CORNELL UNIVERSITY
SIBLEY SCHOOL OF MECHANICAL AND AEROSPACE ENGINEERING

MAE 3780 : MECHATRONICS

“Cube Craze” Final Project Report

Authors:
Sparsh Gupta (sg2326)
Christopher Chan (cec272)
Emma Renner (ejr234)

December 11, 2018

Table of Contents

`

1. Introduction

The objective of the "Cube Craze" final project was to design, build, and program a fully autonomous robot capable of moving blocks off of your side of the arena and onto your opponent's. To win a match, a robot must have fewer blocks on their side of the playing field after 1 minute.

Despite these restrictions, the competition allowed for many innovative designs. Given the unpredictability of events during a match, we decided to prioritize reliability and consistency over complexity. Our robot design included a simple plow with a two-phase strategy that switched from purely offensive to purely defensive. This strategy worked well, allowing our robot to score two wins and two losses, with only one unexpected incident occurring during the matches.

2. Design Motivation

The mechanical and electrical design of the robot was devised by focusing on the following considerations:

I. Number of variables during competition

There were a large number of unpredictable variables during the competition, such as the opponent's strategy and robot design, the distribution of the blocks on the arena, and the starting position of our robot. There were also variables associated with the available components used for the robot. The 9V battery pack provided full power for only a short time, after which its actual power supply diminished. This meant that the speed of the wheel motors was variable and would likely change between matches.

In order to ensure a consistent strategy, it would be necessary to favor a more robust and simpler design that is able to perform its function despite the occurrence of unpredictable events.

II. Overall competition strategy

It was naturally necessary to consider the general winning strategy when designing the robot. We opted for a extreme but reliable strategy that was broken into two phases: a purely offensive '*Sweeping*' phase and a purely defensive '*Anchoring*' phase.

- **Sweeping:** This phase involved a quick zig-zag motion at the start to gather a large portion of the blocks in a plow within the first ten seconds of the match. Since this was purely offensive, it was vital to ensure that the robot stayed on its trajectory despite external impacts like contact with opponent's robot.
- **Anchoring:** This phase occurs after Sweeping, and involves the robot moving to the opponent's side and completely stopping. In this case too, the robot would have to be resistant to external contact.

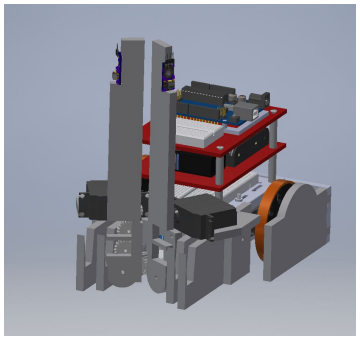
A more detailed analysis of this competition strategy is given in the **Competition Strategy and Relevant Code** section.

III. Aesthetics

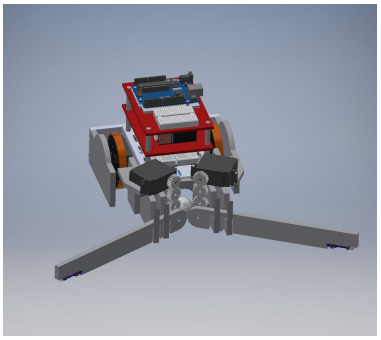
Given that this was in fact a fun competition at the end of the semester with a large audience, we decided to bring a little Christmas spirit with our robot!

Figures 1,2, & 3: CAD Model of the robot. The back QTI Sensors are not displayed in the Bottom view.

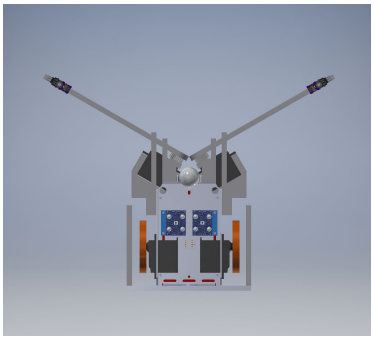
[1] Isometric view



[2] Front View



[3] Bottom View



2.1 Mechanical Design

As seen in Figs. 1, 2, and 3, our mechanical design consisted of two major elements: A front plow with 7" arms to pick up blocks, and back body armor to protect wheels and add weight. Both of these were created by laser cutting acrylic sheets.

