Zumo Robot and Controller Final Project

By: Jonathan Gusler
EECE 4254: Microprocessors
20 April 2019

Table of Contents

Introduction

The goal of this project was to program a ReadyAVR-64 microcontroller to be a remote control for a Zumo Robot. The communication was serial using two HC-05 Bluetooth transceivers, one for the controller and one for the robot. The requirements are as follows:

- Robot should move left, right, forward, and backward at various rates of speed
- When moving backward, a backup alarm on the robot must sound
- Control the robot's motion using the joystick on the ReadyAVR microcontroller
- Controller must always display a visual indication of the robot's speed and direction
- The two Bluetooth transceivers must only recognize each other
- The "master" transceivers must be on the controller, and the "slave" on the robot

Equipment Used

- (1) ReadyAVR-64 microcontroller by 100MHz
- (1) RedBoard by Sparkfun
- (2) HC-05 Bluetooth Module by DSD-TECH
- (1) Zumo Robot for Arduino by Pololu
- (3) 10k Ohm resistors
- (1) RGB LED
- wires/connectors as needed

Hardware and Software Design

Most simply, the joystick on the controller tells the robot what to do. The robot can execute five different motion-related commands, four of which execute while the joystick is being held in a certain position, the fifth happens if the joystick is pushed in and released. The robot can rotate right and left, reverse, and move forward at slower than maximum speed, these are the four, known as "manual" mode operations. The fifth, which is the press and release, not hold, tells the robot to change mode. The non-manual mode is referred to as "cruise control" in the report and "fast" in the code, this mode is the robot moving forward at maximum speed. If pressed again while in this cruise control mode, the robot goes back to "manual" mode, and awaits the next manual mode command. The controller checks every 50 ms for the current joystick state and transmits that state once every 50 ms to the robot. The robot will execute each of these commands for 50 ms before checking the incoming command again. This prevents an overloading of command throughput, which was found to cause delays in execution of these commands anywhere from half a second to about two seconds.

