PORTLAND STATE UNIVERSITY

iSOBOT + Jimmy

Edward Sayers
March 20, 2016

1 INTRODUCTION

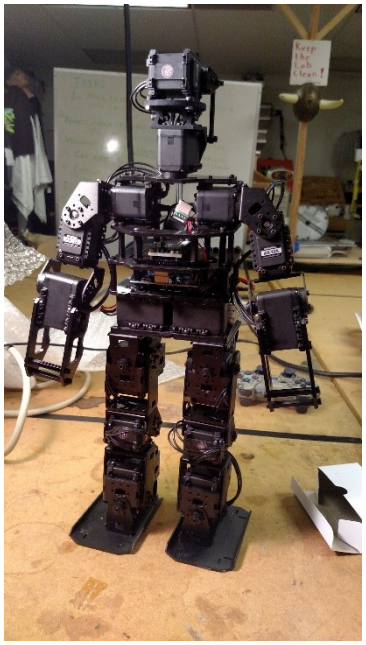
The goal of this project was to develop robots capable of performing in the Robot Theater at Portland State University. My team had two members, Dan Petre and myself. We were given two iSOBOT robots, two KHR-1 robots, and a HR-OS1 (Jimmy) robot to use for this task. In this report I will focus on my contribution to the Jimmy and iSOBOT development. I did not have any involvement with the KHR-1 robots.

2 HR-OS1

We received the unassembled HR-OS1 kit in the third week of class. The kit contains two options for the main CPU, 20 servos, an Arbotix PRO Robocontroller, and all of the hardware needed for assembly. We assembled it together over the course of about ten hours using the instructions provided on the Tossen Robotics web site (http://www.trossenrobotics.com/HR-OS1). The instructions were fairly clear and contained pictures of each step. The kit comes with plenty of spare hardware, so the occasional lost screw was not a problem. The completed robot is shown in Figure 2.1.

The kit came with the option of using either a Raspberry Pi or an Intel Edison as the main CPU. We chose to use the Intel Edison because of its size, built in WiFi and Bluetooth, and lower power consumption. We also chose it because there was not a suitable WiFi connection available to set up the Raspberry Pi at the time. The robot is designed in such a way that changing between the Raspberry Pi and the Intel Edison would not take much work, although the software would have to be set up separately on each platform.

Figure 2.1: Completed Jimmy Robot



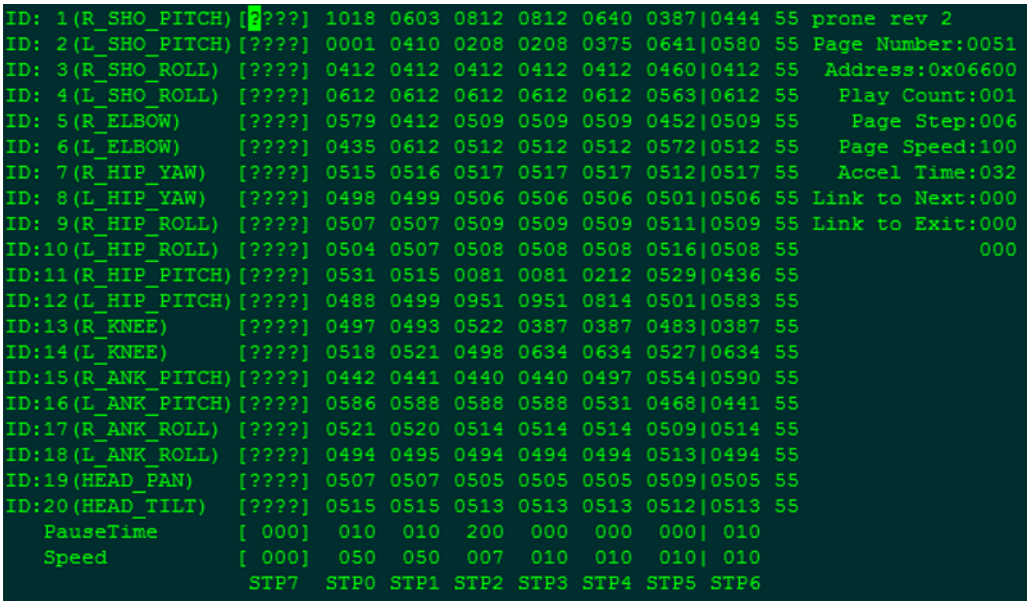
The framework used to control the robot is open source and is available on Github at https://github.com/Interbotix/HROS1-Framework. In addition to controlling the robot, the framework provides several utilities to make customization of the robot easier. It has a robot motion editor (RME), a walking tuner, and a PS3 demo.

