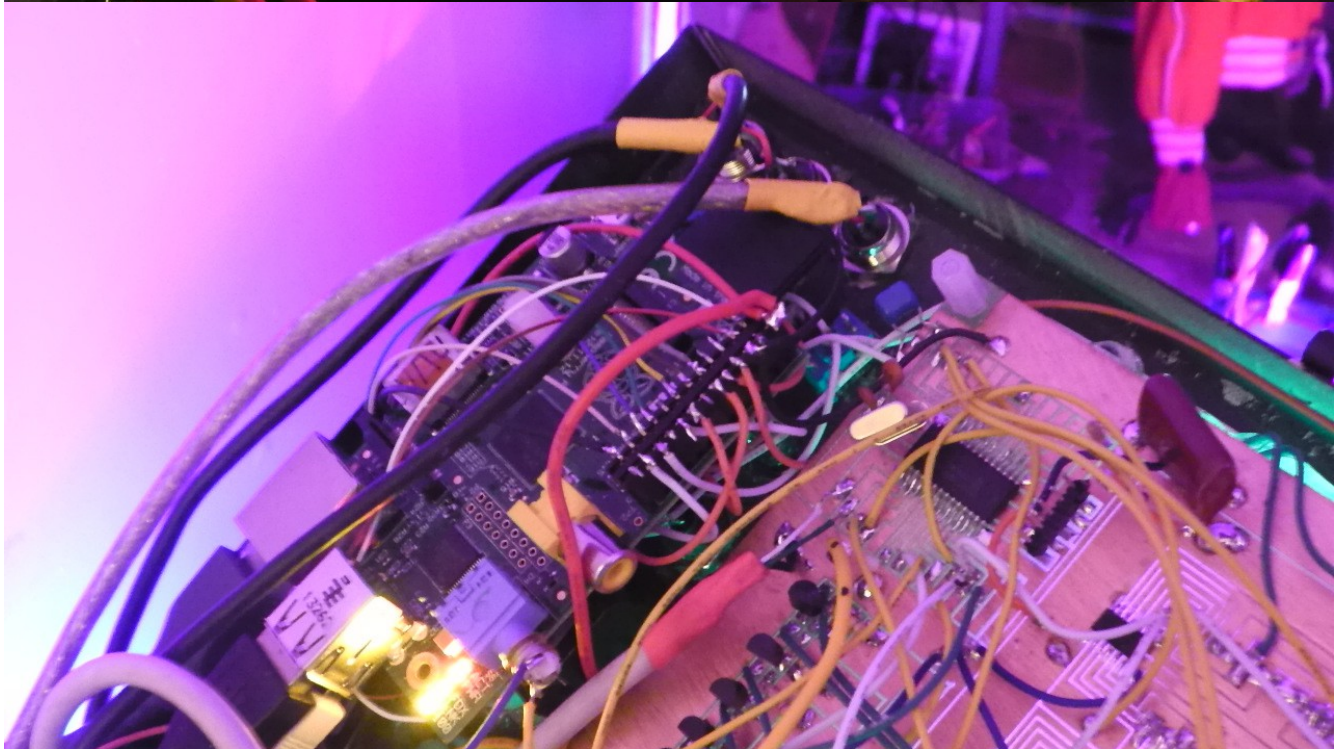
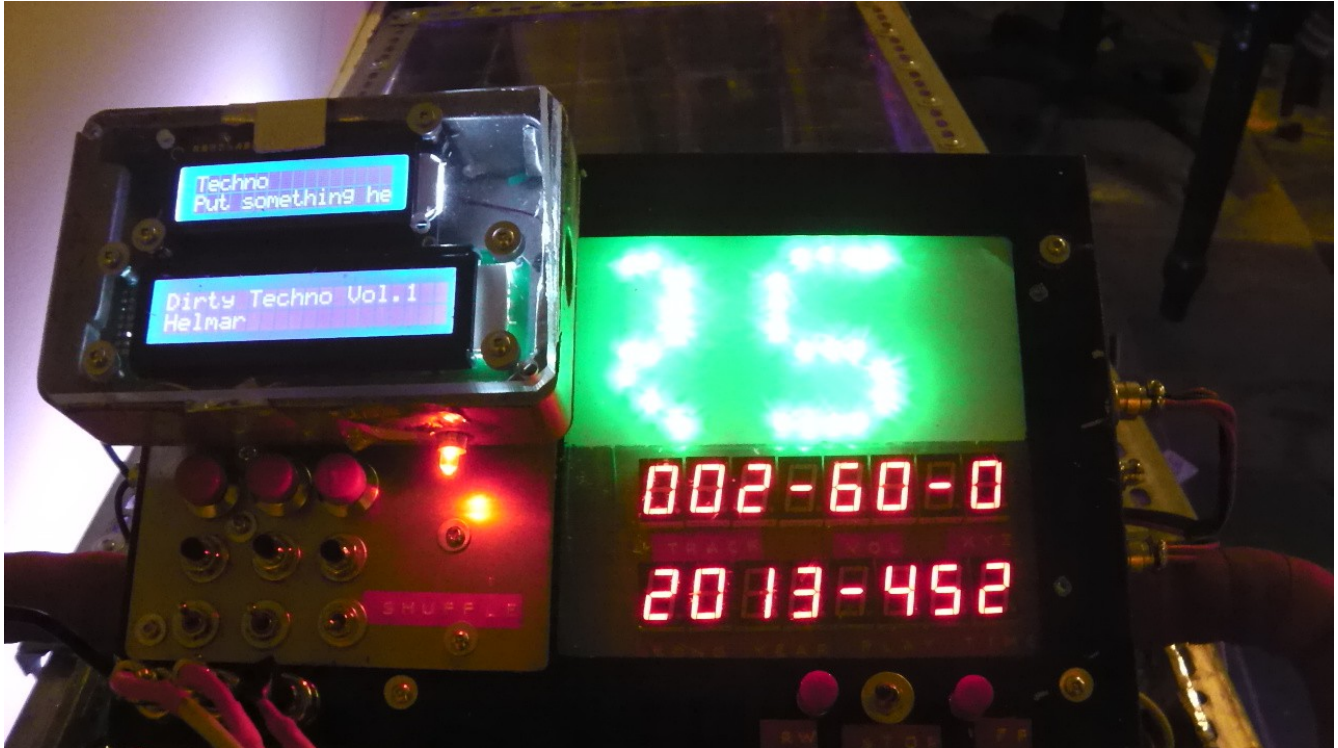


