Nicholas Thorondor's Raspberry Pi Project

This project is the result of many hours of research, experimentation, coding and troubleshooting. It all began with a Raspberry Pi gifted to me by my sister for Christmas 2018. The project has been a great learning experience and I am very happy with the result. I wanted to share that result with others so that they may also replicate this project, learn from it and expand upon it if they so desire.

A big thank you to the developers of the various open source software and packages used throughout this project.

Project functionality

- Airplay server
- Plex server
- Music Visualiser
- Ambilight
- Mood LED
- Image Slideshow
- Web-based GUI control panel

Equipment Needed

Core Devices

- Raspberry Pi 3 B+
- Arduino UNO
- USB sound card
- NAS or external storage device for storing media
- 3.5mm audio cable compatible speakers
- USB Keyboard and Mouse (Wireless or wired)
- SD card w/ USB adaptor or Micro SD card (Preformatted with NOOBS or the OS of your choice for installing the OS onto the Pi)
- TV

Support-Devices

- USB power adaptor with 3+ USB slots
- Powered USB hub
- HDMI splitter (1 input, 2 output)
- HDMI switcher (3 input, 1 output)
- HDMI to AV converter
- USB video grabber (RCA input) (Must have UTV007 Fushicai chipset)

Power Supplies, LEDs and Cases

- 5M RGB LED strip – WS2812B (30 LED's per metre)
- 5V 10A power supply (for the LED's)
- Case for Raspberry Pi (Optional)
- Case for Arduino UNO (Optional)

Cables

- 5V micro USB power plug
- Micro USB male to female power switch (for the Pi)
- USB cable for Arduino UNO
- 3 x 1m USB extension cable
- HDMI cables (will need at least 2, depending on how many devices you are connecting)
- 220-240V (110V in certain countries) power cable (old PC three pin plug will do)
- RCA male to male cable
- Solid core or stranded wire approx. 20AWG (whatever you have available should work as long as it's not too thick of a gauge)
- Female jumper connectors for 0.1" male pin headers, or wire with these jumpers already installed
- Logic level converter 3V to 5V (Not needed necessarily)

Tools

- Philips head screwdriver
- Wire strippers
- Wire cutters
- Soldering iron
- Solder
- Heat shrink wrap (or electrical tape)
- Heat gun
- 0.1" crimping tool for jumpers (only needed if you are making your own 0.1" female jumper wires)

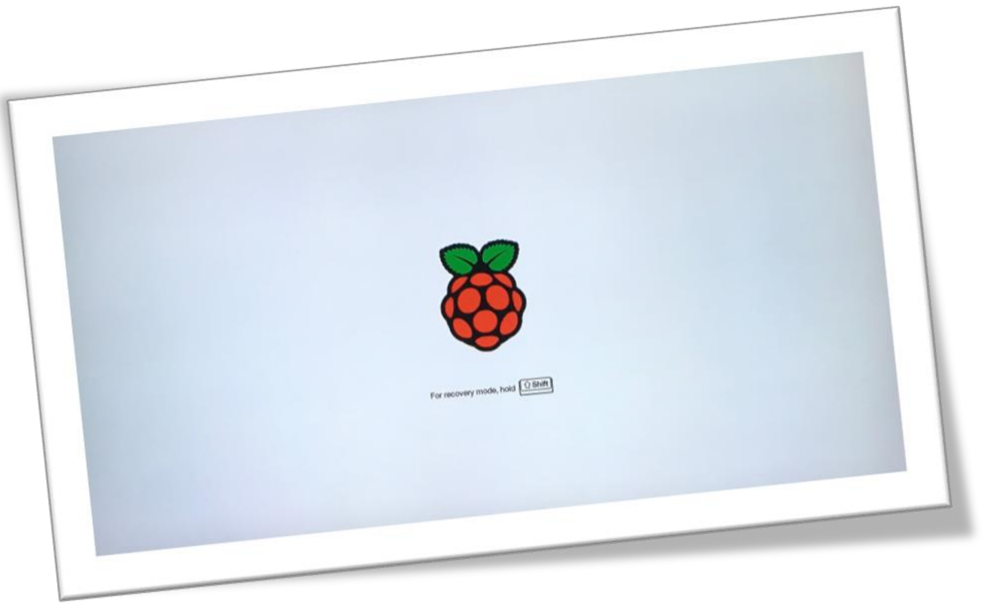
Contents

1. Set up Raspberry Pi hardware and install OS

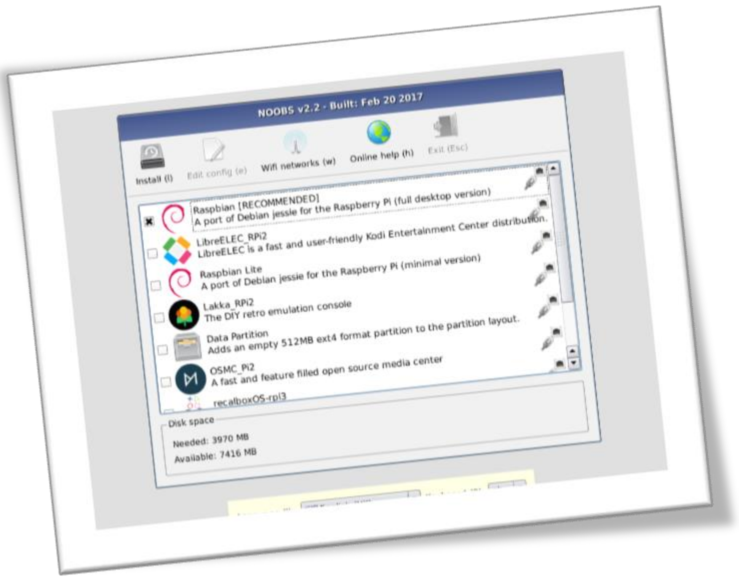
The first step is to get the Raspberry Pi up and running in an operable state. Take an HDMI cable and plug one end into the HDMI port on the Pi, and the other end into a display of some sort, this can be a monitor, TV or any form of display device that can take an HDMI cable as input. In this case I have chosen to hook the Pi up to my TV. Next, plug the 5V micro USB power cable into the micro USB power switch, then plug the end of that into the Pi. Do not plug the other end into a power source just yet, as this will begin booting the Pi immediately if the power button in the switch is depressed.

On the underside of the Pi there is a micro SD card slot, install your micro SD card here if you are using one, otherwise if you are using a standard SD card, plug the USB adaptor into one of the powered USB hubs ports. Finally, plug the USB sound card and keyboard and mouse into the powered USB hubs ports, then plug the hub into the Pi. We are now ready to begin installing the OS.

Turn on your display and plug the Pi into a power source. The Pi will begin booting automatically and a rainbow coloured splash screen should be shown. Following this another splash screen as shown in the following image will be shown. Hold the shift key on your keyboard when presented with this screen to initiate the NOOBS installer.



From the list of OS to install, choose the one you want to use, in this case I went with the recommended standard Raspbian version. Make sure the language is set to what you want, then click Install (or hit 'i' on the keyboard).



[✓ Raspbian [RECOMMENDED]]
A port of Debian jessie for the Raspberry Pi (full desktop version)

LibreELEC_RPi2
LibreELEC is a fast and user-friendly Kodi Entertainment Center distribution.

Raspbian Lite
A port of Debian jessie for the Raspberry Pi (minimal version)

Lakka_RPi2
The DIY retro emulation console

Data Partition
Adds an empty 512MB ext4 format partition to the partition layout.

OSMC_Pi2
A fast and feature filled open source media center

recalboxOS-rpi3

Disk space
Needed: 3970 MB
Available: 7416 MB

The Pi will begin installing the chosen OS onto your SD card, just sit back and wait for the install to complete. Once the install has completed you will be greeted with the desktop of your brand-new Raspberry Pi. At this point it is likely a brief setup will be shown to allow you to set keyboard language, Wi-Fi connectivity and similar. Once this is completed you are ready to roll.

2. Install Plex Media Server

Documentation consulted for this section.
https://thepi.io/how-to-set-up-a-raspberry-pi-plex-server/
https://wiki.archlinux.org/index.php/fstab
https://support.plex.tv/articles/204059436-finding-an-authentication-token-x-plex-token/

https://support.plex.tv/articles/201638786-plex-media-server-url-commands/

2.1. Install Plex

To function as a media server for other devices in your network, the Pi will need to have appropriate software installed. In this case I have chosen to use Plex as my media server. So, the first course of action here is to get Plex installed. Bring up the terminal (command line) which is located in the top taskbar of the Pi's desktop (or hit 'ctl+alt+t'). Before installing Plex, we need to ensure that the Pi is up to date software wise. Type the following commands into the terminal:

```
$> sudo apt-get update
$> sudo apt-get upgrade
```

The first command will check for any available updates, while the second will install these updates onto the Pi. Be patient here as it may take a while.

