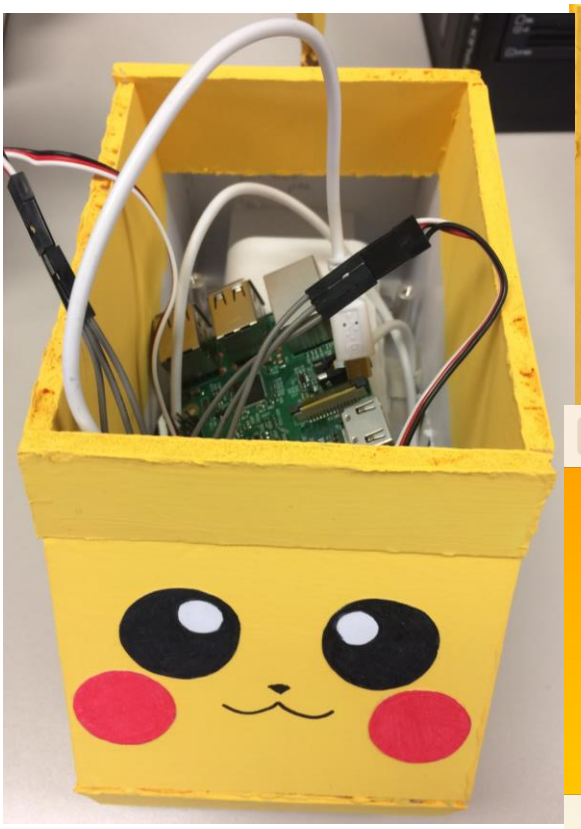
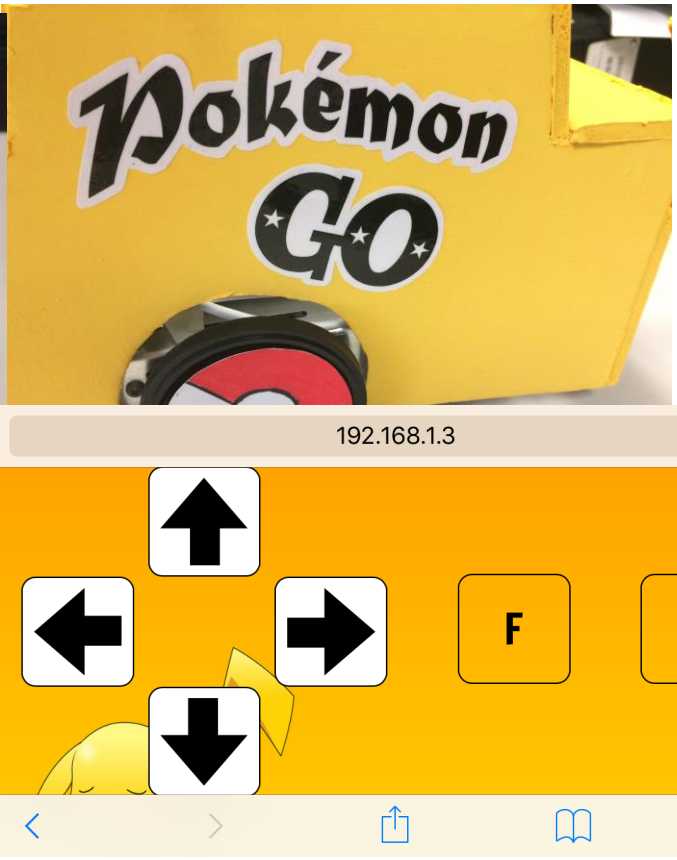
COMPUTER ENGINEERING FINAL PROJECT:

Pokémon Go – Pokémon Themed Boe-Bot Controlled Through a Raspberry Pi UI Game Controller





Kelly Jia

Mr. Muttiah

TEJ4M

January 13, 2016

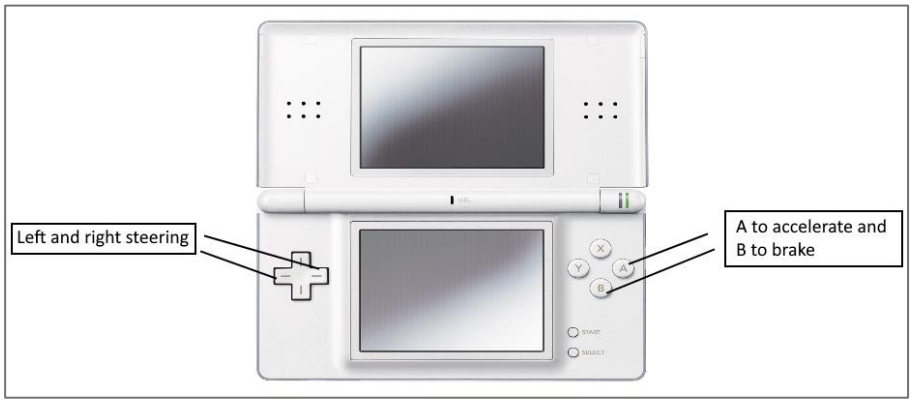
Abstract

The aim of this project was to create a sort of game controller to maneuver the Parallax Boe-Bot using the Raspberry Pi's networking capabilities. Incorporating a Pokémon theme, the Boe-Bot was renamed the "Pokémon Go," referencing one of the most popular mobile apps of 2016. The name was recommended by fellow student Si Yu Zhuang. Research concerning servo motors and programming the movement of the Boe-Bot using Python was conducted, as the Boe-Bot was previously programmed in Basic Stamp. Regarding the controller, a physical controller using a breadboard and push buttons was going to be constructed initially. However, through the teacher's guidance and suggestions, a web-based interface was created using *WebIOPi,* a web application which allows one to debug and control a Raspberry Pi's GPIO pins from a web browser. WebIOPi also included JavaScript, Python, and CSS libraries to control the GPIO pins in those languages. Creating the user interface proved to be the most difficult task of the project as HTML, CSS, JavaScript, and Python code had to be written and integrated into one another to form the application. Also, the different functions to control the GPIO pins using the WebIOPi libraries had to be further researched. In addition, there were many network difficulties regarding the web application that had to be overcome. Ultimately, a successful web application that contained buttons to control the navigation of the Pokémon Go was created.

Introduction

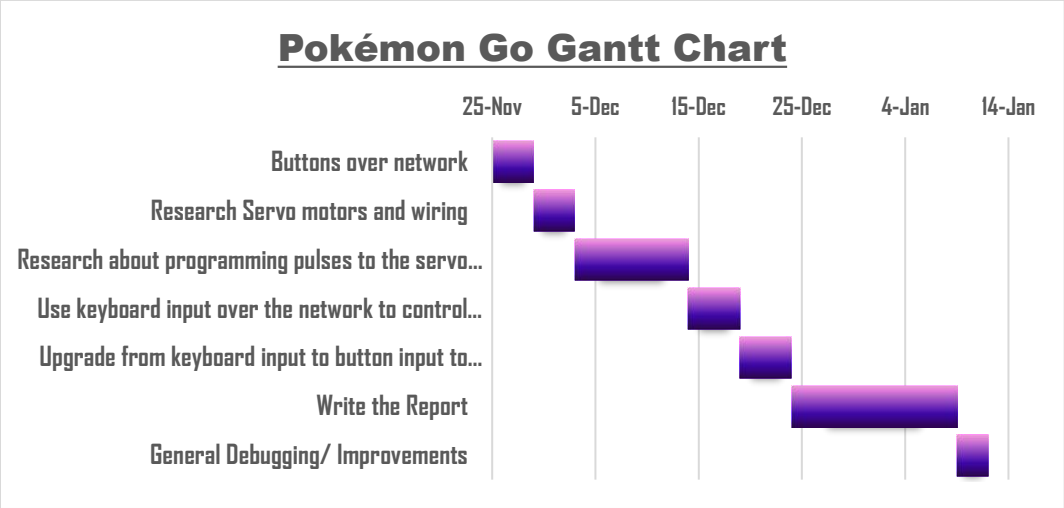
In this project, the number of potential ideas for the final piece is essentially endless. Anything from Tic-Tac-Toe to a security camera to battleship could be made using the Raspberry Pi. However, a project incorporating the Boe-Bot was chosen as there was already previous experience with the Bot-Bot from a previous unit. Furthermore, the Boe-Bot is not too excessive for a one-person project but at the same time, is challenging enough to be used for a culminating project. The button states would be sent over the hotspot to the Raspberry Pi connected to the Boe-Bot to determine its correct maneuvers. The final idea was then to create some sort of game controller with buttons to maneuver the Boe-Bot around. The idea was inspired by Mario Kart, the popular racing game by Nintendo featuring the main characters of the Mario video game franchise. However, instead of Mario, a theme of Pokémon was chosen. The layout of the controller would also be based upon the Nintendo DS controller layout, with 2 main steering buttons (left and right), 1 accelerate/gas button, and 1 brake/reverse button.