# Raspberry Pi based MP3 jukebox with USB display board

2014 Russell Kramer



## Project Goals:

- 1) Launch a Raspberry Pi Python MP3 player script on power up.
- 2) Control volume and song selection with a button matrix
- 3) Extract ID3 tags and send the data to display hardware through USB

## **Background**

Prior to this project I created a LED Rave shopping cart tricycle. This project featured a mobile set of speakers and LED display boards capable of responding to audio sources and displaying text. I found myself irritated with off the shelf MP3 players for two main reasons: 1) None of them would simply turn on and start playing music. They all required me to navigate through various menus and loading screens beforehand. 2) There was no easy way to extract information such as the song title and integrate it into the lighting display. This problem had a few solutions: I could use a microcontroller and MP3 codec chip or hack the firmware of a store bought MP3 player. Ultimately I decided to use a raspberry pi with a software based MP3 codec. At \$50 for the RPi and an 8Gb flash card it cost less than buying an MP3 player or the components required to build one from scratch.

## **Note regarding Rpi audio DAC.**

I have seen numerous sources claiming that the RPi's internal audio DAC is of such low quality it should not be used to play music. This is not true. The Rpi has an eleven bit PWM based DAC, which is comparable to the original line of iPods. The reason for its poor reputation is a hardware quirk not enough people are aware of:

Most audio DACs have a secondary volume control stage. Audio data is fed into the DAC at 100% volume, then the output of the DAC is cut down by the volume stage. The Rpi does not use a secondary volume stage due to its inexpensive simplified design. If the volume is set to 25% (the default for some OSs) the audio drivers will divide data by four before sending it to the DAC. This means the two most significant bits of the data stream are always going to be zero. This turns the RPi's 11-bit DAC into a 9-bit DAC that sounds terrible. People encountering this problem typically solve it by adding an external USB audio DAC rather than investigating the problem.

## **ArchArmLinux**

For this project I decided to use ArchLinuxARM. The selling point of this OSs is that it can boot up in under six seconds due to its minimalist design. This is a very important feature because I wanted to avoid a lengthy delay between hitting the project's power button and a song starting. To further improve startup time I used an 8gb Class 10 (high speed) SD card. ArchLinuxARM is only 2gb, so a 4gb SD card is an option, but this will leave space for only about 100 MP3 files. Someone wanting to use this project with a very large MP3 collection should consider using a 16gb or 32gb SD card.

## Archlinux Setup:

### Required equipment:

- An SD card interface for your computer
- an SD card (recommended 8gb Class 10).
- An Ethernet router or hub.
- Ethernet cable to connect RPi to hub

First follow the installation instructions listed here:

[archlinuxarm.org/platforms/armv6/raspberry-pi](http://archlinuxarm.org/platforms/armv6/raspberry-pi).

