Best Raspberry PI Zero Project For Home Automation Part 1

AWESOME RASPBERRY ZERO PROJECT

Best Raspberry PI Zero Project For Home Automation Part 1

Index Of Contents

Of all the things we do at Raspberry Pi, driving down the cost of computer hardware remains one of the most important. Even in the developed world, a programmable computer is a luxury item for a lot of people, and every extra dollar that we ask someone to spend decreases the chance that they'll choose to get involved.

The original Raspberry Pi Model B and its successors put a programmable computer within reach of anyone with $20-35 to spend. Since 2012, millions of people have used a Raspberry Pi to get their first experience of programming, but we still meet people for whom cost remains a barrier to entry. At the start of this year, we began work on an even cheaper Raspberry Pi to help these people take the plunge.

Today, I'm pleased to be able to announce the immediate availability of Raspberry Pi Zero, **made in Wales** and **priced at just $5**. Zero is a full-fledged member of the Raspberry Pi family, featuring:

- A Broadcom BCM2835 application processor
  - 1GHz ARM11 core (40% faster than Raspberry Pi 1)
- 512MB of LPDDR2 SDRAM
- A micro-SD card slot
- A mini-HDMI socket for 1080p60 video output
- Micro-USB sockets for data and power
- An unpopulated 40-pin GPIO header
  - Identical pinout to Model A+/B+/2B
- An unpopulated composite video header
- Our smallest ever form factor, at 65mm x 30mm x 5mm

Raspberry Pi Zero runs Raspbian and all your favourite applications, including Scratch, Minecraft and Sonic Pi. It is available today in the UK from our friends at The Pi Hut and Pimoroni, and in the US from Adafruit and in-store at your local branch of Micro Center. We've built several tens of thousands of units so

far, and are building more, but we expect demand to outstrip supply for the next little while.

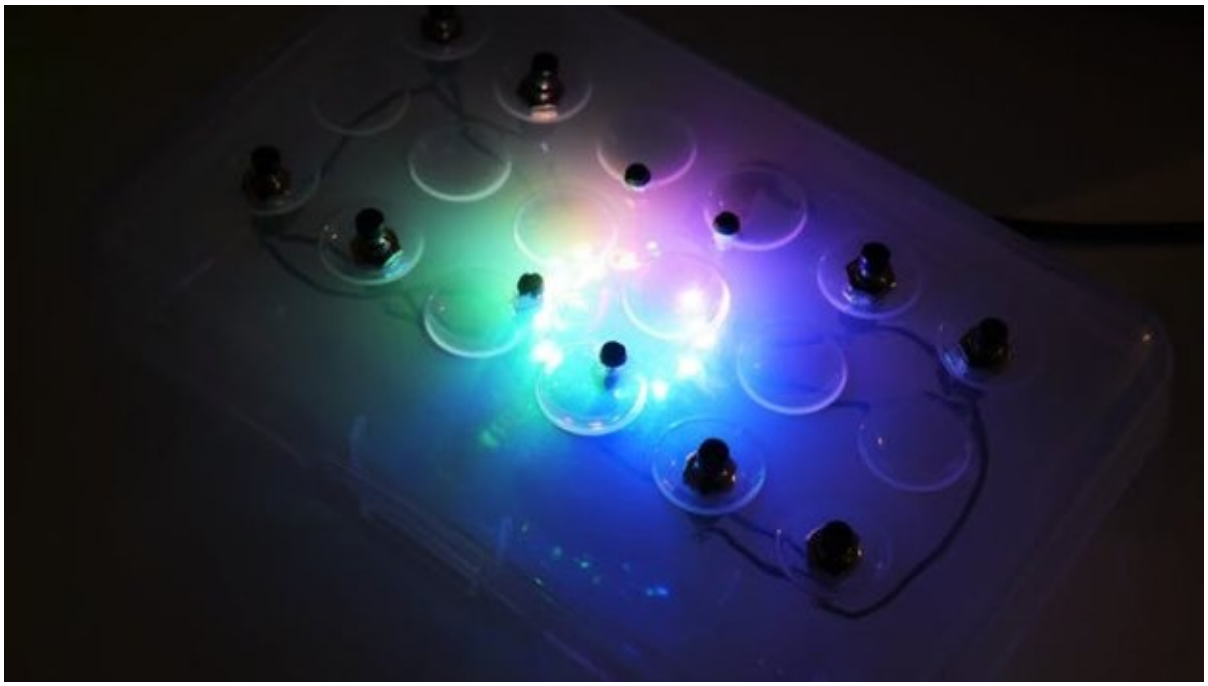
One more thing: because the only thing better than a $5 computer is a free computer, we are giving away a **free Raspberry Pi Zero** on the front of each copy of the December issue of The MagPi, which arrives in UK stores today. Russell, Rob and the team have been killing themselves putting this together, and we're very pleased with how it's turned out. The issue is jam-packed with everything you need to know about Zero, including a heap of project ideas, and an interview with Mike Stimson, who designed the board.

Here Best Raspberry PI Zero For Home Automation Project

Pedal Pi Controller with Circuit Playground

You can find a video for this project on my YouTube channel here.

The Pedal Pi is an open source guitar pedal that uses a Raspberry Pi Zero running Raspbian Lite to run C scripts of digital effects to play around with. It's made by Electro-Smash, an open source hardware company out of the UK. They have some other open source guitar pedals as well, based on Arduino boards but this is their first step into the magical world of Raspberry Pi. They have their own packaged ISO of Raspbian Jessie Lite that includes their self-written effects scripts so that you can change effects via the terminal, since the Lite distro does not have a GUI (of course you can use a version of Raspbian w/ a GUI if that's more your style, but personally for this application I think sans-GUI is the way to go). The only thing that I eventually found to be a bit awkward with the pedal was the act of changing effects. As a guitarist, I'm used to switching things around with my feet so that I can keep playing or not have to awkwardly balance my guitar. Typing into the terminal to change effects was a bit cumbersome so I decided to make a controller to print strings in the terminal to control the Pedal Pi.



Choosing a Board

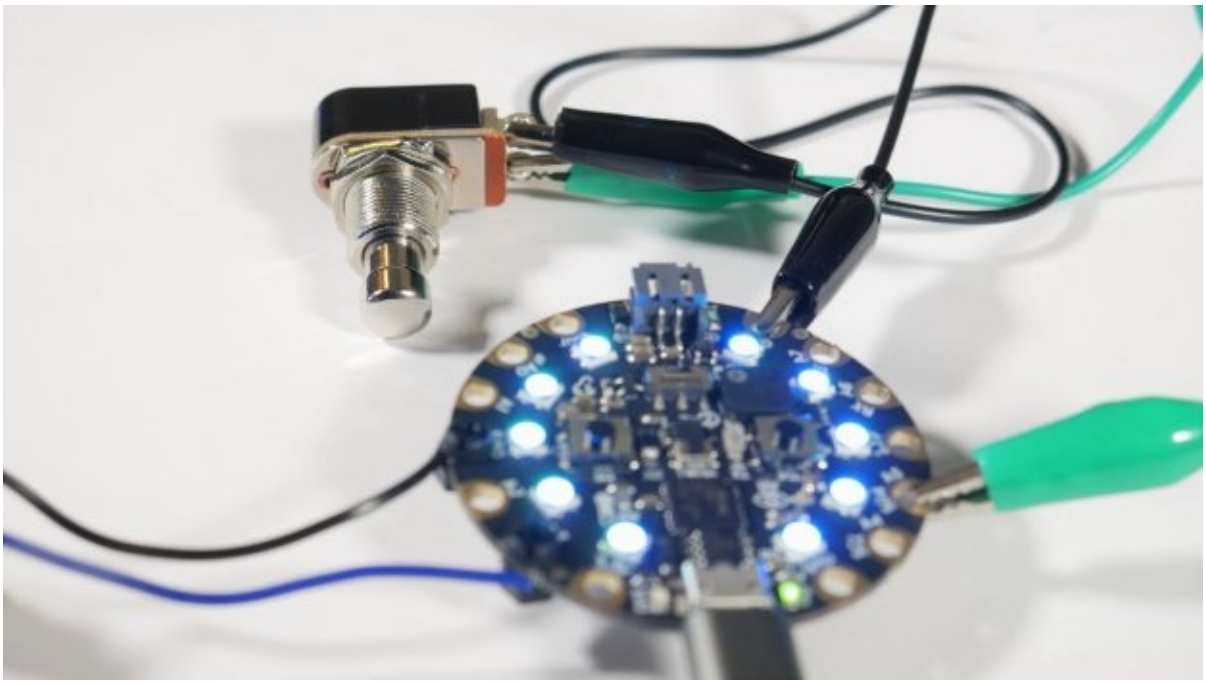
I've been leaning towards Adafruit m0 boards lately for their Circuit Python capability and the built-in USB HID feature without having to flash firmware. For this project I was also going for some aesthetics, namely some LEDs so I

decided to go with the Circuit Playground Express, a circular board that features, well many things (it is a playground after all), but in this case I had my eye on the 10 NeoPixels available to animate. Of course going with the Circuit Playground I wasn't getting the same number of digital I/O that I would with a Metro m0, which I did consider, but I decided instead of loading all of the effects as an option to switch between I would just select my favorites.

Despite the Circuit Python capability with the m0 board though, I did write this in the Arduino IDE instead. I had some issues with printing strings within Circuit Python. It's entirely possible, and more than likely, that it was a mistake in syntax on my part since I am still fairly new to the Python family and syntax is the main aspect of coding that I struggle with. I have some projects in the pipeline that I do hope to use Circuit Python for, but at this time the Arduino IDE seemed to be the easier choice for me for this project.

Speaking of the code, it's available at my GitHub link down below. It can be easily modified for different HID device purposes since this is a very specific need. I go into a lot of detail on how the code is setup and working in the video for this project, which you can find a link for up top where this story started. If you're interested in a write-up detailing the code structure please let me know and I'd be happy to add it to this guide.

Hardware and Housing



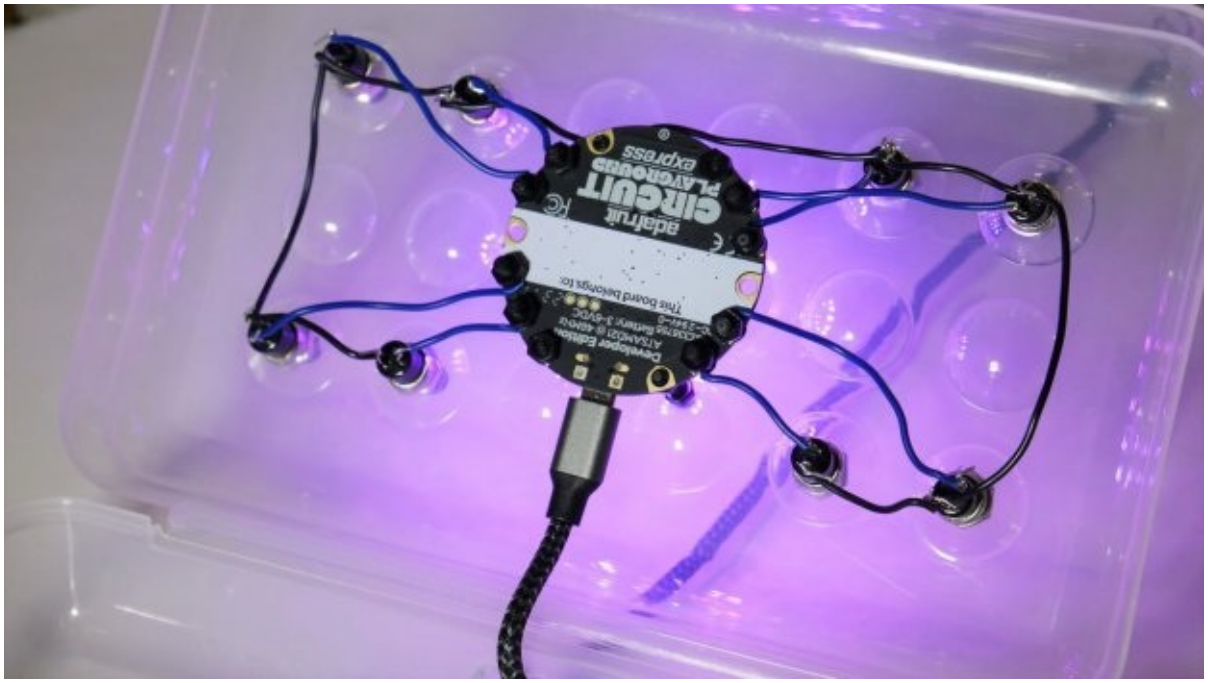
I prototyped my code using aligator clips and a momentary push button that has a similar housing and form factor to a traditional latching switch used in guitar

pedals to turn them on and off. This switch is great and eventually I want to revisit this project and use them for all of the buttons but at this time they're just a bit too pricey, approximately $6-$8/switch. Where I'm using eight switches that adds up fast. I ended up using a different type of momentary button that was a bit smaller but still had a similar form factor, only with a plastic top. I was able to get 10 in a pack for the same price as one of my dream switches so definitely more economical.

For a housing I basically knew going into this that this wouldn't be a final housing. Why? I think this project would benefit greatly from a 3D printed enclosure, which I currently don't have the ability to make. I have this idea for it to be a circle, with the Circuit Playground in the center with the NeoPixels facing up and then the eight switches arranged in a circle around the board; similar in form factor to the RotoVibe pedal made famous by Jimi Hendrix. But right now that isn't an option so instead I opted for a non-traditional housing; specifically a clear plastic pencil box that can be procured for $0.99 to $2 at your local discount store. A few reasons for this: 1. price of course; 2. clear plastic means that the NeoPixels will diffuse quite nicely; 3. It can be easily opened to access the components; and 4. It's easy to mod.

I created the holes in the top of the pencil case in a non-traditional way. When I use a drill in my projects, I'm actually borrowing it from someone else (aka my dad) and he was using his drill while I was working on this project so I had to get creative. I used a sharp and heavy screw to center each hole and then tapped the screw with a hammer to make a dent where I wanted the hole to go. Next, to create the actual hole, I grabbed one of my no longer used for food prep Ikea knives and slowly turned the knife to drill out a hole. It actually worked really well and I had really nice control over the size of the holes, even for the smaller holes to hold the mounting screws for the Circuit Playground board.

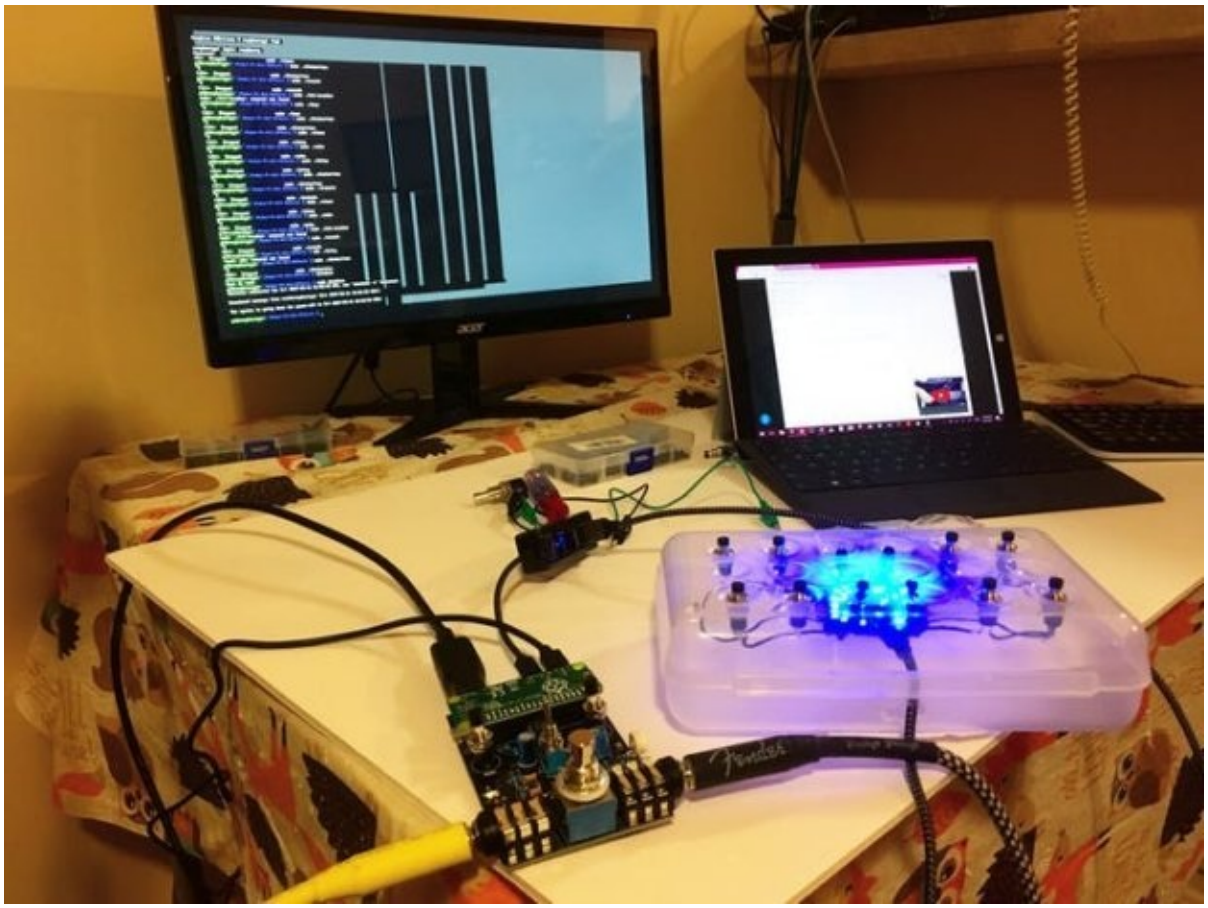
Wiring



In keeping with the minimalist design I decided that I also wanted to avoid soldering for this project. For one thing I'm not finished playing with this Circuit Playground and second as I mentioned previously this isn't the final iteration of this project. Since the Circuit Playground is really designed to be used with alligator clips for quick prototyping I decided to riff on that concept by using some plastic nuts and screws to secure wires to the contact pads around the board.

The switches are also solder free. Because of the mounting solution in the pencil box, the wire is actually tensioned in mid-air around the button terminals, allowing for secure contact. Written out it sounds a bit haphazard but in practice it's actually worked really well. So far zero shorts.

Final Setup



After assembling the hardware and wiring everything up all that's left is to shred with the Pedal Pi sans keyboard (unless you're playing a synth of course). To do this, I first boot up the Pedal Pi and after the boot sequence has finished, it prompts me for a sequence of a username and password. I have one button programmed to send the username string followed by a beefy delay and then the password string followed by a beefier delay. After this, a third string is sent to change into the Pedal Pi effects directory folder. After pressing this button I can switch between effect commands (sudo ./[Fxhere]) by pressing the effect-specific buttons. To end an effect from running I have another button that sends CTRL+Z to end the scripts. This makes for two function buttons and six effects to take up all of the available digital IO on the Circuit Playground. Of course I didn't forget about the NeoPixels! For the start-up button I use the rainbow wave pattern and then for all of the others I use different colors in the color wave pattern that match the effect or command. For example, the CTRL+Z button flashes red, echo flashes blue, distortion flashes orange, etc.

And that's basically it for this project (so far). I'm considering this to be version 1. Of course the hardware and software can be adapted to be a controller or macro keyboard for basically any implementation. I look forward to exploring

the tones of the Pedal Pi a bit deeper now that I can easily change between the effects with the press of a button. Open source FTW!

DOWNLOAD SOURCECODE

PiClock: RGB LED Smart Clock Using Raspberry Pi Zero

Introduction:

This project aims to create a RGM matrix based SMART clock using the RGB Matrix interface boards being sold by Electrodragon and the Raspberry Pi Zero.

The software being used is Raspbian and the board will be interfaced directly in C. The ultimate goal would be to create an SDK for this board to create multiple apps for the P10/P6 display and finally create a nice 3d printed case for the same

The SDK will be capable of:

- Creating your own App-Modules for running different apps on the RPi
- Creating watch-faces for the board
- Enabling connectivity with REST APIs like IFTTT and so on to interact with the on board sensors

Sensors:

The project will host a number of sensors to which the "App Modules" can interface to:

- LDR
- Temperature sensor

- DS1307 RTC clock
- 3d gesture sensor from Flick

Stay Tuned for more from this project. The repo for this project will be at: https://github.com/narioinc/PiClock

Schematics

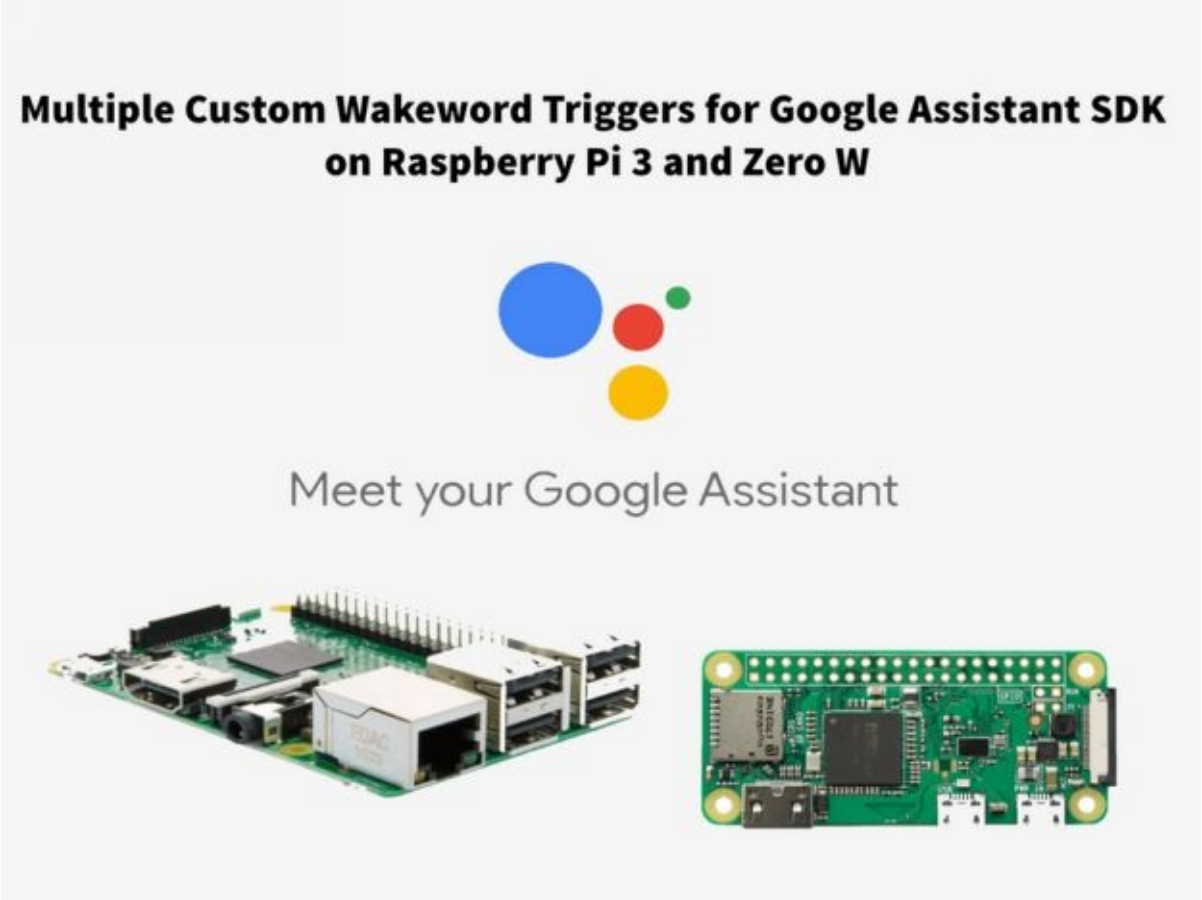
Hzeller Adapter board ref design
Download as zip

Code

PiClock
Download as zip

Multiple Custom Wakewords Activation of Google Assistant

This has been implemented using Snowboy for the hotword service. For some "Mysterious" reason, the assistant.start_conversation of the SDK does not seem to work for forcefully initiated custom wakewords. So I have used the gRPC for custom wakewords and SDK for "Hey Google" wakeword.