To perform the Sweeping strategy effectively, we originally opted to maximise the span of the plow, filling up the entire 18" diameter space we were allowed using plow arms that were 9" long. However, this idea was unsuitable due to the additional weight that the thick acrylic material adds to the robot. While extra weight was actually beneficial to our design as it increased the inertia of our robot and made it harder to push off trajectory, the concerning factor was the moment created by the plow. Since the plow was attached only to the front body of the robot, its weight created a powerful moment tipping the robot forwards. This caused the plow to rub against the ground when moving, which not only slowed down the robot further, but also made the motion very unpredictable as the frictional force was inconsistent. To remedy this, we decided to use a plow with 7" arms. This reduced the moment substantially, allowing the plow to stay slightly elevated above the arena floor. The drawback to this was the smaller span, which meant fewer blocks would be picked up while Sweeping, however during testing the plow was still able to pick up over 50% of the blocks. The plow was folded upward at the start of the match to meet the 8" by 8" starting restriction, and was brought down by two servos connected to a gear train that would activate at the start of the Sweeping phase.

The back body armor served two crucial purposes. First, it protected the wheels and ensured that any contact doesn't dislodge the rubber tyres off the wheels (which happened repeatedly during testing). This meant protection from contact with opponent, and also protection from blocks getting stuck under the robot body. Second, the armor added additional weight to the back of the robot, counteracting the moment from the front plow while also increasing the robot's inertia. Both of these benefits were advantageous to our Sweeping and Anchoring strategies, as they eliminated potential mishaps during competition and made our robot very difficult to move.

2.2 Electrical Design

The robot collectively used 4 QTI sensors, 2 Color sensors, and 2 Wheel motor servos, arranged as seen in Figure 3. We opted to use a greater-than-average number of Color sensors as we needed to reorient the robot when it touched the line dividing the two sides of the arena.

Each QTI was placed in a corner of the robot. However, unlike the back two sensors, the front two were mounted on the ends of the plow instead of the front corners of the robot body, due to the plow's large span. During testing, it was observed that the plow would push its blocks beyond the border and off the arena before these sensors detected the border, if they were placed on the front corners of the body. Thus, mounting the QTIs on the plow was the only method to avoid loss of carried blocks. However, since the plow moved vertically at the start, it was crucial to ensure that the QTIs would be at the right elevation from the ground after the plow has dropped. If the QTIs were touching the ground or if they were too far up, they would detect black continuously and the border detection system would fail. The plow supports were accurately modelled and laser cut to ensure that the QTIs would stop at the right distance after the plow fell.

The two Color sensors were placed horizontally apart in the middle of the robot, which allowed the robot to orient itself perpendicular to the line dividing the two sides of the arena and begin the Anchoring phase. This was necessary to guarantee that the orientation of the robot would be known when it enters the opponent's side, to accurately perform the Anchoring sequence.

`

It was decided to use a larger 30 row breadboard rather than the standard 17 row breadboard attached with the arduino. This larger breadboard was taped on top of the standard one, and gave greater working space to wire in the 12 wires from the 2 Color sensors, 12 wires from the 4 QTI sensors, and the 4 wires from the H-bridges of the two servos. It also allowed us to use the side power lines to transmit a 5V output from the arduino and have a ground channel, which made wiring the components much easier and neater.

3. Competition Strategy

To maximize consistency, we opted to design the Sweeping and Anchoring phases in such a way that the initial starting point of the robot did not affect the program. Additionally, during testing it was noticed that the right motor servo spun faster than the left servo, which caused the robot to deviate to the left when moving forwards and backwards, while also changing its angle to the borders. To counter this effect, pulse-width modulation was used to power the right motor with a square wave of frequency of 25 kHz and a 90% duty cycle.
Please refer to Appendix 7.2 for the overall strategy flowchart for the robot, and is split to easily identity the Sweeping and Anchoring phases.