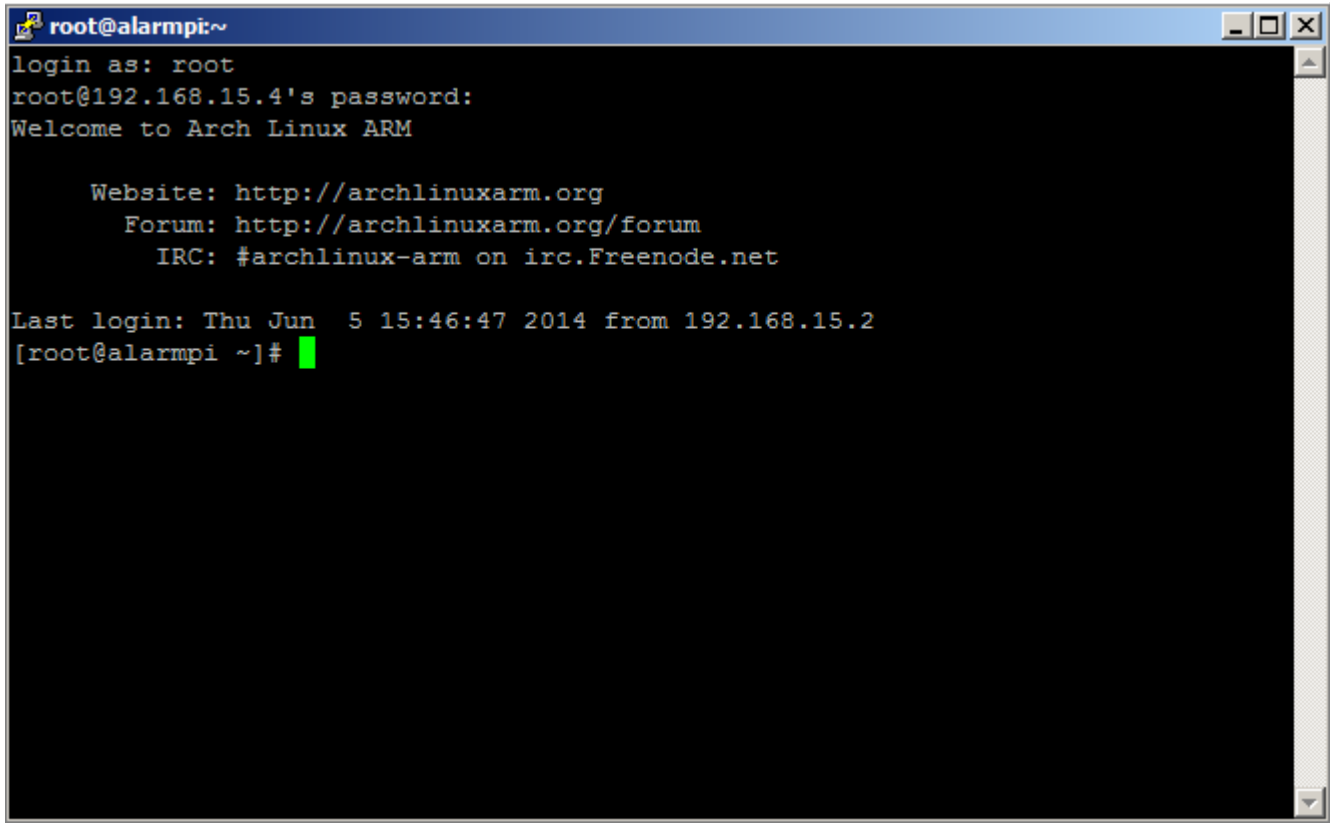
Use the default root username or your filesystem will not match the one presented in this tutorial.

Archlinux runs an SSH server by default. This will allow you to control the RPi through an Ethernet network instead of a monitor and keyboard connected directly to it. The RPi's IP address is assigned through DHCP. You can discover it by logging in directly (keyboard and monitor to the RPi), then entering "*arp -a*". I prefer to use a program called Angry IP Port Scanner ([angryip.org](http://angryip.org)) to search my network for the RPi.

IP	Ping	Hostname	Ports [0+]
192.168.15.1	0 ms	[n/a]	[n/s]
192.168.15.2	0 ms	kramerr-PC	[n/s]
192.168.15.3	[n/a]	[n/s]	[n/s]
192.168.15.4	1 ms	[n/a]	[n/s]
192.168.15.5	[n/a]	[n/s]	[n/s]
192.168.15.6	[n/a]	[n/s]	[n/s]
192.168.15.7	[n/a]	[n/s]	[n/s]
192.168.15.8	[n/a]	[n/s]	[n/s]
192.168.15.9	[n/a]	[n/s]	[n/s]
192.168.15.10	[n/a]	[n/s]	[n/s]
192.168.15.11	[n/a]	[n/s]	[n/s]
192.168.15.12	[n/a]	[n/s]	[n/s]
192.168.15.13	[n/a]	[n/s]	[n/s]
192.168.15.14	[n/a]	[n/s]	[n/s]
192.168.15.15	[n/a]	[n/s]	[n/s]

I know the RPi is 192.168.15.4 because there are only three machines on my network. 192.168.15.1 is the router, 192.168.15.2 is my PC.

Once the Rpi is located download Putty ([chiark.greenend.org.uk/~sgtatham/putty](http://chiark.greenend.org.uk/~sgtatham/putty)) and use its SSH server to connect to the RPi's IP.



```
root@alarmpi:~
login as: root
root@192.168.15.4's password:
Welcome to Arch Linux ARM

      Website: http://archlinuxarm.org
      Forum: http://archlinuxarm.org/forum
      IRC: #archlinux-arm on irc.Freenode.net

Last login: Thu Jun  5 15:46:47 2014 from 192.168.15.2
[root@alarmpi ~]#
```

## Installing Libraries

Download and install Python2:

```
“sudo pacman -S python2”
```

Download and install Rpi GPIO library:

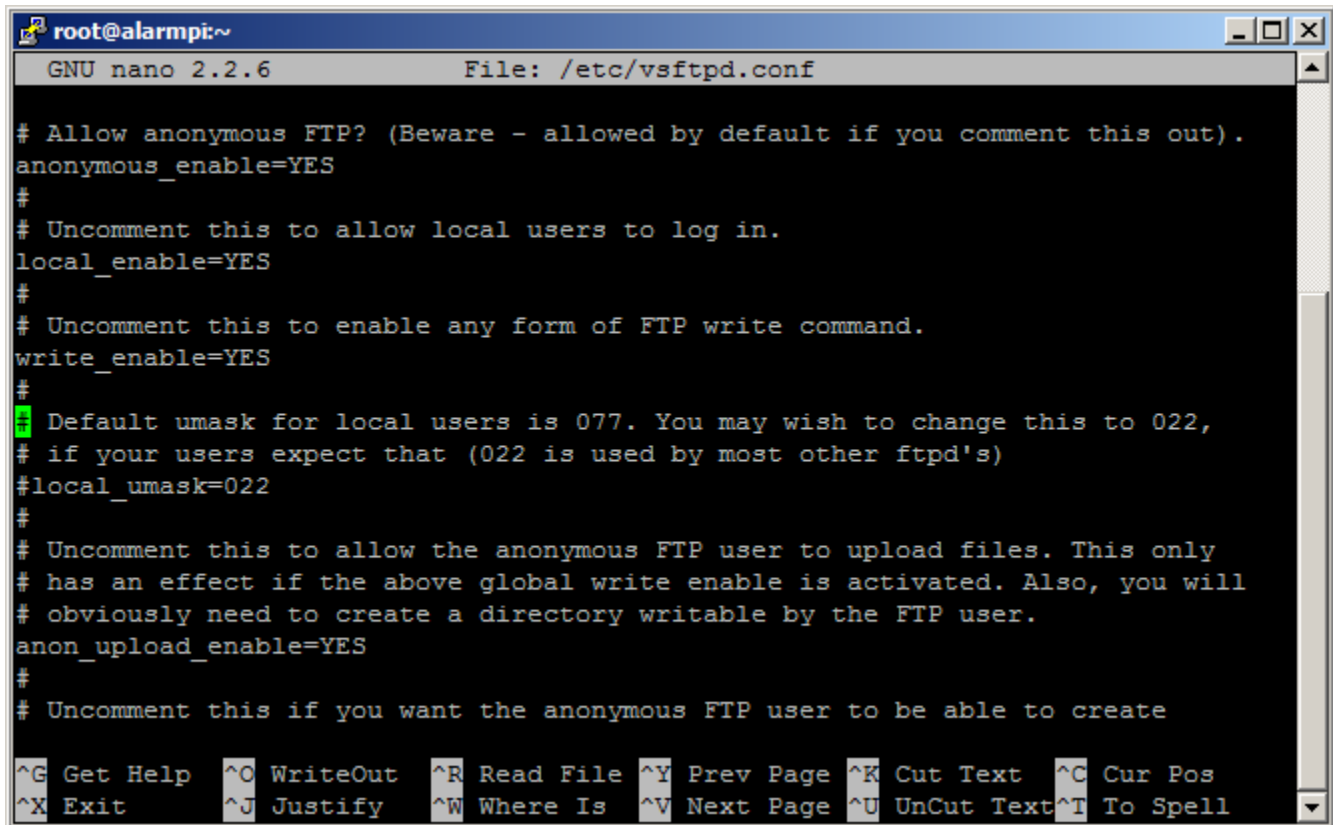
```
“sudo pacman -U http://myplugbox.com/raspberry-gpio-python-0.4.1a-1-any.pkg.tar.xz”
```

Download and install Mutagen MP3 library:

```
“sudo pacman -U https://bitbucket.org/lazka/mutagen/downloads/mutagen-1.23.tar.gz”
```

## Installing FTP server.

Next we are going to create an FTP server using VSFTP. This will allow MP3 files and python scripts to be transferred to the Rpi. Enter “*sudo pacman -S vsftpd*”. Enter “Y” when the package manager asks you to confirm the install. Next modify the configuration file to allow remote users write permission. Enter “*sudo nano /etc/vsftpd.conf*” and modify the file to match the following screenshot.



```
root@alarmpi:~
GNU nano 2.2.6 File: /etc/vsftpd.conf

# Allow anonymous FTP? (Beware - allowed by default if you comment this out).
anonymous_enable=YES
#
# Uncomment this to allow local users to log in.
local_enable=YES
#
# Uncomment this to enable any form of FTP write command.
write_enable=YES
#
# Default umask for local users is 077. You may wish to change this to 022,
# if your users expect that (022 is used by most other ftpd's)
#local_umask=022
#
# Uncomment this to allow the anonymous FTP user to upload files. This only
# has an effect if the above global write enable is activated. Also, you will
# obviously need to create a directory writable by the FTP user.
anon_upload_enable=YES
#
# Uncomment this if you want the anonymous FTP user to be able to create
```

^G Get Help   ^O WriteOut   ^R Read File   ^Y Prev Page   ^K Cut Text   ^C Cur Pos  
^X Exit   ^J Justify   ^W Where Is   ^V Next Page   ^U UnCut Text   ^T To Spell

Use *ctr-O* to save the file, then set the VSFTP to launch on startup with the command “*sudo systemctl enable vsftpd.service*”. Reboot the Rpi to activate VSFTP. You can now connect to the Rpi through Filezilla Client ([filezilla-project.org](http://filezilla-project.org)).

## Adding the Python project.

Using filezilla create the folder “env\_python” in “/usr/bin”. You can also do this using the mkdir command from the SSH terminal. FTP the following files into env\_python:

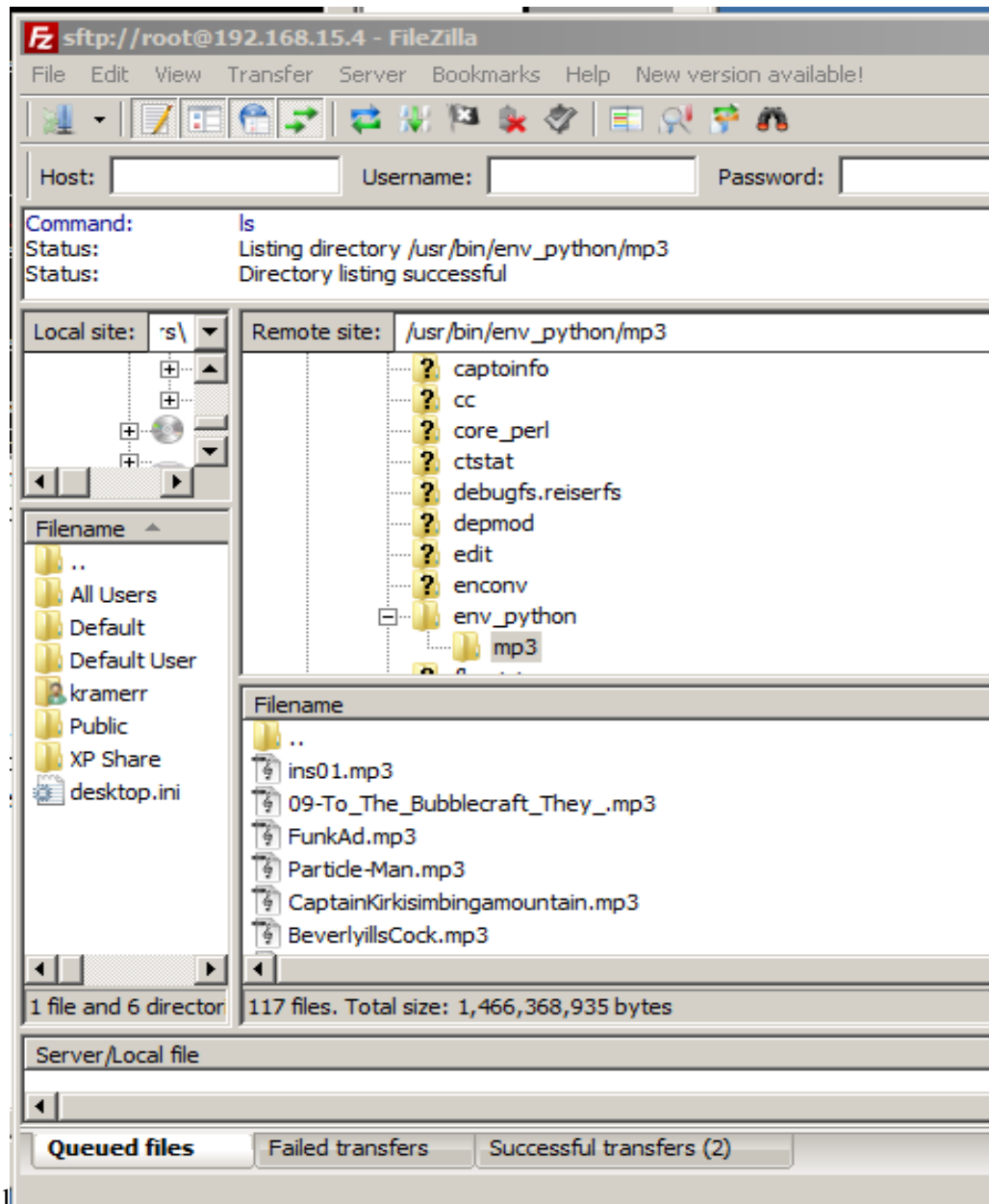
Adafruit\_CharLCD.py , CartBikeControl.py , button-1.mp3.

- **Adafruit\_CharLCD\_dual.py** is a modified version of the Adafruit LCD library capable of running two LCDs in parallel.
- **CartbikeControl.py** is the main script.
- **button-1.mp3** is a simple beeping sound used to indicate a successful startup. (source: SoundJay.com)



## Adding the MP3 files.

Within the “env\_python” directory create a directory named “mp3”. FTP your MP3 files into this directory. Make sure the MP3 filenames do not contain spaces or slashes. Linux does not handle those well.



## Adding Startup script for CartBikeControl.py:

The following command will create a new startup script file and open it for editing:

```
“sudo nano /etc/systemd/CartBike.service”
```

Copy the following text into the contents of the file:

```
[Unit]  
Description=Launches cartbike python script  
After=syslog.target network.target  
[Service]  
Type=simple  
ExecStart=/usr/bin/python2 /usr/bin/env_python/CartBikeControl.py  
RemainAfterExit=yes  
  
[Install]  
WantedBy=multi-user.target
```

Save the file with ctrl-O then exit nano with ctrl-X.

Enable the script file with the following command:

```
“systemctl enable /etc/systemd/CartBike.service”
```