A screenshot from Mario Kart 8 on the Wii U



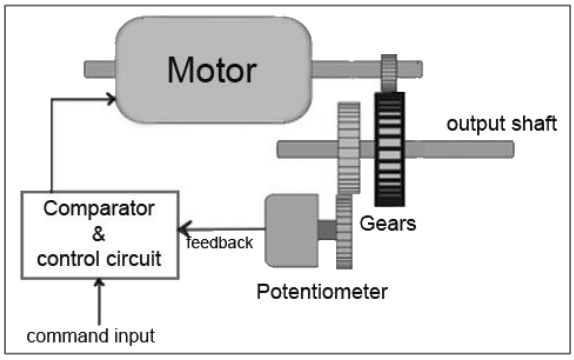
The layout of the Nintendo DS Lite controller with the basic controls used to play Mario Kart

A Gantt chart was used to plan out the whole project by splitting the project into individual tasks with approximate durations/ deadlines of each task. The Gantt chart was generated using the idea that a physical game controller with buttons on the breadboard was going to be created, instead of a web application. However, the dates for the game controller can simply be translated into dates for work on the WebIOPi application.



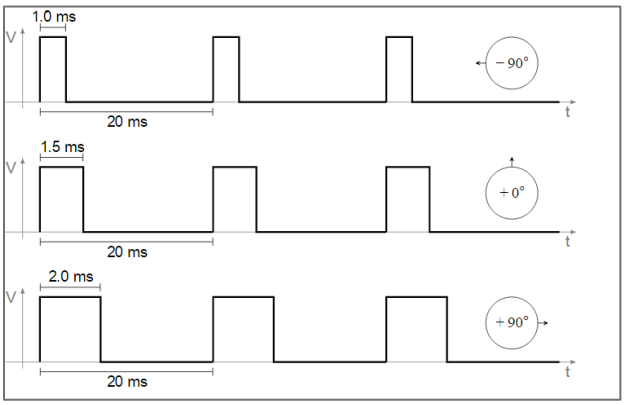
Project Research

Lots of research had to be done in order to complete the project. The first component of research involved investigating how the general servo motor worked, in comparison to the standard electrical motor. This was done as the motors on the Boe-Bot were servo motors. In addition to robotics, it was found that servo motors are most commonly used in computers, CD/DVD players, and toys. The servo motor can precisely control speed and position, by only controlling the position. The physical position of the output shaft is determined by the potentiometer, also known as the "pot". The power supplied to the motor is dependent upon the required position of the motor shaft. If the shaft is at the required position, then power supply stops. If not, then the motor is rotated in the appropriate direction to reach the position. This correct position is sent to the motor by electrical pulses in the control wire. The speed of the motor is in *proportional control*. This means that the motor will run fast when it is far from its required position and run slowly when it is near it.



The components of a servo motor

To control the servo motors, electrical pulses of changing widths (pulse width modulation – PWM) were sent to the motors through a control wire. Generally, a servo motor can turn 90° in each direction, for a maximum of 180°. The motor is in its "neutral position" when the motor is at its 90° position (halfway between 0° and 180°). Some sources will call the neutral position 0°, and have -90° and +90° as the starting and ending positions of the servo. A pulse of 1.5ms (milliseconds) is sent to the motor to obtain this neutral position. To obtain a position of -90°, a pulse of 1ms is sent, and to obtain a position of +90°, a pulse of 2ms is sent. A pulse is sent to the servo motor every 20ms. These pulses can be illustrated in the following diagram:



The different pulses to obtain the 3 main positions of the servo motor

When connecting the Boe-Bot servos to the Parallax microcontroller, there were set spots on the microcontroller for the servos to be connected to. This is not the case with the Raspberry Pi. To connect the servo motors to the Raspberry Pi, Figure 2-13 of the Parallax Boe-Bot Manual was used:

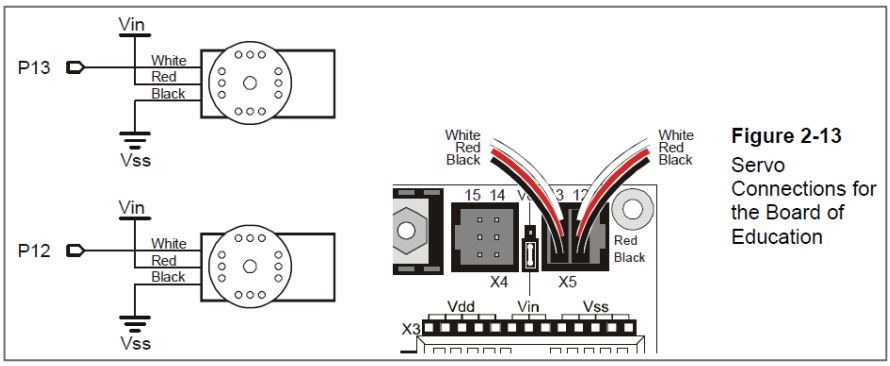
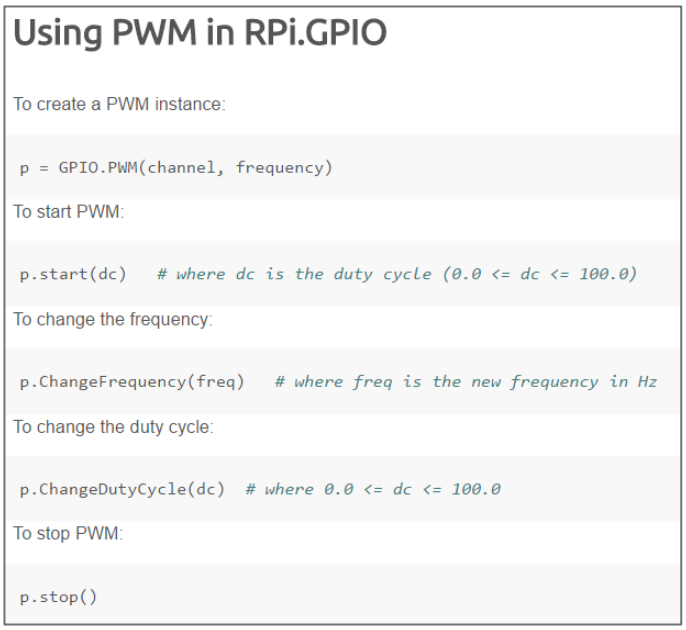


Figure 2-13
Servo Connections for the Board of Education

Each wire of the servo motor had to be physically wired to the pins of the Raspberry Pi. In the manual, pin 13 was connected to the left wheel and pin 12 was connected to the right wheel. However, to connect to the pins on the Raspberry Pi, the pin had to be able to support PWM (pulse width modulation) functionality. Through reading the official Raspberry Pi forums and the Raspberry Pi Stack Exchange, it was said that pins 12 and 18 on the pi supported PWM. To test these pins, a simple program using PWM to brighten and dim a led was taken. The program was by Ben Croston on Sourceforge. The forum also contained code on how to create a PWM instance, and set and change the duty cycles and frequencies.



The Python code to write PWM instances in RPi.GPIO

```python
import time
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BOARD)
GPIO.setup(12, GPIO.OUT)

p = GPIO.PWM(12, 50)  # channel=12 frequency=50Hz
p.start(0)
try:
    while 1:
        for dc in range(0, 101, 5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
        for dc in range(100, -1, -5):
            p.ChangeDutyCycle(dc)
            time.sleep(0.1)
except KeyboardInterrupt:
    pass
p.stop()
GPIO.cleanup()
```

A simple Python program to brighten and dim a led

The program successfully brightened and dimmed a red led on both pins 12 and 18. However, the program also worked on pin 2, a pin that was not supposed to support PWM. Therefore, it was concluded that on the new model of the Raspberry Pi, all the GPIO pins had PWM functionality. Thus, to connect the servo motors of each wheel to the Raspberry Pi, the left wheel was connected to pin 2 and the right wheel was connected to pin 3. These were connected using the white wires of the servo motor (the control wires). After, the 2 red wires were connected to the 5V pins, while the 2 black wires were connected to the ground pins on the Raspberry Pi.

On the other hand, to program the pulses to the servo motors in Python, duty cycles/ ratios and frequency was used rather than angles. Duty cycles are either measured in percentage (0% - 100%), or as a decimal (0.0 – 1.0). A duty cycle describes the percentage of time that the signal is on (high) over a period of time. For example, if the signal spends half its time on and half its time off, then the duty cycle would be 50% or 0.5. Therefore, to find the duty cycles and frequencies needed to program the left and right wheels of the bot, the length of each high pulse and the total length of the pulse (period) needed to be calculated. To do this, calculations from the Boe-Bot Manual were utilized. To move the bot straight forward, both the left and right wheels had to be turning in their appropriate directions at full speed. The left wheel turns counter clockwise while the right wheel turns clockwise. To program the bot, the manual uses a programming language called Basic Stamp, which uses a PULSOUT command with a PAUSE inside a loop to send the pulses to the servo motor. For example, to make the right wheel run at full speed, the manual uses the code:

DO
    PULSOUT 12, 650
    PAUSE 20
LOOP

To figure out the length of the pulse (how long each pulse is high), the manual states on page 50, *"... whatever number is in the PULSOUT command's Duration argument, multiply that number by 2μs (2 millionths of a second = 0.000002s), and you will know how long the pulse will last."* In addition, the manual uses 20ms (0.02s) as the pause, which is the time that the signal is off.