3.1 Sweeping Phase

The purely offensive strategy here relied on gathering at least 50% of the arena's blocks within the first 10 seconds of gameplay, before the opponent had the opportunity to interfere with our trajectory. The angles for the trajectory were calculated using the following criteria:

- The plow must have picked up at least 10 blocks by the end of the Sweeping phase.
- The number of 'sweeps' should be minimized to reduce chances of contact with opponent's robot.
- The number of turns should be minimized as turns take a significant amount of time.
- The robot should finish the Sweeping phase at the boundary between the two sides, and should finish roughly in the center of the arena.

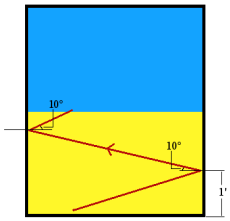


Figure 5: Visual representation of Sweeping trajectory

Based on these criteria, the following Sweeping strategy was designed, which is visually depicted in Fig.5:

1. Despite wherever the robot was placed at the start, orient the robot by hand such that it will hit the right border approximately one foot up from the bottom of the arena (by eye).
2. Quickly make the robot turn left and then right to drop the plow.
3. Upon contacting a border, reorient the robot by turning it until both the left or right QTI sensors detect the border.
4. Turn an additional 10° in the same direction, which makes total angle from the horizontal 10°.
5. Proceed moving straight forward and repeat until the other side's color is detected.

Contacting the right border about one foot above the arena bottom, and turning 10° to the horizontal upon every border contact created an optimal trajectory. This trajectory involved only one full sweep across the

`

board, two turns, and placed the robot close to the center of the arena at its end. Accounting for the span of the plow, this trajectory would also allow us to cover around 70% of the total area where blocks would be placed, thus comfortably ensuring that we pick up at least 50% of the blocks.

Due to the fact that this strategy still left the robot vulnerable to possible contact with the opponent, we opted to add a general code the constantly reoriented the robot whenever a border was hit, rather than hardwiring one sweep in the code. This way, even if the opponent nudged our robot off trajectory, the moment our robot detected a border it was again reorient itself 10° to the horizontal and be back on the trajectory.

3.2 Anchoring Phase

The idea behind this purely defensive strategy was to limit opponent's access to our collected blocks as much as possible. The method used to achieve this was:

1. Reorient robot using Color sensors so it is facing perpendicular to the dividing line.
2. Move the robot forward until the top border is hit (indicated by both front QTI sensors detecting the border).
3. Completely stop robot

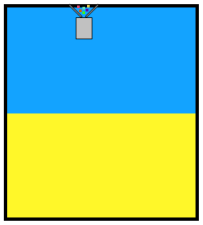


Figure 6: Final stopping representation of the robot.

By stopping the robot at the border (Fig. 6), the opponent would have no ability to access these blocks, other than by pushing our robot out of the way. Given the large weight of our robot, this was quite unlikely to occur, especially given that the opponent would have a high chance of detecting the border and turning away.

3.3 Pulse-Width Modulation for Right Wheel Servo

The right wheel servo (connected to pin PD6) was pulse width modulated with the use of TIMER0 and the two Compare Match Interrupts A and B. This allowed us to add PWM to any pin on the Arduino, giving us great flexibility with the wiring of pins.