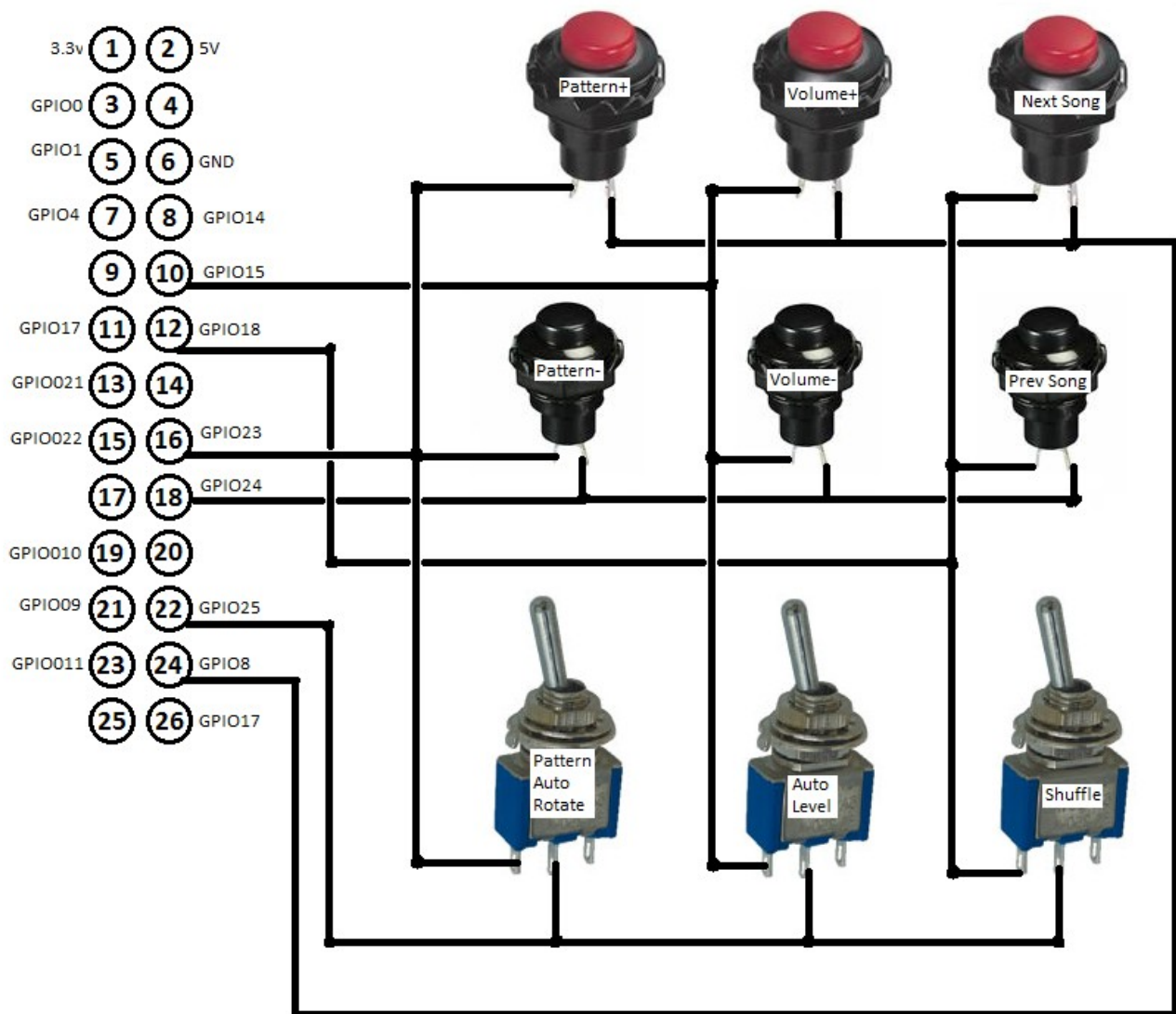
Reboot the Rpi and you should first hear the button-1.mp3 sound effect, then each of your mp3 files played in alphabetical order.



## Adding button matrix:

The button matrix allows you to control the volume and song number of the MP3 files. The pattern number deals with what pattern external lighting equipment uses, and will not be discussed here. The buttons are connected to the Rpi through a matrix, because this allows 9 buttons to occupy only six GPIO pins. An additional row or column of buttons can be added with only the cost of 1 more input button.

