iSobot Controllers

Robotics III ECE 410

Edited by

Waleed Alhaddad ([alhad@pdx.edu](mailto:alhad@pdx.edu))

Saad Alaskar ([alq8t@yahoo.com](mailto:alq8t@yahoo.com))

6/13/16

Author: Mathias Sunardi

# Introduction

This is a report on the system to control an iSobot humanoid robot using custom programs such as Python, C, C++, etc. In this report, the programming language used is Python due to the simplicity of accessing/communicating with the serial port. This system was built upon the work done by Aditya Bhutada’s in his MS thesis [1].

# Components

The components involved in this system includes:

1. [Python programming language (http://www.python.org/getit/)](http://www.python.org/getit/))
2. PySerial – serial port module for Python (<http://pyserial.sourceforge.net/)>
3. Arduino Duemilanove/Uno board with IR emitter circuit
4. [Arduino 1.0 IDE (http://arduino.cc/en/Main/Software)](http://arduino.cc/en/Main/Software))

Two pieces of software were written:

1. isobot.py – a Python module defining the iSobot class
2. isobotIR.ino – the Arduino program to translate command bytes to IR emissions understood by the iSobot

I will first discuss the isobot.py module, then the Arduino component.

## Python – isobot.py

**IMPORTANT PREREQUISITES**: install Python, and the PySerial module. Simply, I recommend installing Python version 2.7.x (I don’t guarantee the code I provide below will work with other Python versions). Please refer to their documentation on how to install them – it’s quite straightforward and involves no manual configurations at all.

The isobot.py is a Python module that contains the definition of the “iSobot” class. It utilizes the PySerial module to connect to the serial port (in this case, a USB port).

The full code is provided in Appendix A.

The module does a few things:

* Defines an “iSobot” class
* In the class, over 200 iSobot command bytes are defined as constants. The bytes were obtained from: http:#minkbot.blogspot.com/2009/08/isobot-­‐ infrared-­‐remote-­‐protocol-­‐hack.html
* Communicates via the serial port (e.g. USB) to an infrared (IR) emitter box (controlled by an Arduino board, built by Aditya Bhutada) to transmit the commands to the iSobot robots. The serial port settings are shown in Table 1 (from [1]):

Table 1: Serial Port settings

|  |  |
| --- | --- |
| **Setting** | **Value** |
| Baud Rate | 38400 |
| Data bits | 8 |
| Stop bit | 1 |
| Parity | None |
| Handshaking | None |

* Calculates the checksum of the command string, and format the command strings.
* Allows users to send/specify commands to iSobot in Mode A and/or Mode B.
* Currently only supports iSobot Type 1 commands (support for Type 0 commands will be added later).

How to use it:

* Normally, you would write some sort of a Python script where you can specify the sequence of actions you want iSobot to do.
* To test/see how it works, just use it from the Python shell:

1. Go to the directory where the isobot.py file is located:

~$ cd directory-where-isobotpy-located/

~$ ls isobot.py

1. Load the Python shell:

~$ python

Python 2.7.1 (r271:86832, Jun 16 2011,

16:59:05)

[GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on Darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

1. Load the isobot module:

>>> import isobot

>>>

1. Create an instance of the iSobot class. Give the port name of the

USB/serial port you are using to connect the IR box1 (make sure it is plugged in before you call this – otherwise, it will return an error):

>>> import isobot

>>> bot = isobot.iSobot(‘/dev/tty.usbserial- A9007KX5’)

>>>

1. Try the lazy method to execute a Type 1 command:

isobotDoType1(action, channel=0, repeat=3)

Notice the parameters:

* 1. action = the command byte. This argument is required.
  2. channel (=0 for Mode A, =1 for Mode B). This argument is optional. If you don’t give provide this argument, the method defaults to 0 (Mode A).
  3. repeat (integer – 0 to whatever). This argument is optional. If you don’t provide this argument, the method defaults to 32.

1. Let’s try the walking forward command for an iSobot in Mode B.

>>> bot.isobotDoType1(bot.CMD\_FWRD, 1)

Note: that we are calling the value of the walk forward byte as “bot.CMD\_FWRD” – this is because the command bytes are defined as constants in the iSobot class, so you do have to refer to them as an instance variable.

1. You should see the output as something like this:

>>> bot.isobotDoType1( bot.CMD\_FWRD, 1, 300 )

Command string: ['2', '9', 'b', '7', '0', '3', '\r']

1 In Windows, it’s usually ‘COM#’ where # is some number (e.g. COM4, COM5, etc.)

2 Some commands/actions must be sent continuously to the iSobot for it to perform the action. For example: walking forward. To make iSobot take multiple steps, the ‘walk forward’ byte must be sent continuously. Sending the command 300 times make the iSobot take about 4 steps. However, most of the other commands may only need to be sent once or twice. For example: saying hello.

Tx 0:

port is open Sending command...

hex: 2

hex: 9 hex: b hex: 7

hex: 0

hex: 3 hex:

------------------

Tx 1:

port is open Sending command...

hex: 2

hex: 9 hex: b hex: 7

hex: 0

hex: 3 hex:

------------------

Tx 2:

port is open Sending command...

… *edited …*

The hope is you would use this lazy method most of the time. If you do want to have finer control over this class, other functions and methods3 are available to you as well:

* makeCmd(self, ch, type, cmd1, cmd2=0)

This function will construct the iSobot command string and return it in a hexadecimal string. I provided a detailed explanation on how the command string is constructed in the source code. See the comments above the implementation of this method in Appendix A.

* + This function takes the parameters:

 ch : channel – 0 for Mode A, 1 for Mode B

3 Just as a distinction in programming jargon: method is a procedure that doesn’t give any return value, function is procedure that returns a value

 type : command type – 0 for Type 0, 1 for Type 1

 cmd1 : command Byte 1. Used in command Type 0 and 1. (Type 1 only takes one byte)

 cmd2 : command Byte 1. Used only in command Type 0 (i.e.

Type 0 takes two bytes). Default 0

o This function returns the command string in hexadecimal

* formatCmd(self, cmd)

This function will convert the hexadecimal string into an array of hexadecimal characters.

* + This function takes the argument:

 cmd : a raw hex string. Pass the output of the makeCmd() function for this argument.

* + This function returns a formatted command string. For example:

>>> cmd = makeCmd( 1, 1, 0xb7)

>>> cmd ‘0x29b703’

>>> formatCmd(cmd)

[‘2’, ‘9’, ‘b’, ‘7’, ‘0’, ‘3’, ‘\r’]

* sendCmd(self, cmd)

This method sends the command string out to the serial port.

* + This method takes the argument:

 cmd : the command string. The string must first be formatted by the formatCmd() function before being used by this method.

* repeatCmd(self, cmd, repeat=300)

This method is the same as sendCmd (sending the command out to the serial port) but allows you to say how many times you want the command to be repeated/sent.

* + This method takes the arguments:

 cmd : the command string. The string must first be formatted by the formatCmd() function before being used by this method.

 repeat : the number of times the command byte (cmd) is to be sent to the iSobot.

The repeatCmd() method essentially calls the sendCmd() method repeatedly. So, if you want to send the command once, you can either:

* use the sendCmd() method, OR
* use the repeatCmd() method with repeat=1.

I also provide some serial port management functions:

* connect(port, baud=38400, databit=8, par=’N’)

This method allows you to (re)connect to a port. If no port argument is provided, it will attempt to connect to the port initially given when the class was instantiated.

* disconnect(self)

This method will close the connection to the serial port (calling Serial.close()). **RECOMMENDED**: that you call this method and close the serial port at the end of your program. Otherwise, when the program quits, it is not always guaranteed that the serial port will be released (based on my experience).

From here, hopefully you will have an idea how to programmatically make iSobot obey your every command (ideas: use genetic algorithm, combine with OpenCV, etc.), or adapt the system to the programming language of your choice. Next, is the Arduino part.

## Arduino

To program your Arduino board, you will need to download and install the Arduino IDE from [http://arduino.cc/en/Main/Software.](http://arduino.cc/en/Main/Software) Once you’ve got it up and running, then you can proceed reading the rest of this report. Is it done? OK, good.

I will not get into depths explaining the Arduino system since that is out of the scope of this report. But here are the basics. The Arduino board is a little beast of a prototyping device. There are many variants of the Arduino board (you can see them here: [http://arduino.cc/en/Main/Hardware),](http://arduino.cc/en/Main/Hardware)) but the one we are using in this project (the Duemilanove or Uno) uses the ATmega328 microcontroller with 32Kbytes of memory running at 16MHz clock. As you can immediately notice, this particular model of Arduino is not suitable for computation-­‐heavy tasks such as image processing, but it is more than enough for simple-­‐yet-­‐sophisticated interface such as communicating with iSobot.

### The IR Emitter

Aditya Bhutada [1] built an IR emitter circuit board (a simple “shield”) that fits with the Arduino board. Pin 7 of the Arduino board is used as the data line that activates the IR LED. The circuit is shown in Figure 1 (taken from [1]). Please refer to his report/thesis for the calculations that were done by Bhutada for the circuit.