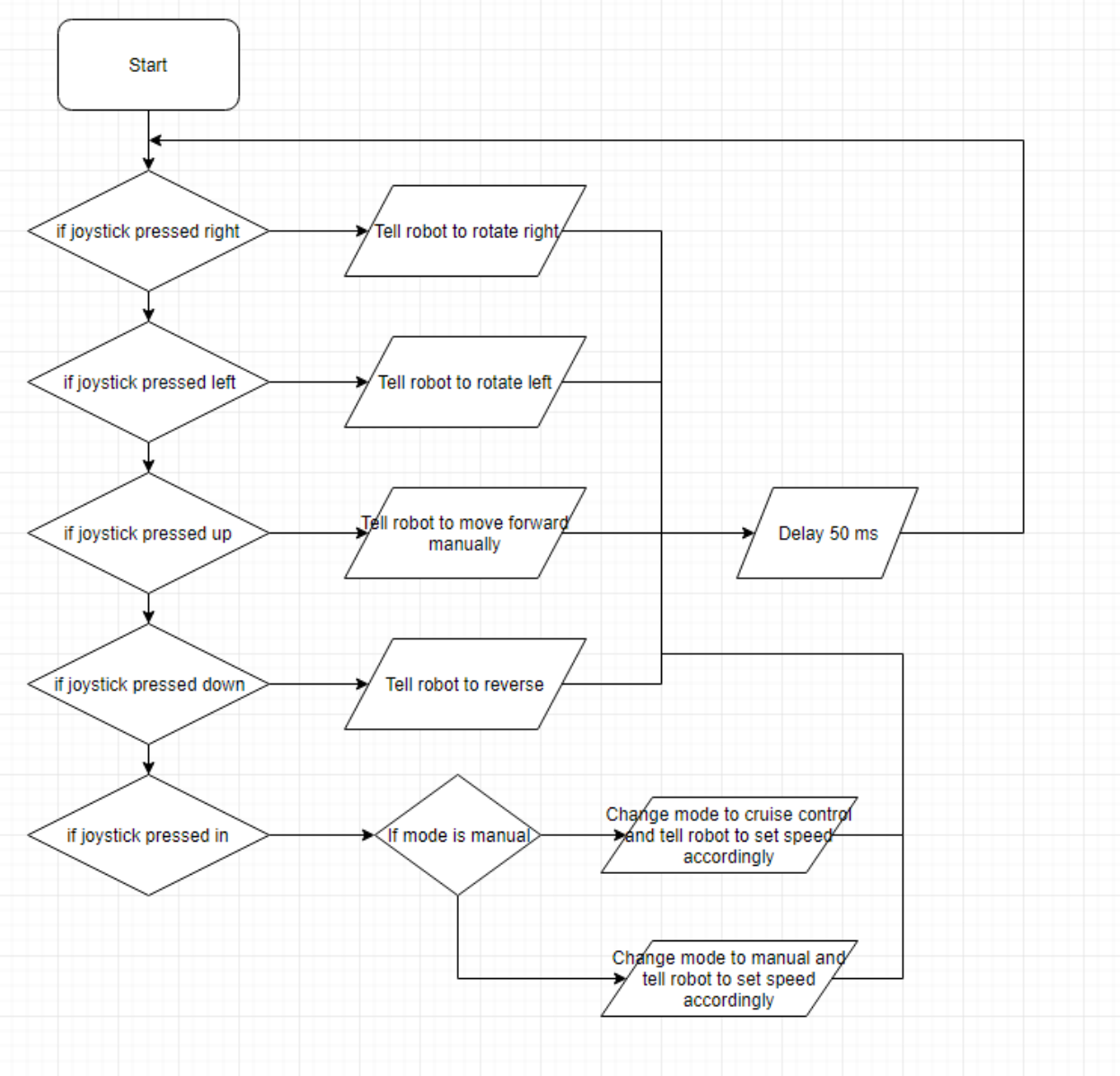

Figure 1: Diagram of simplified controller logic to control robot movement.

Figure 2: Diagram of robot logic to receive and execute commands.

In addition to the movement, the robot also sounds and alarm when reversing, and both the robot and controller indicate direction and mode via LEDs. On the robot, an RGB LED indicates mode and direction by changing color specific to the type of movement and flashing red when reversing. On the controller, two sets of four LEDs mimic each other and indicate the mode and direction of the robot, relative to the robot. If the mode is manual, LED 1 will light, along with a pair of LEDs indicating forward or reverse, or no additional LEDs if stationary. If the mode is cruise control, LED 0 will light. *Figure 3*, *Figure 4*, *Figure 5*, and *Figure 6* show the different configurations for the controller LED sets.

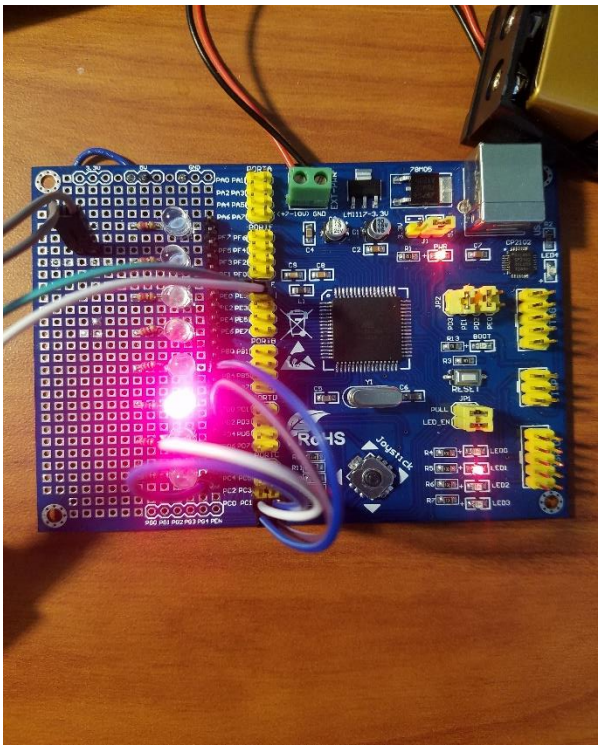

Figure 3: LED configuration for manual mode but not moving forward or backward.