Works with Pi3 as well as Pi Zero Pi Zero - forked and modified from warchildmd's repo ([https://github.com/warchildmd/google-assistant-hotword-raspi)](https://github.com/warchildmd/google-assistant-hotword-raspi)

Features:

- Headless auto start on boot with multiple wakeword activation trigger.
- Locally Control GPIOs without IFTTT, API.AI, ACTIONS.
- Startup audio and audio feedback for wakeword detection.
- Safe shutdown RPi using voice command.

Features coming soon:

- Mute button.
- Neopixel indicator without Arduino.

This is implemented in Python2 so your existing Google Assistant may not work. So please start by making a fresh copy of latest Raspbian.

Install Audio Config Files

- Update OS and Kernel

```
sudo apt-get update
sudo apt-get install raspberrypi-kernel
```

- Restart Pi
- Choose the audio configuration according to your setup. (Run the commands till you get .bak notification in the terminal)

3.1. USB DAC users:

```
sudo chmod +x /home/pi/GassistPi/audio-drivers/USB-DAC/scripts/install-usb-dac.sh

sudo /home/pi/GassistPi/audio-drivers/USB-DAC/scripts/install-usb-dac.sh
```

3.2. AIY-HAT users:

```
sudo chmod +x /home/pi/GassistPi/audio-drivers/AIY-HAT/scripts/configure-driver.sh

sudo /home/pi/GassistPi/audio-drivers/AIY-HAT/scripts/configure-driver.sh
sudo chmod +x /home/pi/GassistPi/audio-drivers/AIY-HAT/scripts/install-alsa-config.sh
sudo /home/pi/GassistPi/audio-drivers/AIY-HAT/scripts/install-alsa-config.sh
```

3.3. USB MIC AND HDMI users:

```
sudo chmod +x /home/pi/GassistPi/audio-drivers/USB-MIC-HDMI/scripts/install-usb-mic-hdmi.sh
sudo /home/pi/GassistPi/audio-drivers/USB-MIC-HDMI/scripts/install-usb-mic-hdmi.sh
```

3.4. USB MIC AND AUDIO JACK users:

```
sudo chmod +x /home/pi/GassistPi/audio-drivers/USB-MIC-JACK/scripts/usb-mic-onboard-jack.sh

sudo /home/pi/GassistPi/audio-drivers/USB-MIC-JACK/scripts/usb-mic-onboard-jack.sh
```

3.5. CUSTOM VOICE HAT users:

```
sudo chmod +x /home/pi/GassistPi/audio-drivers/CUSTOM-VOICE-HAT/scripts/custom-voice-hat.sh

sudo /home/pi/GassistPi/audio-drivers/CUSTOM-VOICE-HAT/scripts/custom-voice-hat.sh
sudo chmod +x /home/pi/GassistPi/audio-drivers/CUSTOM-VOICE-HAT/scripts/install-i2s.sh
sudo /home/pi/GassistPi/audio-drivers/CUSTOM-VOICE-HAT/scripts/install-i2s.sh
```

Those Using HDMI/Onboard Jack, make sure to force the audio:

```
sudo raspi-config
```

Select advanced options, then audio and choose to force audio.

Those using any other DACs or HATs install the cards as per the manufacturer's guide and then you can try using the USB-DAC config file after changing the hardware IDs.

- Restart Pi.
- Check the speaker using the following command:

speaker-test -t wav

Continue After Setting Up Audio

- Download credentials--->.json file
- Place the .json file in/home/pi directory
- Rename it to assistant--->assistant.json
- Using the one-line installer for installing Google Assistant and Snowboy dependencies **Pi3 and Armv7 users use the "gassist-installer-pi3.sh" installer and Pi Zero users use the "gassist-installer-pi-zero.sh" installer. Snowboy installer is common for both.**

4.1 Make the installers executable:

```
sudo chmod +x /home/pi/GassistPi/scripts/gassist-installer-pi3.sh
sudo chmod +x /home/pi/GassistPi/scripts/gassist-installer-pi-zero.sh
sudo chmod +x /home/pi/GassistPi/scripts/snowboy-deps-installer.sh
```

4.2 Execute the installers (Run the snowboy installer first. **Don't be in a hurry and Don't run them in parallel, Run them one after the other:**

```
sudo /home/pi/GassistPi/scripts/snowboy-deps-installer.sh
sudo /home/pi/GassistPi/scripts/gassist-installer-pi-zero.sh
sudo /home/pi/GassistPi/scripts/gassist-installer-pi3.sh
```

- Copy the Google Assistant authentication link from terminal and authorize using your Google account.
- Copy the authorization code from browser onto the terminal and press enter.
- Move into the environment and test the google assistant according to your board.

```
source env/bin/activate
google-assistant-demo
googlesamples-assistant-pushtotalk
```

- After verifying the working of assistant, close and exit the terminal.

Headless Autostart on Boot Service Setup

- Make the service installer executable:

```
sudo chmod +x /home/pi/GassistPi/scripts/service-installer.sh
```

- Run the service installer:

```
sudo /home/pi/GassistPi/scripts/service-installer.sh
```

- Enable the services: **Pi3 and Armv7 users enable both the services for custom wakewords. Pi Zero users enable snowboy services alone:**

```
sudo systemctl enable gassistpi-ok-google.service
sudo systemctl enable snowboy.service
```

- Start the service: **Pi3 and Armv7 users start both the services for custom wakewords. Pi Zero users start snowboy services alone:**

```
sudo systemctl start gassistpi-ok-google.service
sudo systemctl start snowboy.service
```

Restart and enjoy!

Voice Control of GPIOs and Pi Shutdown

The default GPIO and shutdown trigger word is "trigger" if you wish to change the trigger word, you can replace the 'trigger'in the main.py(src folder) and

assistant.py(snowboy folder) code with your desired trigger word.
Similarly, you can define your own device names under the variable name var.
The number of GPIO pins declared should match the number of devices.

For NeoPixel Indicator

- Replace the main.py in src folder with the main.py from Neopixel Indicator Folder.
- Reboot
- Change the Pin numbers in the given sketch according to your board and upload it.
- Follow the circuit diagram given.

Now you have your Google Home Like Indicator!

DOWNLOAD SOURCECODE

Pi0drone: A smart drone with the Pi Zero

This tutorial demonstrates how to build a Linux drone with the Raspberry Pi Zero using a BOM (Bill of Materials) of **less than 200 US$**. The drone uses a real-time capable Linux kernel, a Debian-based file system and Dronecode's APM flight stack compiled for the PXFmini autopilot board. All these components have been put together by Erle Robotics in their OS image for the PXFmini.

Step 1: Assemble the drone kit

Required time: 30 minutes

Once you get all the components start by assembling your drone:

- Get the black frame together and place the motors on top.
- Fix the ESC (Electronic Speed Controllers) to the frame using some tape and connect them to the motors.
- Put together the power (**red**) and ground (**black**) ends of the ESCs into the individual cable (to be connected later to the battery) and fix everything underneath the frame.
- Adjust the power module connectors to the battery ones. There several ways to do this but here's a quick one: a) cut the connectors and solder battery and power module together (do it one at a time,

careful with short circuits!). b) cut the other end of the power module and resolder the battery connector (previously cut) there. c) Done!, this will allow us to easily connect and disconnect the "battery+power module" to the drone.

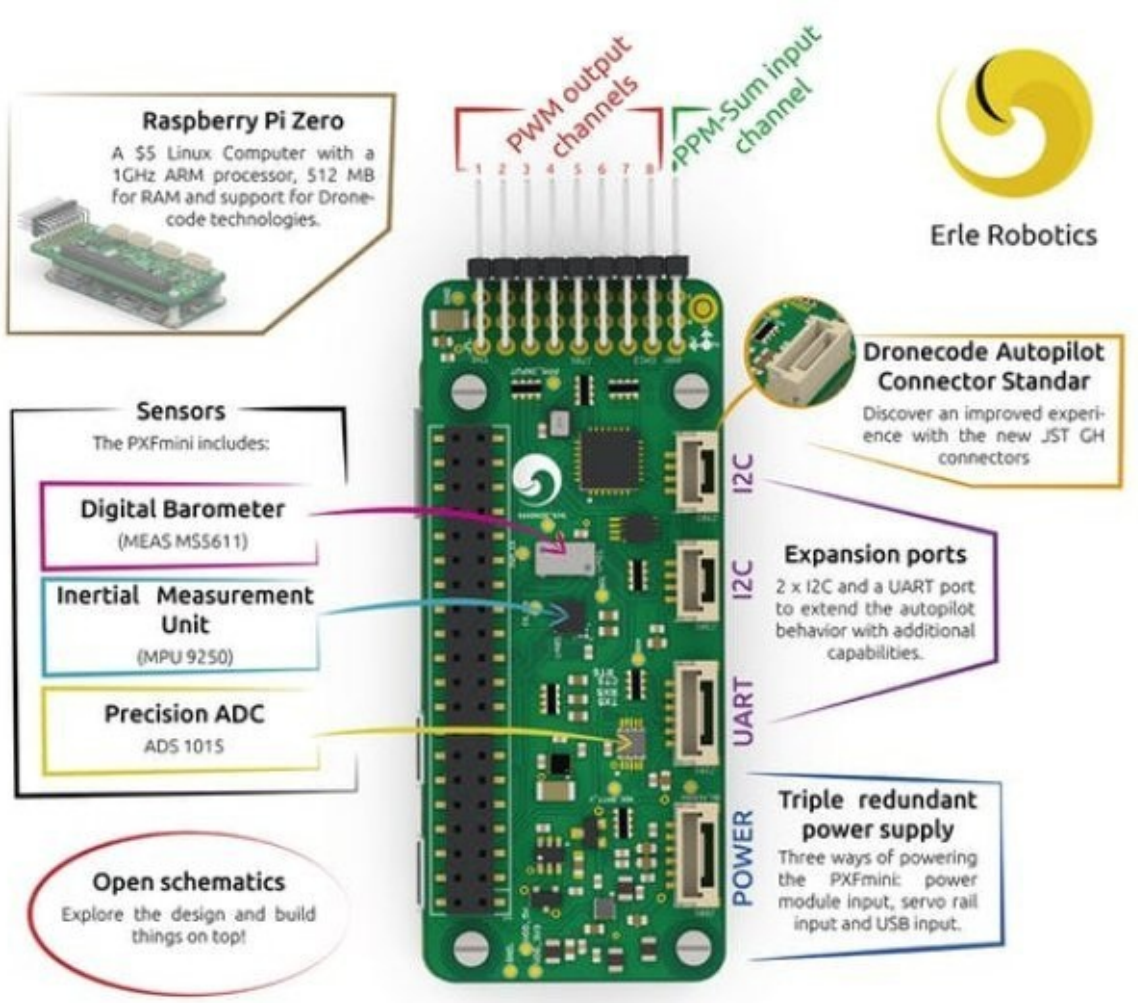
- Place the "battery+power module" pack underneath, use the velcro included in the package to do so.

Step 2: Get the autopilot ready

Required time: 30 minutes

Connect the PXFmini shield on top of the Raspberry Pi Zero as described in the following content:

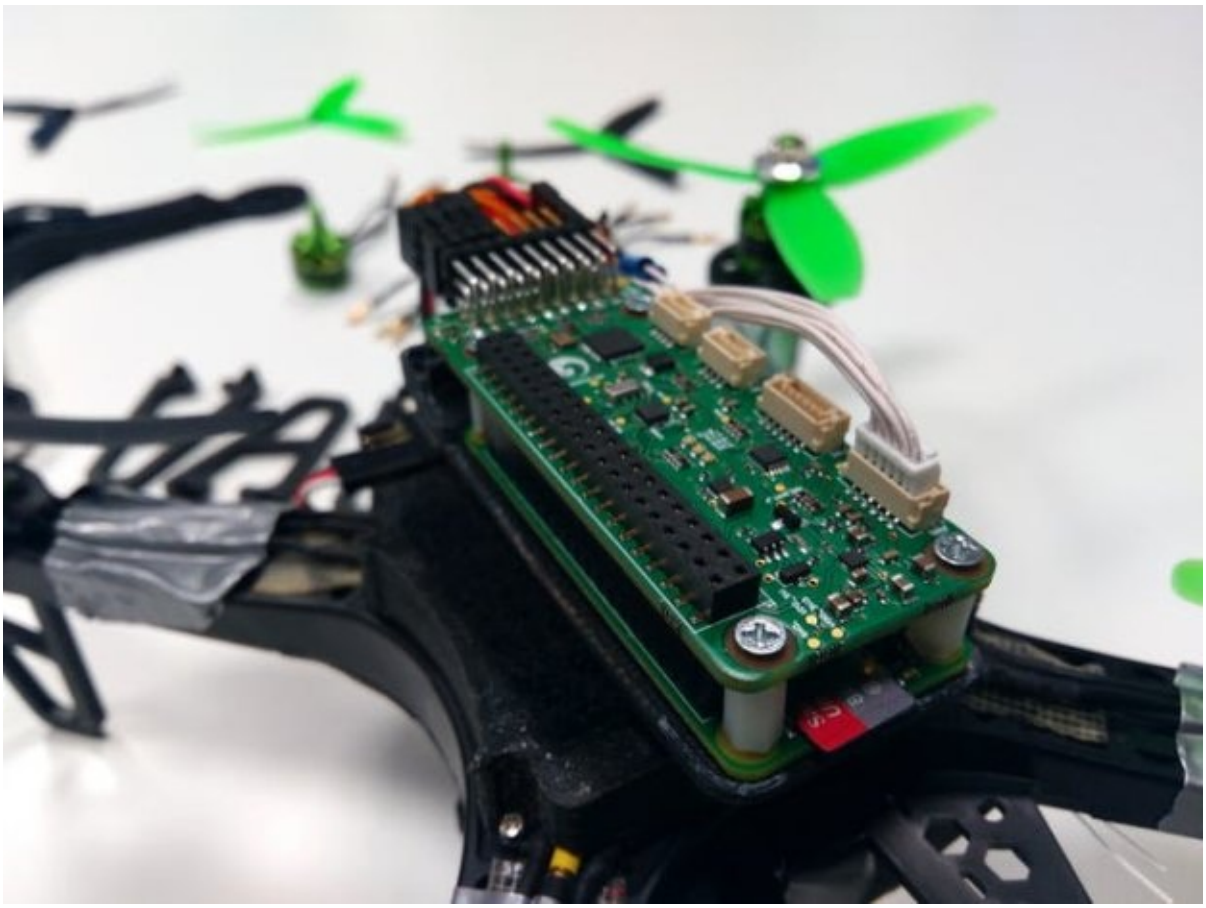
You're almost done but you still need to get the right software on the Raspberry Pi Zero+PXFmini set. This should include the flight stack, an appropriate kernel, enabled daemons that auto-launch on boot, and additional goodies...



Fortunately, if you purchased the PXFmini from Erle Robotics you'll get access to their Debian images which include all this so just fetch a PXFmini compatible Debian image and flash it into a microSD card.

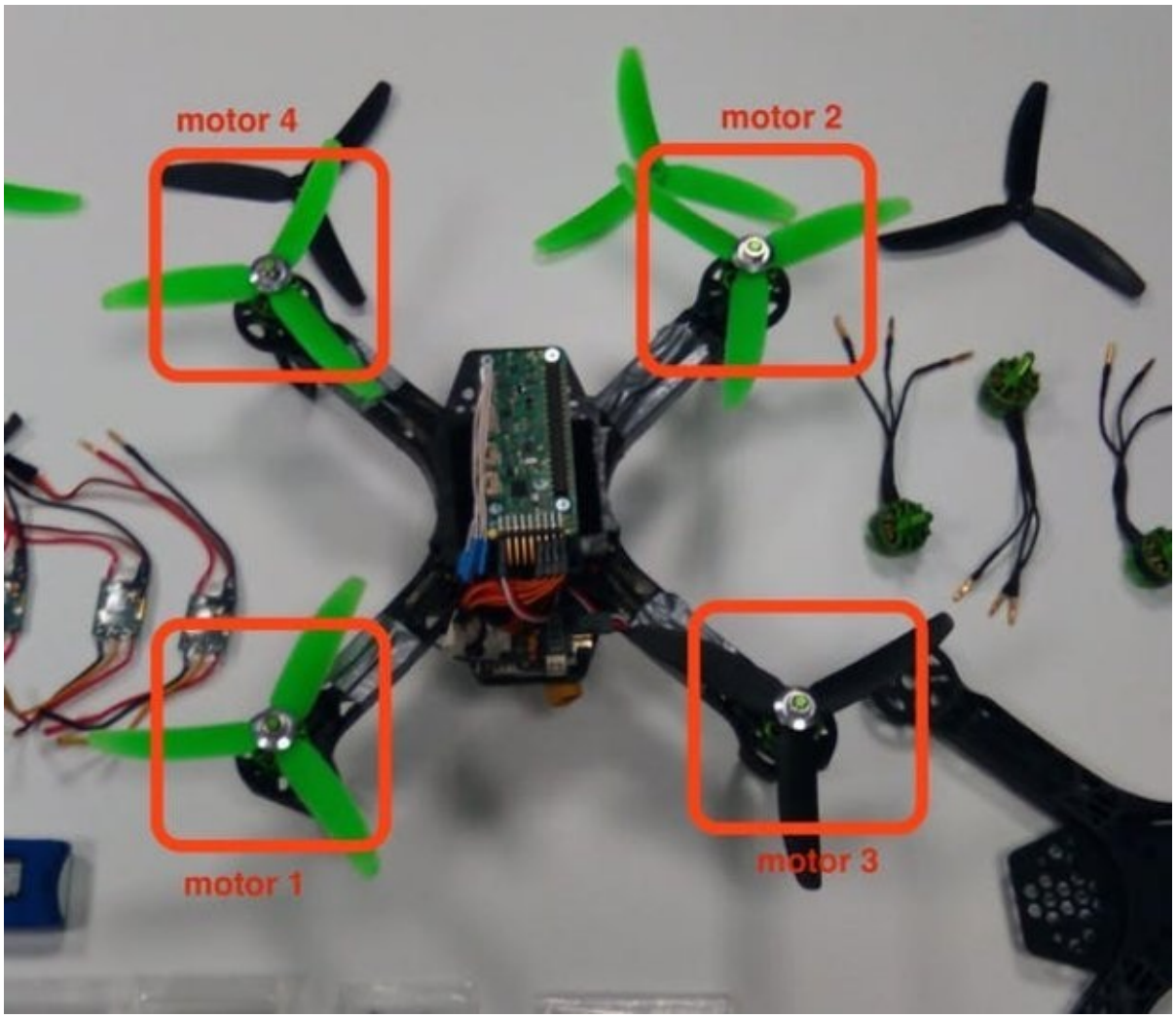
Step 3: Mount the autopilot

Required time: 5 minutes



Mounting the autopilot (Raspberry Pi Zero + PXFmini) in the drone can be done using several methods. Pick yours and connect the JST GH cable from the power module to the PXFmini. This will to power the autopilot when the battery gets connected.

Next is mounting the PWM channels in the autopilot. Get your ESC cables and connect ESC 1 (corresponding with motor 1) to PWM channel 1, ESC 2 to PWM 2 and so on.



Step 4: Mount the propellers and get it flying!

Required time: 15 minutes

There's two kinds of propellers clockwise (marked with an "R") and counter-clockwise. Place the clockwise propellers in motor 3 and 4 and the counter-clockwise ones in motors 1 and 2.

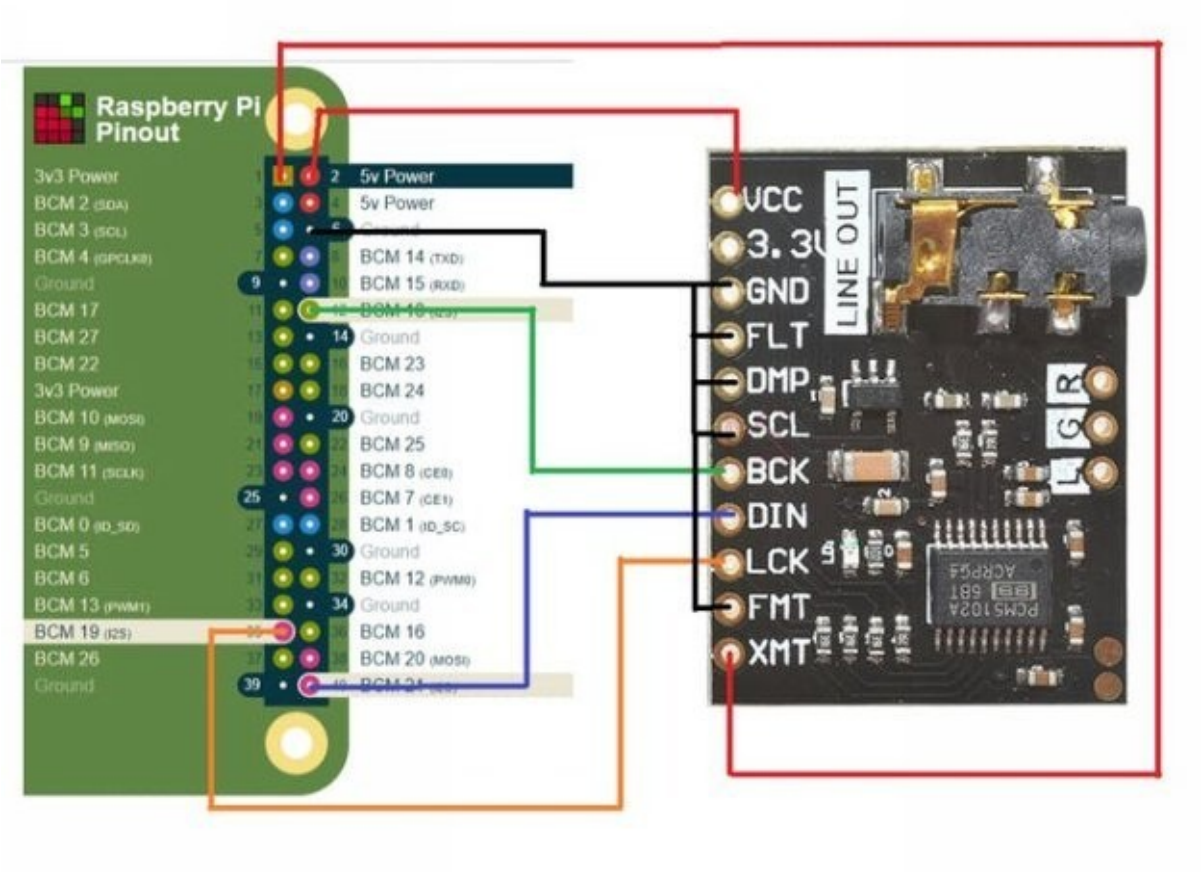
Finally, you'll need a way to control your drone. I propose two methods:

- **WiFi + gamepad**: Have the autopilot create its own WiFi network with a USB dongle (Erle Robotics images support this by default) and use a common gamepath to control the drone through a Ground Control Station.
- **WiFi + ROS**: Do you happen to know about the Robot Operating System (ROS)? You can use a ROS node to visualize the flight mode, state and control the drone. Have a look at https://www.youtube.com/watch?v=XxogeasbHyI.
- **Traditional RC**: Alternatively you could buy an RC controller with PPMSUM-enabled receiver and attach it to the autopilot (to the PPM-SUM input channel).