### The Firmware – isobotIR

The firmware was built on top of the work done by Miles Moody and other hobbyists to decipher the iSobot command protocol (see Moody’s original post here: [http://www.arduino.cc/cgi-­‐](http://www.arduino.cc/cgi-)bin/yabb2/YaBB.pl?num=1237771631).

Unfortunately, the firmware I wrote is a quick-­‐and-­‐dirty code: it includes codes that are specific to my application. A better way to present/package the firmware is as

an Arduino library; a task that I (or you) can do for the next version/project. I will try to explain what was done as best as I can.

You can see the full source code in Appendix B.

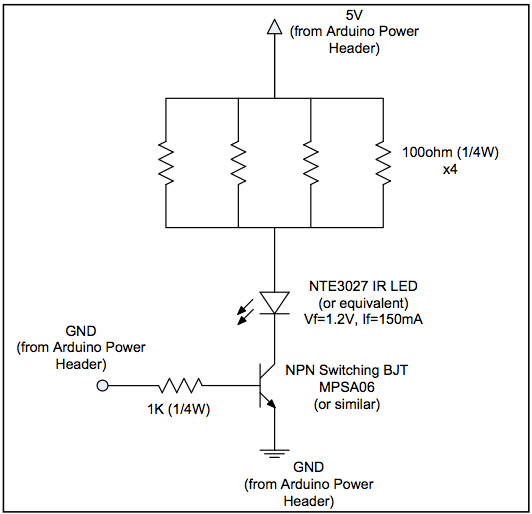


Figure 1: IR Emitter circuit [1]

I will break down and explain each part, but here is the part of the firmware that does the actual work to transmit the bytes as IR signals are as follows (adapted from Moody’s work [2]):

//-------------------info about bits---------------------

----------

#define totallength 22

+18 command

#define channelstart 0

#define commandstart 4

//number of highs/bits 4 channel

//bit where command starts

#define

#define

#define

#define

#define

#define

#define

#define

#define

#define

headerlower 2300

headernom 2550

headerupper 2800

zerolower 300

zeronom 500 //380

zeroupper 650

onelower 800 onenom 1000//850 oneupper 1100

highnom 630

assignments--------------

//doesnt use interrupts so

//lower limit

//nominal

//upper limit

//nominal

//nominal

//---------------------pin

#define TXpin 7

#define RXpin 2 can be anything

#define channellength 4

#define commandlength 18

//---------determined empirically--------------

//----------------------variables----------------------

#define countin 1048576

boolean bit2[totallength]; unsigned long buttonnum; unsigned long x = 0;

unsigned long count = countin; unsigned long buf = 0;

void setup() { Serial.begin(38400); pinMode(RXpin, INPUT); pinMode(TXpin, OUTPUT);

}

void loop() {

// skipped – explained/shown later

}

int SerialReadHexDigit(char digit)

{

byte c = (byte) digit;

if (c >= '0' && c <= '9') { return c - '0';

} else if (c >= 'a' && c <= 'f') { return c - 'a' + 10;

} else if (c >= 'A' && c <= 'F') { return c - 'A' + 10;

} else {

void oscWrite(int pin, int time) {

38khz

for(int i = 0; i < (time / 26) -

//prescaler at 26 for 16mhz, 52 at digitalWrite(pin, HIGH);

//writes at approx

1; i++){

8mhz, ? for 20mhz

}

return -1;

}

// non-hexadecimal digit

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){

(power)

//gives 2 to the

unsigned long

bitshifting and for (int i=0; integer\*=2;

}

integer=1;

pow functions had i<power; i++){

//apparently both

problems

//so I made my own

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);

}

delayMicroseconds(10); digitalWrite(pin, LOW); delayMicroseconds(10);

}

}

#### Firmware Part 1 – Constants and Variables

|  |  |
| --- | --- |
| 1 | //-------------------info about bits------------ |
| 2 | #define totallength 22 //number of highs/bits 4 |
|  | channel +18 command |
| 3 | #define channelstart 0 |
| 4 | #define commandstart 4 //bit where command |
|  | starts |
| 5 | #define channellength 4 |
| 6 | #define commandlength 18 |
| 7 | //---------determined empirically-------------- |
| 8 | #define headerlower 2300 //lower limit |
| 9 | #define headernom 2550 //nominal |
| 10 | #define headerupper 2800 //upper limit |
| 11 | #define zerolower 300 |
| 12 | #define zeronom 500 //nominal |
| 13 | #define zeroupper 650 |
| 14 | #define onelower 800 |
| 15 | #define onenom 1000 //nominal |
| 16 | #define oneupper 1100 |
| 17 | #define highnom 630 |
| 18 | //---------------------pin assignments-------------- |
| 19 | #define TXpin 7 |
| 20 | #define RXpin |
| 21 | //----------------------variables------------------- |
| 22 | #define countin 1048576 |
| 23 |  |
| 24 | boolean bit2[totallength]; |
| 25 | unsigned long buttonnum; |
| 26 | unsigned long x = 0; |
| 27 | unsigned long count = countin; |
| 28 | unsigned long buf = 0; |
| 29 |  |

Moody defined several constants in his code, but in this project/application we only need a few of them. That is, in the scope of this project, you can ignore most of those constants, but do pay special attention to the following (highlighted items above):

-­‐ (line 2) #define totallength 22:

* This value is used in the buttonwrite() function.
* It refers to the number of bits in a type 1 iSobot command. Type 0 commands have 30 bits. As you can see, this firmware currently only focuses on type 1 commands. You can make this firmware to support type 0 commands4 as your next/future project.

-­‐ (line 9) #define headernom 2550:

* This value is used in the buttonwrite() function.
* It refers to the 2.5 ms signal (at 38kHz – explained below) that needs to be sent to iSobot as the header signal, indicating that a command is about to be sent.

-­‐ (line 12) #define zeronom 500:

* This value is used in the buttonwrite() function.
* It refers to the gap (logic 0) between bursts (logic 1) in the signal. For logic 0, the signal is preceded by 0.5ms of logic 0, followed by a 0.5-­‐ 0.6ms burst of logic 1.

-­‐ (line 15) #define onenom 1000:

* This value is used in the buttonwrite() function.
* It refers to the gap (logic 0) between bursts (logic 1) in the signal. For a logic 1, the signal is preceded by 1.0ms of logic 0, followed a 0.5-­‐ 0.6ms burst of logic 1.

-­‐ (line 17) #define highnom 630:

* This value is used in the buttonwrite() function.
* It refers to the duration of the bursts of logic 1. This is the original value used by Moody [2] which seems to work fine with my system. Bhutada’s reported using 0.5ms, while profmason [3] probed the signal to be at 0.55ms. You can try different values which may work better.

-­‐ (line 19) #define TXpin 7:

* This value is used in the setup() and loop().
* It refers to the output (i.e. TX) pin of the Arduino board that drives the IR LED.

-­‐ (line 22) #define countin 1048576:

* This value is used in the loop() function.
* It is used as the initial value for the variable:

 unsigned long count = countin;

* It refers to the value of a 6-­‐digit hex string (220).
* I needed it to convert the hex characters received into the 22-­‐bit command string (in binary).

4 I have not fully confirmed this, but type 0 commands seems to involve manual and individual control over iSobot’s arms and/or walking [(http://minkbot.blogspot.com/2009/08/isobot-­‐](http://minkbot.blogspot.com/2009/08/isobot-)infrared-­‐remote-­‐protocol-­‐ hack.html)

I will skip the details on the variable declarations, as they are relatively straightforward. The only variables you might want to pay attention to are:

-­‐ unsigned long x = 0;

-­‐ unsigned long count = countin;

-­‐ unsigned long buf = 0;

These variables have type ‘unsigned long’ because they are used to calculate the 22-­‐ bit command string (3 bytes). Regular ‘int’ type only holds up to 2 bytes, while ‘unsigned long’ holds up to 4 bytes. As I mentioned above, the variable ‘count’ is initialized to have the value of the countin constant (line 28).

#### Firmware Part 2 – setup()

1

2

3

4

5

void setup() { Serial.begin(38400); pinMode(RXpin, INPUT); pinMode(TXpin, OUTPUT);

}

The setup() method along with the loop() method are the core constructs in an Arduino code. They are the absolute minimum methods you must implement.

In the setup method, you define things like: pin assignments, serial port initialization, etc.

In fact, as you can see above, those are exactly the only things we did:

-­‐ (line 2) Serial.begin(38400):

* We initialize communication with the serial port at 38400 baud rate.

-­‐ (line 3) pinMode(RXpin, INPUT);

* Assign pin #2 (see value of the constant RXpin) as input line (we are not using this pin in this project).

-­‐ (line 4) pinMode(TXpin, OUTPUT);

* Assign pin #7 (see value of the constant TXpin) as the output line. In this case, this pin drives the IR LED.

I will explain the loop() method last, after all the other methods are explained. That way, I hope the explanation of the loop method will make more sense.