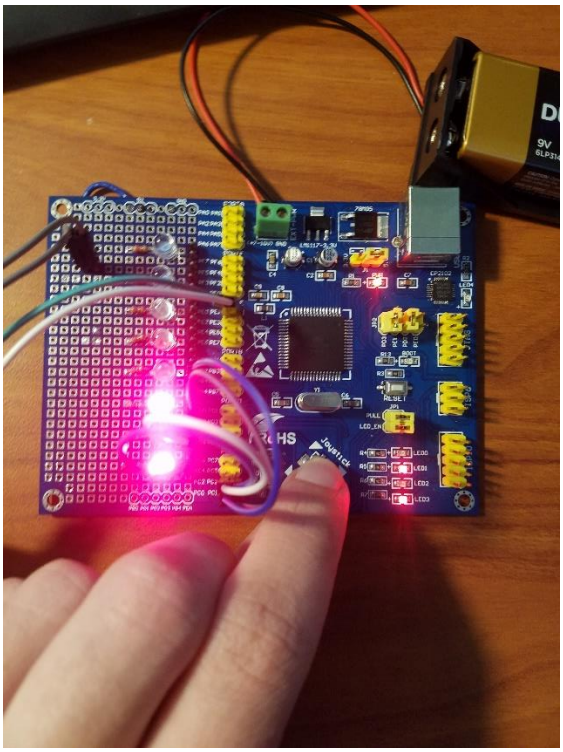

Figure 4: LED configuration for reversing.

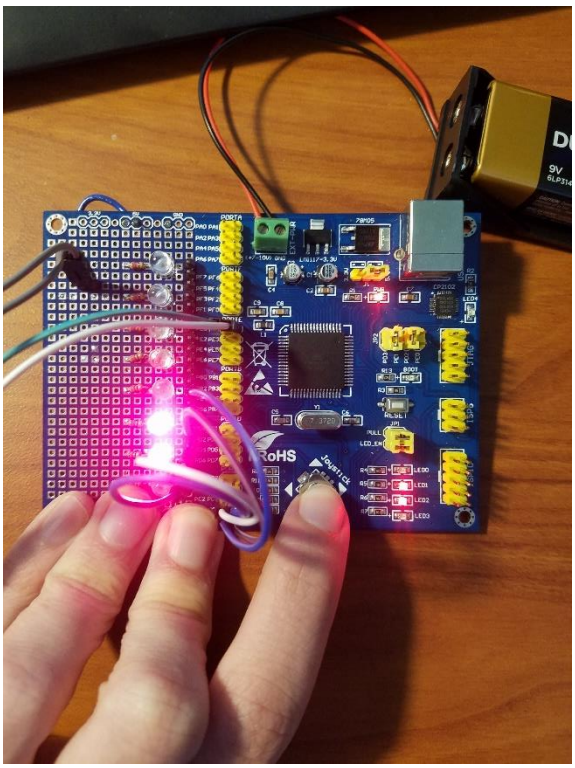

Figure 5: LED configuration for moving forward in manual mode.

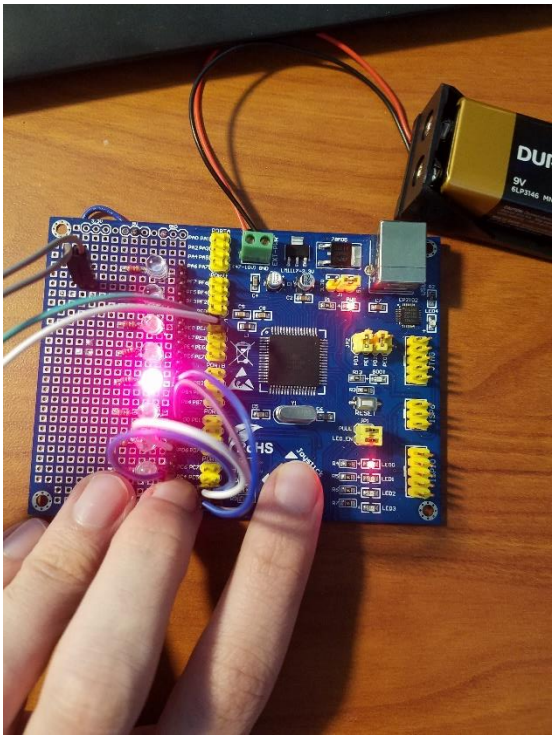

Figure 6: LED configuration for cruise control mode.