Next, we need to make sure the HTTPS transport package is installed. It most likely is, but best to run the following command just in case to make sure (if it is installed a message telling you just that will be displayed):

```
$> sudo apt-get install apt-transport-https
```

Now we need to add a repository key to the Pi for the dev2day website, which is a repository containing the Plex media server. Don't worry if you do not understand the following command, it is a little confusing, but just know that it is adding the key supplied by the dev2day website to the list of trusted keys on the Pi, meaning that we can add the dev2day repository to the Pi's package source list.

```
$> wget -O - https://dev2day.de/pms/dev2day-pms.gpg.key | sudo apt-key add –
```

Now add the dev2day repository to the Pi's package source list:

```
$> echo "deb https://dev2day.de/pms/ jessie main" | sudo tee
/etc/apt/sources.list.d/pms.list
```

Finally, update the package list one more time and then download the Plex package:

```
$> sudo apt-get update
$> sudo apt-get install -t jessie plexmediaserver
```

We are not done yet. Now we have to set the system permissions to allow Plex to be run under the Pi user account (the default user). Open up the config file with the following command:

```
$> sudo nano /etc/default/plexmediaserver.prev
```

Now find the line that says `PLEX_MEDIA_SERVER_USER=plex`. Change the `plex` to `pi`, hit ctl+x, then y and finally hit the ENTER key to save the changes to the file. From now on this save process will not be specified, it is assumed when changes are made to a file you will save the file upon exit. Restart the plex media server to allow this change to take effect, and then reboot the Pi using the following commands:

```
$> sudo service plexmediaserver restart
$> reboot
```

And that's it, you now have a Plex server installed and running each time the Pi is booted. However, before you can access any media, you must populate the Plex server with your media files. If you are using a SMB/CIFS NAS or similar network storage device to store your media you will most likely need to mount it to the Pi's filesystem before you can access any of those files. Rather than having to manually mount the NAS every time you boot the Pi, we will create a new rule in the fstab file which will automatically mount the NAS when the Pi is booted. First, type the following command into the terminal to create a new directory for mounting the NAS to, and then open the fstab file for editing:

```
$> sudo mkdir /mnt/NAS
$> sudo nano /etc/fstab
```

Now navigate to the bottom of this file to a newline, and enter the following as one continuous line (no line breaks), replacing the IP address, pathname and password accordingly:

```
//192.168.0.20/Videos /mnt/NAS cifs noauto,nofail,x-systemd.automount,x-
systemd.requires=network-online.target,x-systemd.device-
timeout=10,password=12345 0 0
```

Note that the above line may need to be edited to suit your environment, this is simply an example of what worked for me and my setup. To get detailed information regarding fstab and what will work with your setup consult the ArchLinux documentation - https://wiki.archlinux.org/index.php/fstab

Finally, we are ready to add the media files to the Plex server. Open up a web page, by default this will be in the top taskbar of the Pi and enter the following into the address bar to be directed to the Plex server dashboard and begin setting up your server settings (name, categories, file locations etc.).

```
localhost:32400/web/
```

If you need any help here consult the Plex documentation - https://support.plex.tv/articles/200264746-quick-start-step-by-step-guides/

You should now be able to access and play the media files you added to the Pi's Plex media server through other client devices connected on your network, assuming they are using the Plex client app of course.

2.2. Programmatically Refresh Plex Library

As the end goal of this project is to use a web-based GUI to control the functionality of the PI, we need a way for the Plex server to programmatically scan for changes in the libraries and refresh the library to reflect these changes. To do this I took advantage of the curl utility and the fact that Plex allows you to control the server from the URL bar. I created a simple script shown below by following the GitHub link, that carried out the necessary commands to sync the libraries. I would suggest creating a directory called *scripts* under your */home/pi* directory on your Pi to store each of the scripts used throughout this project.

https://github.com/nicholasthorondor/Project-Pi/blob/master/plexscan.sh

This script has the effect of carrying out a manual scan of the libraries with id 1 and 2. The output is discarded, and the command is run in the background. The token is unique to each system, I have simply listed a sample token for demonstration purposes. To find the actual token go to a video file's info in plex and open the xml document for the file. At the top of this xml document there will be the unique token value.

Once you have your token you can find the numeric value associated with each of your libraries by entering the following into the URL bar, substituting in your token value. The output will be an XML document that you can scan through looking for library identification numbers.

```
localhost:32400/library/sections?X-Plex-Token=YourTokenGoesHere
```

More info can be found in the Plex documentation - https://support.plex.tv/articles/201638786-plex-media-server-url-commands/

https://support.plex.tv/articles/204059436-finding-an-authentication-token-x-plex-token/

At this point you have two options. The first being to set this script to run automatically at given time intervals. The advantage of this is that it does not require any user input to sync the plex libraries. The disadvantage being that there will be periodic load on the Pi's CPU as the script is run consistently even if changes to the plex library haven't occurred.

The second option is to hook this script into the web control panel that is built in Section 10 of this document. The advantage of this method is that the sync will only occur if executed, therefore removing the consistent periodic CPU load from the automatic method. The disadvantage being that you need to remember to hit the update plex button via the control panel to propagate any changes to the Plex libraries.

Both methods will work just fine, it is entirely personal choice as to which you implement. The following shows how the automatic syncing is done via a cron job, or if you would prefer the manual control panel method, the code for that is detailed in Section 10.

Auto update Plex

To set up a cron job to execute the script, called plexscan.sh, every minute, first open your crontab.

```
$> crontab -e
```

Then enter the following at the end of the file.

```
SHELL=/bin/bash
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/home/pi/scrip
ts

* * * * * /home/pi/scripts/plexscan.sh
```

And that's it, all done.

3. Build/Modify 3.5mm Speaker System

Documentation consulted for this section.
https://www.youtube.com/watch?v=vIPVIvy4ckU

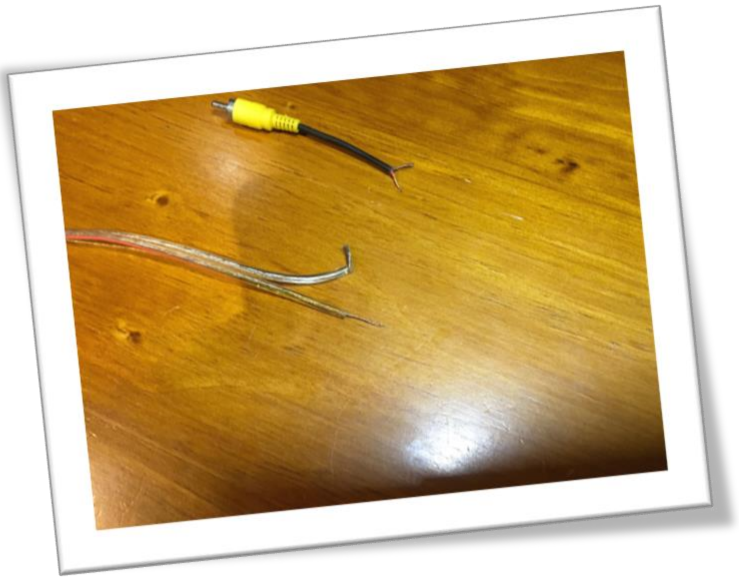
Now that the video aspect of the media server is set up using Plex, we can now turn our attention to the audio aspect, namely AirPlay functionality. However, in order to play music, we need to set up a speaker system for the Pi. In my case I am going to be using and modifying an old computer speaker system that I had lying around. This section just acts as a sort of journal of the processes I took to modify the speaker system. If you already have a fully functional 3.5mm speaker system ready to go skip this section, if not, or you are curious, read on.

The speakers I am using are actually from two different systems. One is a computer sound system with a small subwoofer and two mini speakers implementing RCA connections, the other is an old iPad docking station with two Perspex speakers running standard speaker wire and connectors.

I will be using the small subwoofer and modifying the Perspex speakers to be fitted with RCA connectors rather than the current connectors they are installed with. In order to do this, I will be using a spare RCA cable I had lying around, cutting the connector ends off and splicing them onto the Perspex speakers wires. The result will be the ability to plug the Perspex speakers into the small subwoofer and take advantage of their superior audio quality in contrast with the original mini speakers.

First, I cut an end off the spare RCA cable, as well as one of the ends of the speaker wire connected to the Perspex speakers.

Next, I used wire strippers to strip the wire back on both the speaker wire and RCA plugs to expose the individual positive and negative wires.



The plastic-coated wire inside the RCA connectors is the positive, while the raw copper is the negative. I made sure to keep this polarity in mind when splicing to the standard speaker wire. Before I began splicing, I made sure a heat shrink tube was slipped onto one end of the RCA connector wire being spliced. Using a small amount of solder, I spliced the positive speaker wire to the positive RCA connector wire and repeated for the negative wire. Then I positioned the heat shrink tube over the splice and used the heat gun to shrink the tubing.

And there we have it, a modified speaker that will be able to interface with my standard computer mini subwoofer. I repeated these steps for the other speaker and tested the audio output to make sure everything was working. We can now move onto configuring the Pi to become an AirPlay receiver.

4. Configuring AirPlay on the Pi

Documentation consulted for this section.
https://thepi.io/how-to-set-up-a-raspberry-pi-airplay-receiver/
https://pimylifeup.com/raspberry-pi-airplay-receiver/
https://github.com/mikebrady/shairport-sync
https://github.com/mikebrady/shairport-sync/blob/master/scripts/shairport-sync.conf
https://github.com/mikebrady/shairport-sync/blob/master/TROUBLESHOOTING.md

4.1. Install Required Software

To get AirPlay functioning correctly it will require installing a number of packages and libraries which are needed in support for the primary package shairport-sync.

To begin make sure your Pi is up to date.

```
$> sudo apt-get update
$> sudo apt-get upgrade
```

Next install the necessary support packages.

```
$> sudo apt-get install autoconf automake avahi-daemon build-essential git
libasound2-dev libavahi-client-dev libconfig-dev libdaemon-dev libpopt-dev
libssl-dev libtool xmltoman libsoxr-dev
```

Once that's done the shairport-sync repository needs to be cloned to the Pi. This is the software that is responsible for enabling the AirPlay functionality on the Pi.

```
$> cd ~
$> git clone https://github.com/mikebrady/shairport-sync.git
```

With that done change into the newly cloned directory.

```
$> cd shairport-sync
```

Now build shairport-sync.

```
$> autoreconf -fi
$> ./configure --with-alsa --with-avahi --with-ssl=openssl --with-systemd --
with-metadata --with-soxr
```

After the build process has finished, compile and install.

```
$> make
$> sudo make install
```

Finally, set the newly installed shairport-sync to start when the system boots.

```
$> sudo systemctl enable shairport-sync
```

That's it, the install is done and should be ready to go.

4.2. Configure USB Soundcard

By default, shairport-sync defaults to using the standard 3.5mm jack of the Pi. As this project uses a USB soundcard to drive the audio output/input this requires a few configuration entries to be changed to ensure the USB soundcard is the default audio device used by shairport-sync.

First, find the name associated with the USB soundcard by opening alsamixer and looking at the name in the 'Item' section of the GUI assigned to the USB soundcard device.

```
$> alsamixer
```

Next, find out what card number the USB soundcard is listed as using the following command.

```
$> aplay -l
```

The output of this command will show all the audio devices that are present on the Pi. Look for the one that matches the name of the USB soundcard you are using and take note of the card number associated with it.