#### Firmware Part 3 – SerialReadHexDigit(char digit)

1

2

3

int SerialReadHexDigit(char digit)

{

byte c = (byte) digit;

|  |  |  |
| --- | --- | --- |
| 4 | } | if (c >= '0' && c <= '9') { return c - '0';  } else if (c >= 'a' && c <= 'f') { return c - 'a' + 10;  } else if (c >= 'A' && c <= 'F') { return c - 'A' + 10;  } else {  return -1; // non-hexadecimal digit  } |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |

The SerialReadHexDigit() function takes a ‘digit’ argument in the form of a hexadecimal *character* (0..9,A..F) – *not* the actual hexadecimal value. It is important to note that in this system, it was assumed that the software (i.e. my Python code) is sending the command string one hexadecimal digit at a time. This is because the serial port only buffers one byte at a time, so you cannot send the whole command string at once.

However, the hexadecimal digit being sent is represented as an ASCII character. As you can see in Figure 2 below5 the character ‘9’ has decimal value of 57, and character ‘A’ has decimal value of 65. For this reason, we need this method to convert these characters into computable (i.e. decimal) values to construct the actual command string (in binary bits).

5 Source: <http://www.asciitable.com/>

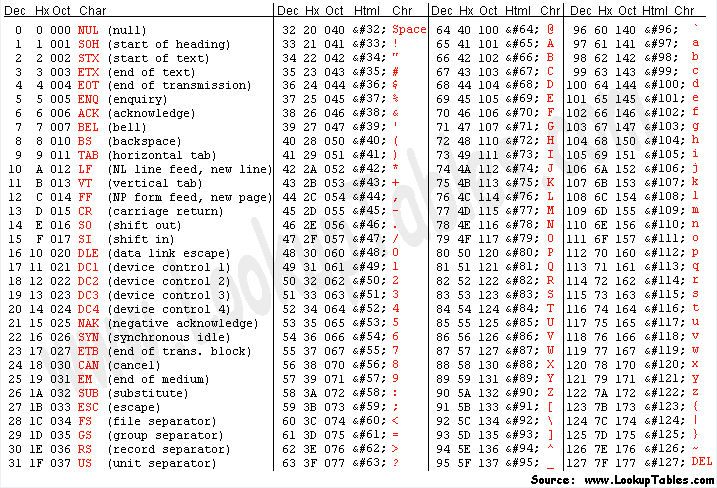


Figure 2: ASCII codes

So, in order to get the actual decimal values of the hexadecimal characters, the SerialReadHexDigit function does the following (pseudo code):

If ‘digit’ is a character in

{‘0’,’1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’}:

return the ASCII decimal value of ‘digit’ minus ASCII decimal value of 0,

else, if ‘digit’ is a character in

{‘a’,’b’,’c’,’d’,’e’,’f’} (lowercase): return the ASCII decimal value of ‘digit’

minus ASCII decimal value of ‘a’ (lowercase a) plus 10,

else, if ‘digit’ is a character in

{‘A’,’B’,’C’,’D’,’E’,’F’} (uppercase):

return the ASCII decimal value of ‘digit’ minus ASCII decimal value of ‘A’ (uppercase a) plus 10,

else return -1 (other characters are invalid)

The SerialReadHexDigit function can be represented in Table 2:

Table 2: SerialReadHexDigit function

|  |  |
| --- | --- |
| (char) digit | (int) SerialReadHexDigit(digit) |
| ‘0’ | 0 |
| ‘1’ | 1 |
| ‘2’ | 2 |
| ‘3’ | 3 |
| ‘4’ | 4 |
| ‘5’ | 5 |
| ‘6’ | 6 |
| ‘7’ | 7 |
| ‘8’ | 8 |
| ‘9’ | 9 |
| ‘a’ or ‘A’ | 10 |
| ‘b’ or ‘B’ | 11 |
| ‘c’ or ‘C’ | 12 |
| ‘d’ or ‘D’ | 13 |
| ‘e’ or ‘E’ | 14 |
| ‘f’ or ‘F’ | 15 |

#### Firmware Part 4 – ItoB(unsigned long integer, int length)

|  |  |
| --- | --- |
| 1 | void ItoB(unsigned long integer, int length){  //needs bit2[length]  Serial.println("ItoB"); // for debugging purposes  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();  } |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |
| 12 |
| 13 |

The ItoB() (stands for ‘Integer to Binary’) method takes the integer form of the command string, and stores the binary bits into the array bit2. Notice that the array bit2 was declared with length 22 (see the variable declaration line 28).

#### Firmware Part 5 – power2(int power)

|  |  |
| --- | --- |
| 1 | unsigned long power2(int 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;  } |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |

This function takes the argument ‘power’ and calculates/returns 2power .As Moody commented (yes, those comments are his original comments on the code), he wrote this function because the built-­‐in bit shifting and power functions did not suffice.

#### Firmware Part 6 – buttonwrite(int txpin, unsigned long integer)

|  |  |
| --- | --- |
| 1 | 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);  } |
| 2 |
| 3 |
| 4 |
| 5 |
| 6 |
| 7 |
| 8 |
| 9 |
| 10 |
| 11 |

The buttonwrite() method takes the arguments:

-­‐ txpin : the pin number which drives the IR LED

-­‐ integer : the integer value of the command string (3 bytes – hence the ‘unsigned long’ type)

This method essentially does all the iSobot communication protocols. The constants given above already gave some indication of the protocol, but the following image illustrates the protocol (taken from [1]).

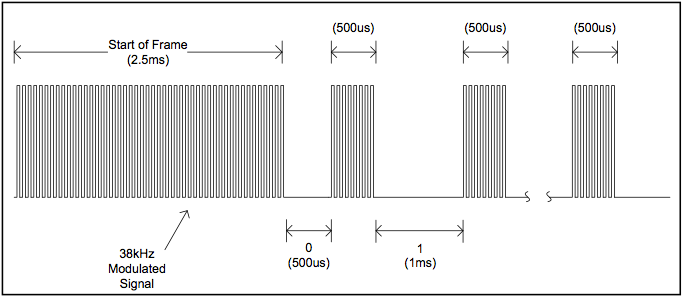


Figure 3: iSobot IR communication protocol

As shown in the illustration, the message must be initiated with the “Start of Frame” signal which lasts for 2.5ms. The message itself is modulated at 38kHz. After the “Start of Frame”, the actual binary bits of the message is then sent as (this is a repeat from above):

-­‐ logic 0 : 0.5ms gap/logic 0 followed by a 0.5ms burst of logic 1.

-­‐ logic 1 : 1.0ms gap/logic 1 followed by a 0.5ms burst of logic 1.

So, the buttonwrite() method performs this protocol as follows:

1. (line 3) ItoB(integer,22) : this prepares the command string into an array of binary (i.e. Boolean) bits.
2. (line 4) oscWrite(txpin, headernom) : this sends the “Start of Frame” signal.
3. (line 5 through 8) send each bit (in the array bit2) according to the protocol of sending logic 0 and 1 above, using the method oscWrite(). Notice, the arguments ‘zeronom’ (line 6) and ‘onenom’ (line 7) are defined in the constants above, and refer to delay for logic 0 (0.5ms) and 1 (1.0ms), respectively. The argument ‘highnom’ (line 8) refers to the duration of the burst of logic 1.
4. (line 10) Give a delay of about 0.2ms before the next command can be read.

#### Firmware Part 7 – oscWrite(int txpin, int time)

1

2

3

4

5

void oscWrite(int pin, int time) {

approx 38khz

for(int i = 0; i < (time / 26) -

//prescaler at 26 for 16mhz, 52 at digitalWrite(pin, HIGH); delayMicroseconds(10);

//writes at

1; i++){

8mhz, ? for 20mhz

6

7

8

9

digitalWrite(pin, LOW);

delayMicroseconds(10);

}

}

This method takes the arguments:

-­‐ pin : the pin number to drive (in this case, to drive the IR LED).

-­‐ time : the burst duration under 38kHz.

The value 26 (line 2) is the prescaler used to make the signal being sent is at 38kHz rate, since we are using a 16MHz clock for the ATmega328 of the Arduino board (Arduino Duemilanove or Uno). If you are using an 8MHz clock, then the value of the prescaler is 52. I had to empirically try different values for the delays between logic 1 and logic 0 for the IR LED (lines 4 and 6). Some values may make the iSobot to not always respond to/execute every single command being sent. I found 10 yields a pretty good result (i.e. all commands are accepted and executed).

#### Firmware Part 8 – loop()

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

void loop() {

while (Serial.available() > 0){

//Serial control

char switcher= (byte) Serial.read(); if (switcher == '\r') {

Serial.print("Break: "); Serial.println(buf,HEX); buttonwrite(TXpin, buf); buf = 0;

count = countin; delayMicroseconds(300); break;

}

x = SerialReadHexDigit(switcher); x = x \* count;

//Serial.println(x,BIN); buf = buf + x;

count = count / 16;

//Serial.print("Buffer: ");

//Serial.println(buf,HEX);

} // end while

}

The loop() method (as previously mentioned) with the setup() method are the two methods *at minimum* you must implement for an Arduino sketch/program. The loop method is your main method which continuously and repeatedly runs when your Arduino board is connected to a power supply.

Here, in the loop method, the while loop (line 2) will keep collecting input from the serial port as long there is a byte ready at the port (Serial.available() > 0). If there is not any byte ready at the serial port, the loop method will just keep … looping, and doing nothing since there is nothing to be done outside the while loop. However, if there is a series of bytes from the serial port, it will be collected and calculated to construct the command string (line 14 through 18).