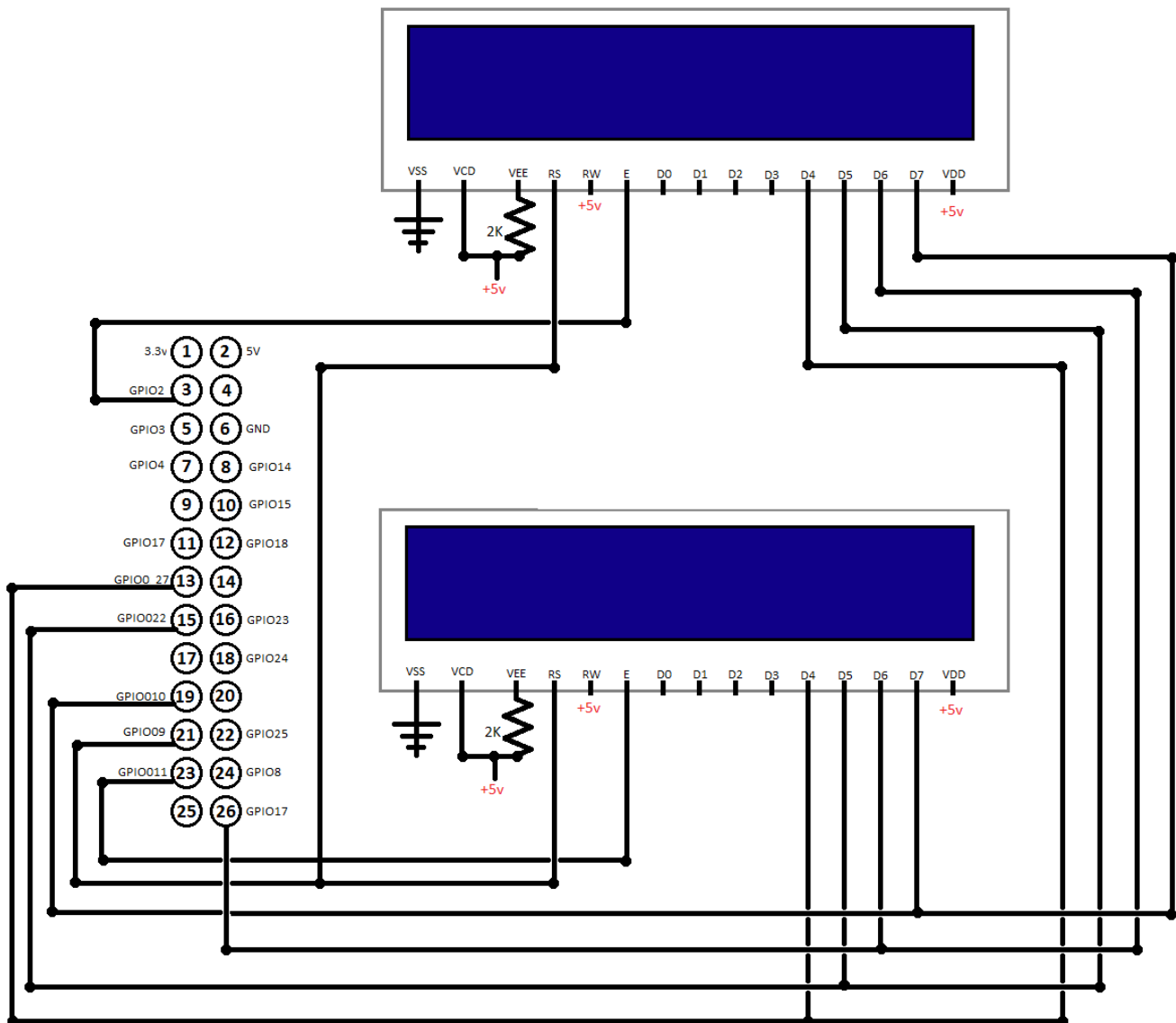
I tried to keep a consistent pattern when assigning buttons to functions. The top row are red pushbuttons which increment a variable. The middle row are black pushbuttons which decrement a variable. The bottom row are toggles which enable some option like cycling through songs randomly instead of sequentially.



## Adding LCD Output.

I had a pair of 16x2 GDM1602K LCDs in my junk drawer, and decided to add them to the jukebox. They display system status, song genre, artist name, and album name on each of the four lines. Genre, artist, and album are extracted from Mp3 ID3 tags.

Adafruit provides a very good GDM1602K python library, however I wanted to run two LCDs in parallel. This means making some minor modifications to the adafruit library. The circuit is shown below. The data and RS wires of each LCD are connected together. Each enable (E) pin is connected to an independent Rpi GPIO output, which allows data to be addressed to each of them individually.



## Adding USB display board.

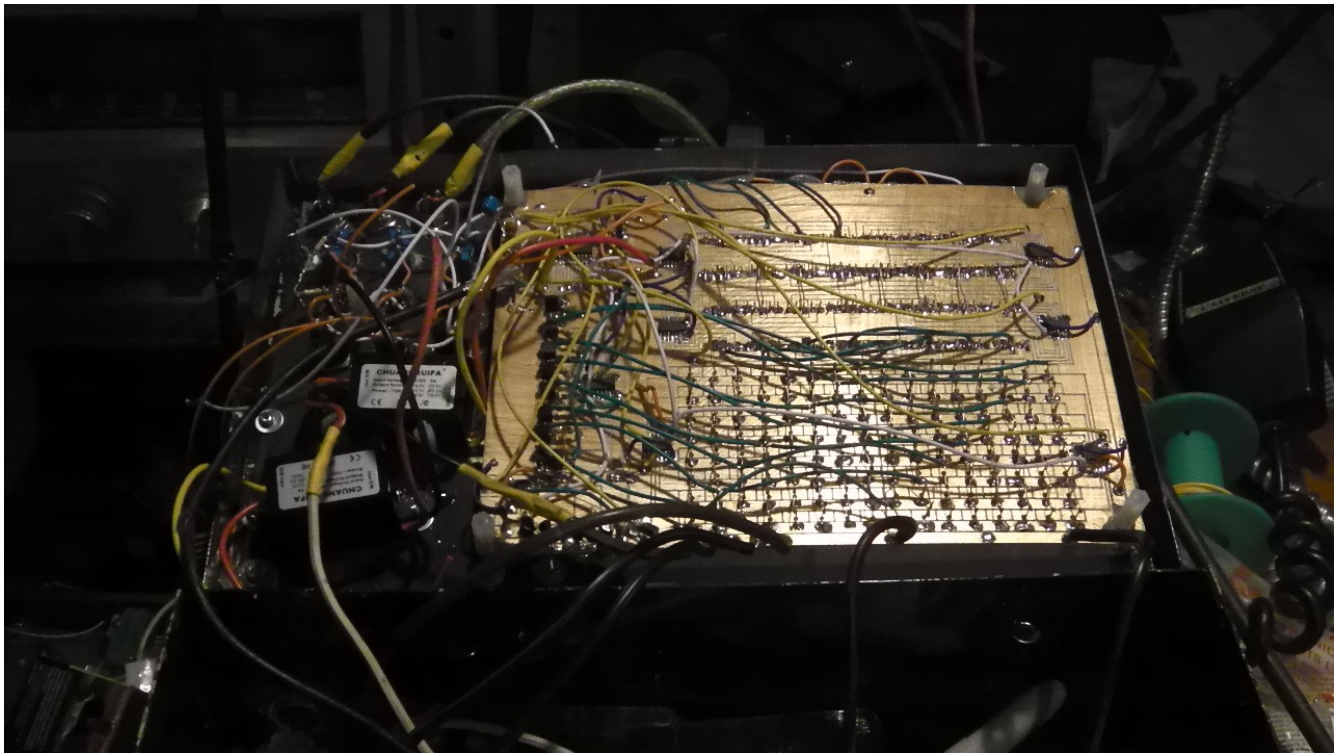
The USB display board is created with a PIC18F2550. This chip has an internal USB transceiver.