Now to edit the config file. The shairport-sync config file is found at /usr/local/etc/shairport-sync.config. Open this file to begin editing.

```
$> sudo nano /usr/local/etc/shairport-sync.config
```

There are a few lines that need editing. Navigate to the parameters in the `alsa{}` section near the bottom of the file. Here change the following entries, substituting '1' for the card number of the USB soundcard and 'PCM' for the name associated with the USB soundcard if yours differ.

```
output_device = “hw:1”;
mixer_control_name = “PCM”;
```

Next, you create/edit the /etc/asound.conf file,

```
$> sudo nano /etc/asound.conf
```

and fill it with the following, taking note to change the card names and numbers to the values associated with your setup, as discussed earlier.

```
pcm.!default {

    type plug

    slave.pcm {

        type dmix

        ipc_key 1024

        slave {

            pcm "hw:1"

            period_time 0

            period_size 1920

            buffer_size 19200

        }

    }

}

ctl.!default {
    type hw

    card 1

}
```

This file sets the USB sound card to be the default audio device and is also looked at by shairport-sync for the values it contains. In this case these are suggested values recommended by the shairport-sync troubleshooting guide as a potential preventative for choppy audio output.

That will finish setting the Pi up as an AirPlay receiver, and it should be visible as a device to stream to from an AirPlay compatible device.

4.3. Choppy Audio

In my case, the audio that was being streamed to the Pi was extremely choppy, even with the preventative configuration in the /etc/asound.conf file. If this is also happening in your set-up the following steps will hopefully alleviate that issue.

First, make sure the USB soundcard is plugged into the powered USB hub, not the Pi directly, to ensure enough power is being supplied to the USB soundcard.

There is also a chance that your Wi-Fi connection is not stable enough to handle the AirPlay stream without choppiness occurring. This could be due to being too far away from the router/modem distributing the Wi-Fi or could be due to excessive traffic on the network or other similar situations. In my case the Wi-Fi connection was not stable enough due to my proximity to the router/modem being too far away. I solved this problem by installing a Wi-Fi extender in the room in which my Pi was located. This extender rejuvenated the Wi-Fi signal to full strength, which in turn removed the audio choppiness from the AirPlay stream.

Having done that, hopefully AirPlay functionality should be complete and fully functional. If not I would suggest having a look at the troubleshooting section of the software to see if that can help - https://github.com/mikebrady/shairport-sync/blob/master/TROUBLESHOOTING.md

5. LED Hardware Installation

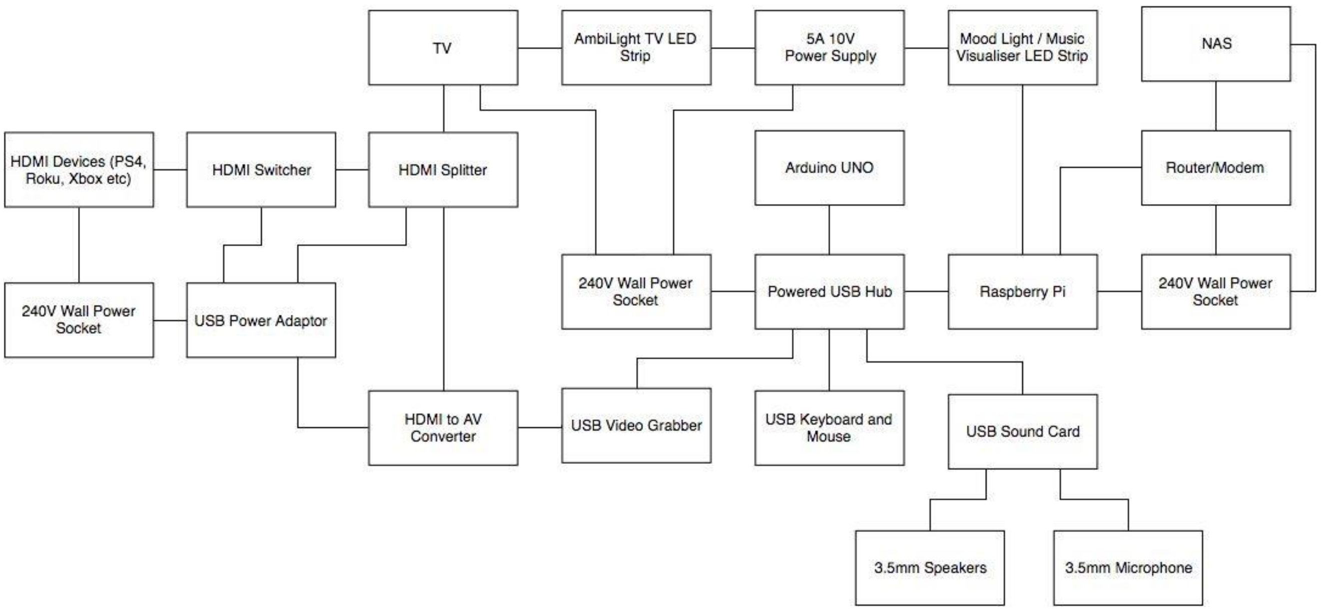
Documentation consulted for this section.
https://core-electronics.com.au/tutorials/ws2812-addressable-leds-raspberry-pi-quickstart-guide.html
http://awesomepi.com/part-3-using-ambilight-for-every-hdmi-device-an-ultimate-step-by-step-tutorial/
https://www.instructables.com/id/Ambilight-System-for-Every-Input-Connected-to-Your/

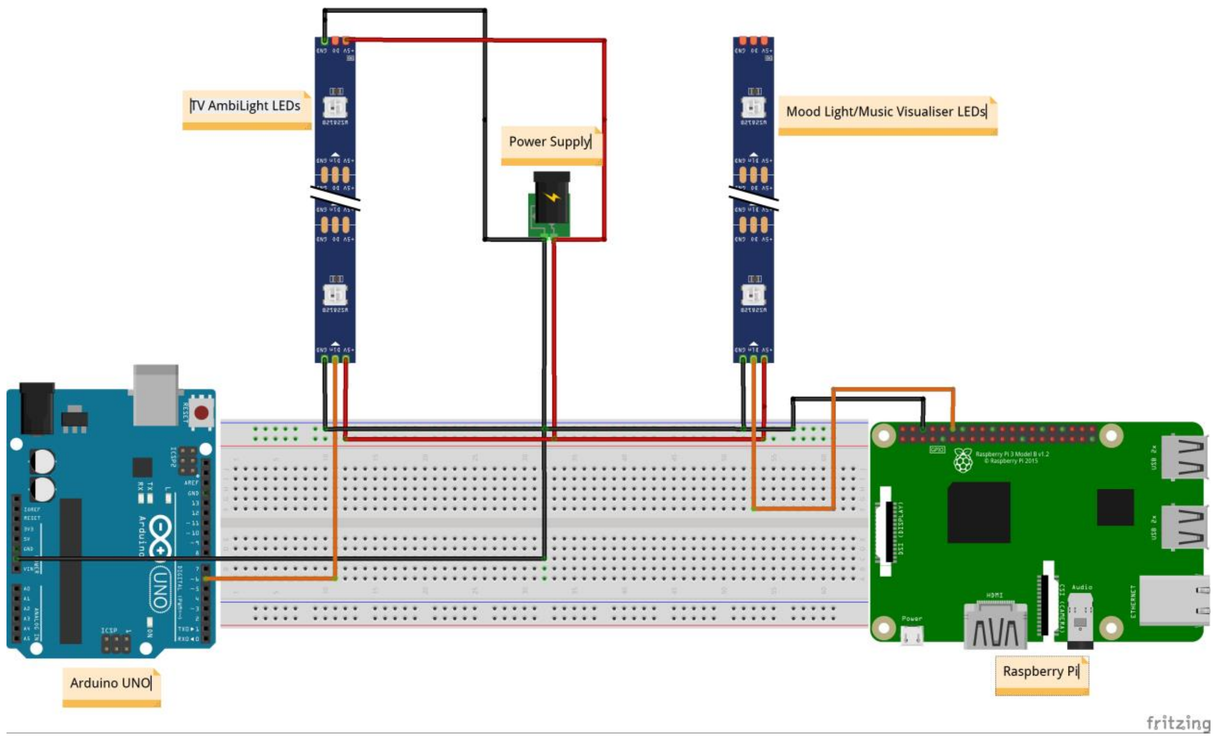
Now that the media server and audio functionality is set-up, we can start considering the LEDs. However, before we start working on getting the varying LED functionality up and running, the hardware needs to be wired up and installed onto the back of the TV.

5.1. Architecture Diagrams

Before jumping into the installation, it would be best to describe how each component will be connected and wired up. This is best done through visual means; therefore, I have included a high-level diagram of the relation between each component's connection, as well as a wiring diagram for the LEDs just below. Note that the wiring diagram uses a breadboard in order to keep the wiring clear and understandable and also does not show the USB connection between the Arduino Uno and the Pi. Take careful note of the shared ground connections when wiring everything together. One other important thing to note is the separation of the LEDs on the back of the TV, which are used exclusively for the AmbiLight functionality, and the LEDs used for the mood light and music visualiser.

As shown in the wiring diagram, both sets of LEDs are powered and grounded via the same external power supply. However, the data line for the AmbiLight LEDs on the back of the TV is connected to the Arduino UNO, while the data line for the mood lights and music visualiser is connected to the Pi. This separation of data lines is required due to the nature of Arduino sketches (scripts) running non-stop once powered on. Essentially the LEDs on the back of the TV set-up for AmbiLight functionality continually receive a data signal from the Arduino while it is powered on, even if the AmbiLight functionality is turned off via the web control panel created in Section 10, as this simply stops the Hyperion (AmbiLight) service on the Pi, it does not power down the Arduino. This means that if the mood light and music visualiser were also connected to the same data line, conflicts would occur, and the display would either be erratic or not occur at all. In my case, when I initially tested it this way, my Pi froze.