The order of the command string being sent from the serial port (i.e. by the iSobot class) is the highest hexadecimal digit to the lowest digit. So, for example: the command string is ['2', '9', 'b', '7', '0', '3', '\r']. Then, the string will be sent per character in order from left to right: ‘2’ then ‘9’ then ‘b’ and so on. Because of this design choice, the decimal value of the command string is calculated from the highest value first. Hence, the multiplier ‘count’ starts from 1048576 (see constant declarations above, line 22), and after each digit, ‘count’ is divided by 16 (line 18) since it is in hexadecimal (4 bits). The total decimal value of the command string is stored in the variable ‘buf’.

The last character ‘\r’ (newline character) indicates the end of the command string. Thus, when the newline character is detected, the command string is assumed to have been constructed, it is then processed and passed to the buttonwrite() method to be transmitted as IR signals (line 8). After the signal has been transmitted, the ‘buf’ and ‘count’ variables are reset, and a new command is ready to be accepted (after a 0.3ms delay).

## Lingering Issues

There are a few issues that have not been addressed:

-­‐ There is no programmatic way to tell when iSobot is finished with an action (i.e. there is no method we can call from the iSobot to check when it is done executing one command so we can send the next command). At least, there are no known ways to do that at the time this report was written. If another command is sent, it will immediately be executed, without completing the previous command. This may or may not be a feature or bug, depending on how you design your program around this … behavior. The best I could come up with so far is to manually determine how long it takes to complete an action (if I want the iSobot to complete the action) and give the appropriate time delay in my program before sending the next commands. In other occasion, when I am more concerned about synchronizing the iSobot with

other media (music or video), I will prioritize matching the delays according to the timing on the media rather than waiting for the action to complete.

-­‐ Currently, the goal of this project was to realize the Act 4 of the Portland Robot Theater, which involves synchronizing the robot actions to the ECE 2011 Graduation ceremony music video, played by Jay Penev and the ECE faculty and staff. However, there is currently no direct/programmatic synchronization between the robot actions/commands and the music/video. The video was launched from a Python program as a separate process, and following that, the sequence of actions for the iSobots is executed. The timing of the actions for the iSobots was determined manually by hand. Needless to say, the synchronization is currently poor.

-­‐ The IR emitter is currently tethered to a PC. This makes it very difficult to have a good theater because the emitter must be placed somewhere where it has direct view of the iSobots’ IR receiver, while at the same time tethered via a USB cable to a laptop/desktop. It occurred several times during testing/demo that the iSobot is in some position which blocks the IR line-­‐of-­‐ sight, making it not executing several commands.

-­‐ Controlling iSobot in two different modes. The iSobot can operate in either Mode A or Mode B. The mode is selected by a switching a physical switch on the back of the iSobot. There is currently no way of changing modes on-­‐the-­‐ fly (i.e. via a command). When multiple iSobots are on the same mode – let’s say there are two iSobots and both are on Mode B, a command for Mode B sent to the iSobots will be executed by both iSobots simultaneously. It creates an interesting illusion of synchronization. However, we may want each iSobot to do different actions executed at the same time to make for a more interesting performance. For this, the iSobots need to be in different modes. Since we only have one IR emitter, we cannot do this currently.

Moreover, it probably will involve a more complex program – maybe one that requires using threading for simultaneous executions.

-­‐ The current system does not support command Type 0 (control of individual arms, directional walking).

## Conclusions and Future Work

I have created a Python class called iSobot that would (hopefully) make it easier for the next students and future projects to quickly work with the iSobot to do their every whim (or most of it). I also provided a more detailed explanation on the Arduino program for the IR emitter and the iSobot command protocol. I provided the source code both for the Python class and the Arduino program in Appendix A and B, respectively. Additionally, a Python program I wrote as a preliminary version of Act 4 of the Robot Theater project is given in Appendix C.

There are still a number of lingering issues that have been addressed yet. I would like to see future works that build on top of this report address: support for command Type 0, a more elegant solution to timing for command execution, and individual/separate controls for multiple iSobots.

# References:

[1] Bhutada, Aditya, 2010 ‘Universal Event and Motion Editor for Robots‟ Theatre’, MS thesis, Portland State University, Portland, OR.

[2] Moody, Miles, 2009, *I-­‐Sobot hacked or Pro Mini shield*, viewed 4 April 2012,

[<http://www.arduino.cc/cgi-­‐](http://www.arduino.cc/cgi-)bin/yabb2/YaBB.pl?num=1237771631>. [3] Mason, Martin, 2008, *ISOBot IR hacking*, viewed 4 April 2012,

[<http://profmason.com/?p=627](http://profmason.com/?p=627)>.

[4] MichWorks, 2009, *iSobot Infrared Remote Protocol Hack,* viewed 4 April 2012,

[<http://minkbot.blogspot.com/2009/08/isobot-­‐](http://minkbot.blogspot.com/2009/08/isobot-)infrared-­‐remote-­‐protocol-­‐ hack.html>

# Appendix A – isobot.py

import serial,time,re,sys class iSobot:

# iSobot Command byte list

# Source: http:#minkbot.blogspot.com/2009/08/isobot- infrared-remote-protocol-hack.html

#

# Standard commands CMD\_RC = 0x07 CMD\_PM = 0x08 CMD\_SA = 0x09 CMD\_VC = 0x0a

CMD\_1P = 0x13 # left punch CMD\_2P = 0x14 # right punch

CMD\_3P = 0x15 # left side whack (arm outwards) CMD\_4P = 0x16 # right side whack

CMD\_11P = 0x17 # left + right punch CMD\_12P = 0x18 # right + left punch CMD\_13P = 0x19 # left up-down chop CMD\_14P = 0x1a # right up-down chop CMD\_21P = 0x1b # both up-down chop CMD\_22P = 0x1c # both down-up chop

CMD\_23P = 0x1d # right + left punch, both up-down chop, both whack

CMD\_24P = 0x1e # look left, up-down chop CMD\_31P = 0x1f # look right, up-down chop CMD\_32P = 0x20 # "c'mon, snap out of it" slap CMD\_34P = 0x21 # both whack

CMD\_1K = 0x22 # left wide kick CMD\_2K = 0x23 # right wide kick CMD\_3K = 0x24 # left kick CMD\_4K = 0x25 # right kick CMD\_11K = 0x26 # left side kick CMD\_12K = 0x27 # right side kick CMD\_13K = 0x28 # left back kick CMD\_14K = 0x29 # right back kick

CMD\_31K = 0x2a # right high side kick CMD\_42K = 0x2b # right soccer/low kick CMD\_21K = 0x2c # left + right high side kick

CMD\_22K = 0x2d # right + left soccer/low kick

CMD\_23K = 0x2e # combo kick low-left, high-side-right,

left

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| CMD\_24K | = | 0x2f | # | another left kick |  |
| CMD\_31K | = | 0x30 | # | right high kick |
| CMD\_34K | = | 0x31 | # | split |
| CMD\_1G = | | 0x32 | # | Block! "whoa buddy" |
| CMD\_2G = | | 0x33 | # | right arm block |
| CMD\_3G = | | 0x34 | # |  |
| CMD\_4G = | | 0x35 | # | both arms block |
| CMD\_11G | = | 0x36 | # | dodge right (move left) |
| CMD\_12G | = | 0x37 | # | dodge left (move right) |
| CMD\_13G | = | 0x38 | # | headbutt |
| CMD\_14G | = | 0x39 | # | right arm to face |
| CMD\_21G | = | 0x3a | # | taunt1 |
| CMD\_22G | = | 0x3b | # | hit & down |
| CMD\_23G | = | 0x3c | # | dodge right, left, block | left, head, |
| fall down  CMD\_A = 0x3d CMD\_B = 0x3e  CMD\_1A = 0x3f # "Roger!" raise right arm CMD\_2A = 0x40 # weird gesture  CMD\_2A = 0x41 # "All your base are belong to isobot" CMD\_3A = 0x42 # "absolutely not!" flaps both arms CMD\_4A = 0x43 # bow/crouch? and get back up  CMD\_11A = 0x44 # "Good morning!" raise both arms, stand on left foot  CMD\_12A = 0x45 # "Greetings I come in peace" wave right arm  CMD\_13A = 0x46 # "Y'all come back now, you hear!"  CMD\_14A = 0x47 # "Wassap!?" opens both arms sideways over and down  CMD\_21A = 0x48 # "Greetings human" raise left arm and  bow  CMD\_22A = 0x49 # "It's an honor to meet you!" bow and shake right hand  CMD\_23A = 0x4a # "Bye bye" CMD\_31A = 0x4b # "Bon voyage!"  CMD\_32A = 0x4c # \*clap\* \*clap\* "Thanks! I'll be here all week" raise right arm  CMD\_33A = 0x4d # "T-t-that's all robots!" raise left arm, stand on left foot  CMD\_41A = 0x4e # "Domo arigato from isobot-o" CMD\_42A = 0x4f  CMD\_43A = 0x50 CMD\_111A = 0x51 CMD\_222A = 0x52 CMD\_333A = 0x53  CMD\_11B = 0x54 # Walk forward + "Give me a bear hug" CMD\_12B = 0x55 | | | | | |

CMD\_13B = 0x56 CMD\_14B = 0x57 CMD\_31B = 0x58 CMD\_22B = 0x59 CMD\_23B = 0x5a CMD\_24B = 0x5b CMD\_31B = 0x5c

CMD\_32B = 0x5d # "woe is me ... what to do ... what to do" bow, shakes head

CMD\_33B = 0x5e # "No no .... not again. ... No no" CMD\_234B = 0x5f # "Oh, I can't believe I did that" CMD\_41B = 0x60 # "I throw myself into a mercy" (?) CMD\_42B = 0x61 # "Oh, like a dagger through my heart" CMD\_43B = 0x62 # Same as 44B but no voice

CMD\_44B = 0x63 # "Ouch, that hurts!" CMD\_112A = 0x65 # points left "wahoo" CMD\_113A = 0x66 # pose northwest "hoo-ah!" CMD\_114A = 0x67 # points left "kapwingg" CMD\_124A = 0x6b # "iz nice. you like?"