Code Reference 1: Initializations for PMW on Pin PD6

```
void initPMW()
{
    sei();
    TCCR0A = 0b00000000; //Normal port operation, OC0A & OC0B disconnected
    TCCR0B = 0b00000011; //Set Prescalar to 64, don't Force Output Compare A
    TIMSK0 = 0b00000000; //TIMER0 Compare Match Interrupts disabled
    OCR0A = 9; //Square wave gives 5V output for 9 ticks
    OCR0B = 1; //Square wave gives 0V output for 1 tick
    TCNT0 = 0; //Reset TIMER0

}
```

`

A Prescaler of 64 was used as this was the quickest tickrate where the Arduino was able to consistently supply the servo with the required square wave (Code Ref. 1). Each square value lasted 10 ticks total, giving the square a frequency of (16MHz / (64 * 10)) = 25 kHz. Compare Match Interrupt A was used to turn the pin from high to low, while Compare Match Interrupt B was used to turn the pin from low to high (Code Ref. 2). By changing the values of the two compare integers OCR0A and OCR0B, the duty cycle of the square wave could be adjusted.
By setting OCR0A to 9 and OCR0B to 1, we allowed the square wave to stay at high power (5V) for 9 ticks and drop to 0V for 1 tick. This gave us a duty cycle of 90%, which in testing was ideal and allowed the robot to maintain a straight line motion when moving forward and backward.

Code Reference 2: Compare Match Interrupt Codes for TIMER0

```
ISR(TIMER0_COMPA_vect) //Occurs when TIMER0 = 9
{
    if(((PIND & 0b01000000)) && start == 1) //If Pin PD6 is High...
    {
        PORTD &= 0b10111111; //Set PD6 to Low and restart TIMER0
        TCNT0 = 0;
    }
}

ISR(TIMER0_COMPB_vect) //Occurs when TIMER0 = 1
{
    if((!(PIND & 0b01000000)) && start == 1) //If Pin PD6 is Low...
    {
        PORTD |= 0b01000000; //Set PD6 to High and restart TIMER0
        TCNT0 = 0;
    }
}
```

4. Analysis of Strengths and Weaknesses

The night before competition at 10pm, we realized something was wrong with our H-bridge and in the end we burned a TA, melted our breadboard, and had to rebuild our robot from scratch.
Ultimately, this summarizes the strength of our team and robot: adaptability and simplicity. We knew if we allowed our code to run for the full minute, something would ultimately go wrong and our robot would not act how we wanted it to. Another strength of our robot is the plow. The plow deploys as the match starts, coming down at an angle, forming a V. This allows the robot to capture blocks and move around with the blocks still within the plow. Finally, the weight of the plow allowed for sufficient anchoring when our robot was stopped, prohibiting it from being pushed around by other robots.

One weakness of our robot was the additional weight of the acrylic, although it anchored the robot, it also caused it to go slower than anticipated. Also, due to our lack of time to test the robot, we could not get the robot to orient itself perpendicular to back border once it reached the other color. This caused the robot to end at random angles, sometimes causing blocks to be pushed off the side.

5. Competition Performance

It was pleasing to see that our robot gave a good performance, winning 2 matches and losing 2 matches. In none of the matches was the robot able to avoid contact with the opponent, and as a result the robot was never able to follow the ideal Sweeping trajectory. Thus, programming a general self-correcting strategy was perhaps the successful idea that secured us two wins.

`

In our two winning games, contact with the opponent veered our robot off course, however this was actually advantageous to us as it gave the robot access to new trajectories on the arena, allowing the robot to pick up even more blocks than our original trajectory was designed for. In both the matches the robot successfully picked up 12-14 blocks and anchored itself on the opponent's side, winning us the match.

In our two losing games, we encountered a problem we had not anticipated during testing: contact during the transition from Sweeping to Anchoring. After aligning perpendicular to the dividing line, our robot was programmed to move forward until *both* front QTI sensors simultaneously detected the border. However, in both of these matches, the opponent happened to push our robot during this forward movement, turning it to an unexpectedly large angle. As our robot moved toward the top border with this angle, one of the front QTIs would detect the border first, followed by the other. However, the angle the was large enough that by the time the other QTI detected the border, the first QTI had already surpassed the border and no longer detected it. Thus at no point did both the front QTIs detect the border simultaneously, and the robot continued to move forward and push most of its collected blocks off the arena. This allowed the opponent to move the remaining (uncollected) blocks to our side and win the match.