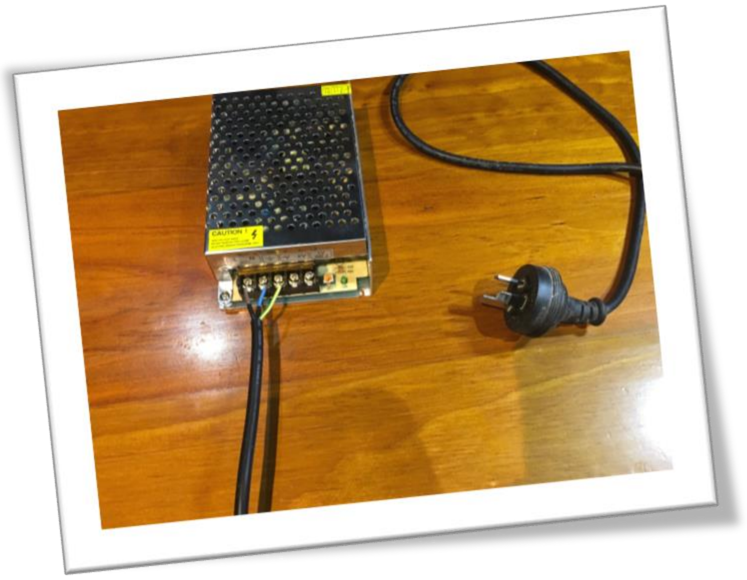


5.2. Wiring and Hardware Installation

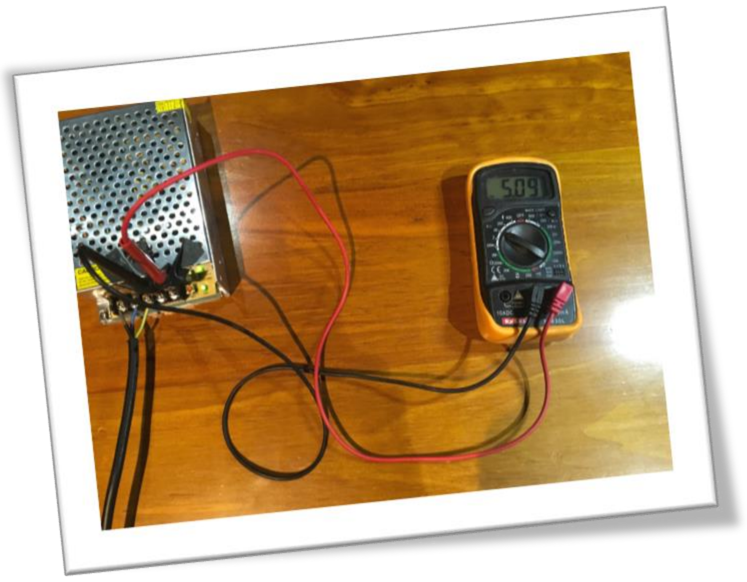
With diagrams in hand the hardware installation can begin. First, the LEDs will need to be cut to length for each section of the TV, ie. top, bottom, sides. Then cut the left-over strip to length for the mood light/music visualiser setup you are using. Jump to the bottom of this section to see how I have set-up my LEDs for this aspect of the project. When cutting the LED strip, make sure to cut in the middle of the copper connectors designed for linking sections of the strip together and solder them back together with small gauge wire, joining the input side of each cut to length LED strip to the output side of the adjoining strip until the circuit is re-established between each individual strip.

As the strip of LEDs going around the TV is quite long, it is best practice to wire up power and ground connections to both ends of the LED strip. This ensures that an even power level is distributed throughout the LED strip, preventing any unwanted flickering. This can be seen in the wiring diagram and is not required for the music visualiser / mood lights as the LED strip used for this aspect will likely not be long enough to warrant it.

The LED power supply used here needs to be wired up with a 240V power lead (or 110V in certain countries). I used an old PC power lead and cut the three-pin connector end off, stripped the wires back and attached to the appropriate connection terminals of the power supply (brown wire to L terminal, blue wire to N terminal, green/yellow wire to ground terminal which looks sort of like a multi crossed T). This follows a standard Australian 240V power connection, if you live in another country your connections and wire colours may vary. I would suggest getting an electrician to wire the power supply up for you if you have any doubts, regardless of country, as any mistake could have severe consequences.



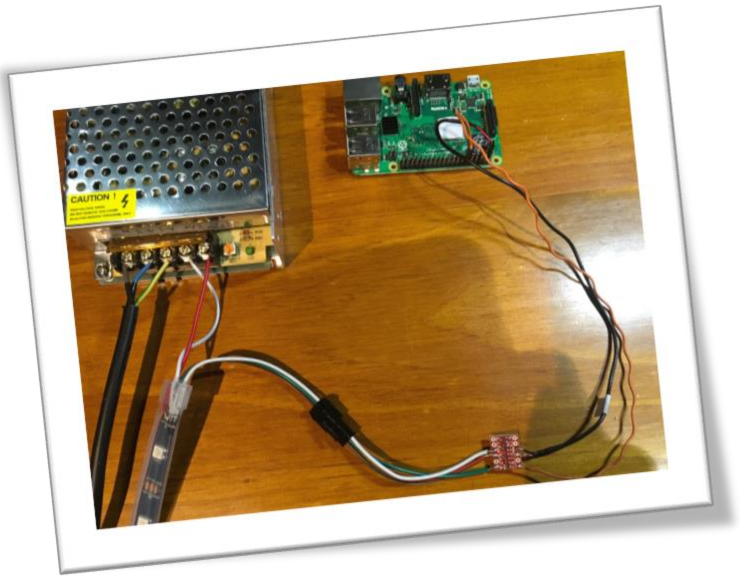
After the power supply is wired up you can test the output voltage using a multimeter to ensure it is within the 5V region. Again, if you undertake this step be very careful, as you are dealing with high voltage current terminals just next to the 5V terminals.



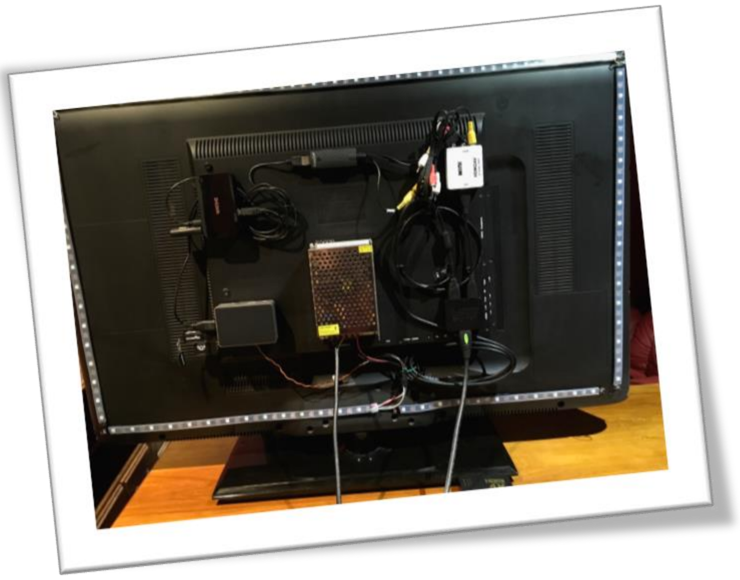
Once this is done, and assuming the power supply is giving out around 5V, then you can proceed with wiring up the connections for the LED strips, Arduino UNO and Raspberry Pi as shown in the wiring diagram. 5V power connections go to the V+ terminal of the power supply, while ground connections go to the V- or COM terminal. In the case of the Pi, when attaching the ground and data line to the GPIO pins use the 0.1" female jumper wires if you have them available. If not, you will need to install a female jumper connector for 0.1" male header pins onto one end of the wire using the crimping tool mentioned in the tools required at the start of this project guide. The Arduino UNO can simply have its connections soldered straight to the board.

Before handling any wires make sure the power supply is unplugged from the wall, and every device is powered off completely.

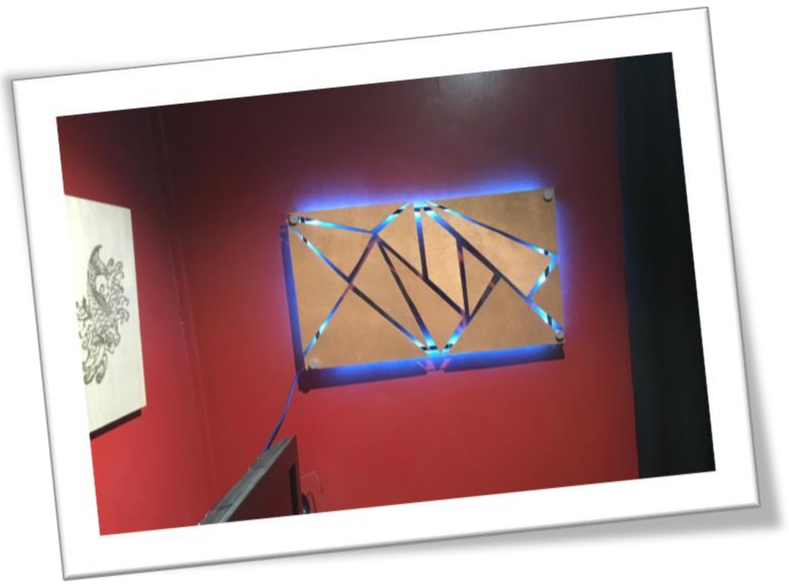
It is best practice to wire the LED's to a logic level converter rather than going directly to the Pi itself due to the voltage difference between the two. However, as the LEDs are powered directly by the power supply, I found that the logic level converter was not necessary. In fact, due to the cable lengths I am running to get the connections to my wall mounted mood lights, I found that the use of the logic level converter resulted in flickering of the LEDs. The removal of the logic level converter fixed this problem, so in my case I did not end up using the logic level converter, however your results may be different. I suggest you try without the logic level converter initially, and if there are issues try with the logic level converter to see if it helps. An example of the use of the logic level convertor is shown in the following image, which was taken during testing while creating this project guide. Do not follow this wiring, use the wiring diagram provided earlier, this image is simply to illustrate the use of the logic level convertor in the event that you need to install one for your set-up.