CMD\_131A = 0x6c # both arm wave left right left CMD\_132A = 0x6d # drunk

CMD\_113B = 0x6e # "no please make it stop." "please i can't take it anymore" "no no" lying down and get up

CMD\_114B = 0x6f # "yippe yippe" 3 times, goal post arms CMD\_121B = 0x70 # "ho ho ho ... <something-something>

isobot"

CMD\_122B = 0x71 # "yeehaaw" both arm wave left right CMD\_123B = 0x72

CMD\_124B = 0x73 # stand on one foot, goal post arms, "wow that's amazing"

CMD\_131B = 0x74 # bow, arms over head and down CMD\_132B = 0x75

CMD\_133B = 0x76 CMD\_134B = 0x77 CMD\_141A = 0x78

CMD\_143A = 0x79 # sit cross legged CMD\_144A = 0x7b # ... owl?

CMD\_211B = 0x7c

CMD\_212B = 0x7d # "Ahh, let me get comfortable. I'm too sexy for my servos" lie down, flips over, gets up

CMD\_213B = 0x7e

CMD\_221B = 0x80 # balancing act + bleeps (+) CMD\_222B = 0x81 # looks like a push up CMD\_223B = 0x82

CMD\_224B = 0x83 # "You can count on me" CMD\_232B = 0x85

CMD\_233B = 0x86

CMD\_A =

times

CMD\_B = CMD\_AB = CMD\_AAA CMD\_BBB

0x8a

# flip forward back forward about 3

0x8b

0x8c

=

=

CMD\_BAB

CMD\_ABB CMD\_BBA CMD\_ABA CMD\_ABAB CMD\_AAAA CMD\_FWRD CMD\_BWRD CMD\_FWLT CMD\_FWRT CMD\_LEFT CMD\_RGHT CMD\_BKLT CMD\_BKRT CMD\_411A CMD\_412A CMD\_413A CMD\_444B CMD\_444A

=

=

=

=

0x8d

0x8e 0x8f 0x95

0x97

0x98

#

#

#

#

"BANZAI"

chicken dancing

3 times

(+)

=

=

=

=

=

=

=

=

CMD\_241B = 0x88

CMD\_242B = 0x89

# headstand

giant robot motion

= 0x99

= 0x9a

= 0xb7

= 0xb8

= 0xb9

= 0xba

= 0xbb

= 0xbc

= 0xbd

= 0xbe

= 0xc7

= 0xc8

= 0xc9

= 0xca

= 0xcb # nothing CMD\_LVSoff = 0xd3

CMD\_HP =

CMD\_NOIMP CMD\_END = MSG\_NOIMP MSG\_NOIMP MSG\_RUP MSG\_RDW MSG\_RRT MSG\_RLT MSG\_LUP MSG\_LDW MSG\_LRT MSG\_LLT

0xd5

= 0xd6 0xd7

= 0x848080

= 0x848080 0x878280

0x808280

0x8480f0

0x848080

0x84f080

0x841080

0xec8080 0x0c8080

# Bonus Commands

CMD\_TURNON = 0x01 CMD\_ACTIVATED = 0x02 CMD\_READY = 0x03 CMD\_RC\_CONFIRM = 0x04

same

CMD\_RC\_PROMPT = 0x05 CMD\_MODE\_PROMPT = 0x06

CMD\_IDLE\_PROMPT = 0x0B # = 0x0C,= 0x0D,= 0x0E all the

CMD\_HUMMING\_PROMPT = 0x0F CMD\_COUGH\_PROMPT = 0x10 CMD\_TIRED\_PROMPT = 0x11 CMD\_SLEEP\_PROMPT = 0x12 CMD\_FART = 0x40 # 2A CMD\_SHOOT\_RIGHT = 0x64 CMD\_SHOOT\_RIGHT2 = 0x68 CMD\_SHOOT2 = 0x69 CMD\_BEEP = 0x6a

CMD\_BANZAI = 0x7F # "TAKARA TOMY" CMD\_CHEER1 = 0x90

CMD\_CHEER2 = 0x91 CMD\_DOG = 0x92 CMD\_CAR = 0x93 CMD\_EAGLE = 0x94 CMD\_ROOSTER = 0x95 CMD\_GORILLA = 0x96 CMD\_LOOKOUT = 0xA1

CMD\_STORY1 = 0xA2 # knight and princess CMD\_STORY2 = 0xA3 # ready to start day CMD\_GREET1 = 0xA4 # good morning CMD\_GREET2 = 0xA5 # do somthing fun CMD\_POOP = 0xA6 # poops his pants CMD\_GOOUT = 0xA7 # ready to go out dancing

CMD\_HIBUDDY = 0xA8 # .. bring a round of drinks CMD\_INTRODUCTION = 0xA9

CMD\_ATYOURSERVICE = 0xAA CMD\_SMELLS = 0xAB CMD\_THATWASCLOSE = 0xAC CMD\_WANNAPICEOFME = 0xAD CMD\_RUNFORYOURLIFE = 0xAE CMD\_TONEWTODIE = 0xAF

# 0xB0 - nothing? CMD\_SWANLAKE = 0xB1 CMD\_DISCO = 0xB2 CMD\_MOONWALK = 0xB3 CMD\_REPEAT\_PROMPT = 0xB4 CMD\_REPEAT\_PROMPT2 = 0xB5 CMD\_REPEAT\_PROMPT3 = 0xB6

# 0xB7-= 0xC4 single steps in different directions CMD\_HEADSMASH = 0xC5

CMD\_HEADHIT = 0xC6

# 0xCC-= 0xD2 - unknown (use param?)

# after exercising one of these I am getting only beeps instead of voice/sounds

# (looks like a tool to synchronize sound with moves) CMD\_HIBEEP = 0xD3

# = 0xD4 - unknown (use param?) CMD\_BEND\_BACK = 0xD8 # same untill = 0xDB

CMD\_SQUAT = 0xDB # also = 0xDC # doesn't work (both) CMD\_BEND\_FORWARD = 0xDD

CMD\_HEAD\_LEFT\_60 = 0xDE CMD\_HEAD\_LEFT\_45 = 0xDF CMD\_HEAD\_LEFT\_30 = 0xE0 CMD\_HEAD\_RIGHT\_30 = 0xE1 CMD\_HEAD\_RIGHT\_45 = 0xE2 CMD\_HEAD\_RIGHT\_60 = 0xE3

# seems identical to A & B getups CMD\_GETUP\_BELLY = 0xE4 CMD\_GETUP\_BACK = 0xE5

# E6 unknown CMD\_HEAD\_SCAN\_AND\_BEND = 0xE7 CMD\_ARM\_TEST = 0xE8 CMD\_FALL\_AND\_LEG\_TEST = 0xE9 CMD\_THANKYOUSIR = 0xEA CMD\_ILOVEYOU\_SHORT = 0xEB CMD\_3BEEPS = 0xEC CMD\_FALL\_DEAD = 0xED CMD\_3BEEPS\_AND\_SLIDE = 0xEE

# EF-FF unknown serialPort = 0

#

# Initialize class

#

def init (self, port='/dev/cu.usbserial-A8008pQc', baud=38400, databit=8, parity=None):

print "Initializing iSobot!" self.\_port = port

#port='/dev/tty.usbserial-A8008pQc' # Mac default

USB

#port='/dev/tty.usbserial-A9007KX5' # The other Mac

USB port

try:

self.\_serialPort = serial.Serial(port, baud,

bytesize=databit, parity='N') #UNCOMMENT TO RUN #UNCOMMENT TO RUN

self.\_serialPort.open()

if self.\_serialPort.isOpen():

print "Serial port is opened."

except Exception as e: # Catch exception in case serial connection fails

print "Unable to connect to serial port." print e

sys.exit(1)

#

# Construct command string

# Returns integer. To use: convert returned value using hex() then process as array of characters excluding '0x'

# How to construct isobot command string:

## command = [channel (1 bit)]:[type (2 bits)]:[checksum (3 bits)]:[commandbyte1 (8

bits)]:[commandbyte2 (8 bits)]:[params (8 bits)]