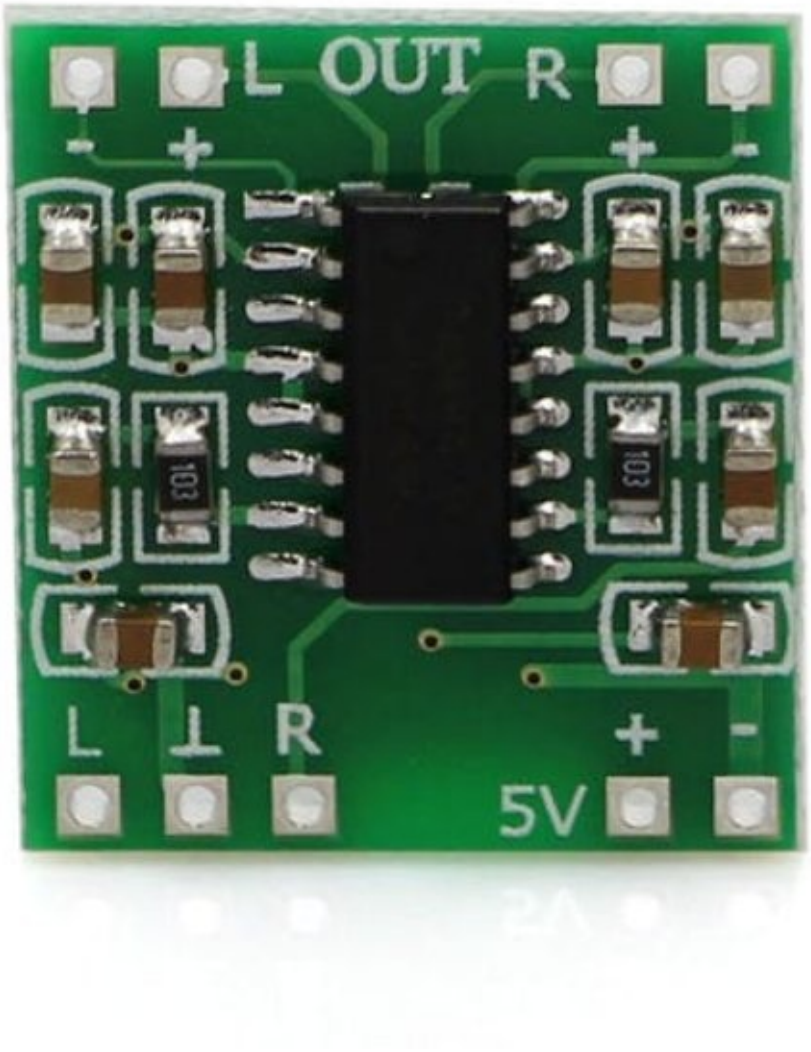
Audio DAC HAT for Pi with Headphone Jack and 3W Speaker Out

Commercially available pHAT DAC costs about $12 and adds just a headphone jack to your Pi Zero. On the other hand, the $14 Adafruit Speaker bonnet provides, two 3W audio channels for the loss of the headphone jack. The DIY HAT shown below strikes a balance between both the commercial HATS, by providing both the headphone output, as well as speaker outputs. Whats more interesting is the cost of implementation of the DAC, which is about $6 or $7. So, at quarter of the cost of two DAC's you are getting the functionality of both.

Wiring diagram for I2S DAC:



Connect the Left, Ground and Right pin from I2S DAC to Left, Ground and Right pin of the PAM8403 amplifier. Also connect the 5V and Ground pins from Pi to the PAM amplifier's supply terminals.

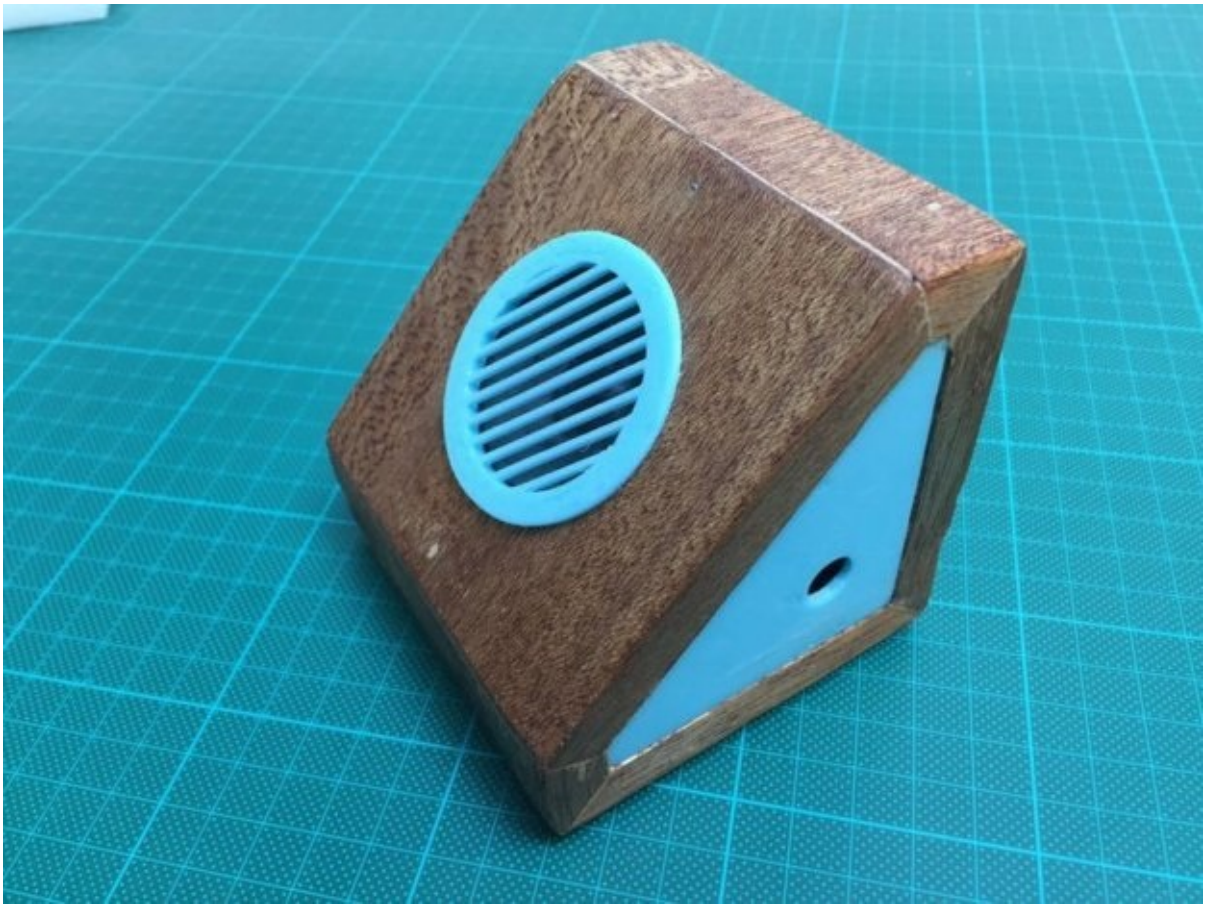


For installation of the homemade DAC on a Raspbian, you can use the pHAT DAC's one line installer command:

curl -sS https://get.pimoroni.com/phatdac | bash

Raspberry Pi Zero AirPlay Speaker

Looking for a new project to build around the Raspberry Pi Zero, I came across the pHAT DAC from Pimoroni. This little add-on board adds audio playback capabilities to the Pi Zero. Because the pHAT uses the GPIO pins, the USB OTG port remains available for a wifi dongle. Perfect for a small wireless speaker project!



Hardware

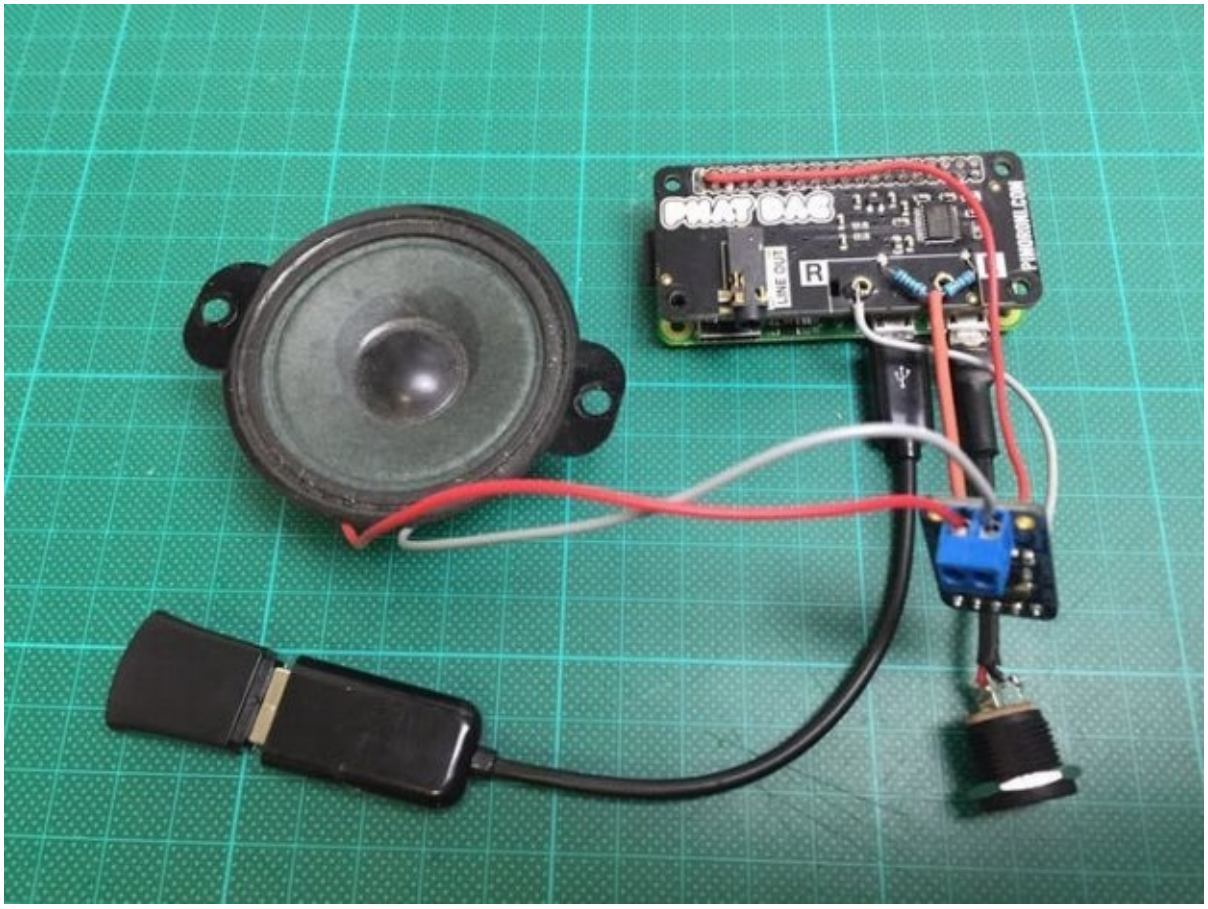
The project is fairly simple and requires following components:

- Raspberry Pi Zero
- pHAT DAC
- Mono 2.5W Class D Amplifier
- WiPi wifi dongle
- Two 100 ohms resistors
- Speaker (4-8 ohms)

The Raspberry Pi Zero is obviously the brains of the project and will run the Shairport software to wirelessly stream music to. The pHAT DAC is a neat little

add-on board adding audio to the Raspberry Pi. It has a jack output, and the possibility to add RCA connectors to it. The fact that the RCA connectors are not presoldered is a bonus, as it exposes the audio lines and keeps a low profile. A small mono amplifier from Adafruit then takes the audio from the pHAT and amplifies it (what else?), playing audio from the speaker. A wifi dongle connected via the USB OTG port provides wireless network connectivity for streaming.

I decided to make a mono speaker to keep things smaller. Making this project with stereo support would imply having a second speaker and replace the mono amplifier by a stereo one.



I know this isn't the nicest way to convert stereo to mono (at all?), but it works. I tried to tackle the problem from a software point of view by downmixing the stereo to mono, but with no success. If I do manage to find a proper solution, ideally in software, I'll be sure to update this post. If anyone has tips on how to achieve this in a simple way, feel free to leave it in the comments!

Software

On the software side, nothing too difficult either.

I started off from the latest Raspbian Jessie image which can be downloaded from the official Raspberry Pi website.

Using “dd”, I put the downloaded image on a 8Gb microSD card and then used it to boot the Pi Zero from.

```
sudo diskutil list
sudo diskutil unmountDisk /dev/disk3
sudo dd if=Downloads/2015-11-21-raspbian-jessie.img of=/dev/disk3 bs=1m
sudo diskutil unmountDisk /dev/disk3
```

Once booted, I set up the wifi in the graphical desktop environment by selecting the correct SSID and entering the wifi password. With the Pi Zero connected to the network, I could update the software.

```
sudo apt-get update
sudo apt-get upgrade
```

Then came the time to install the project specific software: support for the pHAT DAC and the AirPlay software.

pHAT DAC

A tutorial on how to install and use the pHAT DAC is available on the Pimoroni website. I did things slightly differently though, as I didn’t disable the default sound driver.

Device-tree overlay is used to describe hardware. As the pHAT DAC uses the same hardware as the HiFi Berry, the same overlay can be used by appending the following lines to the config file:

```
pi@raspberrypi:~ $ sudo nano /boot/config.txt

# pHAT DAC
dtoverlay=hifiberry-dac
```

After rebooting, I listed the audio devices using the “aplay” application, and there it was: *card 1 – HiFi Berry*.

```
pi@raspberrypi:~ $ aplay -l

**** List of PLAYBACK Hardware Devices ****
card 0: ALSA [bcm2835 ALSA], device 0: bcm2835 ALSA [bcm2835 ALSA]
Subdevices: 8/8
Subdevice #0: subdevice #0
Subdevice #1: subdevice #1
Subdevice #2: subdevice #2
Subdevice #3: subdevice #3
Subdevice #4: subdevice #4
Subdevice #5: subdevice #5
Subdevice #6: subdevice #6
Subdevice #7: subdevice #7
card 0: ALSA [bcm2835 ALSA], device 1: bcm2835 ALSA [bcm2835 IEC958/HDMI]
```

```
Subdevices: 1/1
Subdevice #0: subdevice #0
card 1: sndrpihifiberry [snd_rpi_hifiberry_dac], device 0: HifiBerry DAC HiFi pcm5102a-hifi-0 []
Subdevices: 1/1
Subdevice #0: subdevice #0
```

To make it the default for audio playout, I updated the asound.conf file and replaced every reference to “card 0” by “card 1”.

```
pi@raspberrypi:~ $ sudo nano /etc/asound.conf

pcm.!default {
type hw
card 1
}

ctl.!default {
type hw
card 1
}
```

A final reboot ensured everything was applied.

ShairPort

Shairport is an Airtunes emulator, allowing compatible iOS devices or iTunes to stream audio to the device running it.

A few dependencies need to be met before Shairport can be installed and run.

```
pi@raspberrypi:~ $ sudo apt-get install git libao-dev libssl-dev libcrypt-openssl-rsa-perl libio-socket-inet6-perl libwww-perl avahi-utils libmodule-build-perl
pi@raspberrypi:~ $ git clone https://github.com/njh/perl-net-sdp.git perl-net-sdp
pi@raspberrypi:~ $ cd perl-net-sdp/
pi@raspberrypi:~/perl-net-sdp $ perl Build.PL
pi@raspberrypi:~/perl-net-sdp $ sudo ./Build
pi@raspberrypi:~/perl-net-sdp $ sudo ./Build test
pi@raspberrypi:~/perl-net-sdp $ sudo ./Build install
```

With the dependencies taken care of, the actual Shairport software can be installed.

```
pi@raspberrypi:~ $ git clone https://github.com/hendrikw82/shairport.git
pi@raspberrypi:~ $ cd shairport/
pi@raspberrypi:~/shairport $ make
```

At this stage, it’s possible to test if everything was installed properly by manually running the shairport.pl script.

```
pi@raspberrypi:~/shairport $ ./shairport.pl -a AirPi
```

Afetr confirming everything works as expected, the shairport application can be daemonized in order to have it automatically start at boot.

```
pi@raspberrypi:~/shairport $ sudo make install
pi@raspberrypi:~/shairport $ sudo cp shairport.init.sample /etc/init.d/shairport
```

```
pi@raspberrypi:~/shairport $ sudo chmod +x /etc/init.d/shairport
pi@raspberrypi:~/shairport $ sudo update-rc.d shairport defaults
```

Finally, the shairport file needs to be modified in order to specify the name of the AirPlay device. This could be anything you want. In my case, I picked something generic, such as “AirPi”.

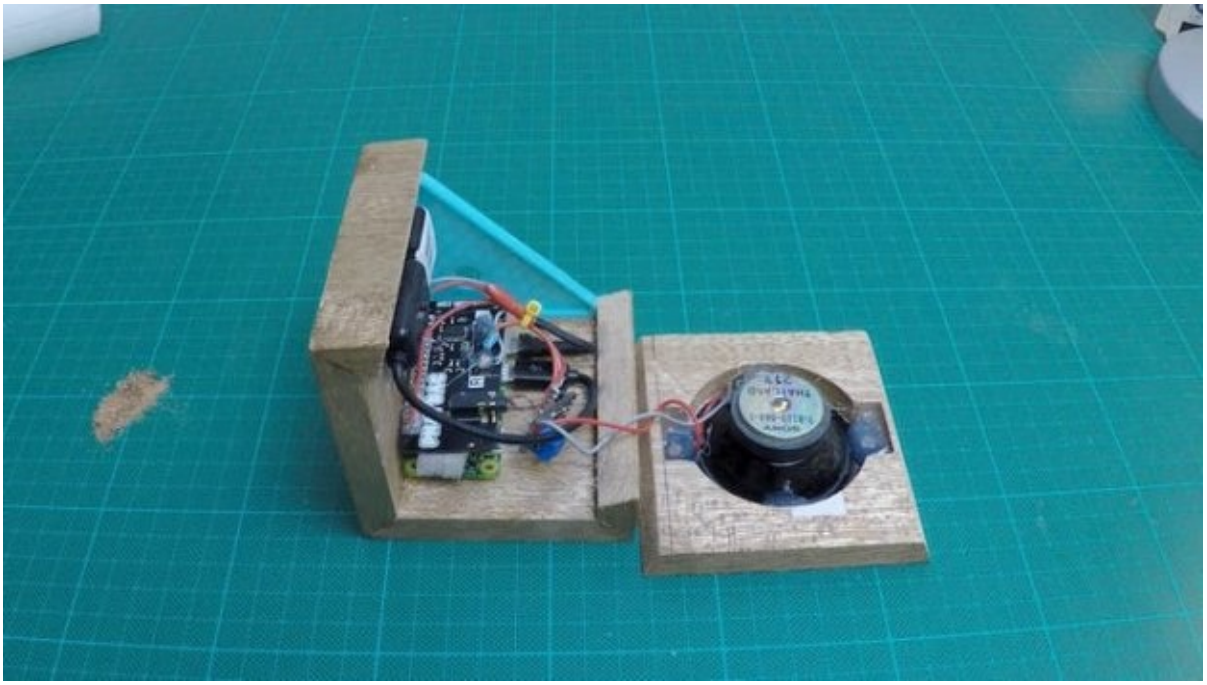
```
pi@raspberrypi:~/shairport $ sudo nano /etc/init.d/shairport

#DAEMON_ARGS="-w $PIDFILE"
DAEMON_ARGS="-w $PIDFILE -a AirPi"
```

Reboot the Pi. Shairport should be running automatically.

Enclosure

Time to package the working AirPlay speaker into something nice, by making a good looking enclosure for it.

This was actually the hardest part of the project. Mainly because I wanted to make it out of wood and with a slightly tricky shape. It meant doing some maths before cutting pieces at the right length using the miter saw and then ensuring the correct angles were cut as well in order to properly connect the pieces. As I’m not a woodworker, and the tools at my disposal are not the most suited either, the results are not always as accurate as you’d expect. That’s where sanding paper and wood filler come to the rescue …



Some accents were given to the build by adding 3D printed parts: the side panels and the speaker grill. One of the side panels was not glued into place and can be removed if needed, in order to access the electronics. I was hesitating to paint the

3D printed parts in a different color for a chrome or brass look, but ended up leaving the pieces as is. It gives the build a little funky side, no ?

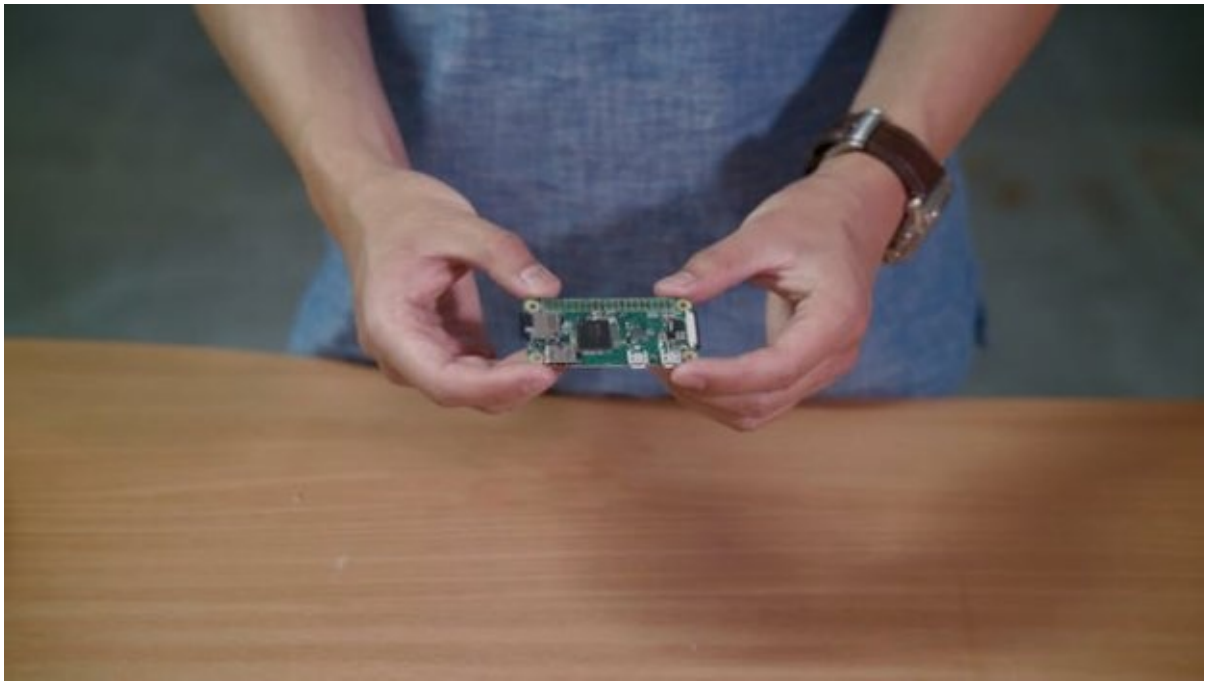
Automated Indoor Gardener

With how busy our lives are, it's sometimes easy to forget to pay a little attention to your thirsty indoor plants until it's too late and you are left with a crusty pile of yellow carcasses. Instead of constantly replacing those plants, we'll show you how to make a compact, automated, Raspberry Pi-powered gardener to water and light your plants. This gardener's memory is impeccable, and never forgets to water your plant.

Wire the Electronics

Step 1

We started by wiring the electronics for the gardener. This project is controlled by a Raspberry Pi Zero W. You don't need to have wifi for this project because the code runs off a scheduler, but you could also extend the functionality by connecting to a smartphone IoT app like Blynk.

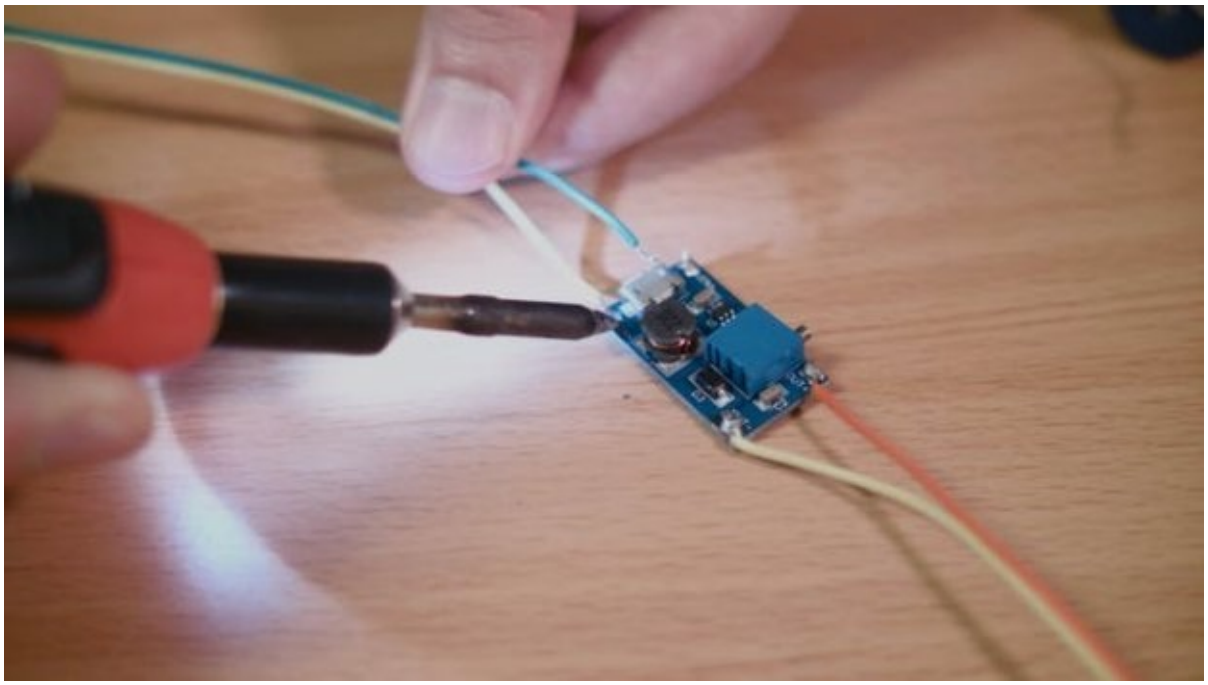


The electronics were attached to a 3D printed case we designed for this project. Both halves were printed out of PLA and had enough space to hide the extra wiring.