Once all the wiring is complete, you will then need to adhere each of the components onto the back of the TV and attach the USB, HDMI etc cabling between each device. Note that you only need to connect the yellow RCA plug of the USB video capture device, as this is the connection that transmits the video signal. I used an adhesive Velcro tape to attach the components to the TV, though any double-sided tape or similar will also suffice. You will likely need to disconnect any preconnected wires to the LED power supply and Pi while adhering to the TV, then reconnect them once everything is in place. The following image shows an example of the placement of the components. Keep in mind this is in an unfinished early testing state, with not all components and wiring being completed, the Arduino UNO is missing for example. The layout remains similar in the final form, though is more refined. Though unfinished, the image should provide you with an idea of how to go about laying out the components but will ultimately come down to the shape and style of your TV.



Finally, before the wiring and hardware installation is complete, the music visualiser/mood light LED design needs to be considered. This can be as simple as running the LED strip being used for this aspect on the back of the TV, the same as the AmbiLight LEDs, or it can be as creative as you like. I decided to get creative here and designed a wall mounted feature piece to display the LEDs. I took inspiration from dv8 Studio, an artist I discovered online and adapted the piece to suit my set-up and the materials I had available. The materials used include, Perspex, plyboard, copper paint, and some sign display hangers, as well as a variety of tools to cut and drill the materials. I used a standard length of cat 5e ethernet cable to wire the connection. The result can be seen in the image below.



Before placing your TV back into place, make sure to count and take note of the total number of LEDs on each of the AmbiLight and music visualiser/mood light strips, as well as the number of LEDs along the top, sides, and bottom of the TV. Also make sure to count the number of LEDs that would fit within the gap at the bottom of the TV, between where the LED strip starts and ends. These numbers will be needed later in configuration settings.

6. LED Music Visualisation

Documentation consulted for this section

https://github.com/scottlawsonbc/audio-reactive-led-strip
https://www.instructables.com/id/How-to-Record-Voice-With-USB-Microphone-Then-Play-/
https://github.com/Toms42/audio-reactive-led-strip/commit/fa492bbffc13cc59820ffff1bf8767daad969620
https://github.com/Toms42/audio-reactive-led-strip/commit/fa492bbffc13cc59820ffff1bf8767daad969620
https://github.com/Toms42/audio-reactive-led-strip/blob/master/python/microphone.py
https://github.com/scottlawsonbc/audio-reactive-led-strip/issues/42
https://github.com/scottlawsonbc/audio-reactive-led-strip/issues/110
https://www.raspberrypi.org/forums/viewtopic.php?t=215428

Now that the LEDs are wired up and the hardware is installed, we can move onto getting those LEDs to actually do something. We'll start with the music visualiser.

First make sure the 3.5mm microphone is plugged in correctly to your USB soundcard and is functioning correctly. You can follow the steps at this link to ensure you are getting correct microphone input/output https://www.instructables.com/id/How-to-Record-Voice-With-USB-Microphone-Then-Play-/.

6.1. Software and Dependencies

Once the microphone is capturing audio the next step is to get the required Raspberry Pi WS2812B Adafruit libraries and python dependencies that the music visualisation software relies on.

Install the python dependencies needed for the software to work first.

```
$> sudo apt-get update
$> sudo apt-get install python-numpy python-scipy python-pyaudio
```

Next, we install the WS2812B libraries. I used the one-line installation command created curtesy of Core Electronics – https://core-electronics.com.au/tutorials/ws2812-addressable-leds-raspberry-pi-quickstart-guide.html.

```
$> curl -L http://coreelec.io/33 | bash
```

At this point you can test the LEDs by running the strandtest.py script inside the rpi_ws281x/python/examples directory.

```
$> cd rpi_ws281x/python/examples
$> sudo nano strandtest.py
```

Once this file is opened for editing you will see configuration options at the top. Change the number of LEDs to the number of LEDs on your music visualiser strip. The rest of the settings should be fine default. Save and exit the file.

Now, the moment of truth, run the strandtest.py script.

```
$> sudo python strandtest.py
```

If everything is wired up correctly, the LED strip will light up and begin displaying a series of test patterns. If not, and assuming no errors where reported due to incorrect configuration etc, you will need to troubleshoot your wiring and connections, making sure to follow the wiring diagram in Section 5.1, as well as ensuring your solder connections are solid.

With the LED libraries installed and working we now need to install the software to power the visualisation. The software being used in this project was developed by Scott Lawson and can be found on his Github repository https://github.com/scottlawsonbc/audio-reactive-led-strip. Download, copy or clone this repo to your Pi. I placed the contents into a directory called *music_visualiser* under my */home/pi* directory.

6.2. Configuration

Now onto the configuration. First, we need to set the USB sound card as the default audio device. Part of this was already done in Section 4.2, however you may also need to edit the alsa.conf file to change the default audio device to the USB sound card. (This step may not be required, I did not need to do it, but your results may vary).

```
$> sudo nano /usr/share/alsa/alsa.conf
```

Change

```
defaults.ctl.card 0
defaults.pcm.card 0
```

To

```
defaults.ctl.card 1
defaults.pcm.card 1
```

Next, we need to set a few system parameters to ensure the best support for the LEDs, as they are run using PWM. Again, this may not be necessary, I simply list it here for completeness. I would suggest trying without these settings enabled first, then come back and try them only if problems occur. First, open up the /boot/config.txt file.

```
$> sudo nano /boot/config.txt
```

Find and uncomment/add the following lines.

```
hdmi_force-hotplug=1
hdmi_force-edid_audio=1
```

Find and comment out/disable the following lines.

```
dtparam=audio=on
diable_audio-dither=on
```

Save and exit the file. Now, open up the following file and add the line below.

```
$> sudo nano /etc/modprobe.d/raspi-blacklist.conf
```

```
blacklist snd_bcm2835
```

And that's that.

Now we need to change some of the config options for the visualisation software. Navigate to the repository clone you downloaded to your Pi and open up the config.py file located in the python directory. For example, if you placed the repo in the *music_visualiser* directory you would enter the following command.

```
$> nano ~/music_visualiser/python/config.py
```

In this file you will need to set the following values; line numbers proceed the attributes in parenthesis for ease of finding them within the file.

```
(36) LED_INVERT = False
(45) DISPLAY_GUI = False
(48) DISPLAY_FPS = False
(51) N_PIXELS = 50 (the number of LEDs you have in your music visualiser strip)
(103) MIN_VOL_THRESHOLD = 1e-2
```

Save and close the file once these changes are made.

Now a few changes need to be made to the microphone.py file as the Pi has issues with audio buffer overflow using this software. These changes are curtesy of Tom Scherlis (Toms42) on Github, and the full file edit contribution can be found here: https://github.com/Toms42/audio-reactive-led-strip/blob/master/python/microphone.py. Open up the microphone.py file located in the python directory of the repo.

```
$> nano ~/music_visualiser/python/microphone.py
```

Edit as follows.

Change line (19) to :
`y = np.fromstring(stream.read(frames_per_buffer, exception_on_overflow=False), dtype=np.int16)`

Add a line after line (20) and enter the following:
`stream.read(stream.get_read_available(), exception_on_overflow=False)`

Finally, we can now change what visualisation style we want the software to incorporate. The default is Spectrum. You can see each of the effects in an example video found here: https://www.youtube.com/watch?v=HNtM7jH5GXg. If you do decide to change the effect, open up the visualization.py file located in the python directory of the repo. On line (251) you can change the effect:

```
visualization_effect = visualize_spectrum
```

And that's it, all set. You can now start up the visualiser by running the virtualization.py script from the command line while playing some music out of your speakers.

```
$> sudo python ~/music_visualiser/python/virtualization.py
```

If all went well, you should see the LEDs light up respond to changes in the music. If not, double check your configuration and make sure your hardware connections between the Pi, the power supply and the LED strip used for the music visualiser are solid.

Once the visualisation is functioning as intended, we still need a way to clear the LEDs when the visualiser is turned off, as it often leaves LEDs displaying the colour they were outputting when the process was terminated. In order to do this a simple custom script needs to be created. I used the Raspberry Pi WS2812 libraries strandtest.py script as a base and stripped it back to the bare minimum for clearing colour from the LEDs.

The custom script named led_clear.py can be found by following the GitHub link below. Essentially the script simply instantiates a Neopixel object and the sets the LEDs colour to nothing (0,0,0), which causes them to display no colour at all, quite simple really. Again, I would suggest placing this script within your scripts directory that you created in Section 2.2.

https://github.com/nicholasthorondor/Project-Pi/blob/master/led_clear.py

For now, the script can be activated by simply calling it from the command line.

```
$> sudo python ~/scripts/led_clear.py
```

In Section 10, which details the web control panel, the script will be used in multiple places to ensure the LEDs are cleared when changing between LED modes or shutting down/rebooting the Pi, meaning that no residue, or unwanted colour will be displayed through the LEDs.

7. LED Ambilight

Documentation consulted for this section

https://www.raspberrypi.org/magpi/program-arduino-uno-raspberry-pi/
https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-installation
https://hyperion-project.org/threads/diy-amblight-project-guide-hyperion-ws2801-ws2812b-apa102.8/
https://hyperion-project.org/wiki/HyperCon-Guide
http://awesomepi.com/part-2-let-there-be-light-installing-ambilight-software-hyperion-in-openelec

https://www.instructables.com/id/Ambilight-System-for-Every-Input-Connected-to-Your/
http://awesomepi.com/part-3-using-ambilight-for-every-hdmi-device-an-ultimate-step-by-step-tutorial/

We now come to one of the best parts of the project – AmbiLight. The AmbiLight feature will allow the LEDs on the back of your TV to dynamically reflect the colour at the edge of the image being displayed, creating a more immersive visual experience, especially in video games. The AmbiLight feature will be accomplished using the Hyperion software running on the Pi but will also heavily rely on the Arduino UNO to drive the LED colours. Before going any further it is important to check that the USB video capture device being used is compatible with Hyperion. I recommended in the parts list that the device must have a UTV007 Fushicai chipset. This is what worked for my set-up, after having purchased a video capture device using the MACROSIL chipset and not having it work. It may also be possible to use a STK1160 chipset, though this is only from what I have read, I can not be certain. For absolute certainty go for the UTV007 Fushicai chipset.