Thus, the following calculation was used to find the length of the pulse, the duty cycle, and the frequency for turning the **right** wheel forward at full speed:

Length of pulse = Duration Argument × 2 μs
= 650 × 2 μs
= 650 × 0.000002s
= 0.0013s

Length of one cycle = Length of pulse + Length of pause
(Period) *= 0.0013s + 0.02s*
= 0.0213s

Duty cycle = Length of pulse / Length of one cycle
= 0.0013s / 0.0213s
= 0.061
= 6.1%

Frequency = 1 / Period
= 1 / 0.0213s
= 46.9 Hz

The same calculations were done to find the length of the pulse, the duty cycle, and the frequency for turning the **left** wheel forward at full speed. However, instead of PULSOUT 12, 650, PULSOUT 13, 850 is used instead.

Length of pulse = 0.0017s

Length of one cycle = 0.0217s

Duty cycle = 0.078 (7.8%)

Frequency = 46.1

In addition, to obtain a full stop with the wheels (the motor is at neutral position), a pulse of 1.5ms with a pause of 2.0ms is needed; indicated in the duty cycle research and the manual on page 49. Using the same previous formulas, the duty cycle for a stopped wheel was calculated to be 0.07 (7%). This means to slow down the right wheel, the duty cycle would be increasing from 6.1% and going towards or past 7% (going past means the wheel would turn in the opposite direction). Simultaneously, to slow down the left wheel, the duty cycle would be decreasing from 7.8% and going towards or past 7%. Thus, with all the calculations finished, they were then programmed into Python to test the bot and see if the calculations were accurate or if

they need adjusting. The code was based on the previous program of brightening and dimming a led. It was tested to see if firstly, the wheels moved at all, and secondly, if the bot moved in the right directions. Ultimately, the actual development of the project was begun.

Product Development

```
import RPi.GPIO as GPIO
import time

"""
pin 3 - right wheel
pin 2 - left wheel
"""
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(3,GPIO.OUT)
GPIO.setup(2,GPIO.OUT)
right=GPIO.PWM(3,46)
left=GPIO.PWM(2,46)

left.start(7.8)
right.start(6.1)
time.sleep(4)

right.stop()
left.stop()
GPIO.cleanup()
```

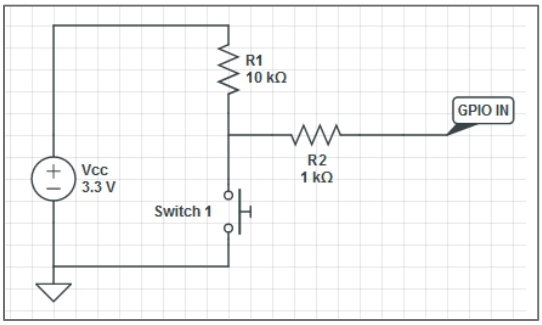
The test program was written using the previous calculations of the duty cycles and frequencies of the left and right wheels. “right = GPIO.PWM (3,46)” and “left = GPIO.PWM (2,46)” creates two PWM instances, one for each wheel, with their corresponding pin number and frequency. In the calculations, the right wheel had a frequency of 46.9 while the left wheel had a frequency of 46.1. For simplicity’s sake, the frequency in the program was set to 46 for both and only the duty cycle would be adjusted. During the actual testing, both wheels successfully started running, but it seemed that the left wheel was more powerful and the bot was moving diagonally towards the right instead of straight ahead. This indicated a hardware issue with the servo motors, as the motors for each wheel were not equal in power. Instead of switching the servo motors, software was used to compensate. The duty cycle of the left wheel was lowered by 0.1 at a time and retested until the bot ran relatively straight ahead. The bot finally ran straight forward when the duty cycle of the left motor was **7.2**.

The next part was to figure out turns with the bot. Instead of the specific 90° turns or 45° turns that were coded in the previous units, a mere slight left and right turn was desired. This was because in the final product, the turning is going to be controlled by a button; so as the button is held down, the bot turns only slightly for each cycle. To turn the bot to the left, it was learned in the previous unit to set the left wheel to turn at full speed in the opposite direction while the right wheel stays the same; vice versa for turning to the right. However, as only a minor turn is needed in this project, the wheel is only going to be slowed down a little instead of turning full speed in the opposite direction. To do this, the duty cycles are only going to be adjusted slightly and then tested. To start off, to gain a right turn, the left wheel stays the same while the right wheel is slowed down. This means the duty cycle of the right servo motor is going to be a bit higher than its original – 6.1. After testing, a duty cycle of **6.5** succeeded. Similarly, for a left turn, the right wheel stays the same while the left wheel is slowed down, so the duty cycle of the left motor will be smaller than 7.2. A duty cycle of **6.75** worked well after several trials.

```
while True:
    if (GPIO.input(14) == False):
        left.ChangeDutyCycle(6.75)
        right.ChangeDutyCycle(6.1)
    elif (GPIO.input(24)==False):
        left.ChangeDutyCycle(7.2)
        right.ChangeDutyCycle(6.5)
    else:
        left.ChangeDutyCycle(7.2)
        right.ChangeDutyCycle(6.1)
```

Following this, the next step of the project was to be able to control the turns with push buttons. There would be 2 buttons – one to turn left and one to turn right, and when no button is pressed, the bot would simply go straight. The following while loop was written to do this, with pins 14 and 24 connected to the buttons. However, when the buttons were connected and the program was tested, the buttons didn’t seem to be working. After some teacher guidance, it was discovered that the

buttons weren't working because they were connected without a resistor. In previous assignments where buttons were involved, they seemed to work without the use of a resistor. However, in this case, they did not. Hence, a button wiring diagram was followed to properly connect the buttons. After the wiring was done, the bot was successfully turning with the buttons.



The wiring diagram that was used to properly wire the push button

As the Boe-Bot component of the project was complete and successful, the next step was to create the game controller to control the bot. The original plan was to use a bread board and push buttons to create a makeshift game controller. However, as there still remained substantial time, a web-based application was suggested to make the controller so the bot would be controlled using a mobile phone. An application called *PiUi* was suggested, which allowed one to add a user interface to any pi project using an Android or iOS phone. Unfortunately, after further research, it was stated that PiUi stopped development approximately 3 years ago and it may not be supported on the newer models of the Raspberry Pi. Because of this, another web application had to be found to support UI with the Raspberry Pi, and *WebIOPi* was discovered. Along with this, an extremely thorough and helpful tutorial was found that contains step-by-step instructions on setting up one's first WebIOPi application; from the downloading of WebIOPi onto the Raspberry Pi to the HTML, CSS, and JavaScript code itself. The tutorial was posted on the Connectedly forum by Jerry Hildenbrand.

First and foremost, in the Pi's command line, *sudo apt-get update* and *sudo apt-get upgrade* was run to ensure the Raspberry Pi contained all the newest versions of the installed packages. Following that, *sudo apt-get install python-dev* and *sudo apt-get install python-rpi.gpio* were called to make sure that Python on the Raspberry Pi had all the necessary updates to be able to fully communicate with the GPIO pins. Next, the actual WebIOPi (Version 0.7.1) package was downloaded and installed using the following commands:

```
$ wget http://sourceforge.net/projects/webiopi/files/WebIOPi-0.7.1.tar.gz
$ tar xvzf WebIOPi-0.7.1.tar.gz
$ cd WebIOPi-0.7.1
$ sudo ./setup.sh
```

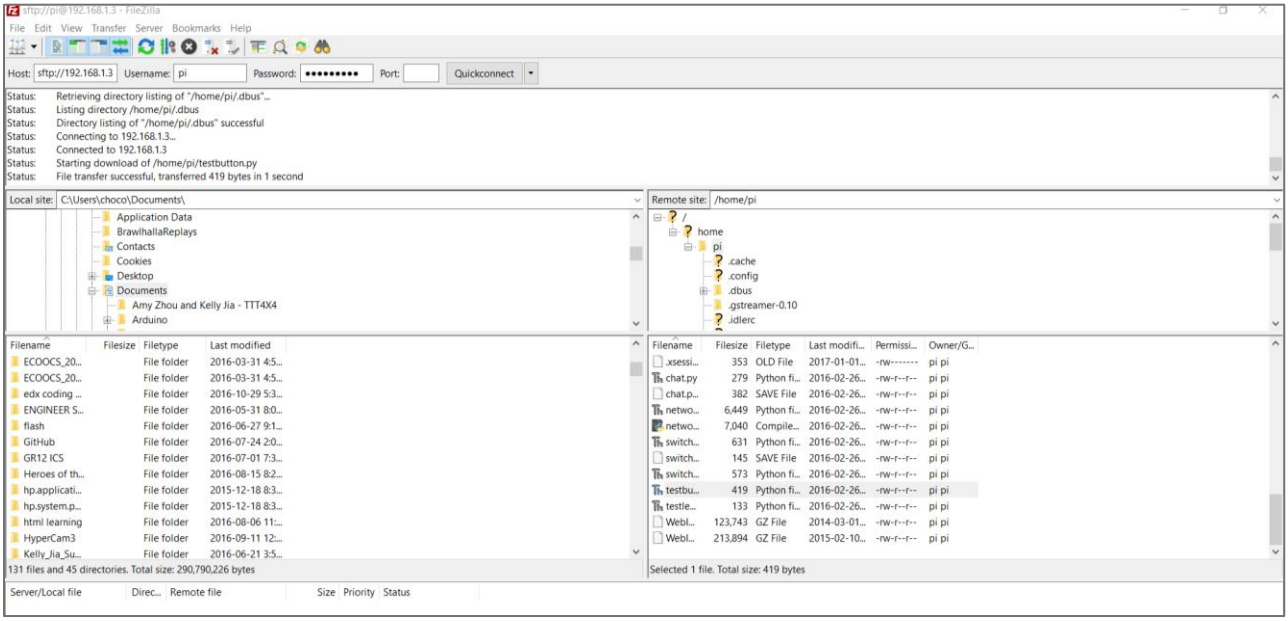
The next step was to test the application by using the command *sudo webiopi -d -c /etc/webiopi/config.* This ran the application with debugging enabled in the console, so if any programs had any errors, it would show up in the command terminal. As no errors were found, this meant that WebIOPi was up and running. After making sure both the phone and Raspberry Pi was connected to the Raspberry Pi Hotspot, http://192.168.1.3:8000 (the IP of the Raspberry Pi and port 8000) was entered into the browser on the