Step 2

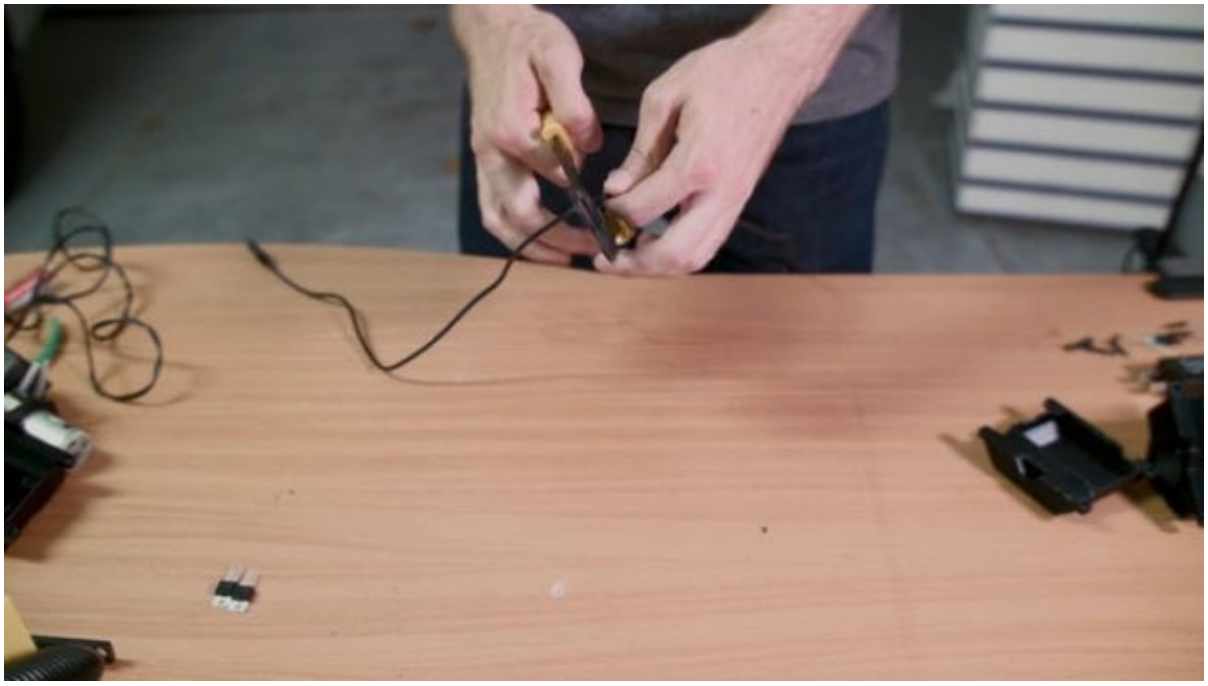
To power the 12v pump, we tried used a 12v boost adapter that was connected to the 5v input on the Raspberry Pi. However, we noticed that our Raspberry Pi power supply wasn't able to output enough current for the pump motor to run.

We decided to switch the 12v boost adapter out for an external 12v switching power supply. You could also use a 12v battery pack or a 12v wall adapter. It'll need to output about 3 amps at 12v.



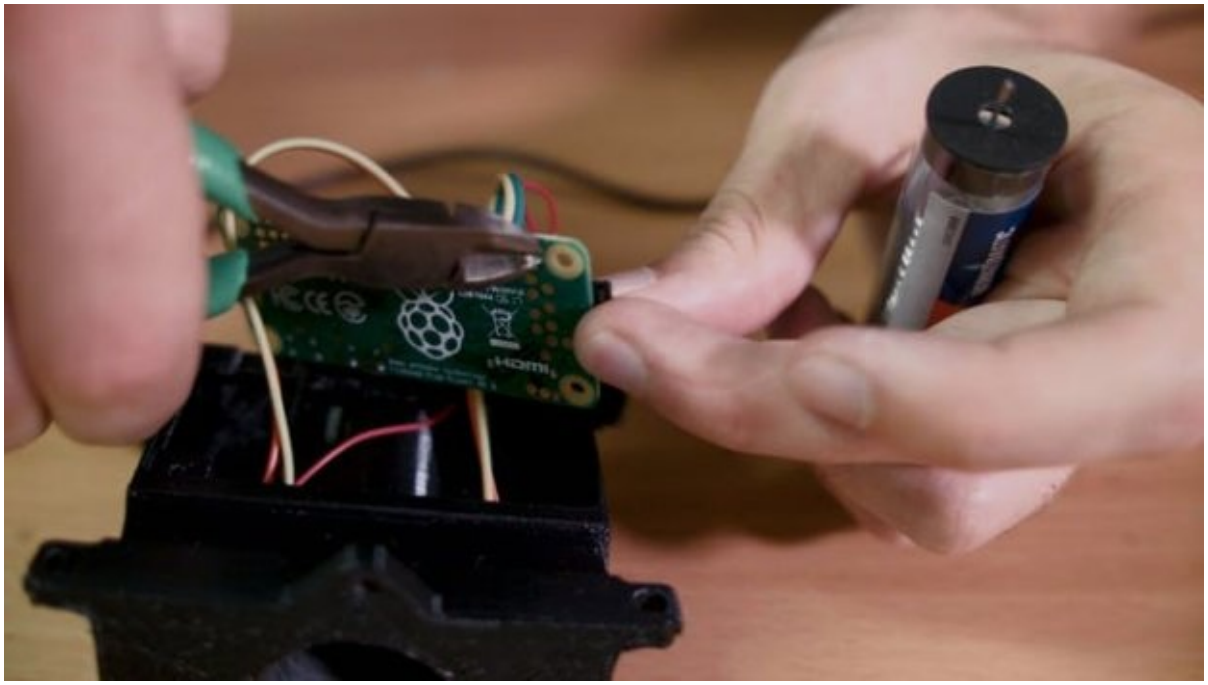
Step 3

Next, the USB connector was removed from the end of the grow light to expose the 5v power and ground wires. The red wire was soldered directly to a 5v output pin on the Raspberry Pi.



Step 4

The ground wire from the grow light was soldered to the drain (middle) pin on one of our N-Channel MOSFETs. The source (right) pin was wired to ground on the Raspberry Pi and the gate (left) pin was wired as a signal wire to the GPIO pin 20 on the Pi. When running, pulling the GPIO pin 20 high will turn on the light in this configuration.



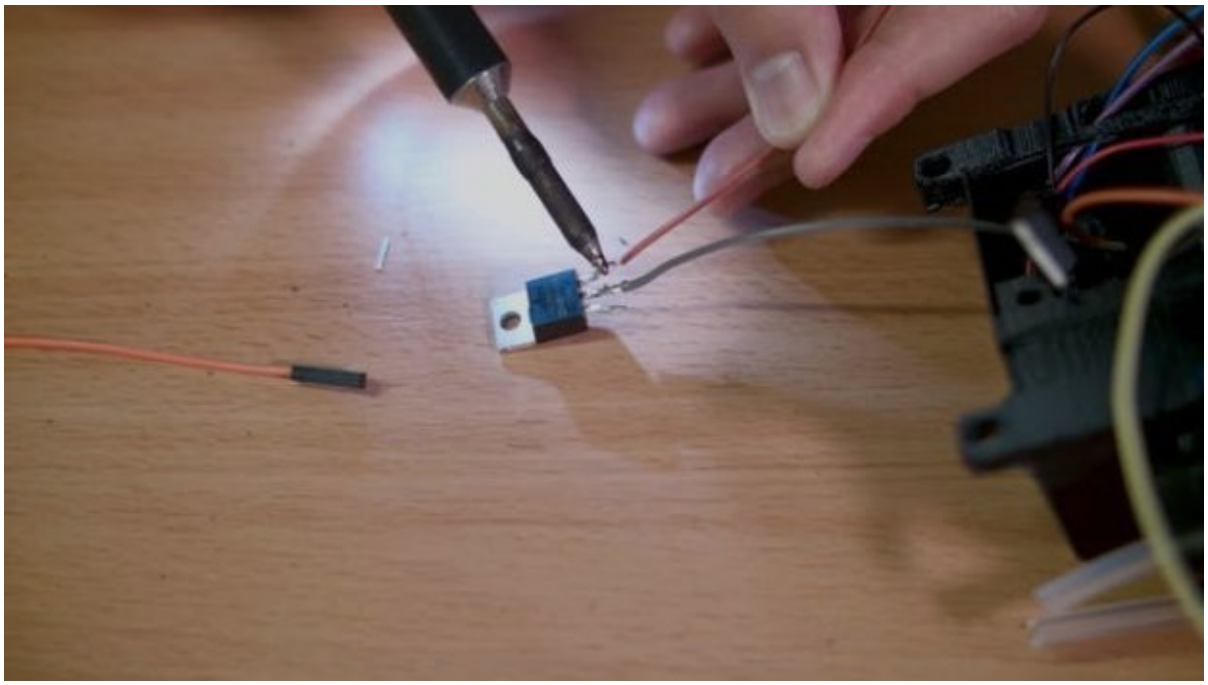
Step 5

Two wires were soldered to the connectors on the pump motor. The motor was then inserted into the slot for the pump and the wires passed through a small opening at the back. One of the wires was connected directly to the 12v power supply.

Step 6

We added an N-Channel MOSFET for the pump with a similar configuration. The 12v power supply ground wire was attached directly to a ground pin on the Raspberry Pi.

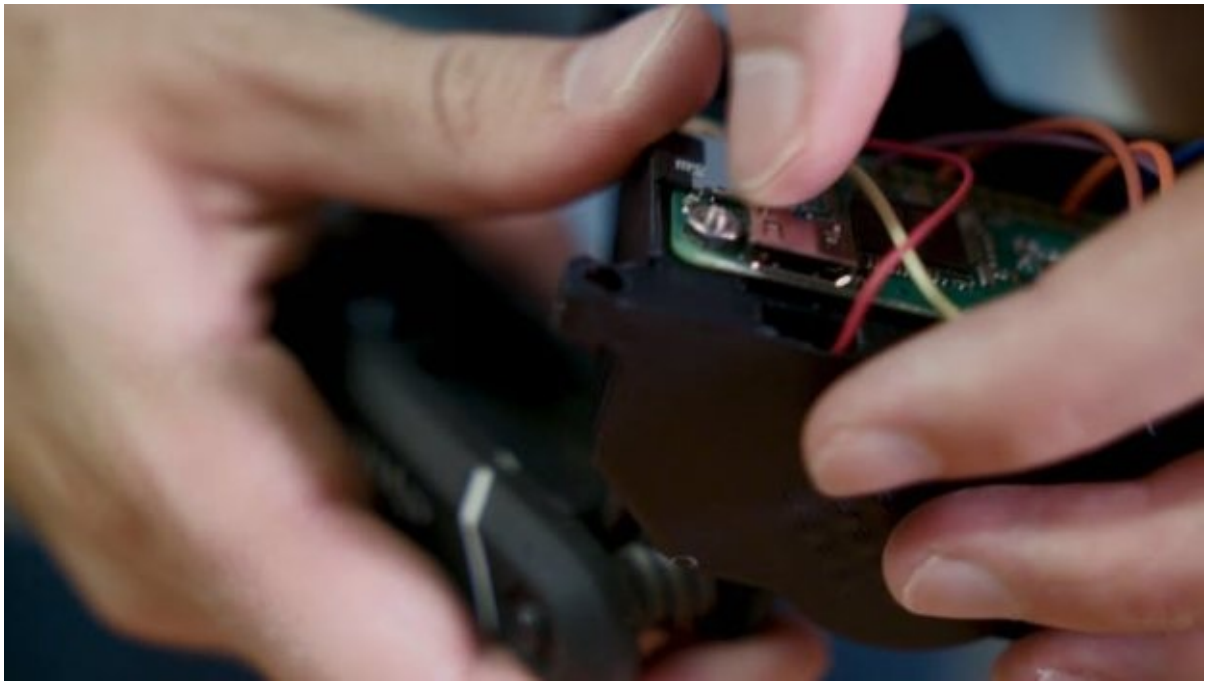
- Drain: ground wire from the pump.
- Source: ground pin on the Raspberry Pin.
- Gate: GPIO pin 12 on the Raspberry Pi.



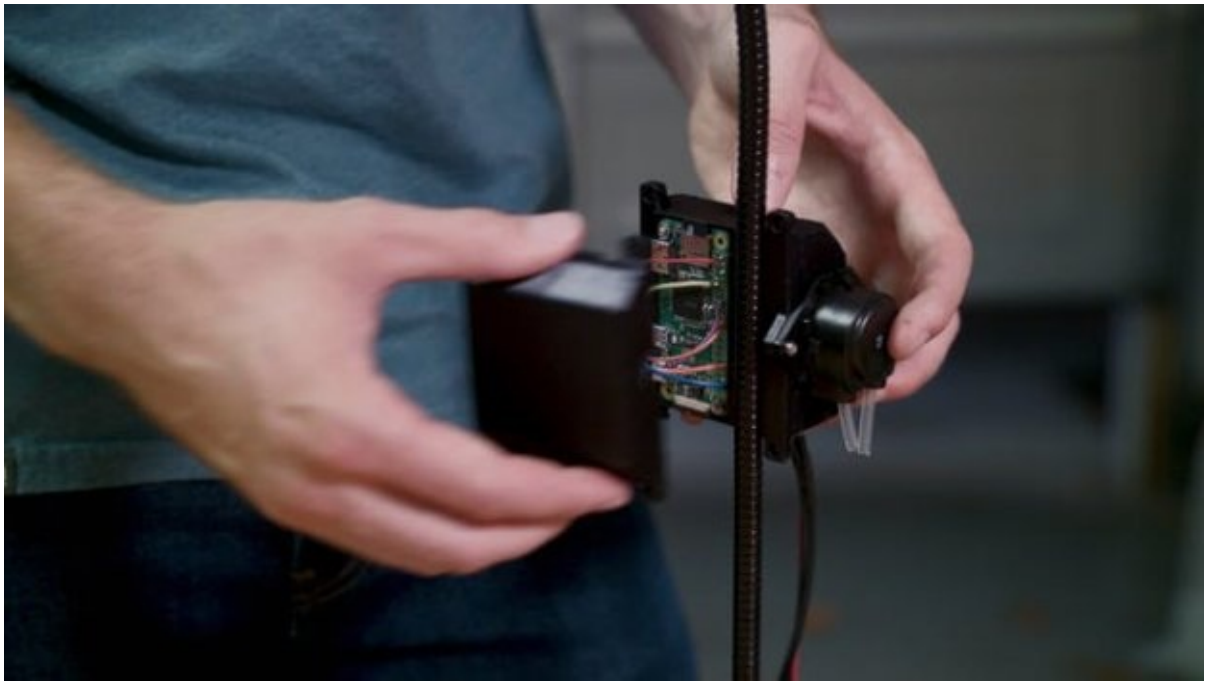
See our full circuit diagram below for more details about wiring the electronics.

Assembly

After the electronics were soldered, we tucked the loose wires into the opening on the back of the case. There are a couple holes on the back of the case that can be used to feed two M3 bolts through to attach the Raspberry Pi. Two M2.5 Bolts were also used to secure the pump on the side of the case.



The case has a small cutout for the stem of the 5v grow light. 4 M4 Bolts were used to attach the two halves of the case so that they sandwiched the stem and held the case in place about halfway up the light.



Run the Code

Download the code from our Github repository and navigate to into the folder.

```
cd Automated-Gardener
```

Step 1

Open the file with vim (if you don't have vim installed, you can install it with:

```
apt-get install vim
vim gardener.py
```

Step 2

Press 'i' to edit. Modify the pin variables if your signal wires are connected to different pins on your Raspberry Pi.

```
LIGHT_PIN = 20
PUMP_PIN = 12
```

Step 3

If you scroll down to the bottom, you can see where the schedule is set:

```
# Turn water on every 30 minutes for 10 seconds
schedule.every(30).minutes.do(threaded, water, forLength=10)
# Other scheduling examples
#schedule.every().hour.do(threaded, light, forLength=300)
#schedule.every().day.at("10:30").do(threaded, light, action=GardenerAction.turnOn)
#schedule.every().day.at("12:30").do(threaded, light, action=GardenerAction.turnOff)
#schedule.every().monday.do(threaded, water, forLength=30)
#schedule.every().wednesday.at("13:15").do(threaded, light, forLength=30)
```

`schedule.every(30).minutes.do(threaded, water, forLength=10)` turns the pump on for 10 seconds every 30 minutes. To change the schedule, you can uncomment some of the

scheduling examples by removing the # at the start of the line and changing the time/day. For example, if I wanted to turn the light on for 30 minutes on Wednesday at 2:00 pm, I would write:

```
schedule.every().wednesday.at("14:00").do(threaded, light, forLength=1800)
```

Step 4

After you've modified the gardener file, press esc to exit edit mode, then :wq to save and quit. Install a couple dependencies before you start the program.

```
sudo pip install schedule
sudo pip install rpi.gpio
```

Run the program.

```
python gardener.py
```

Step 5

Press control-c to quit. Get the current working directory by running:

```
pwd
```

Step 6

Copy the path, then open rc.local

```
sudo vim /etc/rc.local
```

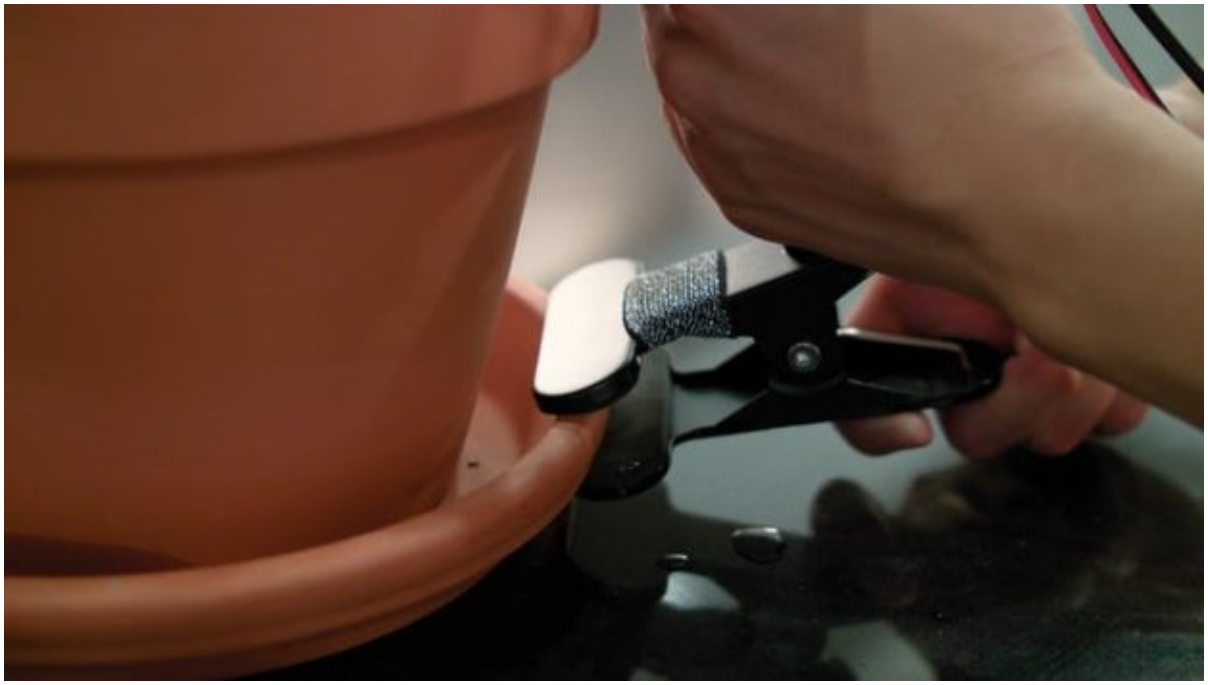
Press i . Before exit 0 , add:

```
python <pwd output>/gardener.py &
```

Press esc then :wq! to save and quit. When you reboot the PI, the program should start!

Attach the Pump Tubing

After testing the code, we attached the gardener to the plant's pot with the light clip. Our gardener was attached it to the water dish, but you can also attach it to the rim of the pot.



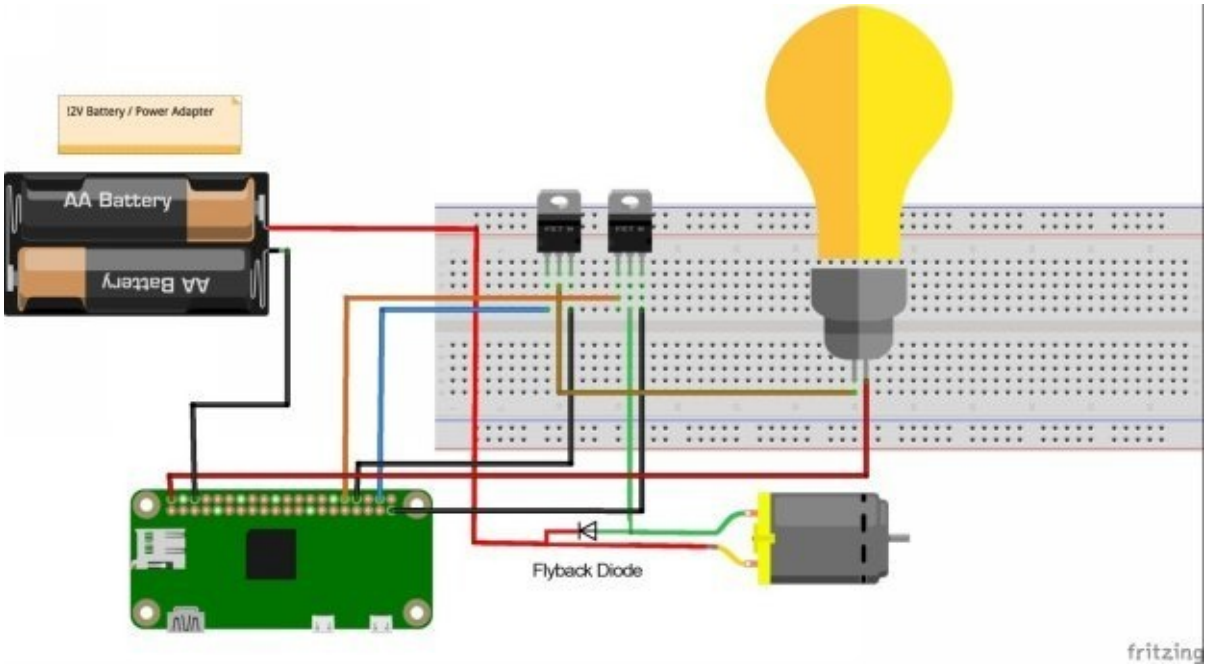
Two silicon tubes were cut and attached to the existing tubes peristaltic pump. The right one was placed in a cup of water near the pot, the other was positioned near the base of the plant because our pump was wired to flow from right to left. You might want to turn the pump on to determine the flow direction of the water.

Finally, make sure to plug in the Raspberry Pi, turn on your 12v power supply, and toggle the grow light on.

Show It Off

You've finished making the gardener! Sit back, relax, and let automation take over from here!