So in order to find out what chipset the video capture device is running on we can run either of the following commands on the Pi. Of course, make sure the video capture device is plugged into the powered USB hub and is actually on first.

```
$> lshw
```

You can also use the following command, though it will most likely need to be installed first.

```
$> sudo apt-get update
$> sudo apt-get upgrade
$> sudo apt-get install lsusb
```

```
$> lsusb
```

These commands will bring up a list of the hardware attached to the Pi. Find the video capture device in the list and take note of its chipset name. It should say UTV007 and/or Fushicai. If not then it is likely your video capture device is not running a recognised chipset, and therefore will not work. If this is the case, I would recommend continuing on with guide and see if Hyperion functions by the end of this section. If not, you will likely need to purchase a new USB video capture device, specifically with the specified UTV007 Fushicai chipset. On the other hand, if it s showing up under that chipset name then all is good, carry on.

7.1. Set-up Arduino

The first step is to set-up the Arduino UNO. Make sure the Arduino UNO is plugged into the powered USB hub using its USB cable, which in turn is plugged into the Pi. Also ensure that the connections between the Arduino UNO and the LED strip are correct, as shown in Section 5.1. In order to get the LED strip on the back of the TV to work with the Hyperion software installed on the Pi (detailed in Section 7.2), we need to create a sketch (script) on the Arduino UNO. Before we can do that however, we need to install the Arduino IDE so we can actually write the sketch.

```
$> sudo apt-get update
$> sudo apt-get upgrade
$> sudo apt-get install arduino
```

With the Arduino IDE installed we can now run a test to ensure the Arduino Uno is interfacing correctly with the LEDs. We will need the Arduino NeoPixel libraby in order to run this test. This can be found at the following GitHub page.

https://github.com/adafruit/Adafruit_NeoPixel

Download this repo and transfer it onto your Pi. Once the repo is on your Pi, rename it to *Adafruit_NeoPixel,* and move this renamed repo to the *~/sketches/libraries* directory.

```
$> mv Adafruit_NeoPixel-master Adafruit_NeoPixel
$> mv Adafruit_NeoPixel ~/sketches/libraries
```

Now to get this test sketch running we need to open the Arduino IDE. This can be done by navigating to Menu->Programming->Arduino IDE from the Pi's desktop (click the raspberry in the top left corner of the desktop). Once the IDE is open navigate to the *Tools* menu in the top of the IDE. In this menu make sure the settings in *Board* and *Port* are correctly set. The board should be set to Arduino UNO, while the port will be dependant upon the serial port device name used to plug the Arduino UNO into the powered USB hub. In my case this was called /dev/ttyUSB0. With those settings configured we can no run the test sketch. Navigate to the *File->Examples->Adafruit_NeoPixel-strandtest* from the top bar of the IDE. This will open up a code editor. Change the Pin number on line 6 to the Pin you soldered the data line connection to. If you followed the diagram in Section 5.1 this will be Pin number 6 and can be left default. Next, change the first parameter of the function call on line 16 to the number of LEDs in your LED strip on the back of the TV, then ensure the RGB channel order setting is correctly set in parameter three. By default, this will be GRB, but can be changed if your LED strip has RGB channel order. Finally, verify the code by clicking the 'tick' button in the top left of the code editor window. This will make sure the code has no errors. Once that is done, click the Upload button (right facing arrow) next to the Verify button to upload the sketch to the Arudino UNO. Once the upload is done, and assuming everything is working as intended, the LEDs should begin displaying the test patterns, and will continually cycle through while the Arduino UNO is powered on. As usual, if you encounter errors or problems double check configurations and connections to ensure everything is correct.

Now that we have run a successful test it is time to create the actual sketch used to get the AmbiLIght functionality working. We will be using the FastLED library for this sketch. Download the library repo from the following GitHub link, and place it on your Pi as usual, rename it to *FastLED*, and place it in the *~/sketches/libraries* directory, as you did with the NeoPixel library.

https://github.com/FastLED/FastLED/archive/master.zip

```
$> mv FastLEd-master FastLED
$> mv FastLED ~/sketches/libraries
```

Next, download the sketch from the following GitHub link.

https://github.com/nicholasthorondor/Project-Pi/blob/master/hyperion_uno.ino

Once you have finished editing the script for your configurations, place it onto your Pi in the *~/scripts* directory. Open the Arduino IDE, load the sketch you just placed in the scripts directory by going to *File->Open* and navigating to the ~/scripts directory. Once the sketch is loaded in the code editor change the number of LEDs on line 4 to match your own number, as well as the data pin and RGB channel order if they differ from default, on line 9 and 11 respectively. Now verify the code, and then upload the sketch to the Arduino UNO as described earlier for the NeoPixel test. Once the upload is done the LEDs should display red, green and blue in sequence before going blank. This means the sketch is working, it is just waiting for the other aspect of the set-up – Hyperion.

7.2. Hyperion

Now that the Arduino UNO is set-up and ready to go we work on installing and configuring the Hyperion software so the LEDs actually reflect the colour of the image on screen. First, we are going to need to install the Hyperion software. I created a directory called *hyperion* on my Pi to keep everything neat.

```
$mkdir ~/hyperion
```

Clone the Hyperion repo onto your Pi. Note this is entered as one line on the command line.

```
$> sudo curl -L --output install_hyperion.sh --get
https://raw.githubusercontent.com/tvdzwan/hyperion/master/bin/install_hyperion.s
h
```

Move this file to the newly create Hyperion directory if it not already here.

```
$mv install_hyperion.sh ~/hyperion
```

Now install Hyperion by running the installation script.

```
$> sudo sh ./install_hyperion.sh
```

Next, install Java. This may already be installed on your Pi.

```
$> sudo apt-get install openjdk-7-jre
```

For the last bit of software, you will need to install HyperCon to actually configure Hyperion settings. The download can be found at the following link.

https://sourceforge.net/projects/hyperion-project/files/hypercon/HyperCon.jar/download

Move this .jar file to your Pi, placing it in the *hyperion* directory.

Now we are ready to begin configuring Hyperion. Open HyperCon by entering the following command from the *hyperion* directory.

```
$ java -jar ./HyperCon.jar
```

You will be greeted with a configuration GUI with a number of tabs at the top. Following is a list of the settings that need to be changed according to each tab. If a setting is not mentioned, simply leave it as its default value.

Let's begin configuring.

HARDWARE

Device

- Change '**Type'** to Adalight
- Change '**Output'** to the USB serial name of the Arduino UNO, eg. /dev/ttyUSB0
- Change '**Baudrate'** to 500000
- Change '**RGB Byte Order'** to GRB or RGB depending on what your LED channels are

Construction

- Set '**Direction'** LEDs run in, clockwise or counter clockwise, ie. LED flow direction from FRONT of TV.
- Check '**Led top right**' etc if your LEDs are all the way into the corner of the TV otherwise leave unchecked.
- Enter number of LEDs across '**LEDs Horizontal**', '**LEDs Right**', '**LEDs Left**' and '**Bottom Gap**'
- Change '**1st LED Offset**' to match where the strip begins on you tv setup. Use the visual display for a guide. When LED number 0 is at the same place on the visual representation as it is on your TV, that is the correct offset.

Black Border Detection

- Ensure '**Black Border Detection**' is enabled. Set '**Threshold**' to 14. This is due to the video grabber displaying blacks as a grey, so the value needs to be quite high to register the black bars on the top and bottom of a traditional movie display.

PROCESS

Smoothing

- Change '**Update Frequency [Hz]**' to 25 ms.

GRABBER

Internal Frame Grabber

- Uncheck '**Enabled**'

Grabber V4L2

- Check '**Enabled**'
- Set '**Device**' to /dev/video0
- Set '**Video Standard**' to PAL
- Set '**Frame Decimation**' to 2
- Set '**Size Decimation**' to 4
- Set '**Crop**' values (will need to experiment with values, process discussed below)
- Set each colour '**Signal Threshold**' to 0.1

EXTERNAL

- Disable everything

SSH

- Leave all default unless you want to SSH into the Pi to change Hyperion settings

Save the Hyperion configuration file to the *hyperion* directory by clicking the '**Save**' button at the bottom of the GUI. Now click the '**Create Hyperion Configuration**' button and save the config file to the *hyperion* directory for safe keeping.

The crop settings under the '**Grabber**' tab of the Hyperion configuration will depend upon on local set-up. You may not have to enter any values at all, though it is best to check anyway. You will need to take a screenshot of what the USB video grabber is capturing in order to see if there are any black borders around the image.

First, the Hyperion process cannot be running when taking a screenshot. Find the hyperiond process id and kill it.

```
$> sudo pid hyperion
$> sudo kill pid
```

Now run the screenshot function.

```
$> sudo hyperion-v4l2 --screenshot
```

This will take a screenshot and place it in your working directory (the directory you ran the command from). Open it up in the image viewer and see how the image is displayed. If there are black bars you will need to enter crop values in the Hyperion configuration. You can experiment to find the correct values by providing them as parameters to the screenshot command. For example.

```
$> sudo hyperion-v4l2 --screenshot --crop-left 10 --crop-right 10 --crop-bottom
10 --crop-top 10
```

The crop values I ended up with are as follows.

```
Left:21
Right:16
Top:10
Bottom:10
```

If there is no image at all when taking a video grabber screenshot, ie. a black image, try restarting the Pi and other devices and repeating the process. If that doesn't work, double check your configuration settings and physical connections. Failing all that, you may have problems with a device in the architecture, such as your video grabber or HDMI to AV converter.