phone, and the default WebIOPi username and password (webiopi, raspberry) was used. The interface successfully opened and the GPIO Header was displayed showing all the pins of the Raspberry Pi and their input/output state. According to the tutorial, one was supposed to be able to click on the "IN" to turn that GPIO into an "OUT" output pin. In addition, clicking each pin on the diagram should have turned it yellow, to make it high. Unfortunately, none of those things worked and clicking any of the pins had no effect whatsoever. Then, it was noticed that this layout was the older 26 pin layout of the Raspberry Pi, and not the 40 pin model that is being used in class. Because of this, it was not compatible with the Raspberry Pi 3. This resulted in additional research to determine if there have been any updated versions to accommodate the 40 pin model. Fortunately, there were many others who were seeking an update for WebIOPi and on Google Forums, a link was posted to a patch on Github to obtain the 40 pin model. The patch was posted by Keisuke Seya, "doublebind," with instructions on how to download the patch on the Pi. The following commands were needed:

$ wget https://raw.githubusercontent.com/doublebind/raspi/master/webiopi-pi2bplus.patch
$ patch -p1 -i webiopi-pi2bplus.patch
$ sudo ./setup.sh

After the patch was installed, the web application was retested and it was successful. It was able to turn GPIO 2 (the one that was connected to the led) into an output pin, and then turn the pin on to light up the led. This confirmed the testing of the WebIOPi service, and the next/ final step was to create the actual web application incorporating the WebIOPi libraries.

Before writing the code for the web application, a program called *Filezilla* was installed to transfer code files written on the laptop to the Raspberry Pi. It a recommended program from the tutorial. In the Filezilla client, one is able to transfer files in SFTP (SSH File Transfer Protocol) to easily browse and edit files on the Raspberry Pi. As the Raspberry does not have any text editors that support HTML, CSS, or JavaScript, it becomes difficult to write code on it as a standard Notepad-like text editor must be used. There are no color schemes or debug features to aid with writing the code. This was why Filezilla was chosen to be used – as the code could be written on the laptop in Notepad++ and easily transferred to the project folder on the Raspberry Pi. To connect to the Pi in Filezilla, one has to enter "sftp://YOUR.PI.IP.ADDRESS", the username and password of the pi (pi, raspberry was the default), and port 22.

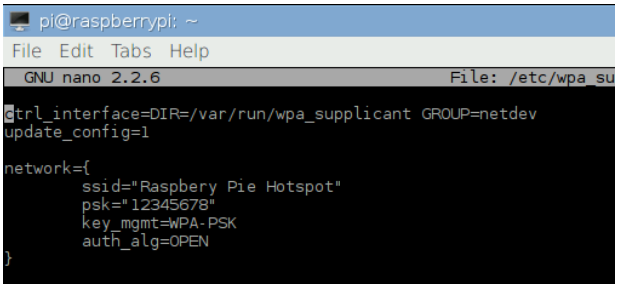


A screenshot of the Filezilla Client in action. The laptop files are on the left while the pi's files are on the right

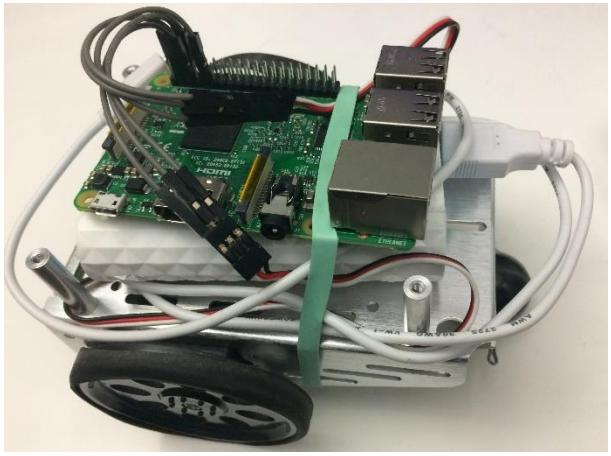
Before the sample code was copied over from the tutorial, research was done on how to launch the WebIOPi application and also have the Raspberry Pi automatically connect to the Raspberry Pi Hotspot at start up. This is because as the bot is a mobile project, it would be extremely inefficient to have to connect the Raspberry Pi to a monitor to connect to the network and run the program in the command terminal every time. To have the Raspberry Pi automatically connect to the network, the network interfaces file in the pi had to be edited. To enter the file, *sudo nano /etc/network/interfaces* was entered into the terminal. When in the file, *auto wlan0* was added to the top of the file. This tells the pi to automatically connect to the wireless LAN at start-up. After this line as appended, the following lines were added to the bottom of the file:

allow-hotplug wlan0
iface wlan0 inet dhcp
wpa-conf /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet dhcp

This tells the Raspberry Pi to allow wlan0 as a network connection and to use */etc/wpa_supplicant/wpa_supplicant.conf* as the configuration file. In the configuration file, a section about the Raspberry Pi Hotspot was added so the Pi could automatically connect to the network.

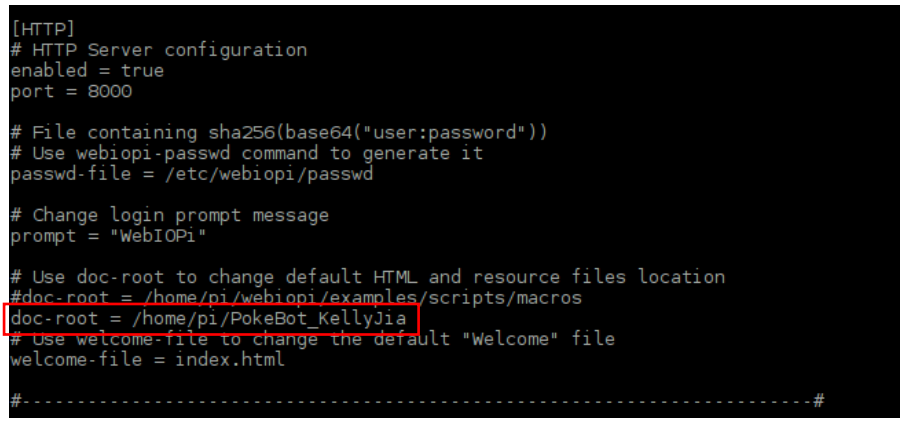


To make the WebIOPi application also run on start-up, *sudo update-rc.d webiopi defaults* was entered into the command terminal. This updates the rc file in the pi to automatically run WebIOPi when it is powered on. Also, since a portable power bank was going to be used to power the bot as it maneuvered around, the WebIOPi application would be launched and the network would be connected to when the power bank is connected. This makes the bot fully mobile, as it does not have to be connected to the monitor to run the application and it does not need to be connected to an outlet. An elastic band was wrapped around the pi and the power bank to hold it all together.



The Raspberry Pi and the power bank assembled and secured on top of the Boe-Bot

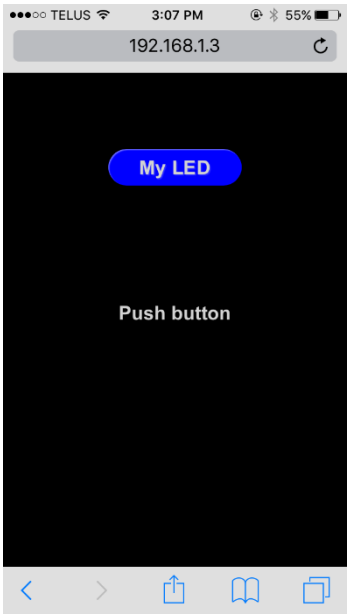
The sample HTML, CSS, and JavaScript code was then copied over from the tutorial and saved in a project folder on the pi. To tell the WebIOPi server to run the code from the project folder instead of the default webpage, the code in the WebIOPi configuration file had to be updated. *sudo nano /etc/webiopi/config* was entered in the command terminal, and in the HTTP section of the file, the doc-root was changed so it pointed to the HTML file in the project folder.