This issue seen during the transition phase of our robot could have been fixed by programming the robot to stop when one of the front QTI sensors detected the border. This would have ensured that the plow and the robot remained within the border. However, if the robot hit the border at a large angle, it would leave an opening for the other robot to scoop in and take our blocks out of. But judging from the complexity of code that is required for this (not to mention anticipating such a case in the first place), it would have been unlikely that any opponents would have managed to steal blocks out of our plow, and thus this would have been a better program for the Anchoring strategy.

During the competition, we also noted two advantages to our design that were very important in deciding the outcome of the match. After Anchoring, our opponents attempted to push our robot much more than we expected. The additional weight of the acrylic was vital in maintaining our position - it prevented our robot from being push off the border and out of the arena. Also, using an Anchoring strategy which involved completely shutting down the robot about 15 seconds into the match was a great idea. Due to the huge number of unpredictable variables in this competition, robots that were programmed to function for the entire 60 seconds of the match often ended up failing, not executing their code properly, or driving off the edge due to unknown errors in the programming. Given that the equipment used for the competition is not robust enough to absorb the unpredictabilities, a strategy of quickly attacking and then completely stopping works best since it avoids the chance of the robot failing over long-term execution of its code.

6. Conclusion

At the end, our team was able to successfully achieve all of our goals. The design and programming features we implemented worked as expected and delivered a solid performance (see Appendix 7.2). Through this competition, we have learned just how different practice can be from theory. While in theory our planned trajectory should have worked most of the time, during the actual match our robot never ran the planned trajectory once due to unpredictable contact with opponents. It also showed us the important of rigorous testing - had we tested our robot against an actual opponent, we would have likely noticed the vulnerability of the robot during the transition from Sweeping to Anchoring. Nevertheless, our robot was able to bring a little christmas cheer into a fun competition, and we are proud of our robot's performance as a whole.

`

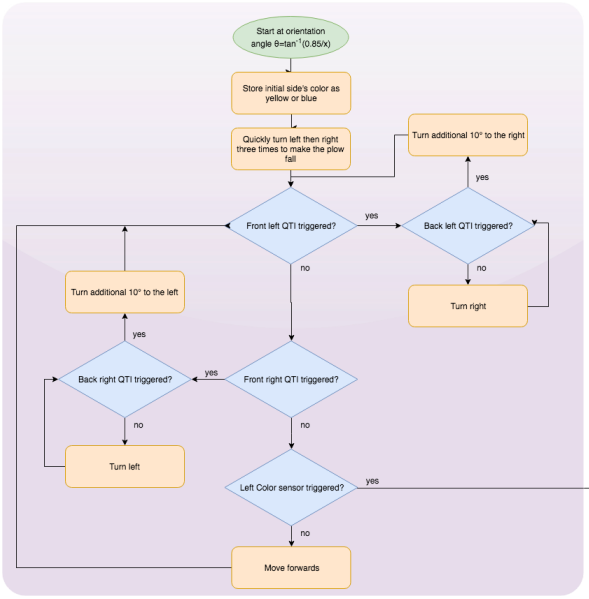
7. Appendix

7.1 Bill of Materials

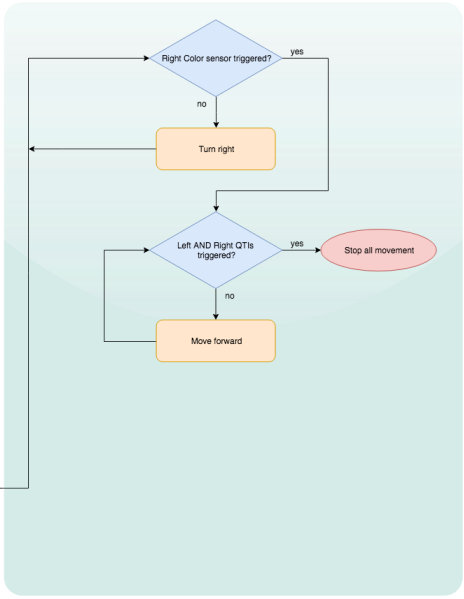
Order	Item	Quantity	Price	Total
1	QTI Sensor	1	$2.00	$2.00
2	Color Sensor	1	$5.00	$5.00
3	Servo Motor	2	$3.00	$6.00
4	12” X 12” Acrylic Sheet	2	$5.00	$10.00
5	Tree Ornaments	1	$3.00	$3.00
6	Garland	1	$3.20	$3.20
7	Wreath Ornaments	1	$3.00	$3.00
8	String Lights, 1m	1	$0.90	$0.90
				$33.10