Before we can start using Hyperion properly, we need to copy the freshly created configuration file to the correct Hyperion directory. Assuming you created the configuration file in the *~/hyperion* directory issue the following command.

```
$> sudo cp ~/hyperion/hyperion.config.json /etc/hyperion/
```

Now restart the Hyperion deamon for the changes to take effect.

```
$> sudo systemctl restart hyperion
```

Assuming all went well, the initial red, green and blue colours of the sketch we made earlier will first appear, then the Hyperion colours will kick in and reflect the display on the TV.

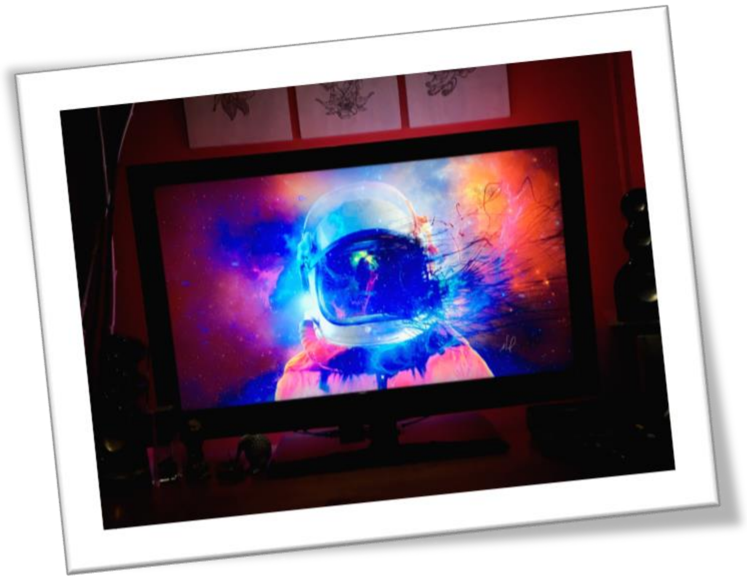
Hyperion works off the hyperiond deamon, so we need to control that to turn it on and off at will. That will be covered in the Section 10 detailing the the web control panel. Until then we can stop the Hyperion service auto starting at boot by using the following command.

```
$> sudo systemctl disable hyperion.service
```

If you want to manually start the Hyperion service again before we create the web control panel enter the following.

```
$> sudo systemctl start hyperion
```

And that's it, all done, you now have a fully functional AmbiLight clone. If not, well you know the drill by now, double check your config settings and physical connections and make sure everything is correct.



8. Image Slideshow

Documentation consulted for this section
https://www.raspberrypi.org/forums/viewtopic.php?t=181916
http://manpages.ubuntu.com/manpages/bionic/man1/fbi.1.html
https://www.raspberrypi-spy.co.uk/2017/02/how-to-display-images-on-raspbian-command-line-with-fbi/

With Hyperion installed and set-up we can turn our attention to the creation of the image slideshow. The slideshow is designed to act as an ambient visual feature, displaying a selection of images on the TV screen, while the AmbiLight responds to each image creating a sort of enlarged, backlit, digital picture frame. The slideshow will be completely scalable regarding the number of images that it can display. Meaning that more images can simply be added, or current images removed, from the directory the slideshow draws from over time, with no configuration changes needed.

8.1. Software and Scripts

To begin, find a selection of images that you wish to use in the slideshow. It looks best when the images resolution matches that of your TV exactly, as no black borders will be present. Place these images onto the Pi. In my case I created a directory called *slideshow* under the *Pictures* directory of the Pi, a dn placed the images inside this new directory.

```
$> mkdir ~/Pictures/slideshow
```

Now, onto the software. The software used to display the images is an open source application called 'fbi'. First, install the fbi package.

```
$> sudo apt-get update
$> sudo apt-get install fbi
```

With that done a custom script can now be created to invoke the 'fbi' software to display all images within the previously created slideshow directory, randomly, on a 30 second interval, in full screen mode. The script can be found at the following GitHub link.

https://github.com/nicholasthorondor/Project-Pi/blob/master/slideshow.sh

This script can be tested from the command line as follows. Use the escape key to exit from the slideshow.

```
$> cd ~/scripts
$> ./slideshow.sh
```

Assuming everything worked and AmbiLight is also active, the slideshow should begin and the LEDs on the back of the TV will adapt to the image displayed.

While the slideshow can be exited using the escape key, this requires input from a keyboard. As the project uses a web-based control panel as detailed in Section 10, we need a way to stop the slideshow programmatically from a script. The following script does just that and can be found at the GitHub link below.

https://github.com/nicholasthorondor/Project-Pi/blob/master/stop_slideshow.sh

8.2. Black Border Removal

Now that the slideshow is set-up and functioning you may have noticed that images are being displayed with black borders, even if they are the exact same resolution as your TV. If this is not happening then feel free to skip this section, if it is, read on.

The problem occurs due to the Pi running on a different resolution than that of the TV. It is fairly simple to fix, only requiring an edit or two within the /boot/config.txt file.

First, open up this file.

```
$> sudo nano /boot/config.txt
```

Find and uncomment the following line within this file.

```
disable_overscan=1
```

Save and exit the file. Now reboot the Pi. Upon rebooting the Pi should now be displaying in the correct native resolution of your TV. When running the slideshow script, the images should now fill the entire screen, with no black bars present, assuming of course their resolution matches that of your TV exactly. If not, you may need to go back into the /boot/config.txt file and uncomment the *overscan* properties for each side of the screen just below the line you just changed earlier. The numbers that apply if you need to take this route will need trial and error as it depends on your TV. Use negative values if there is a black border being displayed. Unfortunately, you will need to reboot the Pi to see the result of each change, so it can take some time.

9. LED Mood Lighting

Documentation consulted for this section

https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library-use

The mood light functionality is actually fairly simple, in fact you can entirely skip this section if you are content using the Hyperion app to control your mood lightning using the LEDs on the back of your TV. I actually created this aspect of the project before I knew Hyperion had a companion app, so it is your choice whether or not to implement the mood lights as shown here, if so, read on.

First, it is necessary to create a custom python script, which uses the base Raspberry Pi WS2812 strandtest.py script installed in Section 6.1 and edits it into a simplified form. The custom script essentially instantiates a Neopixel object and then simply takes three command line arguments in the form of the RGB decimal values (in that order) and sets the colour of the LEDs accordingly. The script, which I have named mood_light_colour.py is available from the GitHub link below. Again, I would suggest placing this script within the *scripts* directory created in Section 2.2.

https://github.com/nicholasthorondor/Project-Pi/blob/master/mood_light_colour.py

You can test this script by calling it from the command line, providing RGB values as arguments.

```
$> sudo python ~/scripts/mood_light_colour.py 255 0 0
```

It should activate the LEDs to display whatever colour those RGB values represent. The script has been designed to continually run in the background, so that it can be identified as a running process later in the web control panel scripts, and turned on/off accordingly, so make sure to kill the process once you are done testing. The led_clear.py script created in Section 6.2 can also be called to clear the LEDs of any colour once you are done.

How to stop the mood light script via command line.
Find process id number of mood_light_colour.py

```
$> ps -ef
```

Kill the process using *pid* discovered above

```
$> sudo kill pid
```

Clear LEDs of residue colour

```
$> sudo python ~/scripts/led_clear.py
```

Calling from the command line directly should work fine, however it is not what we are ultimately looking for. The goal is to hook the mood light functionality into the web-based control panel detailed in section 10. As that section details the code required to make that work, I will not go into detail here, suffice to say that once the web-based control panel has been setup, the mood lights will be able to have colour selection performed via a graphical colour wheel, can be turned on and off at will, and will save the last used colour to a file on the Pi, meaning the colour you used last time will not have to be reselected every time you turn the mood lights on. If you feel this last part about saving the colour to a file is not secure enough for you, then you could think about editing the code and using a cookie to save the colour in place of this method.

Keep in mind that LED strips can come in two varieties, the first being standard RGB channels, and the second being GRB channels. The only difference is that the green channel is reversed with the red, which will cause colours to display incorrectly unless the channels are also reversed in the script. If you are finding the colours displayed by the LED strip do not match the RGB values you are entering then you may have an GRB channel strip. In my case my LEDs are GRB, therefore I need to reverse the red and green input parameters when calling the script. In the supplied script in Section 10 detailing the web control panel, I have reversed the green and red when activating the mood lights to ensure the colour displays correctly. If your LEDs are GRB nothing needs to be changed, however if your

LEDs are RGB then just switch the passed in red and green arguments in the *mood light* (line 135) and *apply colour* (line 156) sections of the control.php script.

10. Web Based Control Dashboard

Documentation consulted for this section
https://howtoraspberrypi.com/how-to-install-web-server-raspberry-pi-lamp/
https://httpd.apache.org/docs/2.4/howto/access.html
https://community.spiceworks.com/topic/835435-restarting-a-linux-server-via-web-browser-php-resolved
https://stackoverflow.com/questions/4400154/deny-all-allow-only-one-ip-through-htaccess
https://iro.js.org/

So far, we have installed a bunch of software and created scripts that essentially provides all the functions the project set out to do. However, the control of these functions is rather cumbersome, having to call scripts from the command line, enable/disable system services etc. What will really finish off this project is the ability to control the Pi via another device, such as a smart phone, tablet or PC. Being familiar with web development, and as I will only be accessing the Pi from within my local network, I chose to implement a local web-based control panel to accomplish this. This is probably not the best solution if security is paramount as certain privileges need to be granted to the system Apache user in order to control aspects of the Pi. In my case this was not a concern, and certain security measures were taken, however other options such as creating an app for your device, or SSHing into the Pi and forgoing the control panel altogether could be another solution for the security minded, or those with unsecure local networks. These other aspects will not be covered in this guide however.

10.1. Web Server Install, Setup and Configuration

Before creating the web app, the Pi needs to be set up as a web server. This is done by installing Apache and PHP.

First, check that the Pi is up to date.

```
$> sudo apt-get update
$> sudo apt-get upgrade
```

Next, install the Apache web server.

```
$> sudo apt-get install apache2
```

Once the server is installed, permissions need to be set. In this case read, write, execute permissions for the pi user and the www-data group for the root web server directory.

```
$> sudo chown -R pi:www-data /var/www/html/
$> sudo chmod -R 770 /var/www/html/
```

This will allow the web server the permissions to run scripts and access files stored on the web server, with a little help from the sudoers file. Keep in mind that if you create a new directory within, you may need to reapply these permissions to ensure the new directory is affected.