The robot motion editor is used to create custom animations. It provides 255 motion pages that can each contain up to seven frames. Each frame defines the position of each servo and the speed at which the individual movements will be made. Each page can be played back by itself or chained to make more complex movements. The RME uses the text-based interface shown in Figure 2.2. Full instructions on the use of the robot motion editor can be found at https://github.com/Interbotix/HROS1-Framework/wiki/rme.

The walking tuner allows for editing various walking parameters such as step size and speed. This can be useful to customize the walking motion to accommodate different surfaces. The full instructions for using the walking tuner can be found at https://github.com/Interbotix/HROS1-Framework/wiki/walk_tuner.

The PS3 demo program allows for remote control of the jimmy using a PS3 Sixaxis controller. The controller can be used to make Jimmy walk or to play predefined motions. There are ten buttons on the controller that can be linked to pages in the RME by editing the corresponding file in the *action_scripts* folder. Each file in the folder has two parameters. The first parameter is the page number to play and the second parameter is a sound file that will be played if the Jimmy has an audio output set up. Full instructions for the PS3 demo can be found at https://github.com/Interbotix/HROS1-Framework/wiki/ps3_demo.

Figure 2.2: RME interface



I created a service for the PS3 demo so that it could be turned on by default when the robot powers on. This way the robot can be controlled by the PS3 controller without first having to log into the machine to enable it. I created the service by adding the following file to the /lib/systemd/system folder. The service can be enabled or disabled using systemctl.

```
[Unit]
Description=PS3 Demo
After=bluetooth.target

[Service]
Type=simple
RemainAfterExit=true
ExecStart=/home/root/HROS1-Framework/Linux/project/ps3_demo/↩
    start_edison_bluez5
Environment="HOME=/home/root"
WorkingDirectory=/home/root/HROS1-Framework/Linux/project/ps3_demo

[Install]
WantedBy=multi-user.target
```

3 iSOBOT

The iSOBOTs are small, remote controlled, toy robots that can perform over 200 predefined moves. They are controlled with an infrared controller. The controller supports two channels (A & B), so two iSOBOTS can be sent different commands. Previous students designed an Arduino controlled transmitter that could send commands to the iSOBOTS. The software was designed to control only one of the robots at a time, so two transmitters were needed to control both robots. We made several improvements to the software to make it so only one transmitter would be needed and to make future integration with a Kinect or RealSense camera easier. A new IR shield for the Arduino was also procured to increase the range in which the robots can be controlled.

Figure 3.1: iSOBOT

The software for the iSOBOTs consists of two parts. The first part runs on the Arduino and it receives commands via a serial over USB connection. Each command consists of four bytes. The first byte defines which of the iSOBOTS the command is to be sent to. If the value of the byte is 0x00, the command is sent to channel A. If the first byte is 0x01, the command

is sent to channel B. The second byte determines the command code to be sent. A list of command codes is included in the code listing in section 4.2. The third byte determines if the command should be repeated. Certain commands—the walk command for instance—require the command to be continually transmitted to work. By having the option to repeat commands, the serial command only needs to be sent once. If the third byte is 0x01, the command will be repeated until told to stop, or told to send a different command. Repetition can be stopped by sending the command code 0x00 to the appropriate channel. The fourth byte is a checksum and is just the XOR of the previous three bytes. If the Arduino receives a valid command, it then calculates the appropriate signal to send to the iSOBOT and sends it. Details on how the signal is constructed can be found in Mathias Sunardi's report which can be found at http://web.cecs.pdx.edu/~mperkows/CLASS_479/Projects-2012/Mathias_Sunardi-iSobot_controller_report.pdf.

The second part of the software runs on a separate computer and sends commands to the Arduino. This section of the code was completely re-written in C#. C# was chosen because it is compatible with both the RealSense and Kinect APIs. The portion of the code that sends commands to the Arduino was written as an object (*Isobot.cs*) so that it can be inserted into other programs easily. The object has the following interface:

Constructors

Isobot()	Creates object without properties set
Isobot(int baudRate, string portName)	Creates object and opens serial on portName at baudRate

Properties

BaudRate	Get or set baud rate
PortName	Get or set port name (COM#)

Methods

SerialConnect()	Connect via serial using BaudRate and PortName properties
SendCommand(Commands cmd, Channel chan, bool repeat)	Send command specified by cmd to channel chan, repeat determines if it the repeat byte is set

Commands and Channel are enums that define valid values for those parameters.