"doc-root = /home/pi/PokeBot_KellyJia" was appended so WebIOPi could run the index.html file written in that folder

In addition, for convenience in the testing stage, a username and password for the WebIOPi webpage was disabled. It made it a lot easier and faster to test updates in the webpage without having to enter a username and password every time the page is refreshed. To do this, *sudo mv/ etc/webiopi/passwd/etc/ webiopi / passwdOLD* was entered in the command terminal.

The sample program had 1 button on the webpage that lit up a led when it was pressed down. It used a WebIOPi JavaScript function "createGPIOButton(gpio, label)" which creates a button that changes the state of the assigned GPIO at each click. The WebIOPi CSS library was then used to change the color of the button depending on the state of the GPIO pin (high or low). This sample program successfully worked to light up a red led!



A screenshot of the sample program on the phone

However, this project involves the use of PWM, and not simple high/low GPIO states. Since python was used to control the PWM pins and not JavaScript, research had to be done on how to call a python function from either HTML or JavaScript. Luckily, WebIOPi contains something called *macros*, which are custom Python functions that can be called from a JavaScript function. To declare a WebIOPi macro in Python, one only needs to put *@webiopi.macro* before declaring the function. However, to use a WebIOPi macro, one must retrieve the GPIO library from WebIOPi by importing the WebIOPi library using *import webiopi* and then *GPIO = webiopi.GPIO*. However, the WebIOPi GPIO's had slightly different syntax than what was used before. For example, to declare a PWM pin before, the following code was used: (All the code from the WebIOPi libraries was referenced from the tutorial and documentation pages of the WebIOPi website).

GPIO.setup(2,GPIO.OUT)
pin = GPIO.PWM(2,50) #pin2, frequency 50
pin.start(7.2) #duty cycle 7.2%

Using the WebIOPi GPIO, the following code was used to declare a PWM pin:

GPIO.setFunction(2,GPIO.PWM) #pin2, automatic frequency 50
GPIO.pwmWrite(2,0.072) #duty cycle of 0.072 (uses decimal and not percent)

And then to call the macro from JavaScript, the following code is used:
webiopi().callMacro("nameOfMacro");

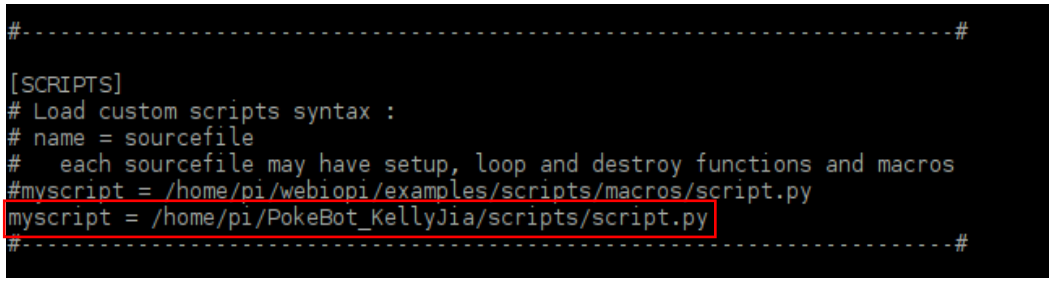
Using the Python code earlier to control the Boe-Bot, the syntax and values were converted to use the WebIOPi GPIO library, and functions were created to move the bot forward, forward left, forward right, backwards, backwards left, backwards right, and a full stop.

```
@webiopi.macro
def leftF():
    GPIO.pwmWrite(leftBtn,0.07)
    GPIO.pwmWrite(rightBtn,0.05)
    time.sleep(0.1)
    webiopi.debug("left")

@webiopi.macro
def rightF():
    GPIO.pwmWrite(leftBtn,0.1)
    GPIO.pwmWrite(rightBtn,0.072)
    time.sleep(0.1)
    webiopi.debug("right")

@webiopi.macro
def leftB():
    GPIO.pwmWrite(leftBtn,0.07)
    GPIO.pwmWrite(rightBtn,0.1)
    time.sleep(0.1)
    webiopi.debug("left")

@webiopi.macro
def rightB():
    GPIO.pwmWrite(leftBtn,0.061)
    GPIO.pwmWrite(rightBtn,0.072)
    time.sleep(0.1)
    webiopi.debug("right")
```

```
@webiopi.macro
def forward():
    GPIO.pwmWrite(leftBtn,0.1)
    GPIO.pwmWrite(rightBtn,0.061)
    time.sleep(0.1)
    webiopi.debug("forward")

@webiopi.macro
def backward():
    GPIO.pwmWrite(leftBtn,0.061)
    GPIO.pwmWrite(rightBtn,0.1)
    time.sleep(0.1)
    webiopi.debug("backward")

@webiopi.macro
def stop():
    GPIO.pwmWrite(leftBtn,0)
    GPIO.pwmWrite(rightBtn,0)
    time.sleep(0.1)
    webiopi.debug("stop")
```

webiopi.debug() allows one to see which function is running by outputting a phrase into the command terminal during execution. The following image shows what this looks like. The debug function was useful to see if the macro was even being executed in the first place, and to see which specific macro is currently running for debugging purposes. In addition, as the WebIOPi GPIO library uses decimals instead of a

percentage for the duty cycle ratio, the duty cycle percentages previously used was multiplied by 0.01. However, some of the new values did not work, most likely to the default 50 frequency of the WebIOPi PWM pin. This was why multiple trials of testing had to be done to find new values to make the bot go forward and turn properly. Furthermore, to tell WebIOPi to run the Python script along the HTML and JavaScript code, the configuration file had to be edited; similarly to changing the doc-root. The following line was appended into the SCRIPTS section of the configuration file:



"myscript = /home/pi/PokeBot_KellyJia/scripts/script.py" was added to load the custom script when WebIOPi is started up

After coding the Python macros, the next step was to create buttons that will call the specific corresponding function. The WebIOPi button function was used, which created a simple button that calls a function on a mouse click event, and calls another function on a mouse release event. To do this, the following code was written inside the webiopi().ready function.

```
var forwardBtn;

//Button with an id of "forward", a label of "Go Forward"
//Calls the function forward() on a mouse click event, and stop() on a mouse release event
forwardBtn = webiopi().createButton("forward","Go Forward", forward, stop)

//Outside of the webiopi.ready() function:
function forward(){
        webiopi().callMacro("forward") //calls the macro named "forward" in the Python script
}

function stop(){
        webiopi().callMacro("stop") //calls the macro named "stop" in the Python script
}
```

This button worked when it was tested on the laptop, but was found to malfunction when tested using the phone. When the button was pressed on the phone, the button would either delay or not be read as pressed at all. Through further research, it was found that the button would not work correctly on the phone because on touch screen devices (such as the iPhone), mouse click events would not work as technically, there is no

mouse cursor. Instead, touch events must be used to detect a finger's "touches" on the screen. Therefore, instead of coding mouse click and mouse release events, touchstart (like mouse click) and touchend (like mouse release) events had to be written. However, since the WebIOPi buttons are predefined to only detect mouse events, something else had to be used. It was decided that simple HTML divs were going to be used as the buttons. Each div was used to represent one button, and they would each have a unique id to be called from the JavaScript program so touch events can be added to it. In addition, all the buttons would have the class "btn" so in the CSS program, only 1 CSS style would have to be written that could be applied to all the buttons. The touch events were initially tested with a simple led program. While the button was held down on the touchscreen, the led would light up, and when the button was released, the led would turn off. After this program was successfully written with the help of an online tutorial, the buttons were then created that were actually going to control the bot. For example, to create a "forward" button, the following code was written:

HTML:

```
<div class="btn" id="forwardBtn">go</div>
```

JavaScript:

```
var forward = document.getElementById("forwardBtn");
forward.addEventListener("touchstart",goForward);
forward.addEventListener("touchend",forwardStop);

function goForward()
{
    ifForward = true;
    webiopi().callMacro("forward");
}

function forwardStop()
{
    ifForward = false;
    if (ifBackward)
    {
        webiopi().callMacro("backward");
    }
    else
    {
        webiopi().callMacro("stop");
    }
}
```

CSS:

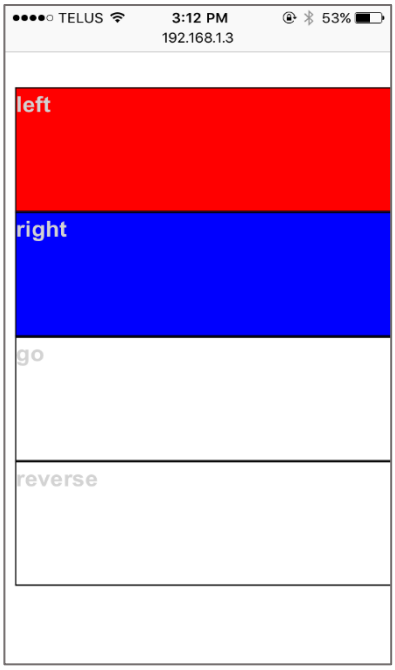
```
.btn
{
    width:300px;
    height:100px;
    border:1px solid black;
}
```

“ifForward” was a global variable that was created in the JavaScript program. This is because for the game controller, the bot would only be able to move if the forward (or backward) button was being pressed at the same time the left/right buttons were pressed. The forward button can be interpreted as the gas pedal in a car. Therefore, in the functions called when the left/right buttons are pressed, the functions would only call the Python macro to turn the bot if the ifForward/ ifBackward variables were true. Initially, the plan was to detect if both the buttons were being touched simultaneously and then execute the macro, but code to detect multiple touch events could not be found.