7.2 Strategy Flowchart

Sweeping Phase

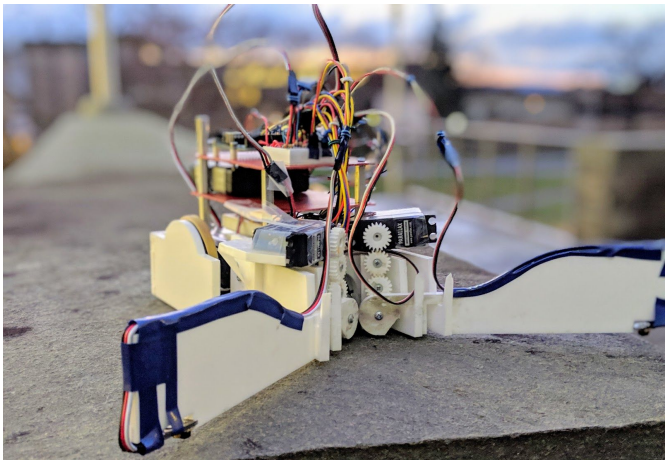


Anchoring Phase



`

7.3 Competition Day Robot



[A] Isometric View, Robot without decorations

[B] Front View, Robot with decorations

[C] The robot, photographed by ASML

[D] Robot successfully defeats opponent

`

7.4 Competition Code

```
/*
* Final Project Code.c
*
* Created: 11/8/2018 7:34:03 PM
* Author : CIT-Labs
*/

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

/*Color Sensors, Periods of waves:
Blue: 16kHz = 48 microsec
Yellow: 10kHz = 79 microsec
Black: 2.5kHz = 267 microsec
*/

/*Motor Control:
[1] PD7(L P0) & PD5(L P1): Right Wheel (Pins 7 & 5)
[2] PD6(R P0) & PD4(R P1): Left Wheel (Pins 6 & 4)
*/

/*QTIs:
[1] PB5: Top Left (Pin 13)
[2] PB4: Top Right (Pin 12)
[3] PB3: Bottom Left (Pin 11)
[4] PB2: Bottom Right (Pin 10)
*/

/*Color Sensors
[1] PB1: Top Right (Pin 9)
[2] PB0: Top Left (Pin 8)
*/

#define rightTime 690 //Movement constants
#define leftTime 750
#define delayTime 1700
#define footTime 3400

int periodct = 0; //Color Sensor variables
int period = 0;
int periodmain = 0;
int blueMin = 230; //Color Sensor compare values
int blueMax = 700;
int yellowMin = 5;
int yellowMax = 220;
int originalColor = 0; //0 - Yellow, 1 - Blue, 2 - Error
int start = 0; //Variable to assist with PWM

//////////////////////////////////////////////////////////
```