Circuit Diagram

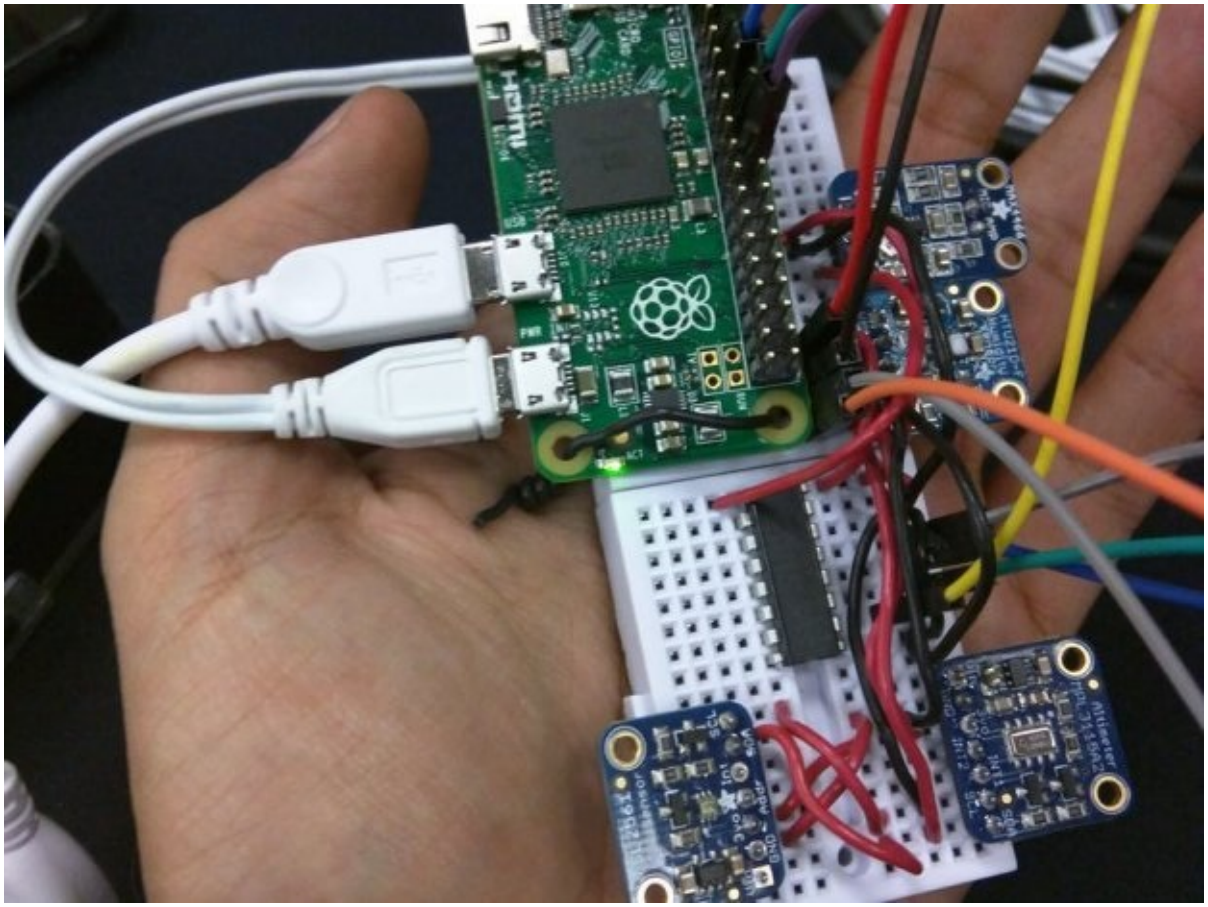


DOWNLOAD SOURCECODE

Smart Environmental Monitoring

Description

The goal is to build a small and easy to use device to monitor temperature, humidity, noise levels, luminosity and atmospheric pressure.



The idea is to have multiple devices spread across the city to send environmental data to the AWS IoT platform for processing and analysis. With this real time data , new public services could be offered, for example:

- Trigger alarms in case of dangerous measurements detected
- Finding out the less polluted places in the city at a given time: parks, squares or any public outdoors places.
- Find out high polluted places to avoid
- Find out the measurements from the nearest monitoring device

This device is intended to be used and managed by anyone with no other requirement than an internet connection and an available outdoor place at home: for example, balcony, a window, a roof, garden, etc. . .

Just by plugging the device , it will automatically start sensing and sending data

to the cloud.

Software

1.- Java SE applications for the Raspberry to read the sensors and send data to the AWS platform in real time. The Pi4j library will be used.

2.- An admin panel for the users of the device. A web application that will allow to restart /shutdown the device and will show the latest measurements.

3.- Examples of web applications using the AWS platform to provide public services:

- Alarms for dangerous measurements
- Show the measurements of the nearest monitoring device from my location
- Provide measurements historic files and graphics

Part 1 - Preparing the Raspberry Pi

1.1 - Installing the OS

1.- Download the latest Rapbian-Lite image from
https://www.raspberrypi.org/downloads/raspbian

2.- Follow the instructions to install the Raspbian in your micro SD card at
https://www.raspberrypi.org/documentation/installation/installing-images/linux.md

Or if you are using Linux you can follow these steps to install Raspbian Lite on your micro SD card:

- Check the device name for your micro SD by running :

df -h

```
[james@fedora22 mnt] $ df -h
Filesystem      Size  Used Avail Use% Mounted on
devtmpfs        3.8G     0  3.8G   0% /dev
tmpfs           3.8G   79M  3.7G   3% /dev/shm
tmpfs           3.8G  1.5M  3.8G   1% /run
tmpfs           3.8G     0  3.8G   0% /sys/fs/cgroup
/dev/sda1       451G  175G  253G  41% /
tmpfs           3.8G  408K  3.8G   1% /tmp
tmpfs           769M  8.0K  769M   1% /run/user/42
tmpfs           769M   28K  769M   1% /run/user/1000
/dev/mmcblk0    7.4G  4.0K  7.4G   1% /run/media/james/3980-8C72
```

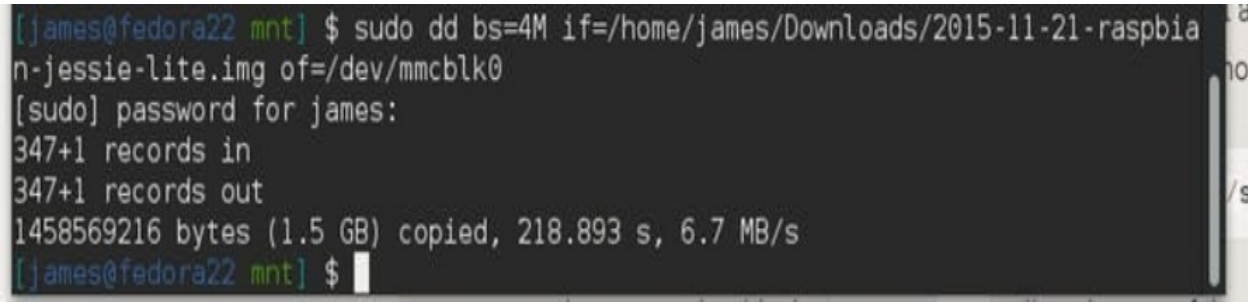
In my case, the device name is /dev/mmcblk0

- Unmount your device with the following command

umount /dev/mmcblk0

- Write Raspbian image to the device , run the following command withroot privileges. Where if is the location of your raspbian img file and of is the sd card device name:

sudo dd bs=4M if=/<path-to-img>/raspbian-jessie-lite.imgof=/dev/mmcblk0



1.2 - Configuring Raspbian

Insert your sd card in the Raspberry , connect it to a monitor using a mini HDMI adapter , connect a keyboard and turn the Raspberry on.

1.- Expand the Filesystem and Enable I2C

- Login as user: pi password: raspberry
- Execute the command sudo raspi-config in the terminal
- Select Expand Filesystem and press Enter
- Select OK and you will return to the main menu
- Select Advanced Options
- Select I2C and press Enter
- Select Yes and press Enter
- Select OK and press Enter
- Select Yes and press Enter
- Select OK and you will return to the main menu
- Select Finish and press Enter
- Select Yes and press Enter to reboot the Raspberry pi

2.- Configure automatic wifi connection

For the raspberry pi to connect automatically to your wifi network , follow the instruction at : https://www.raspberrypi.org/documentation/configuration/wireless/wireless-cli.md

In my case I added the following entry to the /etc/wpa_supplicant/wpa_supplicant.conf file to automaticallyconnect to my wifi network with WPA security , where ssid is mynetwork name and psk is the password:

```
network={
ssid="Huawei-HG8245-BF21"
psk="3219BF21"
proto=RSN
key_mgmt=WPA-PSK
pairwise=TKIP
auth_alg=OPEN
}
```

3.- Enable static IP

Now that your Raspberry connects automatically to your wifi network everytime it is tuned on, you need to specify a static IP address to communicate with your Raspberry

Follow the instructions to configure static IP at : https://www.raspberrypi.org/forums/viewtopic.php?f=91&t=49350

In my case, I set the static IP address to 192.168.100.50

My /etc/network/interfaces file looks like this :

```
auto lo
iface lo inet loopback
iface eth0 inet dhcp
allow-hotplug wlan0
iface wlan0 inet manual
wpa-roam /etc/wpa_supplicant/wpa_supplicant.conf
iface default inet static
address 192.168.100.50
netmask 255.255.255.0
network 192.168.100.0
broadcast 192.168.100.255
gateway 192.168.100.1
```

4.- Restart the Raspberry

- Shutdown the theRaspberry by executing : *sudo halt*

- Remove the keyboard , the HDMI adapter and connect a USB Wifi Adapter

- Turn on the Raspberry

If you performed steps 2 and 3 correctly you should now be able to ssh into the Raspberry

Wait a couple minutes after turning on the Raspberry and then go to step 5

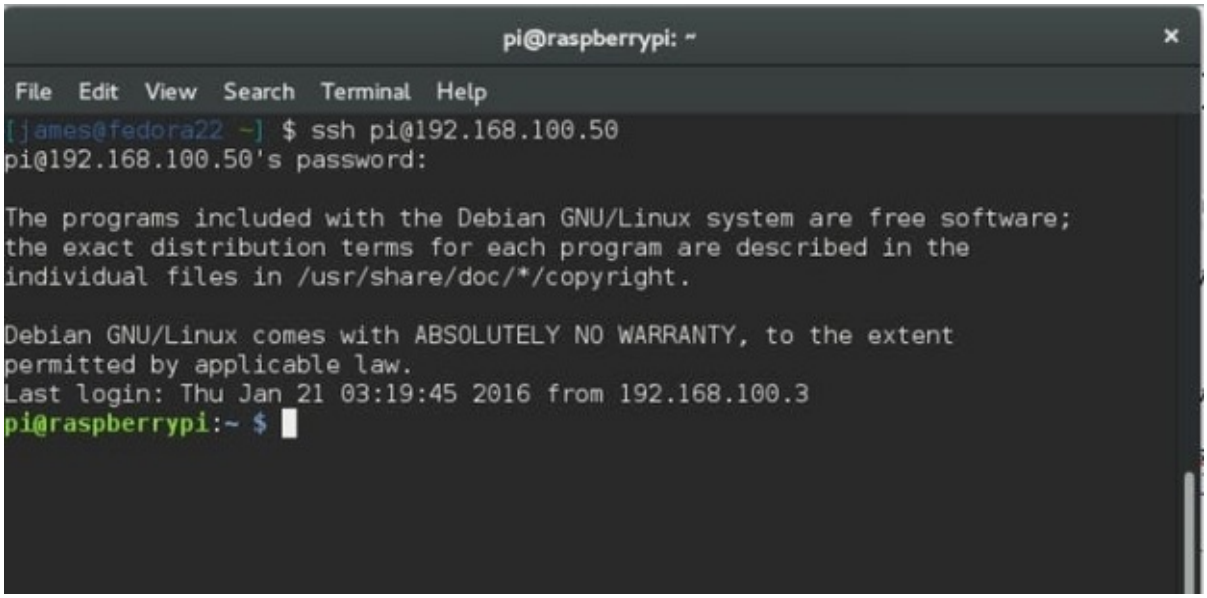
5.- SSH into the Raspberry to install additional software

From a computer connected to your Wifi network ,sse a ssh client software to connect to your Raspberry.

Check more information here:
https://www.raspberrypi.org/documentation/remote-access/ssh/

Or if you are using Linux , execute the following command to ssh into the Raspberry

ssh pi@<ip-address>



- Update repositories

From the terminal execute: *sudo apt-get update*

- Install i2c-tools

From the terminal execute: *sudo apt-get install i2c-tools*

1.3 - Installing the AWS IoT Device SDK for Node.js

SSH into your Rasapberry. From /home/pi , run the following commands

1.- Install the required repositories. Execute the following command

curl -sLS https://apt.adafruit.com/add | sudo bash

2.- Install npm

sudo apt-get install node

3.- Install npm

sudo apt-get install npm

4.- Install the SDK

npm install aws-iot-device-sdk

You should see a folder named node_modules at /home/pi

1.4 - Installing the Java JDK 8

1.- Download the Java JDK from

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Make sure to select Linux ARM 32 Hard Float ABI

2.- Transfer the SDK tar.gz file to /home/pi in the Raspberry

If you are using linux you can execute the following command from the directory where you downloaded the SDK

```
scp jdk-8u65-linux-arm32-vfp-hflt.tar.gz pi@192.168.100.50:/home/pi
```

3.- On the Raspberry. Uncompress the sdk file

```
tar -vxzf jdk-8u65-linux-arm32-vfp-hflt.tar.gz
```

A folder containing the SDK should have been created at /home/pi

Part 2 - Wiring the Sensors and the ADC to the Raspberry PI

2.1 - Description of the Sensors and ADC

The MPL3115A2 Altimeter

This sensor measures pressure, altitude and temperature. It works using the I2C protocol. In this project this sensor is used to measure temperature and pressure. Here is more information about the device:

Official product website

https://www.adafruit.com/products/1893

Official Datasheet:

https://www.adafruit.com/datasheets/1893_datasheet.pdf

Adafruit C++ Library (Not used in this project but useful as reference)

https://github.com/adafruit/Adafruit_MPL3115A2_Library

The TSL2561 Lux Sensor

This is a digital light sensor, it uses the I2C protocol. In this project this sensor is used to measure the Luminosity.

Official product website

https://www.adafruit.com/products/439

Official Datasheet

https://www.adafruit.com/datasheets/TSL2561.pdf

Adafruit C++ Library (Not used in this project but useful as reference)

https://github.com/adafruit/Adafruit_TSL2561/

The HTU21D-F Humidity Sensor

This is a I2C digital sensor. In this project this sensor is used to measure the

Humidity.

Official product website

https://www.adafruit.com/products/1899

Official Datasheet

https://www.adafruit.com/datasheets/1899_HTU21D.pdf

Adafruit C++ Library (Not used in this project but useful as reference)

https://github.com/adafruit/Adafruit_HTU21DF_Library

The MAX4466 Microphone

This is an electret microphone amplifier with adjustable gain. In this project this sensor is used to measure ambient noise (Sound Level Pressure)

Official product website

https://www.adafruit.com/product/1063

Official Datasheet

https://www.adafruit.com/datasheets/MAX4465-MAX4469.pdf

The MPC3008 ADC (Analog to Digital Converter)

The Raspberry pi doesn't have a built-in ADC so we are using the the MPC3008 to convert the analog signal from the MAX4466 Microphone to digital signal. This ADC uses a SPI interface.

Official product website

https://www.adafruit.com/products/856

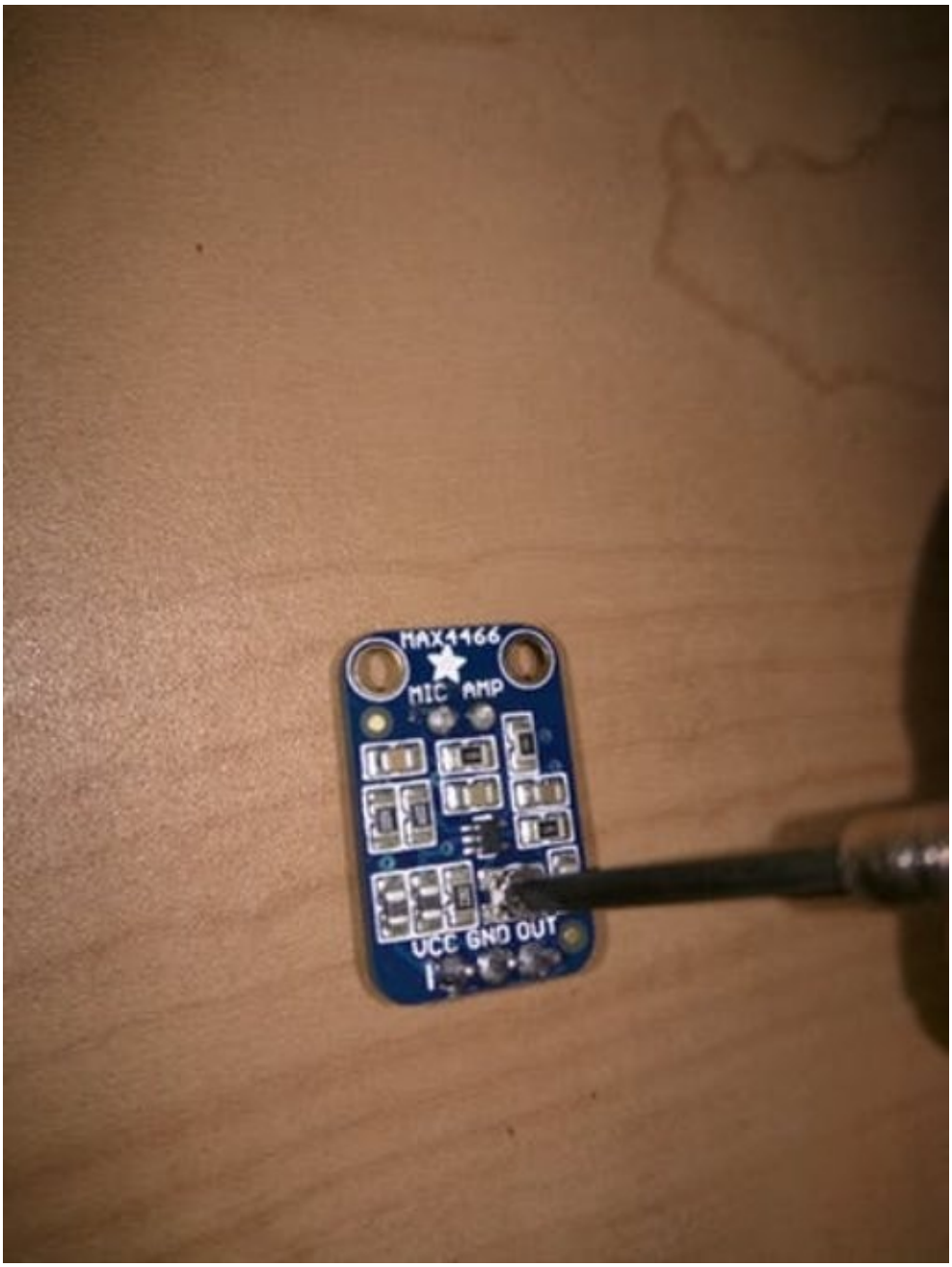
Official Datasheet

https://www.adafruit.com/datasheets/MCP3008.pdf

2.2 Wiring

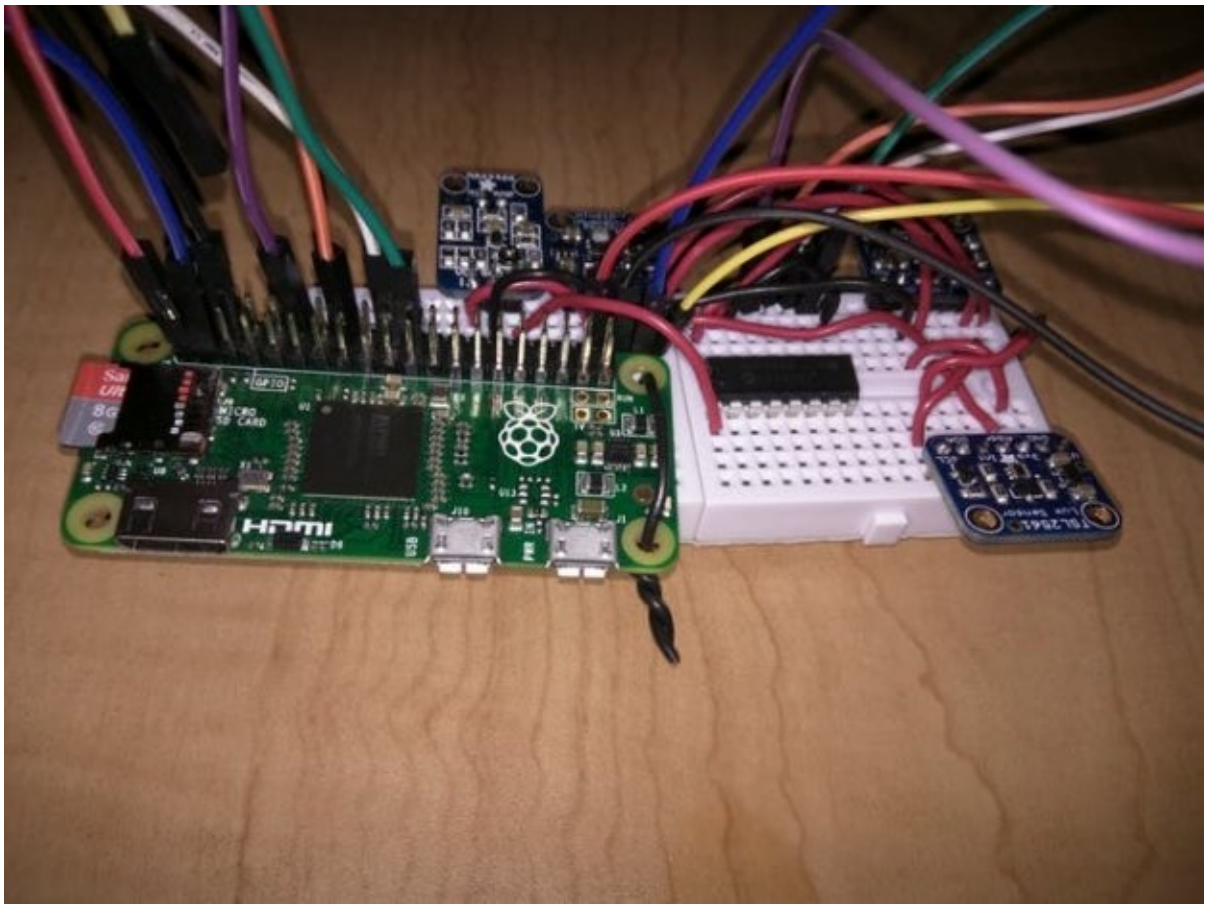
Follow the attached Fritzing diagram to connect the Raspberry , the sensors and the ADC to the breadboard.

Before wiring the MAX4466 we need to set its gain to the maximum level. To do this notice a a small trimmer pot on the back of the microphone. Place it in front of you and turn it counter clock wise all the way, do this very carefully with a small screwdriver. See the following image as reference



After you do this , proceed to to the wiring as indicated in the Fritzing diagram.

Once you finish the wiring and sensor placement your device should look something like this.