Proceeding with the permissions setup, the sudoers file must be edited to grant the required application execution permissions to the www-data user for commands which will be used in the control panel. The reason for this is that the PHP commands within a web application are run as the web server user on the server machine, in this case the user is www-data. This user does not have permission to execute commands related to controlling the Pi, such as rebooting and shutting down. In order to get around this, permissions must be given to the www-data user so that those commands can be executed when the PHP script is run. This is where the potential security hole is introduced, and though the .htaccess file created in Section 10.2 will help to secure the Pi's server, it may not be enough in the case of some users and the environments in which they operate. If this is the case it may be best to look into creating or purchasing an app that accomplishes similar functionality, or as mentioned earlier, not implement a control panel at all, and rather control the Pi using simple SSH.

Open the sudoers file.

```
$> sudo visduo
```

Add this entry just under the root user entry.

```
www-data ALL=(root) NOPASSWD: /sbin/reboot, /sbin/shutdown, /usr/bin/python,
/bin/kill, /bin/systemctl, /usr/bin/fbi
```

What this entry does is give root permissions to the www-data user, while also not requiring a sudo password when executing the comma separated commands. Do not forget the comma between the command paths. I spent much too long debugging a non-existent error due to missing this little comma.

Now that the server is installed and permissions are set, PHP needs to be installed so web scripts can be run.

```
$> sudo apt install php php-mbstring
```

10.2. Security
Before the control panel application can be scripted, Apache needs to be secured to only allow access to the Pi web server from devices you set. This will be done via a .htaccess file at the root of the web server.

In order for the .htaccess file to be valid, Apache needs to be set to allow .htaccess files to override core server settings. This can be set in the Apache config file.

Open the config file.

```
$> sudo nano /etc/apache2/apache2.conf
```

Find the line that a contains a *Directory* directive specifying the /var/www/ path and change the *AllowOverride* from *None* to *All*. The entry will look similar to the following.

```
<Directory /var/www/>
    ….
    AllowOverride All
    ….
</Directory>
```

Now .htaccess files will be valid in any directory under the /var/www/ path, meaning the .htaccess file can now be created.

The purpose of this file is to secure the Pi's web server from any device other than the ones specified as allowed within this file. In order to do that in the most secure manner it is best to set the devices you plan to use to access the Pi's web control panel with static ip addresses. The process to accomplish this varies depending on network setup and the router being used, but will most likely involve finding the MAC address of the Pi and the devices you wish to access the Pi from, and setting a static ip address, from the interior range assigned to you by your ISP, against each MAC address in your router settings. The exact process will be up to you to work out as each router differs.

The other option, which is significantly less secure, is to allow the entire range of ip addresses that your router assigns via DHCP to be allowed. If this is the method chosen, a partial ip address would be entered in place of the full ip address shown below. For example, instead of 192.168.0.10 you would enter 192.168.0, which would allow access to hosts in the 192.168.0.1-192.168.0.254 range. It is best to consult the Apache documentation if you are unsure https://httpd.apache.org/docs/2.4/howto/access.html.

With that out of the way, change to the web server root directory and create the .htaccess file.

```
$> cd /var/www/html

$> sudo nano .htaccess
```

Enter the following directives, replacing the ip addresses with each individual ip of your devices.

```
<RequireAll>
    Require ip 192.168.0.10 192.168.0.20
</RequireAll>

Options -Indexes
```

Once that is done only those devices specified will be able to access the Pi's web server, and directory listing will be prevented, meaning contents of a directory will not be listed which adds a small extra layer of security.

All done, the Pi's web server is now setup and secured, and we are ready to begin coding the actual control panel.

10.3. Control Panel Code

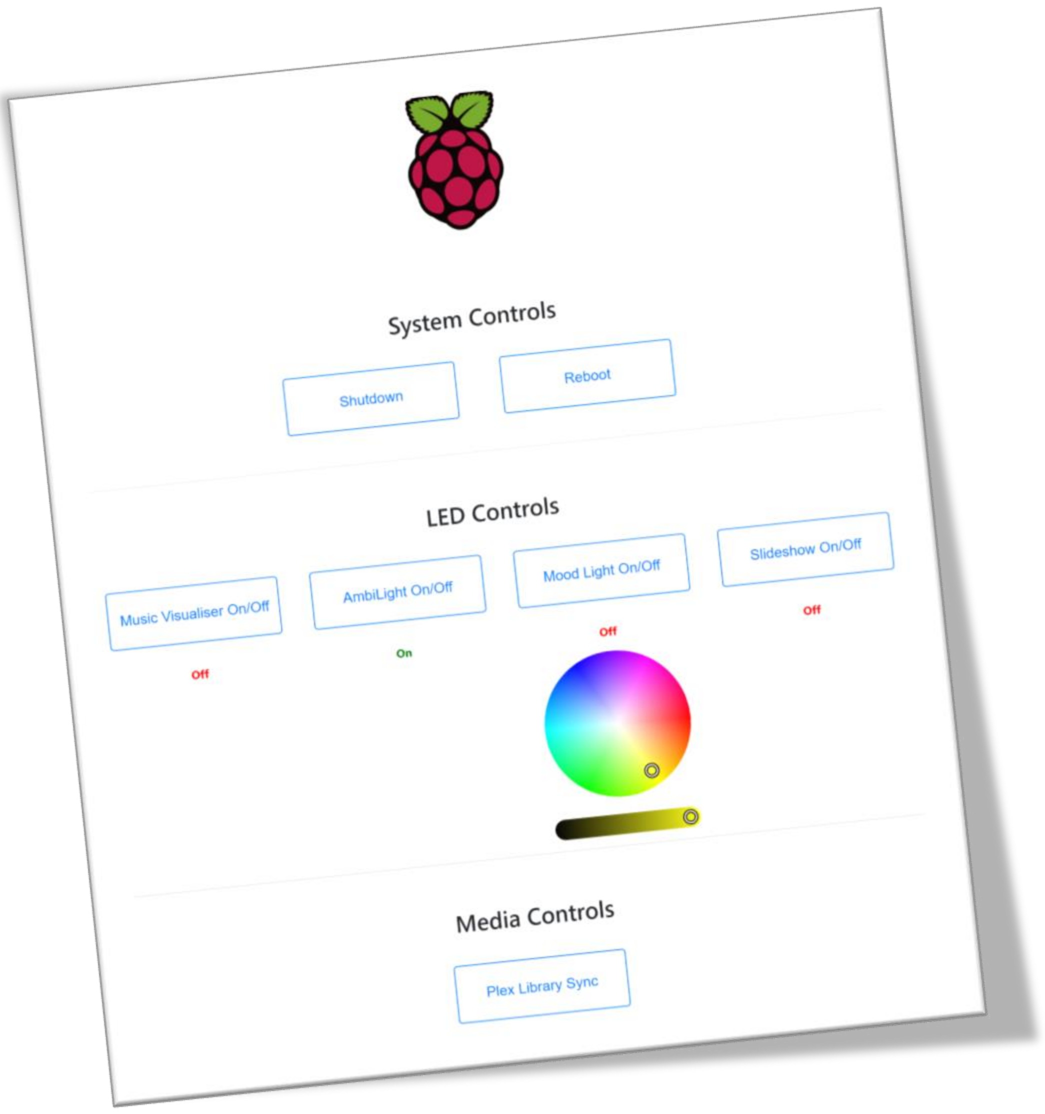
The control panel is simple in design, containing various buttons and a colour wheel to control and change the applications running on the Pi. The functionality that will need to be coded are as follows:

- Shutdown
- Reboot
- Music Visualiser on/off
- Ambilight on/off
- Mood Lighting on/off
- Mood Lighting Colour Selection
- Plex Library Sync
- Ambilight Slideshow on/off

This functionality will be implemented using PHP running on top of a standard HTML/CSS/JS GUI. The BootStrap framework will be used to ensure ease of layout and responsiveness between devices. Each of the scripts that have been created in previous sections to turn on/off the aspects of the system or sync the Plex library will be incorporated into this PHP code and linked to the GUI buttons. The colour wheel used is a plugin developed by James Daniel - https://iro.js.org/. Keep in mind the pathnames in many of the executed commands may differ depending on how you set up your system. Therefore, take note to change any should your scripts be kept in a different directory or similar. The code for the control panel can be found on GitHub at the following link. Copy this code to your Pi and place it within the /var/www/html directory.

https://github.com/nicholasthorondor/Project-Pi/tree/master/raspicontrol

And the final result is a swish new Pi control panel.



10.4. Final Config Changes and Backup

Once all the hardware and software has been setup and installed correctly and everything is functioning as normal, there is one more setting on the Pi that can be changed.

By default the Pi has a power saving mode for the Wi-Fi adaptor. This can cause slowdown or potential network dropouts in some cases. Therefore, I like to turn this function off. I have created a simple script that runs on system boot turning off the Wi-Fi power saving mode using a cron job. There may be another way to do this through the network settings, but that is not my forte, and this works just fine for my needs.

The script can be found at the following GitHub link.

https://github.com/nicholasthorondor/Project-Pi/blob/master/wifi_power_save_off.sh

Place this script within your *scripts* directory, eg. */home/pi/scripts*, then create a cron job, by adding the following line to your crontab, substituting the path to your script if it is different.

```
$> crontab -e

@reboot /home/pi/scripts/wifi_power_save_off.sh
```

Once everything is complete and working, I would recommend creating a backup copy of the Pi's filesystem by copying the contents of its SD card, just so if something goes wrong down the track you will be able to restore without having to repeat all the steps in this guide. If you are using Windows the Win32 Disk Imager software will do the job for you, otherwise Linux and Mac users can use the command line.

II. The END

And that's it, project done! I hope you have enjoyed creating it as much as I did. Now go forth and enjoy some LED fuelled awesomeness.