such a way that people walking by won't bump into the robots and potentially break them