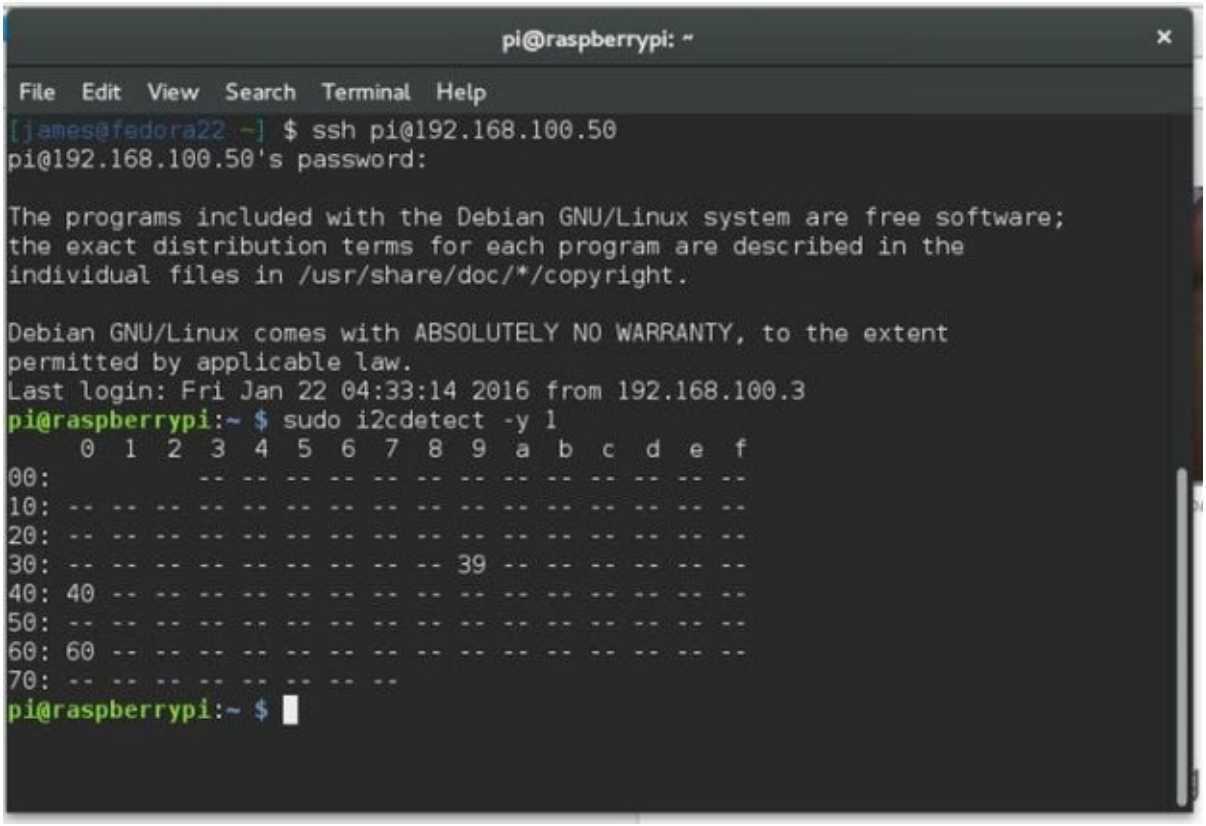
2.3 Verifying the wiring

If all the components were wired correctly , the Raspberry should be able to "see" the sensors.

- SSH into the raspberry and execute the command:

```
sudo i2cdetect -y 1
```

You should see the following output



The numbers 39 , 40 and 70 are the I2C addresses for the Altimeter , Lux and Humidity sensors.

Part 3 - Setting AWS IoT

3.1 - Create a thing device

1.- Login to your AWS iot account and click the "Create a resource" button

2.- Click "Create a thing"

3.- Enter a name and click the Create button

In this case , I named it "Monitoring_Device"

You should now see you device in the main dashboard

3.2 - Create connectivity certificates

1.- Click the name of your newly created device and a menu on the right should open up

2.- Click the "Select a device" button

3.- Choose NodeJs and click the "Generate certificate and policy" button

4.- Links to the created certificates will show up. Download the private key and the certificate

5.- Download the root CA certificate from

https://www.symantec.com/content/en/us/enterprise/verisign/roots/VeriSign-Class%203-Public-Primary-Certification-Authority-G5.pem

3.3 .- Copy the certificates to the Raspberry Pi

1.- On the raspberry pi create a folder for the certificates : /home/pi/certs

2.- Copy the private key , the certificate and the root CA to the raspberry.

If you are using linux you can use scp, example:

```
scp -r /home/james/Downloads/56ca21d279-private.pem.key pi@192.168.100.50:/home/pi/certs

scp -r /home/james/Downloads/56ca21d279-certificate.pem.crt pi@192.168.100.50:/home/pi/certs

scp -r /home/james/Downloads/VeriSign-Class 3-Public-Primary-Certification-Authority-G5.pem pi@192.168.100.50:/home/pi/certs
```

3.- On the Raspberry , rename the root CA certificate to root-CA.crt

```
mv "VeriSign-Class 3-Public-Primary-Certification-Authority-G5.pem" root-CA.crt
```

Part 4 - Installing the Software

4.1 - Source Code

The software consists on a Java 8 application that reads the sensors and a NodeJs application that communicates with AWS Iot

The main Java application code is located at :

https://github.com/alapisco/AWS-IoT-Smart-Env-Monitoring/tree/master/read_monitoring_device

It relies on a java library created for this project located here:

https://github.com/alapisco/AWS-IoT-Smart-Env-Monitoring/tree/master/sensors-libs/src

The java app uses a NodeJs script that uses the NodeJS AWS-IoT SDK to send data to the AWS platform

https://github.com/alapisco/AWS-IoT-Smart-Env-Monitoring/tree/master/update_AWS_device

The whole applications are packaged for deployment on the Raspberry Pi here:

https://github.com/alapisco/AWS-IoT-Smart-Env-Monitoring/tree/master/dist

4.2 - Deploying the app to the Raspberry

4.2.1 Installing the application

1.- Go to the github project page and select the "Download ZIP" button

https://github.com/alapisco/AWS-IoT-Smart-Env-Monitoring

2.- Unzip the github project file and copy the AWS-IoT-Smart-Env-Monitoring/dist folder to /home/pi to the Raspberry

3.- Check the application property file at AWS-IoT-Smart-Env-Monitoring/dist/device.properties and make sure that all properties are correct

KEY_PATH : refers to the private certificate you downloaded after creating the AWS thing device

CERT_PATH : refers to the .crt certificate you downloaded after creating the AWS thing device

CA_PATH : refers to the root certificate you downloaded

CLIENT_ID: refers to the AWS thing device name

REGION: refers to the region your AWS account is set at. You can check this by selecting your device on the AWS IoT dashboard and checking the REST api end point . In my case my endpoint is https://A39XL4FDYC51DK.iot.us-west-2.amazonaws.com/things/Monitoring_Device/shadow so my endpoint is us-west-2

MEASUREMENT_INTERVAL: refers to how often your application will get data from the sensors and send it to AWS IoT. It is defined in milliseconds. By default its set to 10000 (every 10 seconds)

AWS_SCRIPT: refers to the location of the nodejs script that will upload data to AWS IoT

After deploying all your files, you should now be able to run the application manually by executing the following commands in the raspberry :

```
cd dist/
sudo ../jdk1.8.0_65/bin/java -jar read_monitoring_device.jar
```

Your device should now be sending data to AWS IoT. You should see outputs like this:

If you go to AWS IoT and check your device status you should see the latest update and how many times it has been updated (Shadow versions)

4.2.2 Automatically run the app on boot

To automatically execute the application every time you turn on the Rapsberry . SSH into the raspberry and execute the following commands from /home/pi:

```
sudo cp dist/device_start /etc/init.d
sudo chmod 755 /etc/init.d/device_start
sudo update-rc.d device_start defaults
```

Download Schematic
Download Source Code

Raspberry Pi Zero Internet Connected Information Display

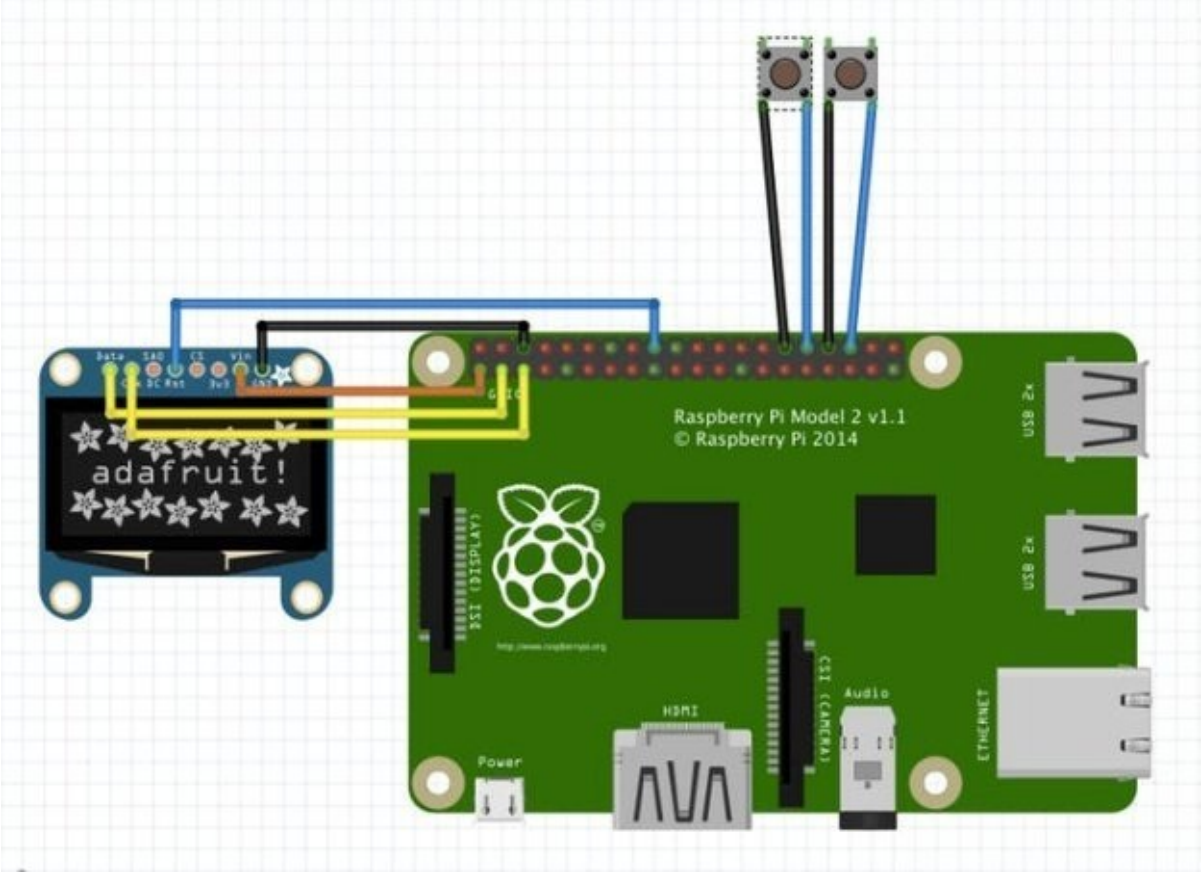
After last week's Pi Zero mod, I thought I'd try a slightly more useful project. Using an Adafruit OLED display, two push buttons, a wifi dongle and a Pi Zero, I made an internet connected information display. The information could be anything: time and date, weather, social media status, etc … The two push buttons are used to cycle through the data and trigger certain actions.

Hardware

The hardware consists of the following components:

- Raspberry Pi Zero
- Adafruit 128×64 SSD1306 OLED
- Edimax wifi dongle
- 2 large push buttons

The wiring of the OLED and buttons is rather straightforward, as is illustrated below:



The wifi dongle is connected the same way as I did previously with the USB hub:

- Pi Zero PP1 to dongle 5V
- Pi Zero PP6 to dongle GND
- Pi Zero PP22 to dongle D+
- Pi Zero PP23 to dongle D-

Finally, to keep everything in place, I designed a simple enclosure just large enough to fit everything in. A back panel is screwed in place to keep all components inside, exposing the power input microUSB port on the side.

A bit of kapton tape prevents exposed contacts to touch each other and keeps the wiring in place. The wifi dongle is positioned at the top for better connectivity. Its bright blue LED shines through the enclosure, giving a clear indication on its status and activity.

The files for the 3D printable enclosure can be found on thingiverse: http://www.thingiverse.com/thing:1193350

Software

For the software side of the project, I started off by creating the microSD card using the latest Raspbian Jessie image available. I booted it from the Pi Zero/USB hub combo with keyboard and wifi dongle connected. I'm using this approach because this project's Pi Zero's USB port has been hardwired to a wifi dongle and a keyboard can no longer be connected.

I configured the wifi by adding the correct SSID and passphrase in the */etc/network/interfaces* file. After testing the wifi connectivity, I put the microSD card back in the correct Pi. With network connectivity, it is possible to log in using SSH and work on the script to display the desired information.

Using the Adafruit OLED SSD1306 Python Library and some custom Python

code, I programmed three different displays:

- Time & date
- Network settings
- Social media subscribers/followers

The left button cycles through the different screens, while the right button triggers a custom action per screen.

In the case of the time and date display, the button simply toggles between 12h and 24h representation. For the network settings, it forces the wifi to reconnect by bringing down the interface and forcing it up again. Finally, to avoid excessive traffic, social media information is only retrieved every five minutes, pressing the button forces the retrieval of information.

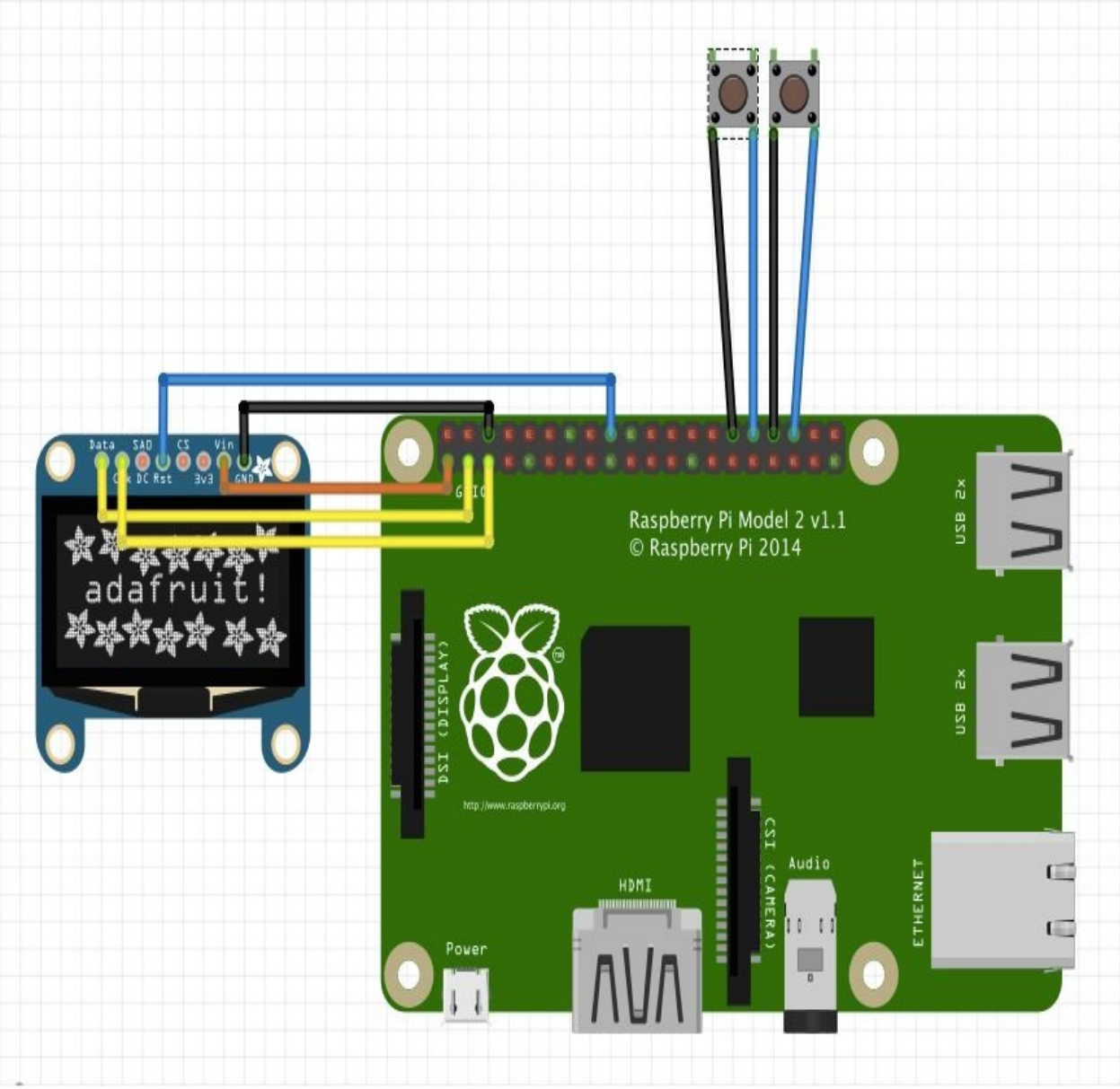
Of course, this is only a subset of what could be displayed. You could fetch weather information, email, latest tweets, etc … You could also have it cycle through the different screen without the need of pushing a button. Anything is possible.

The current code can be found below. It’s far from perfect, but gets the job done.

Did you like this project? You would like to see a specific project? Let me know in the comments!

Schematics

Information Display Schematic



Code

Gist
https://gist.github.com/fvdbosch/d8968325fa5d756045c3.js

Chicken Coop Livestream

We have two chickens, and their coop is located at the back of the garden. Looking for a project idea for element14's Going Green challenge, I decided to build a Pi-based streaming device to check on the chickens and check for eggs.

The idea is to use a Pi Zero with Pi NoIR and a set of IR LEDs which can be enabled/disabled on demand.

Hardware

The hardware required for this project is:

- Raspberry Pi Zero with wifi dongle or Raspberry Pi Zero W
- Raspberry Pi NoIR Camera v2
- Official Pi Zero case with camera lid
- Pi-Supply Bright Pi
- **Optional**: Solar panel with charging circuit and battery

Because the coop is located at the back of the garden, I opted for a regular Pi Zero with external WiFi dongle, ensuring decent connectivity to my home network.

A Pi NoIR combined with Bright Pi's IR LEDs takes care of the night vision in the coop. By glueing Bright Pi to the case's lid and connecting to the Pi's GPIO from the back, a neat little package is obtained.

I'm also experimenting with a 10W solar panel I had at hand from an older kickstarter. The idea would be to have everything powered from a USB battery pack, while recharging it during the day.

This slideshow requires JavaScript.

Software

On the software side of the project, I went for the all-in solution provided by MotionEyeOS, a streaming OS compatible with the Raspberry Pi.

After downloading the software image for my device, I flashed it onto an SD card using Etcher. I copied my *"wpa_supplicant.conf"* file to the SD card, before ejecting and inserting it in the Pi, as per the WiFi Preconfiguration procedure. Upon first boot, the Pi automatically connected to the wireless network and set up the camera. I honestly didn't expect the entire setup to be this easy.

Browsing to the web interface, various options are available to configure frame rate, resolution, camera name, etc.

MotionEye also allows the creation of action buttons. Just what I need in order to enable/disable the BrightPi's IR LEDs. In the *"/data/etc"* folder, I created two files called *"light_on_1"* and *"light_off_1"*.

```
[root@meye-1d01a2ec etc]# ls -l light_o*
-rwxrwxrwx 1 root root 83 Jul 29 20:02 light_off_1*
-rwxrwxrwx 1 root root 83 Jul 29 20:02 light_on_1*
```

Inside those files, a short bit of code to control the Bright Pi via I2C.

```
[root@meye-1d01a2ec etc]# cat light_off_1
#!/usr/bin/env python

from os import system

system("i2cset -y 1 0x70 0x00 0x00")
[root@meye-1d01a2ec etc]# cat light_on_1
#!/usr/bin/env python

from os import system

system("i2cset -y 1 0x70 0x00 0xa5")
```

MotionEye detects these scripts, and adds the button overlays to the video streams.

Power Consumption

One of the goals of this project, is to try to power it from a battery pack and solar panel if possible. In order to hope to achieve that, I tried to lower the power consumption of the project using various little tricks.

A first change is to disable the Pi's and camera's onboard LED. This can be achieved by editing *"/boot/config.txt"* and appending following lines:

```
# Disable the ACT LED on the Pi Zero.
dtparam=act_led_trigger=none
dtparam=act_led_activelow=on

# Disable the LED on the Pi Camera
```

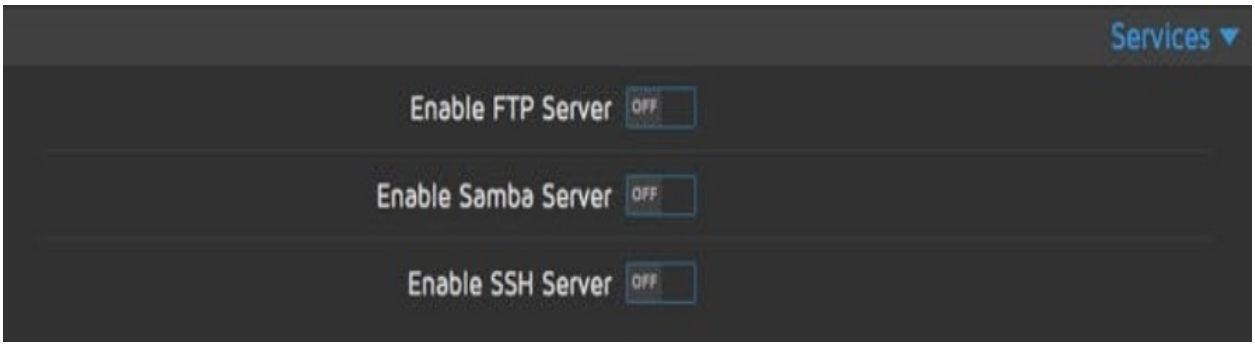
disable_camera_led=1

Another change that can be applied, is to disable the HDMI output, as no display is required for this project. This can be done by adding a cron job using the *“crontab -e”* command, and append:

```
# Disable HDMI
@reboot /usr/bin/tvservice -o
```

While previous changes are applicable to Raspbian, there are some MotionEye specific changes that can be applied as well.

By default, MotionEye enables various services such as FTP, Samba and SSH. Other features such as motion detection are also enabled. It’s always a good idea to disable anything you don’t need, as not only will this save a little bit of power, it will also increase security.



It’s a good idea to explore the settings, including the advanced ones, and disable anything you don’t require.

Finally, the streaming settings play a role as well. The lower the resolution and frame rate, the lower the power consumption. It’s worth experimenting with the settings depending on your streaming needs.

Here are some quick measurements after applying above tricks:

- Measurement Comment
- 140mA idle
- 220mA streaming 320x200 @ 2fps
- 300-400mA streaming 1920x1080 @ 30fps

As I don’t need quality streaming, the lower resolution is fine for my application. Taking a bit of buffer, the average power consumption is 250mA @ 5V, or **1.25W** while streaming**.** About **30W / day**.

My first battery pack had a 5200mAh capacity, but as the cells are 3.7V boosted to 5V, this results (ideally) in **19Wh**. A conversion loss of 10% should be taken into account, meaning the battery would discharge completely after a bit more than half a day.

The current battery pack has a capacity of 20000mAh, (ideally) resulting in

about **74Wh**. Applying the same conversion loss, this battery would still last about two days without recharging.

My solar panel can generate 10W of power in perfect conditions. Assuming an average of 6 hours of sun during the day in summer, in theory, **60W** of energy could be generated.

Long term testing will have to determine if this is a viable solution, though the limited sunlight during the winter will certainly pose a problem.

Infrared

Finally, a little night test using the Bright Pi and Pi NoIR combination.

The IR LEDs can be enabled/disabled on command using the action buttons, as demonstrated in the video below.

Conclusion

Though the solar charging will probably not work out in its current form, it remains a good experiment which will hopefully yield useful data.

The streamer itself works as expected, with MotionEyeOS making the setup and configuration extremely easy. The combination of Bright Pi and Pi NoIR is a good one as well, and being able to control both from within MotionEye is great.

Download Code
Download Motioneyeos
Download ActionButton

Pi0Boat: A Smart Aquatic Vehicle with the Pi Zero

Erle-Boat

Erle-Boat is a Linux-based smart aquatic vehicle that uses robotic frameworks such as ROS (the Robot Operating System) and the award winning APM software autopilot to achieve different navigation modes.

Erle-Boat is ideal for water operations.