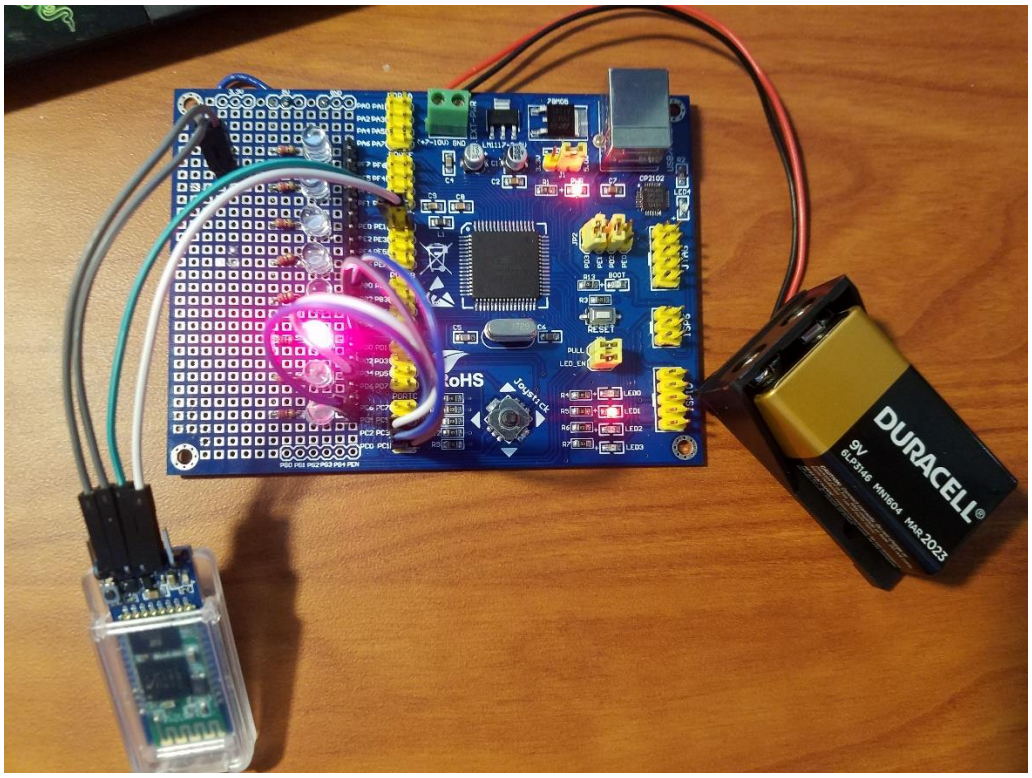

Figure 7: Image of controller.