## channel: 0 -> Mode A, 1 -> Mode B

## type: 00 -> Type 0, 01 -> Type 1

## checksum: How to calculate:

### 1. add the header bits (channel, type, and checksum). For this, just give checksum 0x00 in the calculation.

### After the calculation, this value will be updated.

### 2. Do sum (logical OR) on the sum bits, 3 bits at a time. (see below: implemented as 3-bits right-shift)

### 3. Return the last three bits of this value as the checksum.

### 4. Add the checksum to the header bits (just do normal +)

## commandbyte1: see isobot.py for the command bytes

## commandbyte2: see isobot.py for the command bytes.

Not used in command Type 1

## params: ALWAYS 0x03 (don't know what it is for)

# Example:

## For Mode A (channel bit: 0), Type 1 (type bits: 01), checksum (bits: 000):

### header\_bits = channel:type:checksum

### = 0:01:000

### Notice this is a 6-bits string. You must look at it as a byte.

### header\_bits (as byte, in hex) = 00001000 = 0x08

## For Mode B (channel bit: 1), Type 1 (type bits: 01), checksum (bits: 000):

### header\_bits = 1:01:000

### header\_bits (as byte, in hex) = 00101000 = 0x28

## Walk forward byte: CMD\_FWRD = 0xb7 = 10110111 (see

isobot.py)

## Params: 0x03 = 00000011

## command string in Mode A, Type 1 (checksum not calculated yet): [header\_bits]:[walkforwardbyte]:[params] = [00101000]:[10110111]:[00000011]

## Caculate checksum:

### sum = 0x28 + 0xb7 + 0x03

### = 226 = 0xe2 = 11100010

### take and sum 3 bits at a time (i.e. scan 3 bits at a time from right to left)

### 010 + 100 + 011 (padded with zero) = (1)001

### The total is actually 9 (0x09) but we only use the last three bits. So checksum = 0x01

## Add the checksum to the header bits:

### 0x28 + 0x01 = 0x29 = 00101001

## The command string becomes: [00101000]:[10110111]:[00000011] = 0x29b703

def makeCmd(self, ch, type, cmd1, cmd2=0): param = 0x03

# Different header bytes depending on channel and type. See[: http://minkbot.blogspot.com/2009/08/isobot-](http://minkbot.blogspot.com/2009/08/isobot-) infrared-remote-protocol-hack.html

if ch==0 and type==0: hdr = 0x00

elif ch==1 and type==0: hdr = 0x20

elif ch==0 and type==1: hdr = 0x08

elif ch==1 and type==1: hdr = 0x28

else:

return -1

# Calculate sum of command string. Checksum: 000 if type==0:

sum = hdr + cmd1 + cmd2 + param # For command type 0 (individual/manual arm control?)

elif type==1:

sum = hdr + cmd1 + param # For command type 1 (most commonly used)

else:

return -1

# Calculate checksum

chksum = ((sum & 7) + ((sum >> 3) & 7) + ((sum >>

# byte

6) & 7) &7)

hdrsum = hdr + chksum

# Construct the hex

if type==0:

return hex(((hdrsum (cmd2 << 8) + (param)))

elif type==1:

<< 32) + (cmd1 << 16) +

string for type 0 commands

return hex(((hdrsum << 16) + (cmd1 << 8) +

(param))) # byte string for type 1 commands else:

return -1

#

# Send command to serial port (Arduino + IR - Aditya's

box)

#

def sendCmd(self, cmd):

#if serialPort.isOpen(): try:

print "port is open"

print "Sending command...\n" for c in cmd:

print "hex: %s" % c self.\_serialPort.write(c) #UNCOMMENT TO RUN

#serialPort.close() print "------------------\n"

#else:

except serial.SerialException:

print "Port is not open/available"

#serialPort.close()

#

# Repeat sending command

# Default # of tries: 300. Some actions (e.g. Walk) require the command to be sent for a period of time.

# e.g. sending the Walk FWRD command once, the robot will accept the command but not move forward

def repeatCmd(self, cmd, rep=300): for i in range(rep):

print "Tx %d: " % i self.sendCmd(cmd) time.sleep(0.5)

#

# Format the hex string

#

# Shorthand function for lazy people (like

#

def isobotDoType1(self, action, channel=0, try:

me)

repeat=3):

### Management

#

# Close serial

#

functions ###

port

#

def formatCmd(self, cmd):

# Remove leading 0x in hex string:

[# http://stackoverflow.com/questions/5197959/how-](http://stackoverflow.com/questions/5197959/how-) do-i-remove-hex-values-in-a-python-string-with-regular- expressions

c = re.sub(r'0x','',cmd)

# The string must be 6 digits long. Check; if not, add with a leading 0 (assuming the command is type 1 and can only vary

# between 5 or 6 characters if len(c) < 6:

c = c.zfill(6)

c = c + '\r'

print "Command string: %s" % list(c)

# Return the string as a list of characters:

#

<http://groups.google.com/group/comp.lang.python/browse_thre> ad/thread/6543299e955388e2?pli=1

return list(c) # Must add '\r' at the end of each

string

self.repeatCmd(self.formatCmd(self.makeCmd(channel,1,action

)),repeat)

return 0

except Exception as e:

print "Blargh! Command failed!" print e

return 1

def disconnect(self):

print "Closing serial port ..." try:

self.\_serialPort.close() print "Port is closed."

return 0

except Exception as e:

print "Unable to close port." print e

return 1

#

# Open serial port

#

def connect(self, port, baud=38400, databit=8, par='N'):

if port == '':

print "No port supplied. Will use previously

used port."

port = self.\_port

try:

print "Connecting to port ... %s" % port self.\_serialPort = serial.Serial(port, baud,

bytesize=databit, parity=par)

self.\_serialPort.open()

if self.\_serialPort.isOpen(): print "Serial port is opened." return 0

except Exception as e:

print "Unable to connect to serial port." print e

sys.exit(1)

# Appendix B – isobotIR.ino

//-------------------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;

void setup() { Serial.begin(38400); pinMode(RXpin, INPUT); pinMode(TXpin, OUTPUT);

}

void loop() {

while (Serial.available() > 0){ //Serial control

//msg = Serial.read()

char switcher= (byte) Serial.read(); if (switcher == '\r') {

}

-1; // non-hexadecimal character

Serial.print("Break: ");

Serial.println(buf,HEX); buttonwrite(TXpin, buf); buf = 0;

count = countin; delayMicroseconds(300); break;

}

x = SerialReadHexDigit(switcher);

x = x \* count;

//Serial.println(x,BIN); buf = buf + x;

count = count / 16;

//Serial.print("Buffer: ");

//Serial.println(buf,HEX);

} // end while

}

int SerialReadHexDigit(char digit)

{

//byte c = WaitAndRead(); byte c = (byte) digit;

if (c >= '0' && c <= '9') { return c - '0';

}

}

else if

return else if return

else { return

(c >= 'a' && c <= 'f') {

c - 'a' + 10;

(c >= 'A' && c <= 'F') { c - 'A' + 10;

}

}

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){ (power)

//gives 2 to the

unsigned long

bitshifting and for (int i=0; integer\*=2;

}

integer=1;

pow functions had i<power; i++){

//apparently both

problems

//so I made my own

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) {

approx 38khz

for(int i = 0; i < (time / 26) -

//prescaler at 26 for 16mhz, 52 at digitalWrite(pin, HIGH); delayMicroseconds(10); digitalWrite(pin, LOW); delayMicroseconds(10);

}

}

//writes at

1; i++){

8mhz, ? for 20mhz

# Appendix C – Preliminary Act 4 program

import serial, time

import subprocess, isobot, threading

#

# the iSobot sequence will be running as a separate thread

#

class isobotThread( threading.Thread ): def run(self):

isoport = '/dev/tty.usbserial-A8008pQc'

isoport

print "Connecting to isobot on port: %s ..." % bot = isobot.iSobot(isoport, 38400)

bot.isobotDoType1(bot.CMD\_RC,1,1) # for some reason, the first command always fail/ignored

#for i in range(10000):

# if i > 9000:

# continue

time.sleep(8) # careful

with the delays

bot.isobotDoType1(bot.CMD\_3P,1,1) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_11G,0,1)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_11G,1,1) time.sleep(3)

#bot.isobotDoType1(bot.CMD\_12G,1,1)

#time.sleep(3) bot.isobotDoType1(bot.CMD\_FWRT,1,5) time.sleep(0.5) bot.isobotDoType1(bot.CMD\_FWRT,1,5) time.sleep(2) bot.isobotDoType1(bot.CMD\_BKLT,1,4) time.sleep(0.5) bot.isobotDoType1(bot.CMD\_BKLT,1,5) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_11G,1,1)

#time.sleep(3) bot.isobotDoType1(bot.CMD\_12G,1,1) time.sleep(3) bot.isobotDoType1(bot.CMD\_FWLT,1,4) time.sleep(0.5) bot.isobotDoType1(bot.CMD\_FWLT,1,5) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_BKLT,1,4)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_BKRT,1,4) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_FWRT,1,4)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_FWRT,1,5) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_BKLT,1,4)

#time.sleep(2) bot.isobotDoType1(bot.CMD\_11G,1,1) time.sleep(3)