The main program (*Program.cs*) reads commands from a file and sends them to the Arduino. Each line of the command file contains four pieces of information separated by commas and with no spaces. The first piece of information is the command to be sent. This must correspond to a valid name in the Commands enum found at the end of *Isobot.cs*. The second piece of information must be either "A" or "B" corresponding to the channel to which the command should be sent. The third piece of information sets the repeat byte. It must be either "0" or "1". The last piece of information is a delay value in milliseconds. If this is set to something other than zero, the program will pause for that many milliseconds before sending the next command.

As an example, the following file will cause both robots to play the "GreetHuman" animation,

followed by the "GoOut" animation, followed by the "Dance" animation and there will be a four second pause between starting each animation.

```
GreetHuman,A,0,0
GreetHuman,B,0,4000
GoOut,A,0,0
GoOut,B,0,4000
Dance,A,0,0
Dance,B,0,4000
```

This makes it easy to play and design long sequences of animations without having to hard-code them into the program.

4 iSOBOT Code

The most recent version of the following code can be found at:
https://github.com/esayers/iSOBOT

4.1 Program.cs

```
using System.Threading;
using System.IO;
using System;

namespace iSOBOT
{
    class Program
    {
        static void Main(string[] args)
        {
            // Change COM port as needed
            Isobot bot = new Isobot(38400, "COM3");

            // Open command file for reading
            StreamReader reader = File.OpenText("commands.txt");
            string line;

            while ((line = reader.ReadLine()) != null)
            {
                Commands cmd;
                Channel chan;
                bool repeat;
                int delay;

                string[] sections = line.Split(',');

                // Check for length
                if (sections.Length != 4)
                    continue;

                // Get command
                if (!Enum.TryParse(sections[0], out cmd))
                    continue;

                // Get Channel
                if (!Enum.TryParse(sections[1], out chan))
                    continue;

                // Get repeat
                if (sections[2] == "0")
```

```
                repeat = false;
            else if (sections[2] == "1")
                repeat = true;
            else
                continue;

            // Get delay
            if (!int.TryParse(sections[3], out delay))
                continue;

            bot.SendCommand(cmd, chan, repeat);
            Thread.Sleep(delay);

        }

        bot.SendCommand(Commands.StopRepeating, Channel.A, false);
        bot.SendCommand(Commands.StopRepeating, Channel.B, false);

    }

}
```