Mechanical features

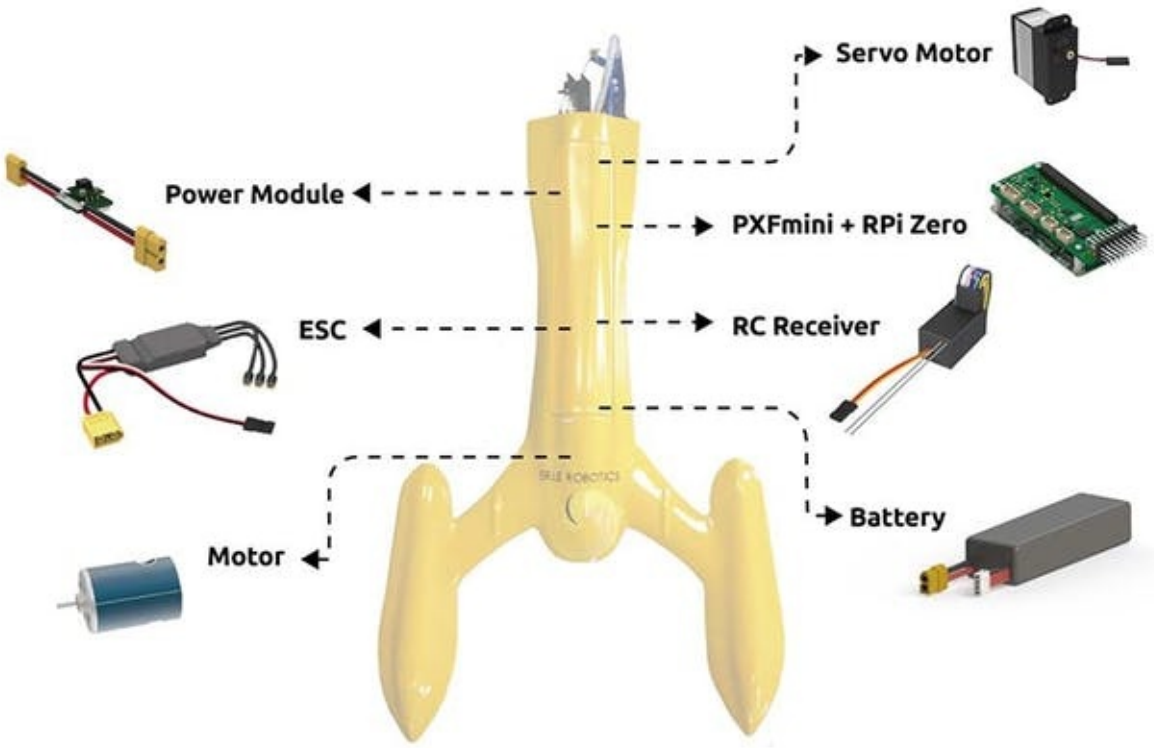
- Dimensions: 520 * 330 mm
- Weight: 770 grams fully assembled (battery included)
- Colour: Yellow

Suggested parts

- PXFmini with WiFi 802.11ac USB dongle, microUSB to USB adapter for WiFi, Flashed MicroSD, Power module (JST GH) and GPS (For PXFmini) [Optional / Improvement]
- Raspberry Pi Zero
- HobbyKing Pod Racer Boat 520mm
- Radio controller

- Battery

Assembly



PXFmini Configuration.

The PXFmini should be configured. Request the latest image for your PXFmini or acquire a flashed one with your order to Erle-Robotics.

Having the PXFmini you need to:

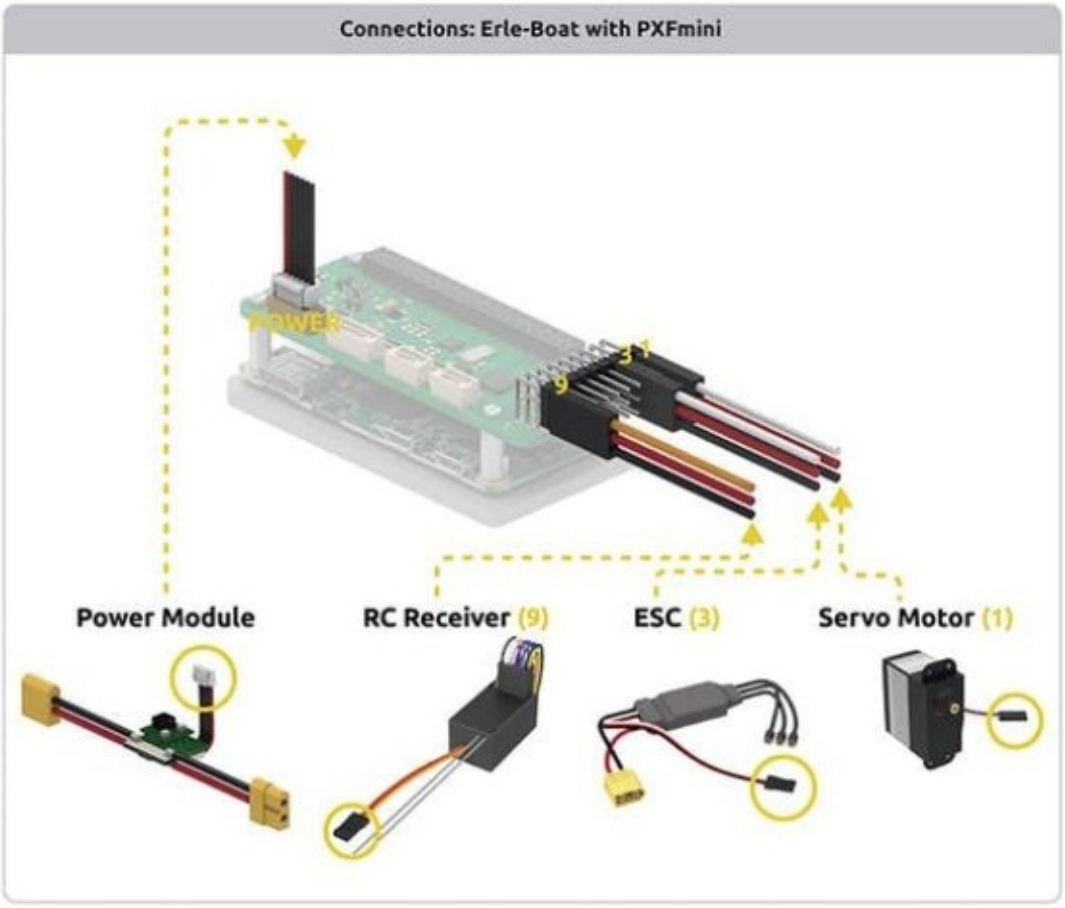
- Set as a Erle-Rover

The easy method is via HDMI with the Erle-Robotics User Interface

- Connect via network to APM

Connect via network to APM and Load Parameters for Erle-Boat

PWM Inputs



The ESC should be connect to the motor and to the PXFmini's PWM #3. The Servo, need to be connected to the PWM #1

Everything in place

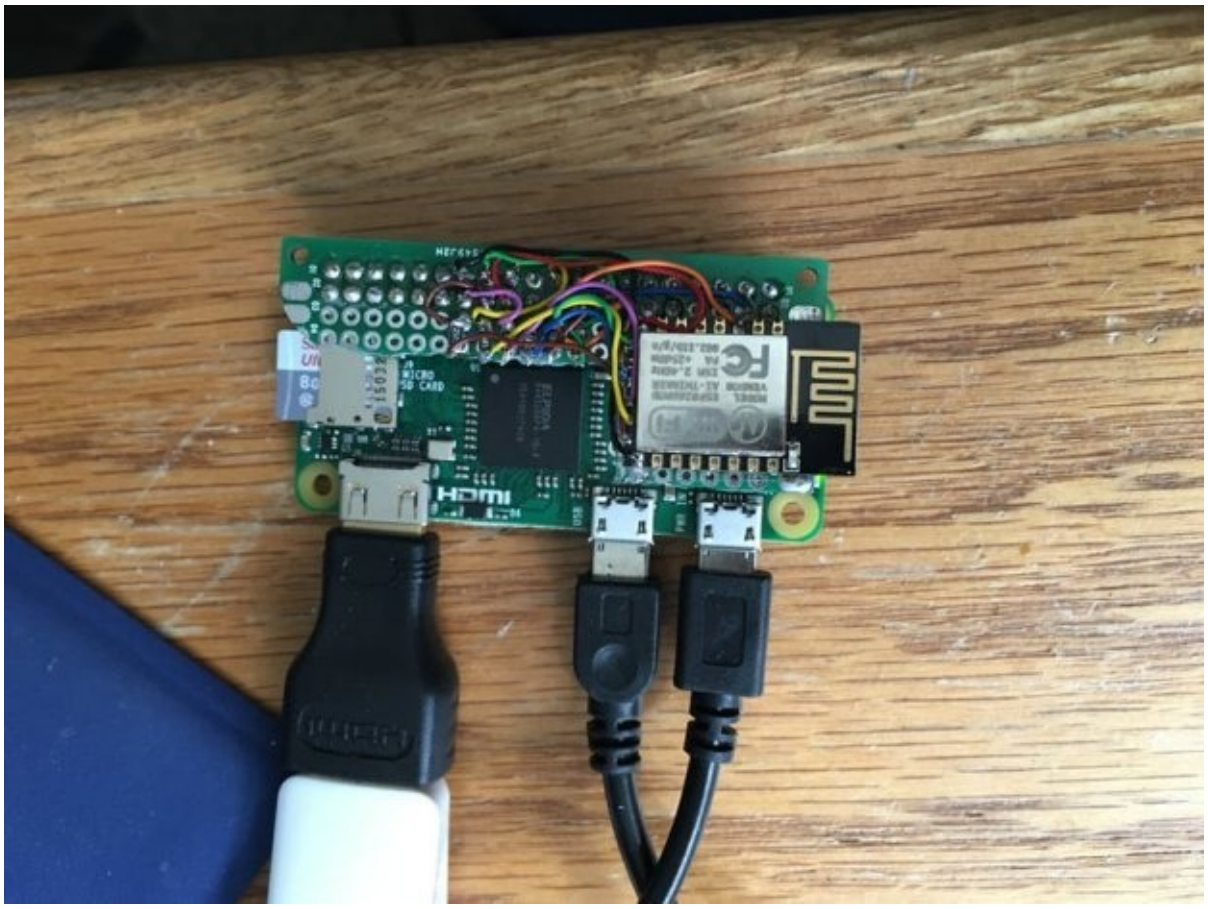
Ensure everything is in place. Check that the motor and servo can move free.

Test

Be careful with the propeller. A suggestion is to make the first test with the Erle-Boat in a controlled area.

Raspberry Pi Zero Wifi Adapter and Windows Backup Server

WiFi on the Zero



I found the Hackaday project to add an ESP8266 Wi-Fi to a Raspberry Pi Zero. It seemed like a great cheap way to get Wi-Fi on my raspberry pi zero without using the USB. This becomes a very minimalist IoT base computer or even a minimal server machine.

For the Wifi, you only need to

- Build the 9 connections between the two boards
- Compile the driver
- Install the driver
- Watch it go.

The software configuration for the PI is in the instruction section of the Hackaday project. Get the driver from the Hackaday instructions.

There's a lot more information in the Hackaday project while they were learning in case you run into trouble.

Driver Doesn't Load

I found that mine has to reset the ESP8266 every few reboots in order to get it working. So, I built a script to do that. It runs 30 seconds after machine startup.

I invoke the script from /etc/rc.local:

```
/home/pi/esp_reset &
```

Here is the script itself:

```
pi@raspberrypi:/etc $ cat /home/pi/esp_reset
#!/bin/sh
sleep 30
sudo modprobe -r esp8089
echo 0 >/sys/class/gpio/export
sleep 1
echo low >/sys/class/gpio/gpio0/direction
sleep 1
echo in >/sys/class/gpio/gpio0/direction
sleep 1
sudo modprobe -r esp8089
```

Console Cable

Then I found that 30 seconds was a long time and sometimes it didn't load. I initially wired up a monitor to the PI but eventually realized that was pretty cumbersome to set it up. I dug around in the spare hardware I had and found a board that could also function as a console adapter for the Pi. It's written up here:

https://www.hackster.io/jweers1/raspberry-pi-console-adapter-f7e9e1?ref=user&ref_id=57327&offset=0

Windows Backup

I've always liked how my Mac is able to do its Time Machine backups.. In Windows 10 I can use "File History" to do basically the same thing.

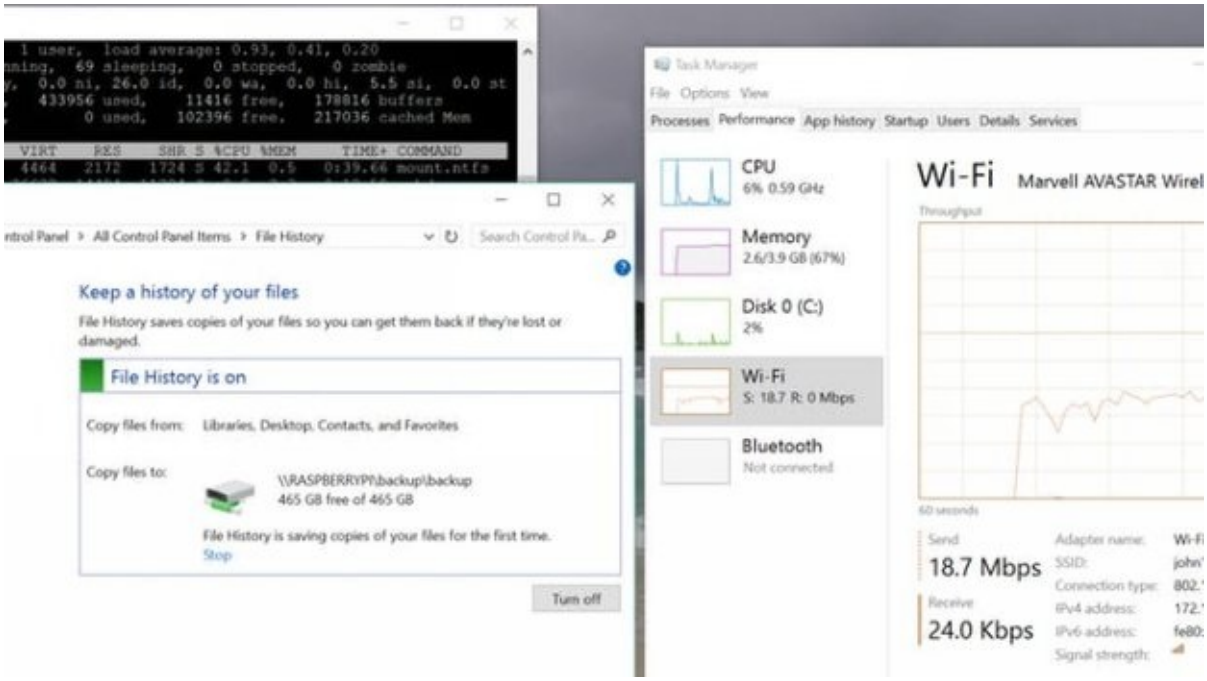
The Zero with Wifi above forms the base server. I found instructions here for installing Samba and setting up a file server:

http://www.cio.com/article/2901051/create-a-home-server-with-raspberry-pi-2.html

Then I plugged in a 500Gb Sata drive to a USB/SATA interface through the USB port on the zero.

Here's what it looks like:

Now all that is needed is to find a dark corner to stick the Server and let it do its job silently.



After placing in the server closet it's been running for several days now 24x7 without any issues.

To Do:

- Stand up Rsync on the PI so I can optionally backup Linux boxes.
- Stand up SSH with a key on the windows laptop so I can backup an SD card automatically. (File history doesn't like the external SD for some reason.)
- Install VPN software so I can backup from anywhere.

Schematics

Download

Code

Download

Smart Conductive 'On Air' Sign!

I always wanted an On Air sign, so I decided to make my own with changing colors, and connected to the internet to let people around me and the world that I am filming, stuck and debugging, on a food break, or done filming.

I painted the sign using conductive paint and I used a Raspberry Pi Zero W and connected it to Bare Conductive's PiCap to read which parts of the paint I touched. I connected three GPIOs to three relays to be able to switch the RBG LEDs off/on. I powered the Pi and the relays with a 5V battery pack and powered the lights with a 12V battery pack.

LEDs + Relays

To be able to control the RGB LEDs, I used a relay to be able to switch each color off and on. Raspberry Pi's GPIOs do not provide enough current to switch the relays or power the LEDs so I built a driver circuit.

The schematic shows how the lights are connected to the relay and to the pi. The Raspberry Pi turns the transistor on, which triggers the relay and the light connected to the relay.

Relay = to be able to switch the LEDs On/Off.

Diode = to provide the path of the current when the coil is switched off. And keep other components from frying.

Transistor = The relay needs a large current to switch, so we use a transistor to amplify the current from the Pi to the relay.

Resistor = To regulate the current from the GPIO to the transistor and have the right current go through the relay.

Check out the schematics to see how it all works!

Raspberry Pi

The Raspberry Pi, should have internet connection to Tweet and will run the script that controls the Pi Cap. No further set ups needed.

Twitter

To tweet my status, I used Raspberry Pi's tutorial on how to get started with Twitter's API. https://www.raspberrypi.org/learning/getting-started-with-the-twitter-api/

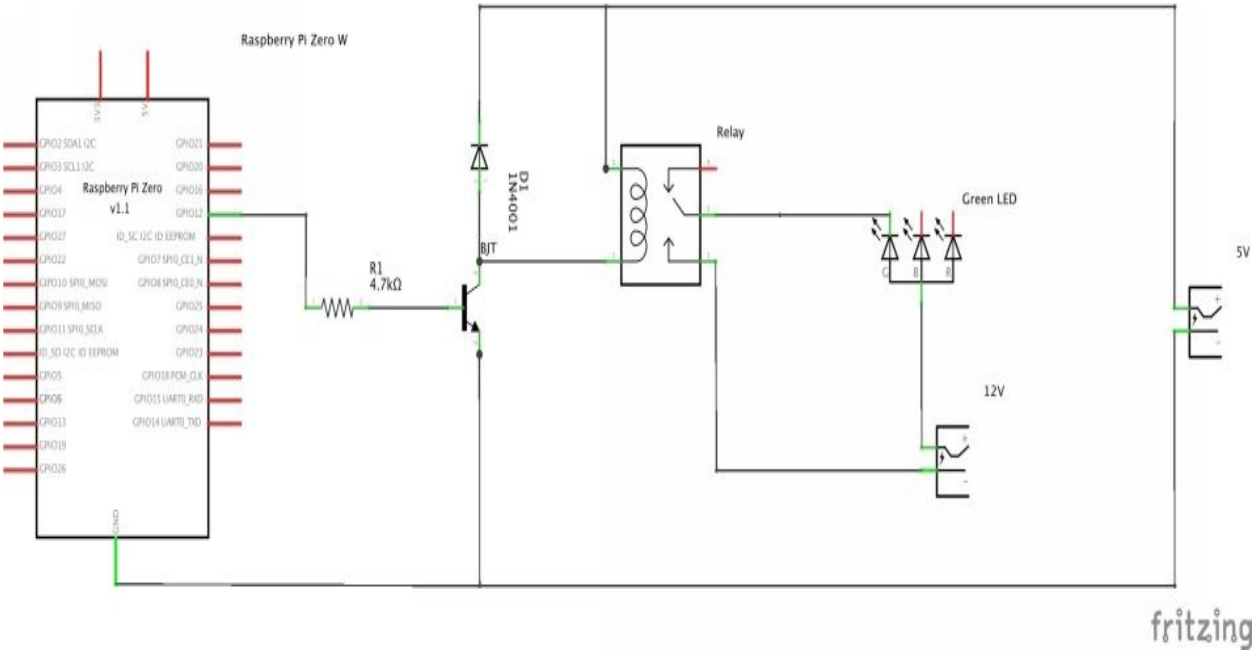
Pi Cap

To get my Pi Cap set up with my Raspberry Pi Zero, I followed Bare Conductive's tutorial: https://www.bareconductive.com/make/setting-up-pi-cap-raspberry-pi-zero/

Schematics

Green Circuit

This is the schematic for connecting one of the LEDs to a Raspberry Pi GPIO. This applies to Red and Blue and go to different GPIOs in the Raspberry Pi.



Code

[Download Sourcecode]

```
import RPi.GPIO as GPIO
import time
from twython import Twython
import signal, sys, MPR121
import datetime

now = datetime.datetime.now()

sensor = MPR121.begin()
touch_threshold = 40
release_threshold = 20
sensor.set_touch_threshold(touch_threshold)
sensor.set_release_threshold(release_threshold)

from auth import (
    consumer_key,
    consumer_secret,
    access_token,
    access_token_secret
)
```

```
twitter = Twython(
    consumer_key,
    consumer_secret,
    access_token,
    access_token_secret
)

#GPIO pin numbers
red = 12
green = 13
blue = 15

GPIO.setmode(GPIO.BOARD)
GPIO.setup(red, GPIO.OUT)
GPIO.setup(green, GPIO.OUT)
GPIO.setup(blue, GPIO.OUT)

def turnColor(R,G,B):
if R == True:
GPIO.output(red, GPIO.HIGH)
else:
    GPIO.output(red, GPIO.LOW)
if G == True:
GPIO.output(green, GPIO.HIGH)
else:
GPIO.output(green, GPIO.LOW)
if B == True:
GPIO.output(blue, GPIO.HIGH)
else:
GPIO.output(blue, GPIO.LOW)

while True:
if sensor.touch_status_changed():
sensor.update_touch_data()
#Loop trough the PiCap's pins
for i in range(12):
if sensor.is_new_touch(i):
if i == 1:
#Neutral State: White LEDs
turnColor(True,True,True)
elif i == 3:
#Started Filming State: Blue LEDs
turnColor(False,False,True)
message = "I started filming a new video! Woooooo! #newvideosoon" + now.strftime("%H:%M:%S")
twitter.update_status(status=message)
print("Tweeted: %s" % message)
elif i == 5:
#Stuck State: Red LEDs
turnColor(True,False, False)
message = "I am stuck. Prototype down. Still filming though.. #debugging #newvideosoon" +
now.strftime("%H:%M:%S")
```

```
twitter.update_status(status=message)
print("Tweeted: %s" % message)
elif i == 7:
#Eating State: Purple LEDs
turnColor(True,True,False)
message = "Pizza time #filmingfood"
twitter.update_status(status=message)
print("Tweeted: %s" % message)
elif i == 9:
#Done Filming State: Green LEDs
turnColor(False,True, False)
message = "I am done filming. Woooooo! #editingtime #newvideosoon" + now.strftime("%H:%M:%S")
twitter.update_status(status=message)
print("Tweeted: %s" % message)
else:
#Neutral State: White LEDs
turnColor(True, True, True)
```

Echo Starts your Car

Using your voice to control everything has always been very interesting. With my shiny new echo I could connect all the usual "connected home" devices, but I really wanted the echo to keep my car warm and toasty on a winter morning. In this instructable we are going to make the Amazon Echo start any car. We will do this by programming a Raspberry pi zero to act like a Belkin Wemo switch, named My Car. You can then ask "Alexa turn ON my car"

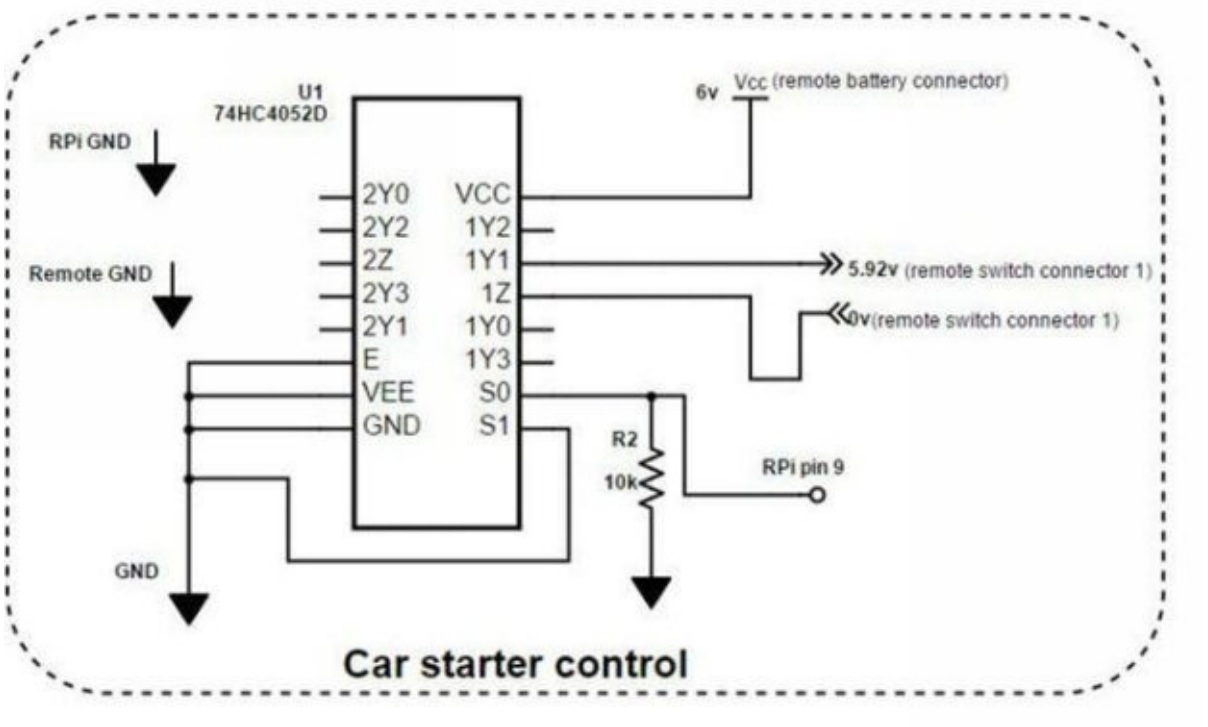
1. Any remoter starter (compustar, drone mobile, Viper(viper has web APIs you can use :))) + 1 extra remote (45$)

2. One 74HC4052 (~1$)

3. Pi-Zero (5$)

5. USB OTG + Wifi Adapter (15$)

Step 1: Build your Circuit



Open up the remote starter remote and disconnect the battery before you start. You could get away with connecting two wires on the opposite sides of the start button on your remote [shown in pic]. Every remote is wired differently, and you might have to figure out which two leads of the switch short when the start button is pressed. Also make sure you draw out the battery positive and battery negative(GND), this will power the 74HC4052 (Analog Mux). Now build the circuit shown in pic. This IC can support multiple buttons, but we are only using one channel right now. Make sure you short the enable pin to GND.