#bot.isobotDoType1(bot.CMD\_12G,1,1)

#time.sleep(3) bot.isobotDoType1(bot.CMD\_21K,1,1) time.sleep(3) bot.isobotDoType1(bot.CMD\_4G,1,1) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_11G,1,1)

#time.sleep(3) bot.isobotDoType1(bot.CMD\_12G,1,1) time.sleep(3)

#bot.isobotDoType1(bot.CMD\_FWRT,1,4)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_FWLT,1,5) time.sleep(2) bot.isobotDoType1(bot.CMD\_BKRT,1,4) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_32B,1,1)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_12G,1,1) time.sleep(3) bot.isobotDoType1(bot.CMD\_FWRT,1,4) time.sleep(0.5) bot.isobotDoType1(bot.CMD\_FWLT,1,5) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_BKLT,0,4)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_BKRT,1,4) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_11G,0,1)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_11G,1,1) time.sleep(3)

#bot.isobotDoType1(bot.CMD\_12G,1,1)

#time.sleep(3) bot.isobotDoType1(bot.CMD\_22K,1,1)

time.sleep(3) bot.isobotDoType1(bot.CMD\_1G,1,1) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_FWRT,1,4)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_FWRT,1,5) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_BKLT,0,4)

#time.sleep(0.5)

bot.isobotDoType1(bot.CMD\_BKLT,1,4) time.sleep(3) bot.isobotDoType1(bot.CMD\_FWRT,1,4) time.sleep(3) bot.isobotDoType1(bot.CMD\_FWRT,1,5) time.sleep(3)

#bot.isobotDoType1(bot.CMD\_BKLT,1,4)

#time.sleep(0.5)

bot.isobotDoType1(bot.CMD\_BKLT,1,4) time.sleep(2) bot.isobotDoType1(bot.CMD\_32B,1,1) time.sleep(2)

#bot.isobotDoType1(bot.CMD\_11G,0,1)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_11G,1,1) time.sleep(3) bot.isobotDoType1(bot.CMD\_12G,1,1) time.sleep(3) bot.isobotDoType1(bot.CMD\_21K,1,1) time.sleep(2) bot.isobotDoType1(bot.CMD\_1G,1,1) time.sleep(2) bot.isobotDoType1(bot.CMD\_11G,1,1) time.sleep(3)

#bot.isobotDoType1(bot.CMD\_12G,0,1)

#time.sleep(0.5) bot.isobotDoType1(bot.CMD\_12G,1,1) time.sleep(3) bot.isobotDoType1(bot.CMD\_22K,1,1) time.sleep(3) bot.isobotDoType1(bot.CMD\_32A,1,1) time.sleep(10)

port = '/dev/tty.usbserial-A9007KX5' try:

print "Connecting to port: %s ..." % port

# Wait a few seconds.

robots play at the same time

# as the music in the time.sleep(8)

--

everybody strumming"

arduino = serial.Serial(port, 9600) # this is for a

second Arduino board that controls activation of two Halloween robots (Appendix D).

except:

print "Failed connecting to serial port", port

try:

if arduino.isOpen():

# Play the video (using VLC) vlc =

subprocess.Popen(["/Applications/VLC.app/Contents/MacOS/VLC ", "ecegraduation.mov"])

if vlc: print "VLC on!"

# Start the isobot thread isobotThread().start()

Adjust this to make the

video starts

print "Song starts arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

print "c'mon y'all

Greenwood's in the band oh

arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

let's clap some hands -

yeah!"

even

print "Strumming ..."

arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

print "Rockin out with famous names, Brano, Holtzmann and McNames oh yeah!"

arduino.write('A') print "Writing A"

print "We're gonna have a bash with Perkowski,

Daasch oh yeah!"

perokwski

on

arduino.flush()

time.sleep(7)

print "Strumming ..." arduino.write('A') print "Writing A" arduino.flush() time.sleep(7)

and

Hall

arduino.write('B') # Mcnames off, print "Writing B" arduino.flush()

time.sleep(7)

print "Strumming ..."

arduino.write('B') # both on print "Writing B" arduino.flush()

time.sleep(7)

print "We might get serious ..."

arduino.write('B') # Mcnames off, print "Writing B" arduino.flush()

time.sleep(7)

perokwski

on

print "Strumming ..."

arduino.write('B') # both on print "Writing B" arduino.flush() time.sleep(7)

print "Remember the first time you failed that class ... digital circuit with Mark Faust oh yeah!"

arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

print "Strumming ..." arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

of you

print "Thank you all

we're gonna miss..." arduino.write('B') print "Writing B" time.sleep(7)

for being you now let's go to

print "We couldn't be anymore proud, to have

Lendaris here with us ...Tymerski, Teuscher, Sutherland ... yadda yadda having fun oh yeah!"

arduino.write('A') print "Writing A" time.sleep(7) arduino.write('A') time.sleep(1) arduino.write('B') time.sleep(7) arduino.flush()

print "Strumming ..." arduino.write('B') print "Writing B" arduino.flush() time.sleep(7)

scream

print "Some of you ...makes

WHY WHY!"

arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

you pull

your hair and

print "Strumming ..."

arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

print "with the help

arduino.write('C') print "Writing C" arduino.flush() time.sleep(7)

of the

lovely staff ..."

for being a part of this all

print "Thank you all the barbecue right there"

arduino.write('B') time.sleep(1)

arduino.write('A')

time.sleep(7) arduino.write('A') time.sleep(10) arduino.flush() arduino.close()

vlc.kill()

# kill the vlc subprocess

print

except:

print "Failed to send!"

# Appendix D – Program of 2nd Arduino to control Halloween robots in Act 4

halloween1 = 11;

halloween2 = 13; msg = ' ';

setup() {

initialize the digital pin as an output. Pin 13 has an LED connected on

most

Arduino

boards:

\*Note: I apologize for the dirty code here.

int

int int int int int char void

//

//

bear1Pin1

bear1Pin2 witchPin1 witchPin2

= 8;

= 9;

= 2;

= 3;

/\*

Controls the dancing puppets

\*/

pinMode(bear1Pin1, OUTPUT);

pinMode(bear1Pin2, OUTPUT); pinMode(witchPin1, OUTPUT); pinMode(witchPin2, OUTPUT); pinMode(halloween1, OUTPUT); pinMode(halloween2, OUTPUT); digitalWrite(halloween1, HIGH); digitalWrite(halloween2, HIGH); Serial.begin(9600); Serial.print("Program init!\n");

}

void loop() {

while (Serial.available() > 0) { msg = Serial.read(); Serial.println(msg);

}

if (msg == 'A') {

//halloweenToggle(halloween1); Serial.println("halloween1 toggled!");

halloweenToggle(halloween1);

//delay(6000);

//halloweenToggle(halloween1);

//delay(6000);

/\* original code

digitalWrite(13, delay(2000); digitalWrite(13,

LOW);

// set the LED on

HIGH);

Serial.println("pin 13

delay(5000);

// wait

// set high");

// wait

for

the

a second

LED off

for

a

digitalWrite(13,

delay(2000); digitalWrite(13, delay(2000);

LOW);

// set the LED

// wait for a

HIGH);

// set the

LED

second

on sec

on

digitalWrite(9,

delay(5000); digitalWrite(9, delay(5000); digitalWrite(8, delay(2000); digitalWrite(8, delay(2000);

\*/

LOW); // Bear

on

HIGH);

LOW); // Bear

off

HIGH);

} else if (msg == 'B') {

halloweenToggle(halloween2); Serial.print("halloween2 toggled!\n");

} else if (msg == 'C') { halloweenToggle(halloween1); Serial.print("halloween1 toggled!\n"); delay(500); halloweenToggle(halloween2); Serial.print("halloween2 toggled!\n");

}

//Serial.println("hello");

//witchOn();

//delay(5000);

//witchOff();

}

void bear1On() { digitalWrite(bear1Pin1, LOW); digitalWrite(bear1Pin2, HIGH); delay(1000);

bear1Idle();

}

void bear1Off() {

digitalWrite(bear1Pin1, HIGH); digitalWrite(bear1Pin2, LOW); delay(1000);

bear1Idle();

}

void bear1Idle() { digitalWrite(bear1Pin1, HIGH); digitalWrite(bear1Pin2, HIGH);

}

void witchOn() { digitalWrite(witchPin1, LOW);

}

void witchOff() { digitalWrite(witchPin1, HIGH);

}

void halloweenToggle(int id) { digitalWrite(id, HIGH); delay(500);

digitalWrite(id, LOW);