4.2 ISOBOT.CS

```
using System;
using System.IO.Ports;

namespace iSOBOT
{
    public class Isobot
    {
        public int BaudRate { get; set; } = 0;
        public string PortName { get; set; } = "None";

        private SerialPort port;
        private byte[] sendBuffer;

        //Default Constructor
        public Isobot()
        {
            sendBuffer = new byte[4];
        }

        //Constructor to open serial connection
        public Isobot(int baudRate, string portName)
        {
```

```
    sendBuffer = new byte[4];
    BaudRate = baudRate;
    PortName = portName;
    SerialConnect();
}

// Destructor
~Isobot()
{
    port.Close();
}

// Connect to transmitter via serial port
public void SerialConnect()
{
    if (BaudRate != 0 && PortName != "None")
    {
        port = new SerialPort(PortName, BaudRate);
        port.DataReceived += new SerialDataReceivedEventHandler(↩
            DataReceivedHandler);
        try
        {
            port.Open();
        }
        catch
        { //todo
        }

        if (port.IsOpen)
        {
            Console.WriteLine(PortName + " is open");
        }
        else
        {
            Console.WriteLine("Unable to open " + PortName);
        }
    }
    else
    {
        Console.WriteLine("Port name or baud rate not set prior to ↩
            opening connection");
    }
}


public void SendCommand(Commands cmd, Channel chan, bool repeat)
{
    sendBuffer[0] = (byte) chan;
    sendBuffer[1] = (byte) cmd;
```

```
            sendBuffer[2] = (byte) (repeat ? 0x1 : 0x0);
            sendBuffer[3] = GetChecksum(sendBuffer[0], sendBuffer[1], ↩
                sendBuffer[2]);
            port.Write(sendBuffer, 0, 4);
        }

        static byte GetChecksum(byte b1, byte b2, byte b3)
        {
            return (byte)(b1 ^ b2 ^ b3);
        }

        private static void DataReceivedHandler(object sender, ↩
            SerialDataReceivedEventArgs e)
        {
            SerialPort sp = (SerialPort)sender;
            string indata = sp.ReadExisting();
            Console.Write("Data Received:");
            Console.WriteLine(indata);
        }
    }

    // List of valid commands
    public enum Commands : byte
    {
        StopRepeating = 0x00,       // Stop repeating the current
        LeftPunch = 0x13,           // Left Punch
        RightPunch = 0x14,          // Right Punch
        LeftWhack = 0x15,           // Left side whack
        RightWHack = 0x16,          // Right side whack
        LRPunch = 0x17,             // Left + Right punch
        RLPunch = 0x18,             // Right + Left punch
        LeftDownChop = 0x19,        // Left up-down chop
        RightDownChop = 0x1a,       // Right up-down chop
        UpDownChop = 0x1b,          // Both hands up-down chop
        DownUpChop = 0x1c,          // Both hands down-up chop
        PunchChopWhack = 0x1d,      // Right + left punch, Both hands up-down↩
            chop, Both hands whack
        LeftUpDown = 0x1e,          // Look left, up-down chop
        RightUpDown = 0x1f,         // Look right, up-down chop
        SnapOutOfIt = 0x20,         // "c'mon, snap out of it", slap
        BothWhack = 0x21,           // Both hands whack
        LeftWideKick = 0x22,        // Left wide kick
        RigthWideKick = 0x23,       // Right wide kick
        LeftKick = 0x24,            // Left kick
        RightKick = 0x25,           // Right kick
        LeftSideKick = 0x26,        // Left side kick
        RightSideKick = 0x27,       // Right side kick
        LeftBackKick = 0x28,        // Left back kick
        RightBackKick = 0x29,       // Right back kick
```

```
RightSideHKick = 0x2a,      // Right side high kick
RightLKick = 0x2b,          // Right side low kick
LeftRightHKick = 0x2c,      // Left + Right high kick
RightLeftLKick = 0x2d,      // Right + Left low kick
ComboKick = 0x2e,           // Combo low-left, right-high, left kick
LeftKick2 = 0x2f,           // Another left kick
RightKick2 = 0x30,          // Another right kick
Split = 0x31,               // Split
Block = 0x32,               // Block, "Whoa buddy"
RightBlock = 0x33,          // Right arm block
BothBlock = 0x34,           // Both arms block
DodgeRight = 0x36,          // Dodge right