As each button had a unique id, in the CSS program, each individual button could have its own style in addition to the “btn” style that was applied to each of them. For example, to make the left button red and the right button blue, the following styles were added in the CSS file:

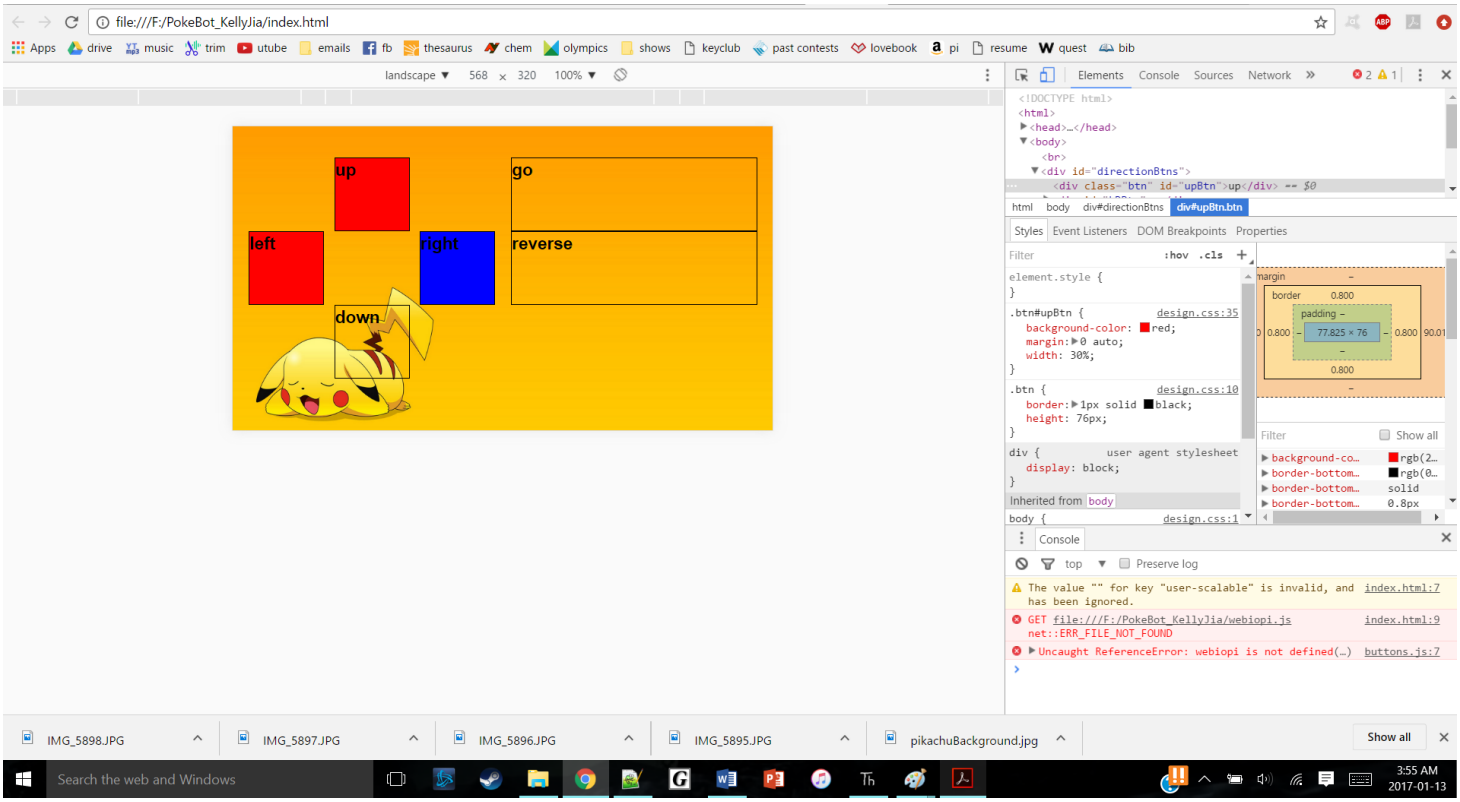
```
.btn#leftBtn
{
    background-color:red;
}

.btn#rightBtn
{
    background-color:blue;
}
```

After the left/right/reverse buttons were also created, the final testing program looked like this:



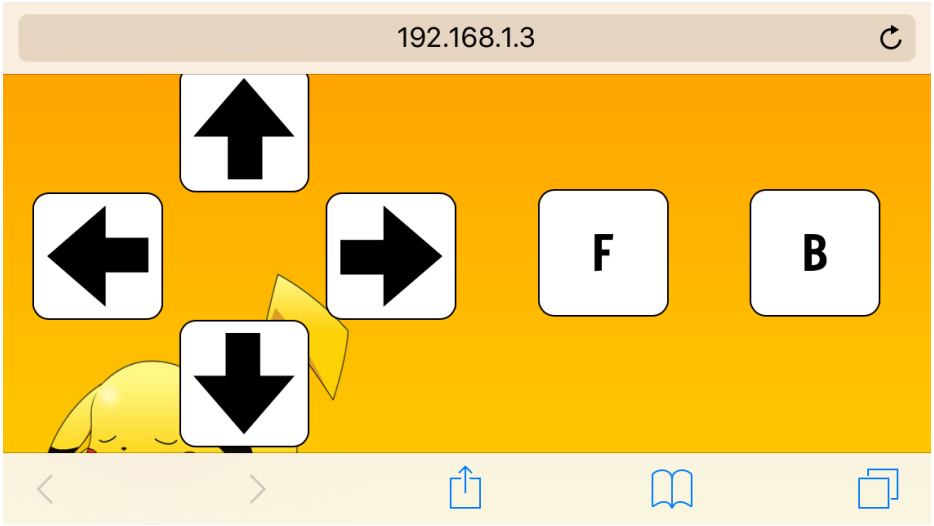
After successful trials of all the maneuvers using the testing program, proper CSS styling was done to create the layout of the game controller. To replicate the layout of the Nintendo DS game controller, 2 additional buttons were added – up and down. These will function the same as forward and backward buttons. To test the layout, the Google Chrome emulator was used. This allows one to see what their webpage will look like

on a variety of phone screens/ sizes. As the iPhone 5 was used to test this project, the dimensions of a landscape iPhone 5 screen was used (568px by 320px).



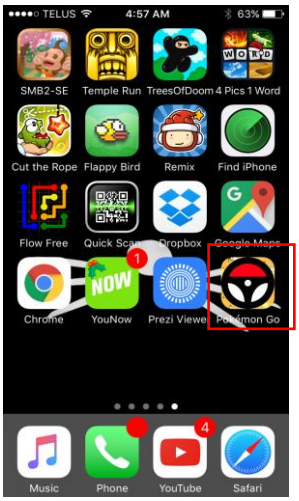
A screenshot of the webpage using the Google Chrome emulator (webpage still in progress)

This is the final layout of the web page on the iPhone 5:



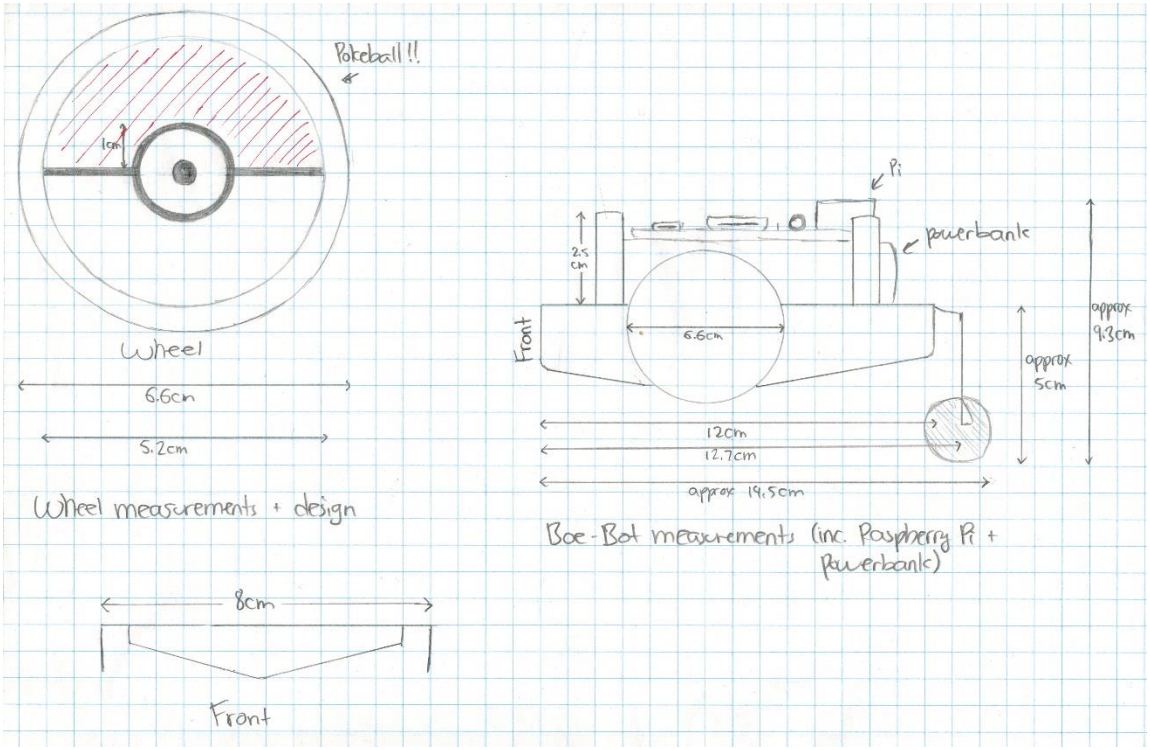
To create the button layout, the HTML div tag was used. Firstly, the direction buttons (up, down, left, right) and the movement buttons (forward, backward) were put into 2 separate div tags. Each button is also contained within its own div tag. This is so they can be controlled separately in the CSS program. Each div was set to take up 47% of the screen, so the extra space can be used for margins on the side of the screen.

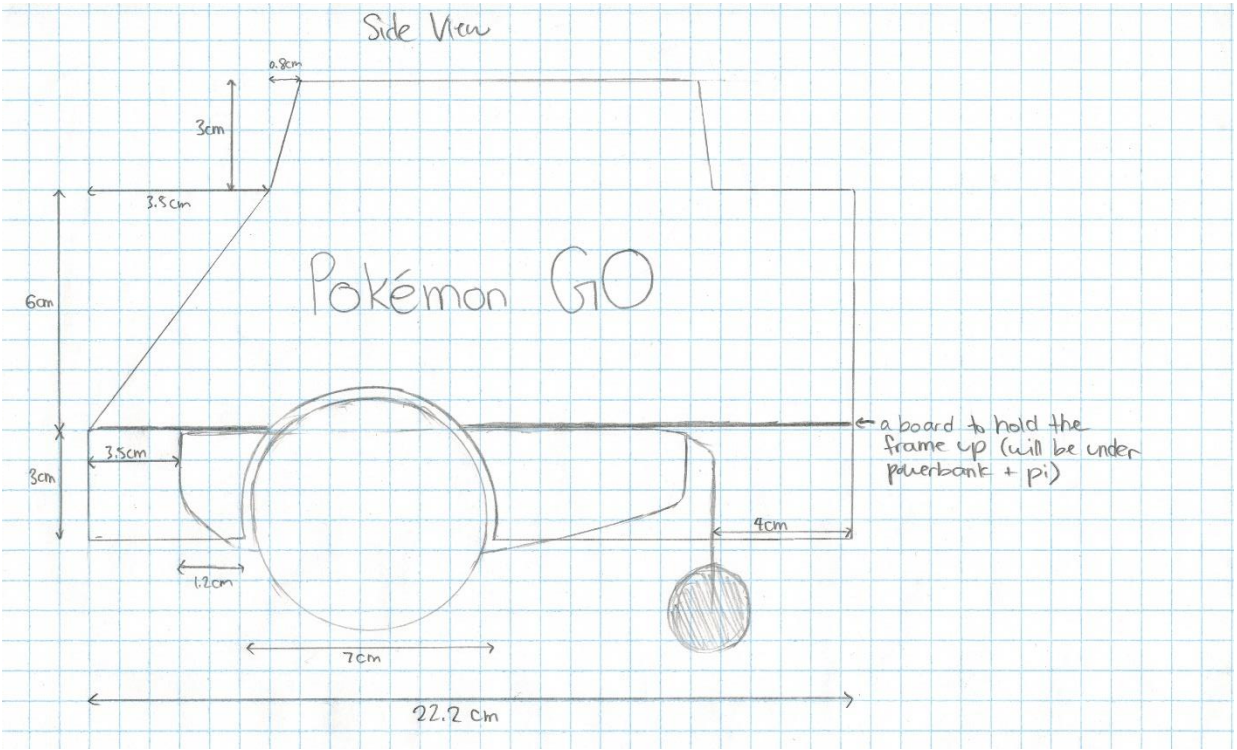
After this, the float: left property was used on the 2 divs to make them side by side on the screen. Within the direction buttons div, the up button was put into its on div tag, the left/right buttons into one div, and then the down button into another div. Since the left/right buttons were put into 1 div, the float: left property was able to be used on the left button and then the float: right property on the right button to get them onto the sides of the div tag. Following this, the up and down buttons were simply centered. In the movement buttons div, since the individual buttons were also encased in their own div tag, the float property was used to make them side by side. Since the forward and backwards buttons had to be aligned with the left/right buttons, an empty div was created above the forward/ backward buttons. After all the buttons were laid out, arrow images were set as backwards of each of the direction buttons and text was centered in the movement buttons. The button corners were then rounded with the border-radius property. Finally, a Pokémon background with Pikachu was set as the background of the whole program. Slight adjustments in pixel sizes were made throughout the creation of the web application – guess and check was used to obtain the right sizes.



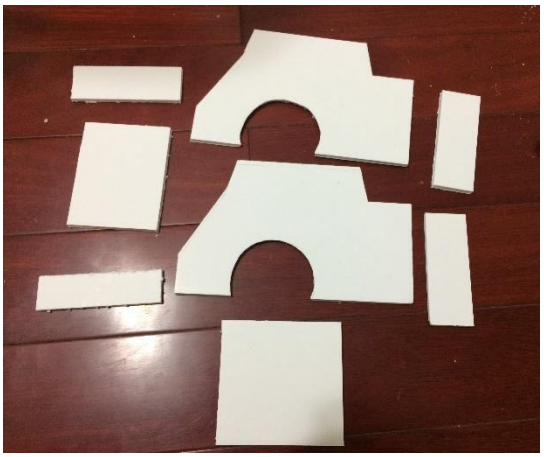
In addition, an icon image was implemented so when the web page was added to the home screen of the iPhone, it would have its own icon, similar to a regular application. *<link rel="apple-touch-icon"sizes="60x60"href="styles/images/logo.jpg"*> was added into the head of the HTML file to add the icon. Through research, it stated that the icon must be size 60x60 for the classic iPhone. If the size was not 60x60, then the icon would not appear and the standard screenshot of the website would be the icon instead.

Following the completion of the software component of the project, the rest of the hardware component was then tackled. With all the wiring to connect the pi to the Boe-Bot and the long cable of the power bank, it all looked messy and unappealing. This was why a car frame was decided to be created so the actual bot could be enclosed inside and the wiring would be hid. Before the frame could be created, the Boe-Bot firstly had to be measured out and a rough design had to be made. This was done manually on grid paper:





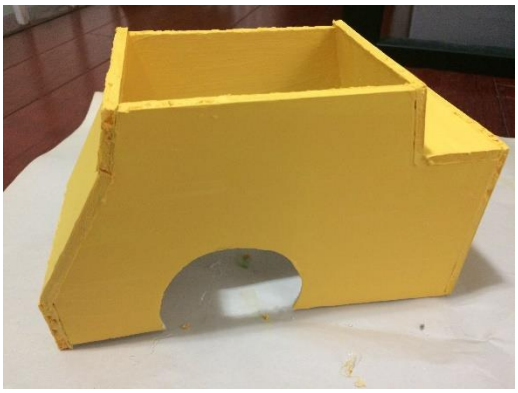
After the measurements was done, the 2 side pieces were measured out and cut from a piece of white foam board with an X-Acto knife. Once the 2 main pieces were cut, then approximation and improvisation was done to cut out the other pieces for the front and back of the frame. They were simply measured based off of the measurements of the side pieces.



All the pieces for the car frame cut from a slice of foam board

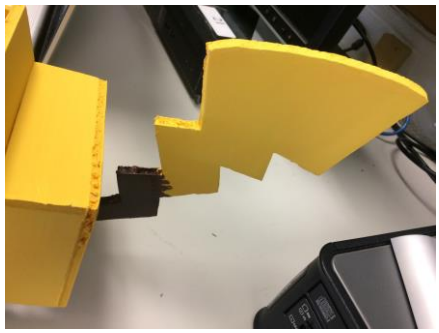
The acrylic gesso used to coat the foam board

Once all the pieces were cut out, they were coated with a layer of acrylic gesso. This was done because the raw surface of the foam board is very smooth and slippery, which causes acrylic paint to appear scratchy when painted on. With a layer of gesso, this roughens up the surface of the board so the paint sticks to it better and thus gives it a nicer appearance. The gesso itself is a white color. Once the gesso has dried, which takes approximately 1-2 hours, a coat of bright yellow acrylic paint was done and the pieces were then assembled using hot glue. Bright yellow was chosen to mimic the iconic Pokémon, Pikachu.