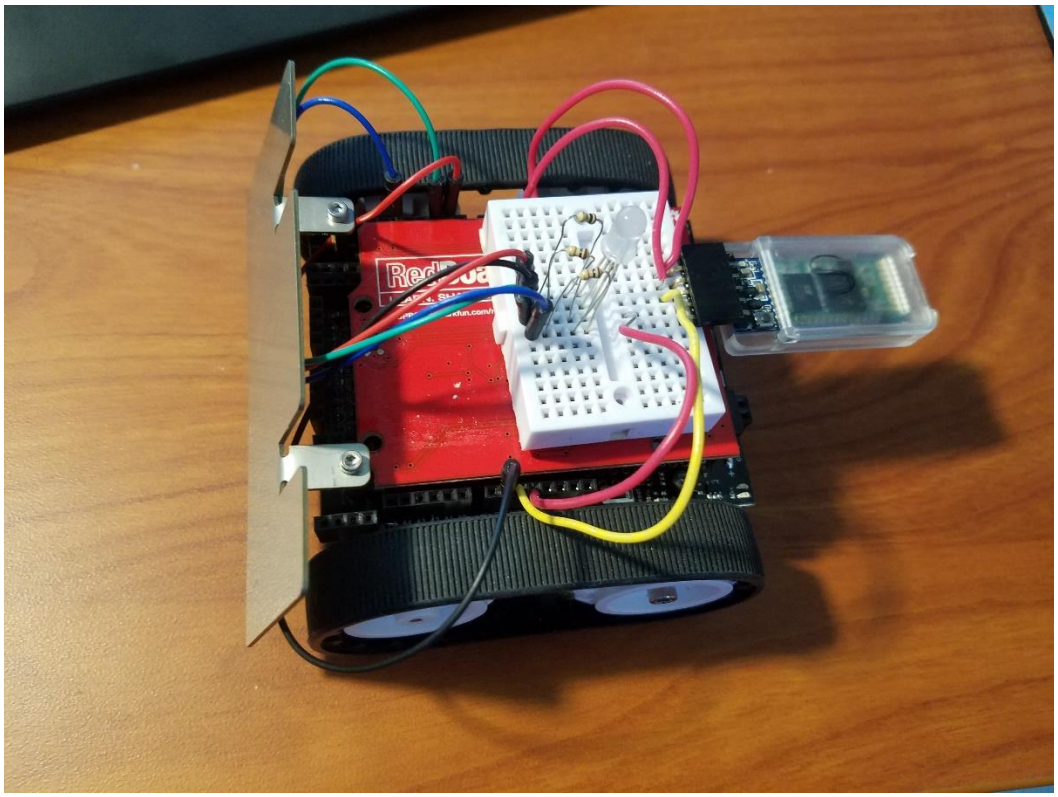

Figure 8: Image of Zumo Robot.

The Bluetooth transceivers are fed 5V power and are connected to defined Serial Receive and Transmit pins on the boards.

Summary

The final product did work to expectations and met the requirements plus more. Along with the requirements, an additional LED was attached to the robot to also indicate speed and direction. I found that a delay between instruction transmissions was needed to not overload the robot with instructions and thus cause large delays. Also, figuring out where to put the code to turn the controllers LEDs on and off took time, as one of the later iterations was turning them on and off so rapidly the human eye could barely detect they were on. Further development would involve figuring out how to optimize the code for the controller as currently it seems possibly more complicated than necessary. However, the robot works as intended so this is not a pressing issue. In addition to being controlled by the controller, there is a dance mode built into the robot that can be activated by commenting and uncommenting some code as described in the code.

Appendix A – Controller Code

```
/*****************************************************************************************
*
 * FinalProject.c
 *
 * Created: 4/12/2019 1:27:44 PM
 * Author: Jonathan Gusler
 *
 * Alright! Time for the final project in Microprocessors! This project is to control a Zumo
robot
 * using the ReadyAVR-64 micro-controller that we've been using all semester. This is done using
 * the joystick. Directions forward and back and speed mode are indicated by LEDs on the board.
 * The controller and robot communicate via HC-05 Bluetooth modules.

*****************************************************************************************
*/
// include file declarations go here
#define F_CPU 7372800
#include <avr/io.h>
#include <stdio.h>
#include <util/delay.h>
#include "UARTLibrary.h"


// Variable declarations
long unsigned int BaudRate = 9600; // Can be 1200, 2400, 4800, 9600, 19200, 38400, or 56700
unsigned char DataBits = 8;        // Can be 5, 6, 7, or 8
unsigned char StopBits = 1;        // Can be 1 or 2
unsigned char ParityType = 1;      // Can be 0 (none), 1 (odd), or 2 (even)
unsigned char character;

// Enumerate the possible states
enum mode

{
        Fast,   // Cruise control
        Manual,
};

// Create a variable to keep track of the program's states
enum mode Mode;

void modeChange(){

        switch(Mode)
        {
                case(Manual):
                Mode = Fast;
                // Set motor speed to low setting
                break;
                case(Fast):
                Mode = Manual;
                break;
        }
}

// Triggered by holding joystick up, switches mode to manual
void driveForward(){
        uart_tx(0x05);
        // Set mode to manual
        switch(Mode)
        {
                case(Manual):
                Mode = Manual;
                break;
                case(Fast):
```

```
                Mode = Manual;
                break;

        }
}

// Triggered by holding joystick down, switches mode to manual
void driveBackward(){
        // Set robot to reverse
        uart_tx(0x01);
        switch(Mode)
        {
                case(Manual):
                Mode = Manual;
                break;
                case(Fast):
                Mode = Manual;
                break;
        }
}

// Triggered by pressing left
void rotateLeft(){
        // Set robot to rotate left
        uart_tx(0x03);
}

// Triggered by pressing right
void rotateRight(){
        // Set robot to rotate right
        uart_tx(0x02);
}

int main(void)
{
        // Initialize the UART
        uart_init(BaudRate, DataBits, StopBits, ParityType);
        // Set PORTC as LED output, direction indicator
    DDRC = 0xFF;
        PORTC = 0xFF;

        // Set all Joystick buttons as input
        DDRB  = 0x00;

        // Set starting Mode
        Mode = Manual;

        volatile char changed = 0;
        volatile char direction = 0;
    while (1)
    {

                if(PINB == 0xFE){ // Joystick right pressed
                        rotateRight();
                        direction = 0x00;
                }
                if(PINB == 0xFD){ // Joystick down pressed
                        driveBackward();
                        direction = 0x01;
                        changed = 0;
                }
                if(PINB == 0xFB){ // Joystick left pressed
                        rotateLeft();
                        direction = 0x00;
                }
                if(PINB == 0xF7){ // Joystick up pressed
                        driveForward();
                        direction = 0x02;
                        changed = 0;
                }
                if(PINB == 0xFF){ // Nothing is pressed
```

```
                direction = 0x00;
            }


            if(PINB == 0xEF){ // Joystick in pressed
                if(!changed){ // Handles bounce issue by only executing once
                    modeChange();
                    changed = 1;
                }
            }

            else{

                uart_tx(0x00);
                changed = 0;
            }

            if(Mode == Fast){ // Mode indicator
                PORTC = 0xFE;
                uart_tx(0x04);
                direction = 0x00;
            }
            if(Mode == Manual){
                PORTC = 0xFD;
                if(direction == 0x01){
                    PORTC = 0xF5;  // If going backwards
                }
                if(direction == 0x02){
                    PORTC = 0xF9;  // If going forwards
                }

            }
            _delay_ms(50);
    }
        return 0;
}
```