Since this was a one-off project I mostly used previously designed breakout boards. I print my boards using only a single layer of copper. This unfortunately means there is a very large number of jumper wires connecting everything.

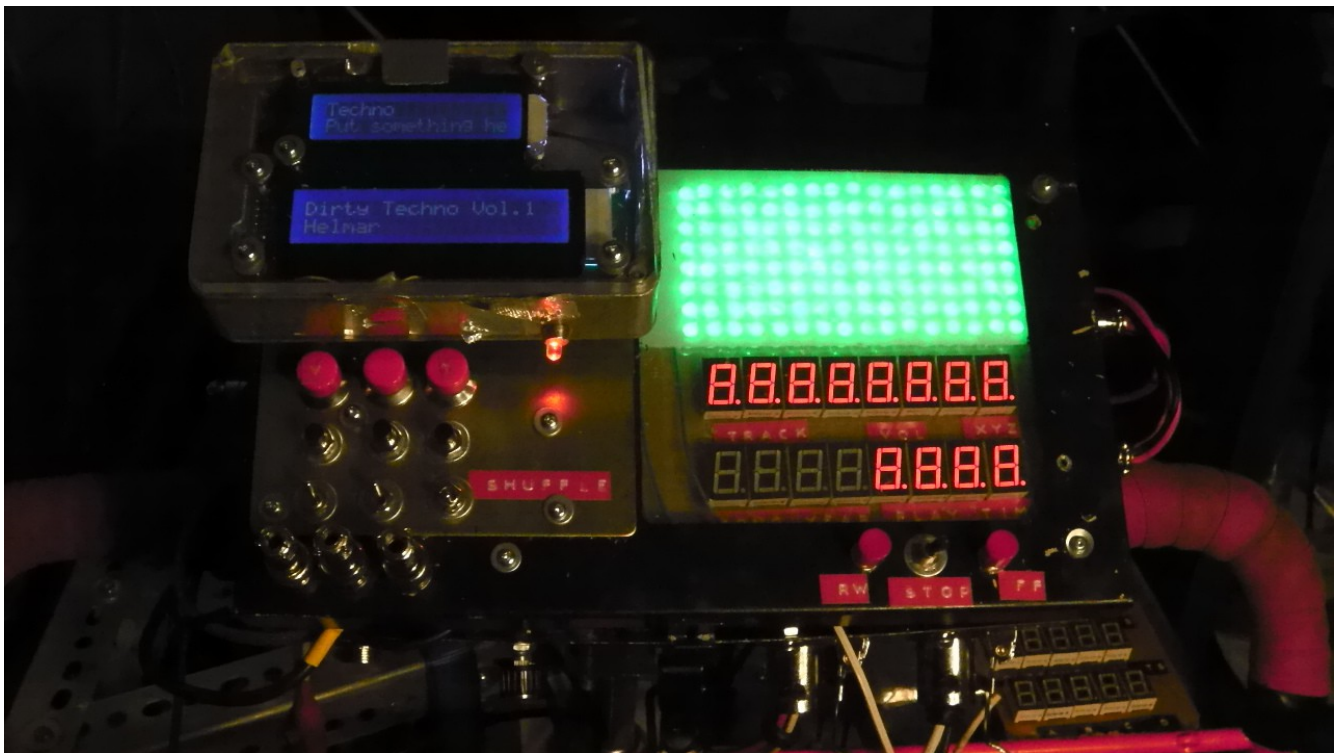
Provided with this project are four PCB files: 4X transistor, 8x transistor, 15F4550BreakoutBoard, LedPanel. There is a .brd file and a .tap file included for both of them. The .tap file is G-code deaigned to be read by routing software like Mach4. The .brd file is an Eagle board file, however these are not designed like regular eagle PCB files. I directly specify the cuts that need to be made in the copper layer rather than dealing with pour layers. Keep this in mind if you are generating your own Gcode or toner transfer sheets.

The DisplayBoardAssemblyDiagram.png explains how to solder components onto the PCBs, and how to connect them together. The parts list is as follows:

Part	QTY
LED 5MM Green 3v, 20mA	128
PIC18F2550 SOIC	1
7-segment LED Display. 0.56" Common Anode 8	16
74HC164 SOIC 6	6
2N3906 TO-92	12
USB Type A plug/cable	1
1k resistor	12
10k resistor	1
20Mhz Xtal	1
470 nF ceramic capacitor	2
22pf ceramic capacitor	2



Assembled USB display Board (Back)



Assembled USB display Board (Front)

## Installing PIC firmware.

After assembling the PIC18F2550 breakout board you will need a PIC programmer such as the PICKIT2 to download the firmware. The PIC programming port is indicated in DisplayBoardAssemblyDiagram.png. Compile the source code with MCC18. The source code is based on the Vellman USB sample program.