`

```
//////////////////////////////////////////////////////

//Color Sensor functions
ISR(PCINT0_vect)
{
    periodct=TCNT1;
    TCNT1=0;
}

ISR(PCINT1_vect)
{
    periodct=TCNT1;
    TCNT1=0;
}

void initColor()
{
    sei(); //enable global interrupts
    DDRB &= 0b11111110; //sets PB0, PB1 to input
    DDRB &= 0b11111101;
    PORTB &= 0b11111110;
    PORTB &= 0b11111101;

    PCICR |= 0b00000001; //enable control register for Port B
    PCMSK0 |= 0b00000001; //enable mask register for PB0
    PCMSK0 &= 0b11111101; //disable mask register for PB1

    TCCR1B=0b00000001; //sets prescaler for TIMER1 to 1
    TCNT1 = 0; //set TIMER1 to 0
}

void initSensor1() //enable left sensor & disable right sensor
{
    PCMSK0 |= 0b00000001; //enable mask register for PB0
    PCMSK1 &= 0b11111101; //disable mask register for PB1
    TCNT1 = 0; //set TIMER1 to 0
}

void initSensor2() //enable right sensor & disable left sensor
{
    PCMSK0 &= 0b11111110; //disable mask register for PB0
    PCMSK1 |= 0b00000010; //enable mask register for PB1
    TCNT1 = 0; //set TIMER1 to 0
}

int getColor()
{
    return periodct;
}

//////////////////////////////////////////////////////
```

`

```
////////////////////////////////////////////////////////

//Movement functions
void moveForward()
{
    PORTD |= 0b00010000;
    PORTD |= 0b00100000;
    PORTD &= 0b10111111;
    PORTD &= 0b01111111;
    TIMSK0 = 0b00000000;
    // Move forwards
}

void moveBackward()
{
    TIMSK0 = 0b00000110;
    PORTD &= 0b11101111;
    PORTD &= 0b11011111;
    PORTD |= 0b10000000;
    start = 1;
    // Move backwards
}

void turnRight()
{
    PORTD &= 0b11101111;
    PORTD |= 0b00100000;
    PORTD |= 0b01000000;
    PORTD &= 0b01111111;
    TIMSK0 = 0b00000000;
    //Turn right
}

void turnLeft()
{
    PORTD |= 0b00010000;
    PORTD &= 0b11011111;
    PORTD &= 0b10111111;
    TIMSK0 = 0b00000110; //Turn left
}

void wait()
{
    PORTD &= 0b11101111;
    PORTD &= 0b11011111;
    TIMSK0 = 0b00000000;
    PORTD &= 0b10111111;
    PORTD &= 0b01111111; //Stop
}
////////////////////////////////////////////////////////
```

`

```
//////////////////////////////////////////////////////

//PWM Compare Match Interrupts
ISR(TIMER0_COMPA_vect) //Occurs when TIMER0 = 9
{
    if(((PIND & 0b01000000)) && start == 1) //If Pin PD6 is High...
    {
        PORTD &= 0b10111111; //Set PD6 to Low and restart TIMER0
        TCNT0 = 0;
    }
}

ISR(TIMER0_COMPB_vect) //Occurs when TIMER0 = 1
{
    if((!(PIND & 0b01000000)) && start == 1) //If Pin PD6 is Low...
    {
        PORTD |= 0b01000000; //Set PD6 to High and restart TIMER0
        TCNT0 = 0;
    }
}
//////////////////////////////////////////////////////

//main function
int main(void)
{
    //Initializations

    DDRD |= 0b00010000;
    DDRD |= 0b00100000;
    DDRD |= 0b01000000;
    DDRD |= 0b10000000; //Initialize movement pins (PD4 - PD7)

    wait(); //Initialize movement pins
    initPWM(); //Initialize PWM for TIMER0

    DDRB &= 0b11111011;
    DDRB &= 0b11110111;
    DDRB &= 0b11101111;
    DDRB &= 0b11011111;
    PORTB &= 0b11111011;
    PORTB &= 0b11110111;
    PORTB &= 0b11101111;
    PORTB &= 0b11011111; //Initialize QTI pins (PB2 - PB5)

    initColor();
    _delay_ms(50);
    init_uart(); //Initialize Color Sensors (left enabled, right disabled)
    initSensor1();
    _delay_ms(100);

    moveForward();
    _delay_ms(1000);
    wait();

    //////////////////////////////////////////////////////
```

```
    //////////////////////////////////////////////////////

    //Get original color
    periodmain = getColor();

    if (((int)periodmain > blueMin) && ((int)periodmain < blueMax))
    {
        originalColor = 1;
    }
    else if (((int)periodmain > yellowMin) && ((int)periodmain < yellowMax))
    {
        originalColor = 0;
    }
    else
    {
        originalColor = 2;
    }
```