}

Latest update by Waleed Alhaddad and Saad Alaskar on 6/13/16

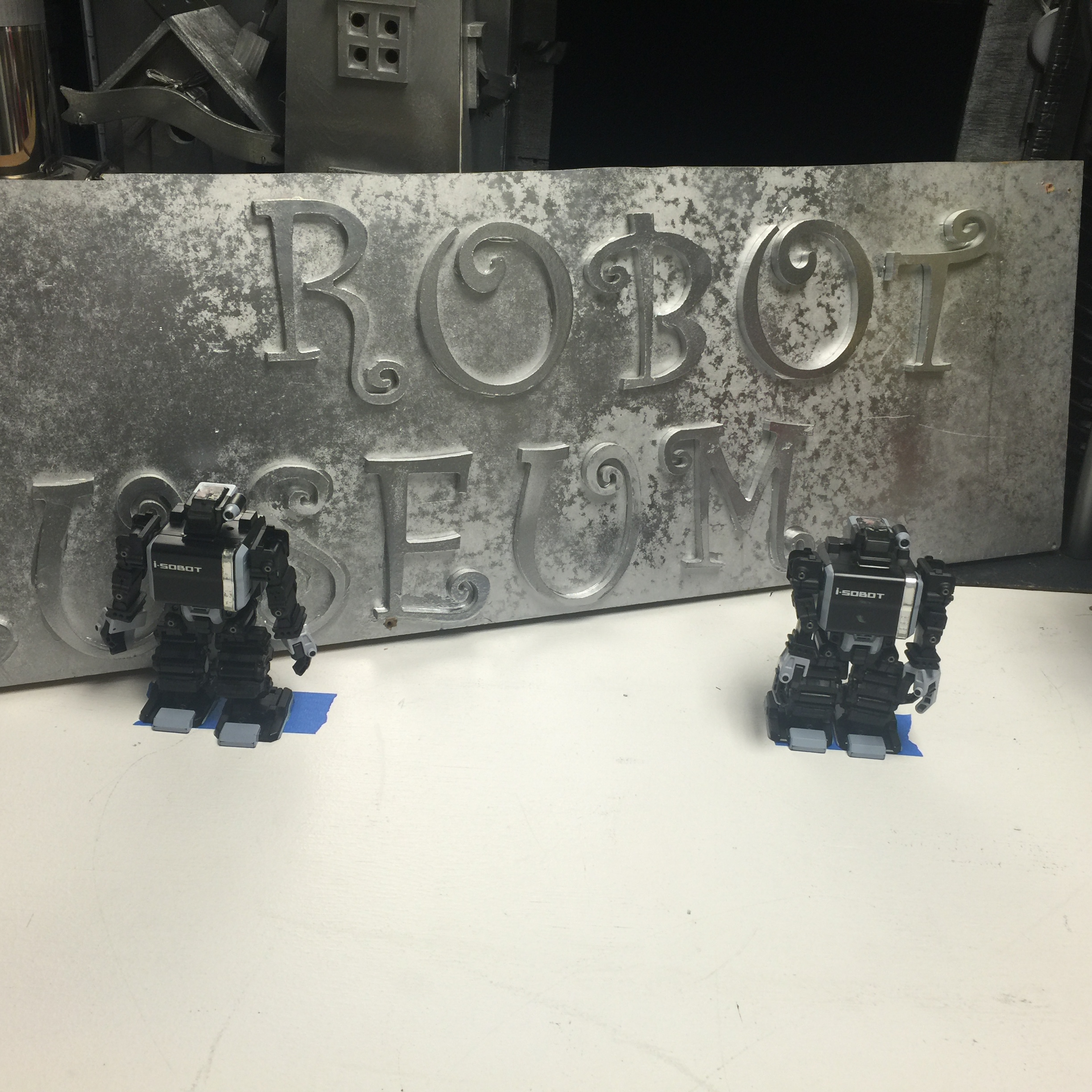
On week 9, Waleed and Saad were assigned to the Isobot robot where we were assigned to re-demo the isobot. We had to go through a bunch of previous term documentation for Isobot in order to figure out how to control these robots and how they do work. We picked Mathias Sunardi report as it was the clearest report to us and also he used python to control the robot which we were familiar with it. Reading through the documentation probably took us around 10 hours of time in order to find, read through all the documentation we found, and understand the code that was in the robots. We faced big obstacle where both documentations didn’t have proper links or the writer didn’t upload all the codes in the report to the github also the two reports used two different methods which created a big confusion. We had to rewrite all the codes that we needed to be able to use it as we cannot simply copy it from the report as the layout of the code will be different and that consumed us a lot of time and a lot of debugging.

After getting all the codes we needed, we found out that one of the IR-emitters was not working and the other was having a weak signal, but it was enough to test the robots with. So we followed Mathias report steps installing the necessary programs and everything worked fine, so we proceeded to install it on the stage and test both ‘Isobot’ robots and we got to work successful. One IR-emitter is not enough to cover the whole stage, so we needed to fix the other IR-emitter that has not been fixed in a long time. After researching we found out that the IR sensor is broken so we removed the broken one and we soldered a new one and it has fixed the problem. After fixing the broken IR-emitter we attached it to the other side of the first emitter so it can cover a big area of the stage. We also labeled the best area that the emitter can cover with a blue tape.

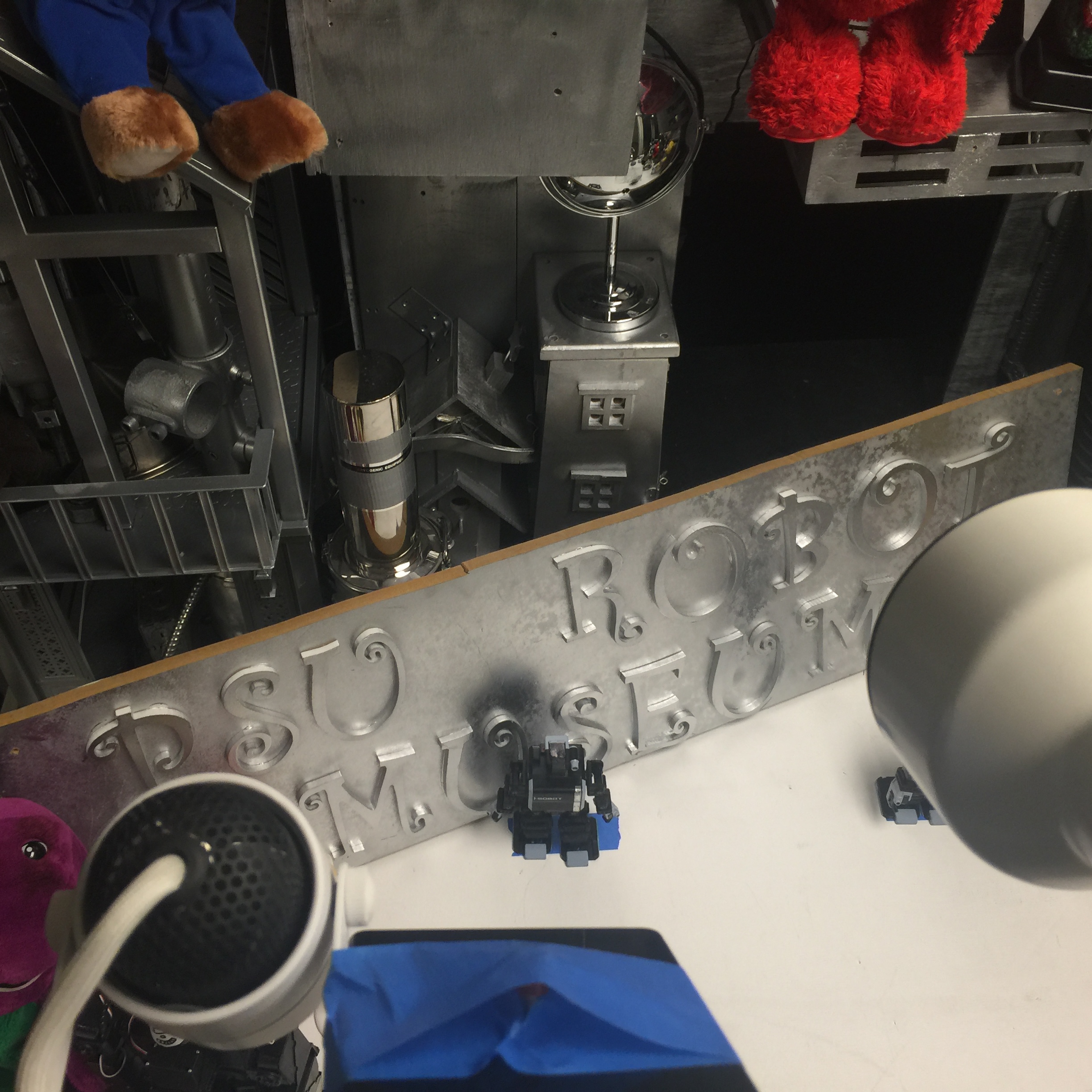
**Pictures of the stage after installing everything**



Overview of what is the initial positioning of the two IR-emitter along with the robots



The two “Isobot robots” under the blue tape that shows the area that the IR-emitter can cover (send signal to).



The angle of the IR-emitter scoping on the first (A) “Isobot” robot



The angle of the IR-emitter scoping on the second (B) “Isobot” robot

Future Work:

1. Create a complete script for the theater for the two robots to interact with each other.
2. Install speakers outside the lab so people outside the lab can listen to the conversation going between the two robots.
3. Use a better command sender method such as Bluetooth so robots position will not be restricted and can be controlled from a distance.
4. Use a better power source since the one in there doesn’t last the robots too long.

Resources:

I have created a github and uploaded to it the Arduino code(.ino) and python code (.py)

<https://github.com/waleedalhaddad/Isobot-2016/>