Step 2: Setup your Pi

Now all you need to do is clone this repo to your home directory on the raspberry pi.

git clone https://github.com/sajingeo/EchoCarStarter

Run the device.py,

cd EchoCarStarter/

sudo ./device.py

Go to settings>connected home on the alexa app and run "**Discover devices**".

You should see a device "**My Car**" show up on the list. Now go ahead and ask Alexa to turn on your car. If it does not work the first time, retry "**Discover devices**"

Step 3: Auto Start the Scripts

Make sure all the scripts inside EchoCarStarter directory are executable (**run chmod 777 ***). Time to get more lazy :)

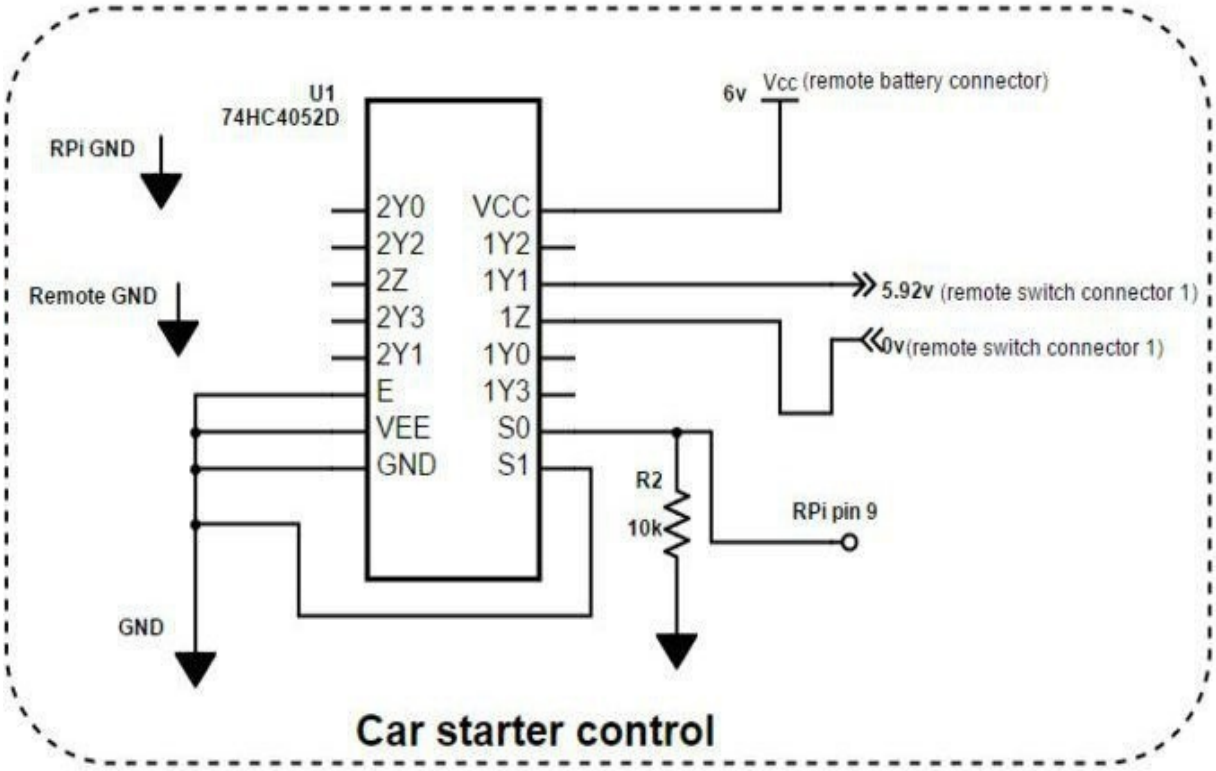
To run the script on startup (boot) edit the **/etc/rc.local** file**sudo nano /etc/rc.local**

add this before exit.

sudo /home/pi/EchoCarStarter/device.py

also run "sudo raspi-config" and enable wait for network option (4) - assuming you have already setup the wifi or Ethernet on the pi zero.

Schematics

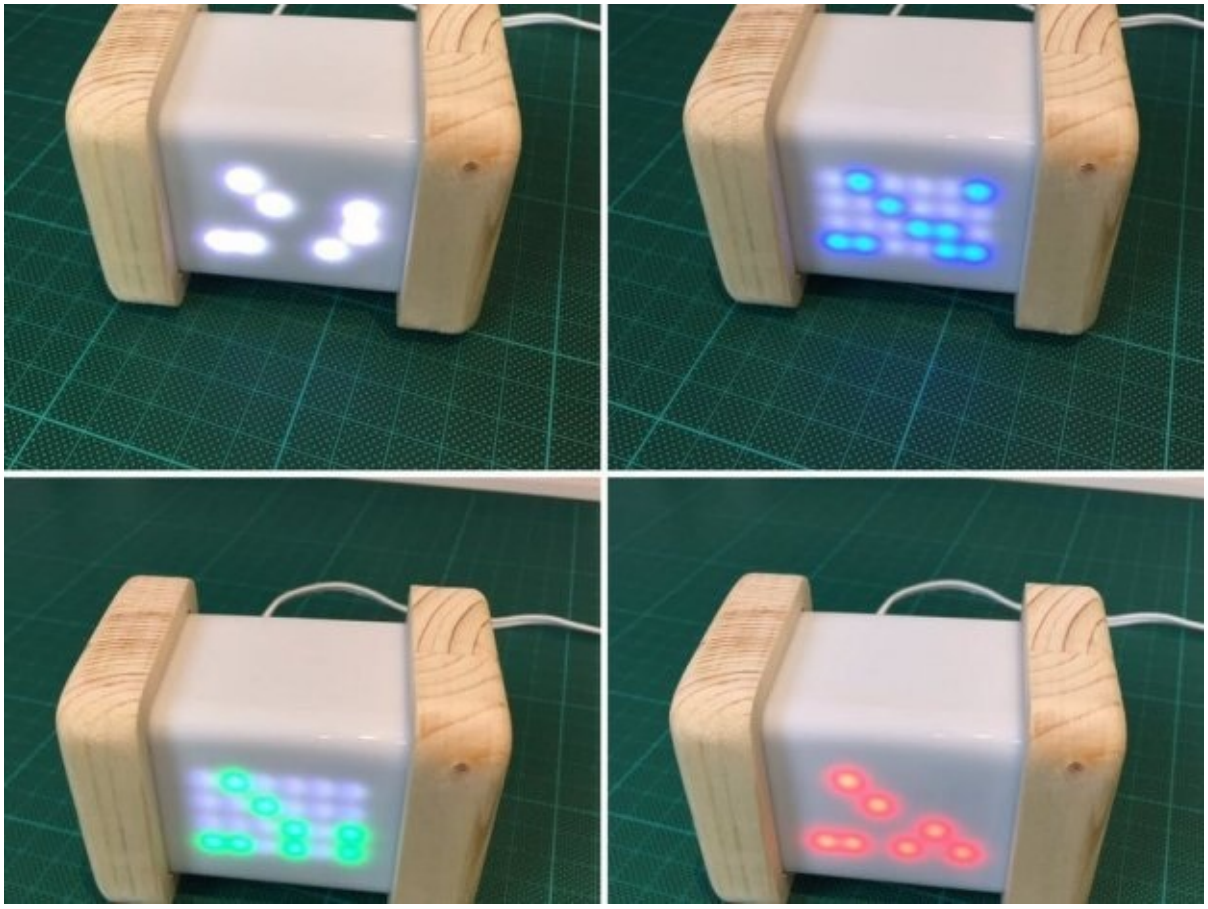


Code

Github
https://github.com/sajingeo/EchoCarStarter

[Download Sourcecode]

Binary Clock



I've had a Unicorn pHAT sitting in a box for a while, but I finally used it in a simple project. Because it has four rows of LEDs, I thought it would be ideal to make a binary clock from.

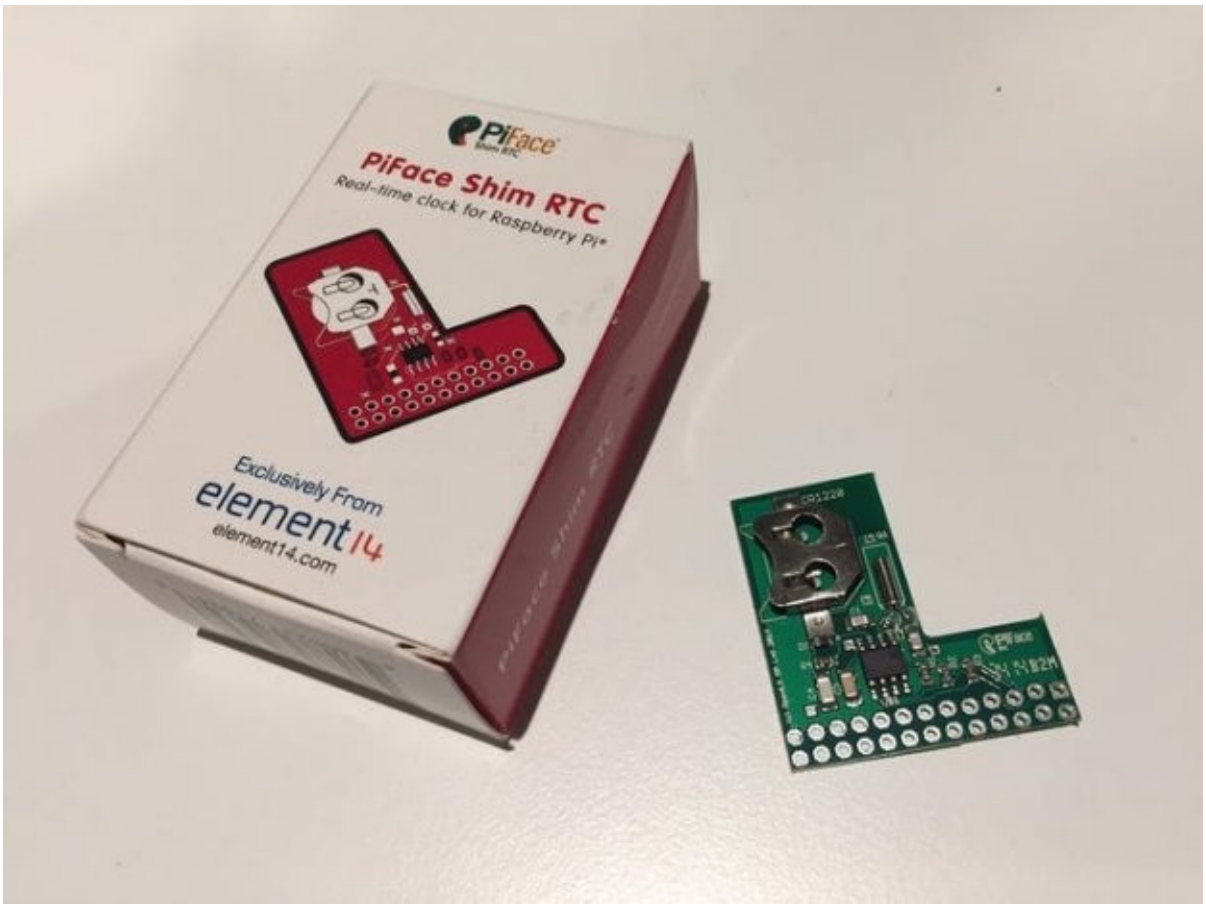
Let me show you how ??

Hardware

This project can be tackled in two ways:

- an online version fetching time via NTP (network time protocol)
- an offline version using a RTC (real time clock)

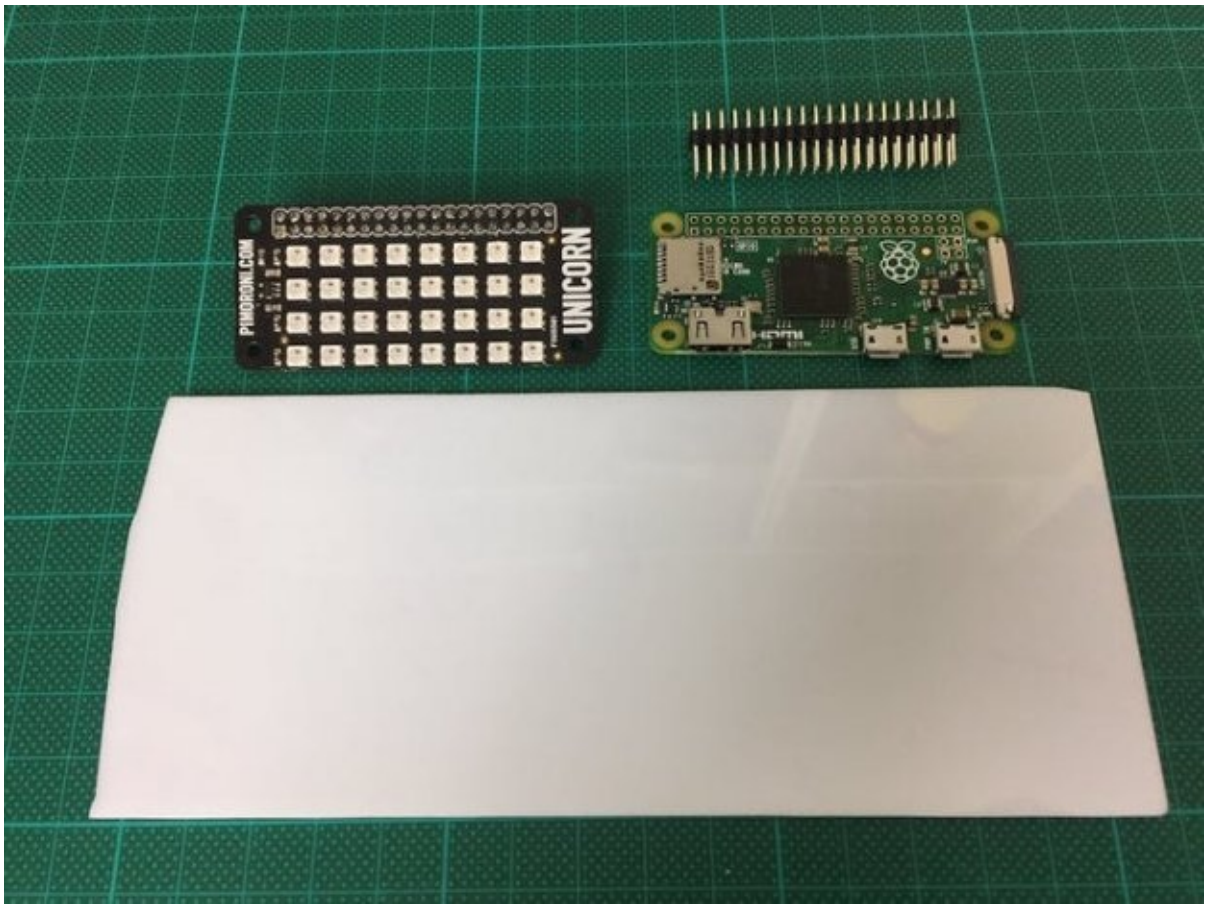
In this instance, I'm focusing on the *online* version, though the offline version could be achieved as easily with an RTC shim such as the one pictured below, which can slide on the GPIO header between the Pi Zero and the Unicorn pHAT:



Electronics

At the center of the project are a PiZero and a Unicorn pHAT. The PiZero fetches the time, converts it to binary and controls the Unicorn pHAT with a Python script. The Unicorn pHAT's RGB LED matrix can be used to display those binary values in any colour.

In order to have the Unicorn pHAT face forward, while still having the Pi Zero connectors at the back, an angled header was used.

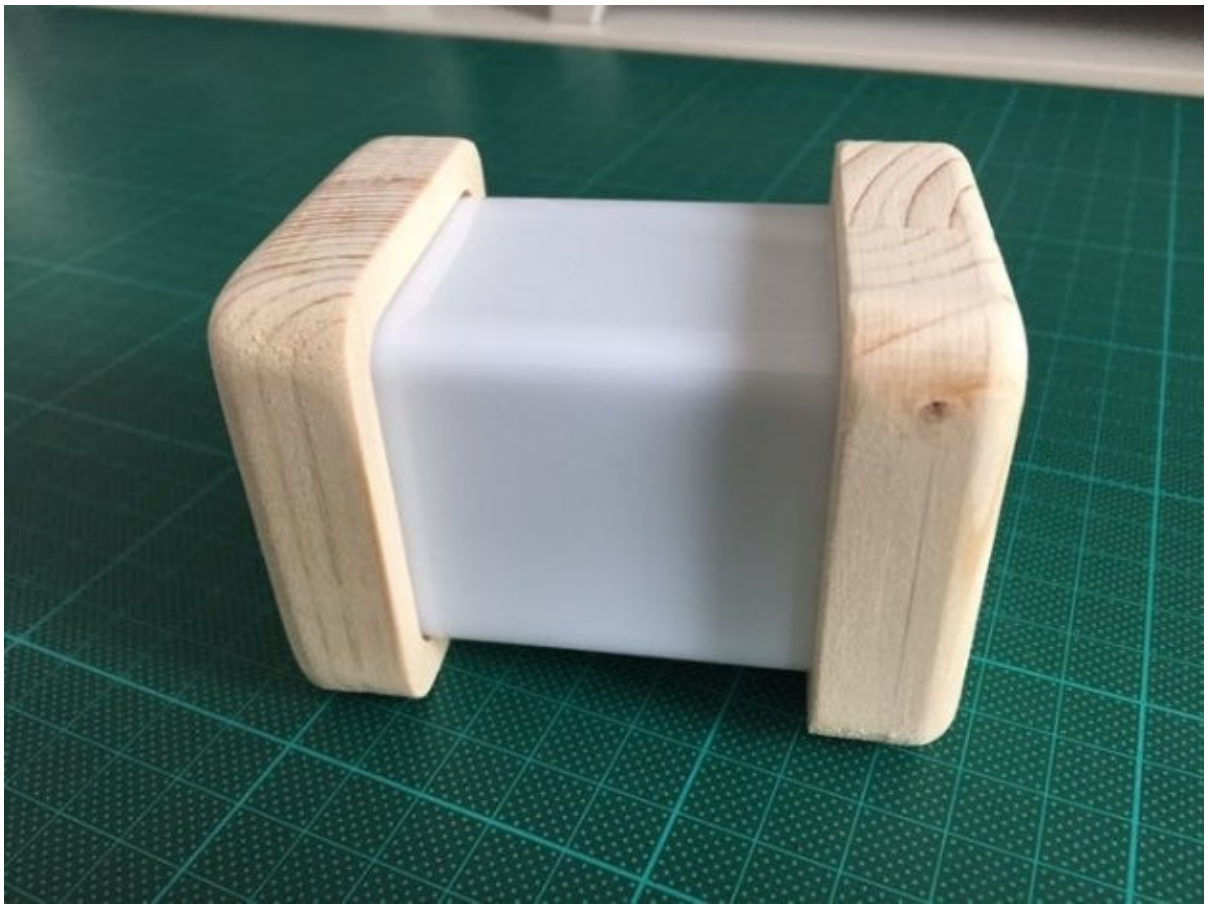


Enclosure

The enclosure is a combination of acrylic and wood, except this time, the acrylic makes for the majority of the enclosure.

To form the acrylic in the desired shape, a small torch was used to heat it and have it bend 90 degrees. Once cooled off, it retains the new shape. Repeating this action two more times, the final shape is achieved. Careful not to put the torch too close to the acrylic to prevent it from burning or leaving marks!

To finish the enclosure, two wooden “end caps” were milled using the CNC and the edges rounded with a router, though a similar result could be achieved with a bandsaw and some manual sanding for example.



Software

On the software side of things, I started off with the latest Raspbian Jessie Lite image from 23/09. Set up network connectivity via wifi by editing the "*wpa-supplicant.conf*" file.

Once network connectivity was set up, I proceeded with the upgrade of the software:

```
pi@binaryclock:~ $ sudo apt-get update && sudo apt-get upgrade -y
```

Don't forget to make sure the time is set to the correct zone:

```
pi@binaryclock:~ $ sudo cp /usr/share/zoneinfo/Europe/Brussels /etc/localtime
```

Unicorn pHAT

Installing the software to control the Unicorn pHAT and its dependencies, is a breeze as always thanks to Pimoroni's installation wizard:

```
pi@binaryclock:~ $ curl -sS get.pimoroni.com/unicornhat | bash
```

I chose a full install, including examples, though they are not required for this project. To know more about the installer, head over to the Unicorn HAT GitHub page.

Binary Clock

For the binary clock, I created s mall script which takes the time, converts it to binary format and lights up the correct LEDs on the Unicorn pHAT. A cronjob ensures the script is started at boot time.

Here’s the full script:

View the code on Gist

Make sure the downloaded script is executable:

```
pi@binaryclock:~ $ sudo chmod +x binary.py
```

Finally, add the cron job:

```
pi@binaryclock:~ $ sudo crontab -e
...
# m h dom mon dow command
@reboot sudo /home/pi/binary.py
```

Upon reboot, you should see the LEDs light up on the Unicorn pHAT!

You are of course free to change the colours for both the time and the background:

Hope you liked this project, let me know in the comments!

Code

Code snippet #5Plain text
pi@binaryclock:~ $ sudo crontab -e

...
m h dom mon dow command
@reboot sudo /home/pi/binary.py

Gist
https://gist.github.com/fvdbosch/11ccbbda153a8e4ea8d7230df79dad65.js

```
#!/usr/bin/env python
```

```
import unicornhat as unicorn
```

```
import time
```

```
unicorn.set_layout(unicorn.PHAT)
```

```
unicorn.brightness(0.5)
```

```
unicorn.rotation(180)
```

```

while 1:
    
    decimal = time.strftime("%H%M%S")
    
    decimal_list = list(decimal)
    
    
    for x in xrange(0, 6):
        
        binary = bin(int(decimal_list[x]))[2:].rjust(4, '0')
        
        binary_list = list(binary)
        
        
        for y in xrange(0, 4):
            
            if binary_list[y] == '1':
```

```
unicorn.set_pixel(x+1,y,255,255,255)
```

```
else:
```

```
unicorn.set_pixel(x+1,y,0,0,0)
```

```

```

```
unicorn.show()
```

```
time.sleep(1)
```

Github
https://github.com/pimoroni/unicorn-hat

Download Sourcecode

To be continued to Book Part 2 .....