`

```
//Begin code

if (!(originalColor == 2)) //if the robot detects blue/yellow as original color, then...
{
    //Sweeping Phase
    initSensor1(); //Use left color sensor for sweeping
    _delay_ms(50);
    periodmain = getColor();

    if (originalColor == 0) //if original color is yellow
    {
    //Perform sweeping until opponent's color is detected
    while(!(((int)periodmain > blueMin) && ((int)periodmain < blueMax)))
    {
        //while neither QTIs nor color sensors detect anything, move forward
        while ((!(PINB & 0b00010000) && !(PINB & 0b00100000)) && (!(((int)periodmain > blueMin) && ((int)periodmain < blueMax))))
        {
            moveForward();
            periodmain = getColor();
        }

        //if Color Sensors detect...
        if (((int)periodmain > blueMin) && ((int)periodmain < blueMax))
        {
            initSensor2();
            _delay_ms(50);

            while (!(((int)periodmain > blueMin) && ((int)periodmain < blueMax)))
            {
                turnRight(); //turn right until both sensors detect opponent's color
            }
            while ((!(PINB & 0b00100000)) && (!(PINB & 0b00010000)))
            {
                moveForward();
            }
            wait();
        }
        else //otherwise if QTIs detect...
        {
            if ((PINB & 0b00100000) && (!(PINB & 0b00010000))) //top-left QTI detects
            {
                while (!(PINB & 0b00001000)) //turn right until bottom-left QTI detect
                {
                    turnRight();
                    _delay_ms(200);
                    moveForward();
                    _delay_ms(100);
                }
                wait();
            }
            else if ((!(PINB & 0b00100000)) && (PINB & 0b00010000)) //top-right QTI detects
            {
                while (!(PINB & 0b00000100)) //turn left until bottom-right QTI detect
                {
                   turnLeft();
                   _delay_ms(200);
                   moveForward();
                   _delay_ms(100);
                }
                wait();
            }
        }
    }
    }
```

`

```
        else //if original color is blue
        {
        while(!(((int)periodmain > yellowMin) && ((int)periodmain < yellowMax)))
        {
            //while neither QTIs nor color sensors detect anything, move forward
            while ((!(PINB & 0b00010000) && !(PINB & 0b00100000)) && (!(((int)periodmain > yellowMin) && ((int)periodmain < yellowMax))))
            {
                moveForward();
                periodmain = getColor();
            }

            //if Color sensors detect...
            if (((int)periodmain > yellowMin) && ((int)periodmain < yellowMax))
            {
                initSensor2();
                _delay_ms(50);

                while (!(((int)periodmain > yellowMin) && ((int)periodmain < yellowMax)))
                {
                    turnRight(); //turn right until both sensors detect opponent's color
                }
                while ((!(PINB & 0b00100000)) && (!(PINB & 0b00010000)))
                {
                    moveForward();
                }
                wait();
            }
            else //otherwise if QTIs detect...
            {
                if ((PINB & 0b00100000) && (!(PINB & 0b00010000))) //top-left QTI detects
                {
                    while (!(PINB & 0b00001000)) //turn right until bottom-left QTI detect
                    {
                        turnRight();
                        _delay_ms(200);
                        moveForward();
                        _delay_ms(100);
                    }
                    wait();
                }
                else if (!(PINB & 0b00100000) && (PINB & 0b00010000)) //top-right QTI detects
                {
                    while (!(PINB & 0b00000100)) //turn left until bottom-right QTI detect
                    {
                        turnLeft();
                        _delay_ms(200);
                        moveForward();
                        _delay_ms(100);
                    }
                    wait();
                }
            }
        }
        }

        wait(); //End of program
    }
    else //if robot doesn't detect blue/yellow as original color, do nothing
    {
        wait();
    }
}
```

`