DodgeLeft = 0x37,           // Dodge left
Headbutt = 0x38,            // Headbutt
RightToFace = 0x39,         // Right arm to face
Taunt = 0x3a,               // Taunt
HitDown = 0x3b,             // Hit and down
MultiBlock = 0x3c,          // Dodge right, left, block left, head, ↩
    fall down
Roger = 0x3f,               // "Roger!" raise arm
Weird = 0x40,               // Weird gesture
AllYourBase = 0x41,         // "All your base are belong to isobot"
AbsolutelyNot = 0x42,       // "Absolutely not!", flaps both arms
BowCrouch = 0x43,           // Bow/Crouch, get back up
GoodMorning = 0x44,         // "Good morning!", raise both arms, ↩
    stand on left foot
IComeInPeace = 0x45,        // "Greetings, I come in peace", wave ↩
    right arm
YallComeBack = 0x46,        // "Y'all come back now, ya hear!"
Wassup = 0x47,              // "Wassup!?", opens both arms sideways
GreetHuman = 0x48,          // "Greetings human", raise left arm and ↩
    bow
Honor = 0x49,               // "It's an honor to meet you!", bow and ↩
    shake right hand
ByeBye = 0x4a,              // "Bye bye"
BonVoyage = 0x4b,           // "Bon voyage!"
IllBeHereAllWeek = 0x4c,    // *clap* *clap* "Thanks! I'll be here ↩
    all week", raise right arm
ThatsAll = 0x4d,            // "T-t-that's all robots!", raise left ↩
    arm, stand on left foot
DomoArigato = 0x4e,         // "Domo arigato from isobot-o"
BearHug = 0x54,             // Walk forward, "Give me a bear hug"
WoeIsMe = 0x5d,             // "Woe is me ... what to do .. what to ↩
    do", bow, shake head
NotAgain = 0x5e,            // "No, no ... not again ... no, no"
CantBelieve = 0x5f,         // "Oh, I can't believe I did that"
Mercy = 0x60,               // "I throw myself into your mercy"
Dagger = 0x61,              // "Oh, like a dagger through my heart"
OuchSilent = 0x62,          // Motions from "Ouch" but without sound
```

```
        Ouch = 0x63,                // "Ouch, that hurts!"
        Wahoo = 0x65,               // point left, "Wahoo"
        Hooah = 0x66,               // "Hooah"
        Kapwing = 0x67,             // point left, "Kapwing"
        IzNice = 0x6b,              // "Iz nice, you like?"
        BothWave = 0x6c,            // Both arms wave left, right, left
        Drunk = 0x6d,               // Drunk
        MakeItStop = 0x6e,          // "No please, make it stop. Please, I ↩
            can't take it anymore. No, no ...", lie down and get up
        Yippe = 0x6f,               // "Yippe Yippe" x3, goal post arms
        HoHoHo = 0x70,              // "Ho, Ho, Ho"
        Yeehaw = 0x71,              // "Yeehaw" both arms wave left, right
        Amazing = 0x73,             // Stand on one foot, goal post arms, "↩
            Wow, that's amazing"
        Bow = 0x74,                 // Bow, arms over head and down
        CrossLegs = 0x79,           // Sit cross legged
        Comfortable = 0x7d,         // "Ahh, let me get comfortable. I'm too ↩
            sexy for my servos", lie down, flip over, get up
        BalancingAct = 0x80,        // Balancing act
        Pushup = 0x81,              // Pushups
        CountOnMe = 0x83,           // "You can count on me"
        Headstand = 0x88,           // Headstand
        ForwardFlip = 0x8a,         // Flip forward, back, forward x3
        Banzai = 0x8f,              // "Banzai" x3
        Chicken = 0x95,             // Chicken
        Dance = 0x97,               // Dancing (long)
        GiantRobot = 0x98,          // Giant robot motion
        GoOut = 0xA7,               // "Ready to go out dancing"
        WalkForward = 0xb7,         // Walk forward
        WalkBack = 0xb8,            // Walk Backwards
        WalkForwardLeft = 0xb9,     // Walk forward to the left
        WalkForwardRight = 0xb1,    // Walk forward to the right
        WalkLeft = 0xbb,            // Walk left
        WalkRight = 0xbc,           // Walk right
        WalkBackLeft = 0xbd,        // Walk Backward to the left
        WalkBackRight = 0xbe,       // Walk Backward to the right
        GetupBelly = 0xE4,          // Get up from belly
        GetupBack = 0xE5,           // Get up from back
    };

    public enum Channel : byte
    {
        A = 0x00,
        B = 0x01
    };
}
```