Appendix B – Robot Code

```
#include <Wire.h>
#include <ZumoShield.h>
#include <SoftwareSerial.h>

int redPin = 6;
int greenPin = 5;
int bluePin = 2;

unsigned char incomingByte = 0;

ZumoMotors motors;
ZumoBuzzer buzzer;
SoftwareSerial mySerial(11, 12); // RX, TX

void setup()
{
  pinMode(redPin,OUTPUT);
  pinMode(greenPin,OUTPUT);
  pinMode(bluePin,OUTPUT);
  mySerial.begin(9600);

  while (!mySerial) {
    ; // wait for serial port to connect. Needed for native USB port
only
  }
}

void loop()
{

// Uncomment for random shenanigans and comment out the serial.read
// const unsigned char Directions[] = {0x00, 0x01, 0x02, 0x03, 0x04,
0x05};
//  incomingByte = Directions[random(0,6)];
//  delay(random(500, 200));



  //Comment out for random shenanigans
  incomingByte = mySerial.read();



  if (incomingByte == 0x00) {
    motors.setSpeeds(0,0);
    setColor(0,0,0);
  }
```

```
  while (incomingByte == 0x05) { // Drive forward from joystick
command
    motors.setSpeeds(200,200);
    setColor(0, 0, 255);         // Blue
    delay(50);
    break;
  }

  while (incomingByte == 0x01) { // Drive backward from joystick
command
    motors.setSpeeds(-200,-200);
    buzzer.playNote(NOTE_A(4), 75, 15);
    setColor(255,0,0);          // Red
    delay(25);
    setColor(0,0,0);
    delay(25);

    break;
  }

  while (incomingByte == 0x02) { // Rotate right from joystick command
    motors.setSpeeds(150,-150);
    setColor(255, 0, 255);       // Purple
    delay(50);
    break;
  }

  while (incomingByte == 0x03) { // Rotate left from joystick command
    motors.setSpeeds(-150,150);
    setColor(255, 255, 255);     // White
    delay(50);
    break;
  }

  while (incomingByte == 0x04) { // Drive forward at cruise control
speed
    motors.setSpeeds(400,400);
    setColor(0, 255, 0);         // Green
    delay(50);
    break;
  }
}

void setColor(int redValue, int greenValue, int blueValue) {
  analogWrite(redPin, redValue);
  analogWrite(greenPin, greenValue);
  analogWrite(bluePin, blueValue);
}
```