In the above pictures, the first image shows a board spanning across the middle of the car frame. That board was used to hold the frame in place, and the Raspberry Pi and the power bank sat on top of it.

To finish off the frame, little designs were added. These designs include a Pikachu face on the front of the car, a tail on the back, and the title “Pokémon Go” on the side.



Conclusion

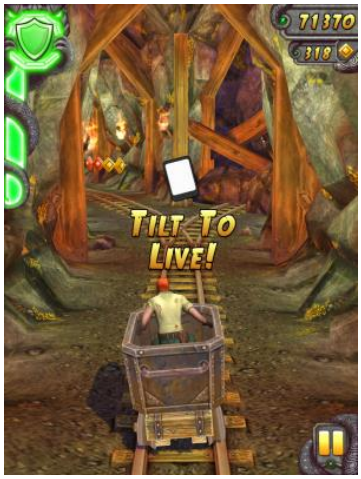
To conclude, a successful web-based game controller to maneuver a Boe-Bot was successfully created using the power of the Raspberry Pi. Research on servo motors and PWM pins was initially done to gain a deeper understanding of how they work and specifically how to use the ones of the Boe-Bot. In previous units, another language called Basic Stamp was used to write code to program the bot; however in this project, Python had to be used. Thus, the bot's maneuvers (going straight and turning) had to be tested with Python to determine the correct duty cycle ratios to turn the wheels. Following this, WebIOPi was used to create the web application for the game controller. WebIOPi contains its own JavaScript, CSS, and Python libraries so it could be used to control the Raspberry Pi's GPIO pins through a web browser. The web application proved to be an extremely difficult process as the WebIOPi libraries had to be researched and learned, and multiple testing's had to be executed to test buttons and different functions. It was however still achieved and a successful game controller was created to control the bot's movements. Furthermore, with the construction of a Pikachu themed car frame, the bot, renamed the Pokémon Go, can drive in style.

With big projects like these, it is always a huge learning experience. Before the project, one could only code basic Python programs to light up a led, and only know the basic Raspberry Pi networking capability with the chat.py program. Through this culminating project, one learned how to use PWM functionalities in Python and use it to control the servo motors of the Boe-Bot. Furthermore, a whole new application, WebIOPi, was discovered, researched, and implemented. WebIOPi is definitely a powerful tool and can certainly be used beyond this project to control other appliances through the mobile phone. As an additional Raspberry Pi was purchased, more projects implementing the Raspberry Pi can be done in the future and in post-secondary. The Raspberry Pi is certainly a powerful device with limitless possibilities.

Further Developments

A Pi Camera connected to a Raspberry Pi

Many further developments can be done to add more elements to the Pokémon Go project. One advancement is to attach a Pi Camera to the top of the Boe-Bot and then sync up the camera with the web application. The camera video would then be sent over the network to the phone screen. This allows the user to physically see what is ahead of the bot so that the bot can be driven at a distance without the user following behind. The camera could also be able to take photos, which would be sent over the network and then saved onto the user's camera roll on the phone. This can be a quite humorous project, as it can be used to spy on teachers down the hall or to friends. However, the camera can also be used a security feature against robberies or other forced entries. A prime example of this is seen in the 1997 *Home Alone 3* movie when the main character Alex manages to monitor a break-in on his house using a video camera strapped to a remote controlled race car.



A screenshot of Temple Run 2

Another possible development is to allow tilting of the phone to control the maneuvering of the bot instead of buttons. Tilting the phone is implemented in many popular games on the App Store such as Temple Run. There may still need to be a forward button, but the left and right turns could be done by tilting the phone left/ right instead of pressing buttons. This allows a more exciting user experience as tilting the phone is somewhat like driving the bot itself. However, as WebIOPi does not contain any predefined functions in its libraries to support mobile tilts, another sort of application or language must be used to program and build it. To implement this, extensive further research must be done to find programming languages that allows one to detect physical phone tilts and then how to actually program the tilt events.

Furthermore, in many racing games or applications such as Temple Run, a big component of the game is the music and sound effects. For the Pokémon Go, a standard buzzer could be used to play a jingle every time the application is booted up or a general theme song during its execution, such as the Pokémon theme song. In addition, there could also be sound effects whenever the bot turns or crashes into an obstacle. In the previous unit, whiskers were added to the bot to detect obstacles in front of it while it was moving. The whiskers can also be implemented in this project to play some sort of “boom” or “crash” sound effect whenever the bot hits the obstacle. These sound effects would give the user an overall more exciting and fun experience when driving the bot around.

Works Cited

Information:

"Automatically connect a Raspberry Pi to a Wifi network." We Work We Play. N.p., n.d. Web. 14 Dec. 2016. <http://weworkweplay.com/play/automatically-connect-a-raspberry-pi-to-a-wifi-network/>.

"Buttons and Switches." Physical Computing with Raspberry Pi. University of Cambridge, n.d. Web. 6 Dec. 2016. <https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/robot/buttons_and_switches/>.

Croston, Ben. "Using PWM in RPi.GPIO." Sourceforge. Slashdot Media, n.d. Web. 2 Dec. 2016. <https://sourceforge.net/p/raspberry-gpio-python/wiki/PWM/>.

Daware, Kiran. "How Does A Servo Motor Work?" Electrical Easy. N.p., n.d. Web. 30 Nov. 2016. <http://www.electricaleasy.com/2015/01/how-does-servo-motor-work.html>.

Dee, Jordan. "Pulse-width Modulation." Sparkfun. SparkFun Electronics, n.d. Web. 31 Nov. 2016. <https://learn.sparkfun.com/tutorials/pulse-width-modulation/duty-cycle>.

Hanselman, Scott. "Make your Website Mobile and iPhone Friendly - Add Home Screen iPhone Icons and Adjust the ViewPort." Hanselman. N.p., 25 July 2008. Web. 12 Jan. 2017. <http://www.hanselman.com/blog/MakeYourWebsiteMobileAndIPhoneFriendlyAddHomeScreenIPhoneIconsAndAdjustTheViewPort.aspx>.

Hildenbrand, Jerry. "How-To: Controlling GPIO Pins via the Internet." Connectedly Forum. N.p., n.d. Web. 12 Dec. 2016. <http://forums.connectedly.com/raspberry-pi-f179/how-controlling-gpio-pins-via-internet-2884/>.

"Make your site work on touch devices." Creative Bloq ART AND DESIGN INSPIRATION. N.p., 13 May 2014. Web. 23 Dec. 2016. <http://www.creativebloq.com/javascript/make-your-site-work-touch-devices-51411644>.

Reed, Frances. "How Do Servo Motors Work." Jameco Electronics. Jameco, n.d. Web. 30 Nov. 2016. <http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html>.

"Responsive Web Design - The Viewport." W3schools. N.p., n.d. Web. 23 Dec. 2016. <http://www.w3schools.com/css/css_rwd_viewport.asp>.

Robotics with the Boe-Bot: Version 3.0. N.p.: Parallax Inc., 2010. Print.

"SFTP." Raspberry Pi. RASPBERRY PI FOUNDATION, n.d. Web. 13 Dec. 2016. <https://www.raspberrypi.org/documentation/remote-access/ssh/sftp.md>.

Singleton, David. "Introducing PiUi: Add a Simple Mobile Phone UI to Your RaspberryPi Project." Blog post. David Singleton. N.p., n.d. Web. 9 Dec. 2016. <http://blog.davidsingleton.org/introducing-piui/>.

Images:

Gargette, Darren. Nintendo DS Lite. Digital image. Generation-N. N.p., n.d. Web. 30 Dec. 2016. <http://www.gen-n.net/reviews/ds/lite.shtml>.

RASPBERRY PI CAMERA BOARD. Digital image. Adafruit. N.p., n.d. Web. 12 Jan. 2017. <https://www.adafruit.com/products/1367l>.

Servo Motor Diagram. Digital image. Electrical Easy. N.p., n.d. Web. 30 Dec. 2016. <http://www.electricaleasy.com/2015/01/how-does-servo-motor-work.html>.

Temple Run 2 Tilt Pic. Digital image. Nardio. N.p., n.d. Web. 12 Jan. 2017. <http://nardio.net/2013/01/18/temple-run-2-review/temple-run-2-tilt-pic/>.

W., Dan "Deezer" Digital image. The Mushroom Kingdom. N.p., n.d. Web. 30 Dec. 2016. <http://themushroomkingdom.net/media/mk8/ss>.

Software Downloads:

"Download FileZilla Client for Windows (64bit)." FileZilla. N.p., n.d. Web. 13 Dec. 2016. <https://filezilla-project.org/download.php>.

"Installation." WebIOPi. N.p., n.d. Web. 12 Dec. 2016. <http://webiopi.trouch.com/INSTALL.html>.

Seya, Keisuke "doublebind" "WebIOPi-0.7.1 Patch for Raspberry B+, Pi2, and Pi3." GitHub. GitHub Inc, 16 May 2016. Web. 13 Dec. 2016. <https://github.com/doublebind/raspi?files=1>.