4.3 iSOBOT_IR.ino

```
// Serial protocol definitions
// FIRST BYTE: 0 - Channel A; 1 - Channel B
// SECOND BYTE: Command to be executed
// THIRD BYTE: 0 - Do not repeat; 1 - Repeat
// FOURTH BYTE: Checksum - XOR of other bytes
#define SERIAL_BUF_LEN 4
#define CHANNEL_BYTE 0
#define COMMAND_BYTE 1
#define REPEAT_BYTE 2
#define CHECKSUM_BYTE 3

// Command offsets
#define CHANNEL_OFFSET 21
#define CHECKSUM_OFFSET 16
#define COMMAND_OFFSET 8

// ------------------ - info about bits-------------------------
#define totallength 22 //number of highs/bits 4 channel + 18 command
#define channelstart 0
#define commandstart 4 //bit where command starts
#define channellength 4
#define commandlength 18
//---------determined empirically-------------
#define headerlower 2300 //lower limit
#define headernom 2550 //nominal
#define headerupper 2800 //upper limit
#define zerolower 300
#define zeronom 500 //380 //nominal
#define zeroupper 650
#define onelower 800
#define onenom 1000//850 //nominal
#define oneupper 1100
#define highnom 630

//---------------------pin assignments---------------
#define TXpin 7
#define RXpin 2 //doesnt use interrupts so can be anything

//---------------------variables----------------------
#define countin 1048576
boolean bit2[totallength];
unsigned long buttonnum;
char msg = ' ';
unsigned long x = 0;
unsigned long count = countin;
unsigned long buf = 0;
```

```
typedef struct serialCommand {    // Structure to store the state of a ↩
    channel
  unsigned long command;      // Command to be sent
  boolean valid;          // Is the current data valid?
  boolean repeat;         // Should the command be repeated?
} serialCommand;

byte serialBuf[SERIAL_BUF_LEN];   // Buffer for serial input
serialCommand isobotState[2];   // Array of state structures, one for each ↩
    channel

// Initial setup of device
void setup()
{
  Serial.begin(38400);
  pinMode(RXpin, INPUT);
  pinMode(TXpin, OUTPUT);
  isobotState[0].valid = false;
  isobotState[1].valid = false;
}


void loop()
{
  for (int i = 0; i < 2; ++i)
  {
    if (isobotState[i].valid)
    {
      buttonwrite(TXpin, isobotState[i].command);

      if (isobotState[i].repeat == false)
      {
        isobotState[i].valid = false;
      }
    }
  }
}

// Handler for incoming serial data
void serialEvent()
{
  while (Serial.available())
  {
    Serial.readBytes(serialBuf, SERIAL_BUF_LEN);

    // Check for valid checksum
    if (serialBuf[CHANNEL_BYTE] ^ serialBuf[COMMAND_BYTE] ^ serialBuf[↩
        REPEAT_BYTE] ^ serialBuf[CHECKSUM_BYTE])
```

```
      continue;

    // Check for valid channel
    unsigned long channel = serialBuf[CHANNEL_BYTE];  // Select channel
    if (channel > 1 || channel < 0)
      continue;

    unsigned long command = serialBuf[COMMAND_BYTE];  // Command word
    boolean repeat = serialBuf[REPEAT_BYTE] ? true : false;     // Should the↩
        command be repeated?

    // Check for stop repeating command
    if (command == 0x00)
    {
      isobotState[channel].repeat = false;
      isobotState[channel].valid = false;
      continue;
    }


    // Build command word without checksum
    command = command << COMMAND_OFFSET;
    command |= (channel << CHANNEL_OFFSET);
    command |= 0x80003;

    // Calculate checksum
    unsigned long checksum1 = ((command >> 16) & 0xFF) + ((command >> 8) & 0↩
        xFF) + (command & 0xFF);
    unsigned long checksum2 = 0;
    for (int i = 0; i < 3; ++i)
    {
      checksum2 += checksum1 & 0x7;
      checksum1 = checksum1 >> 3;
    }

    // Add checksum to command word
    isobotState[channel].command = command + ((checksum2 & 0x7) << ↩
        CHECKSUM_OFFSET);

    // Setup for next execution of main loop
    isobotState[channel].repeat = repeat;
    isobotState[channel].valid = true;
  }
}

void ItoB(unsigned long integer, int length)
{
    //needs bit2[length]
    Serial.println("ItoB");
```

```
    for (int i = 0; i < length; i++)
    {
        if ((integer / power2(length - 1 - i)) == 1)
        {
            integer -= power2(length - 1 - i);
            bit2[i] = 1;
        }
        else bit2[i] = 0;
        Serial.print(bit2[i]);
    }
    Serial.println();
}

unsigned long power2(int power)
{ //gives 2 to the (power)
    unsigned long integer = 1; //apparently both bitshifting and pow ↩
        functions had problems
    for (int i = 0; i < power; i++)
    { //so I made my own
        integer *= 2;
    }
    return integer;
}

void buttonwrite(int txpin, unsigned long integer)
{
    //must be full integer (channel + command)
    ItoB(integer, 22); //must have bit2[22] to hold values
    oscWrite(txpin, headernom);
    for (int i = 0; i < totallength; i++)
    {
        if (bit2[i] == 0) delayMicroseconds(zeronom);
        else delayMicroseconds(onenom);
        oscWrite(txpin, highnom);
    }
    delay(205);
}

void oscWrite(int pin, int time)
{ //writes at approx 38khz
    for (int i = 0; i < (time / 26) - 1; i++)
    {
        //prescaler at 26 for 16mhz, 52 at 8mhz, ? for 20mhz
        digitalWrite(pin, HIGH);
        delayMicroseconds(10);
        digitalWrite(pin, LOW);
        delayMicroseconds(10);
    }
}
```