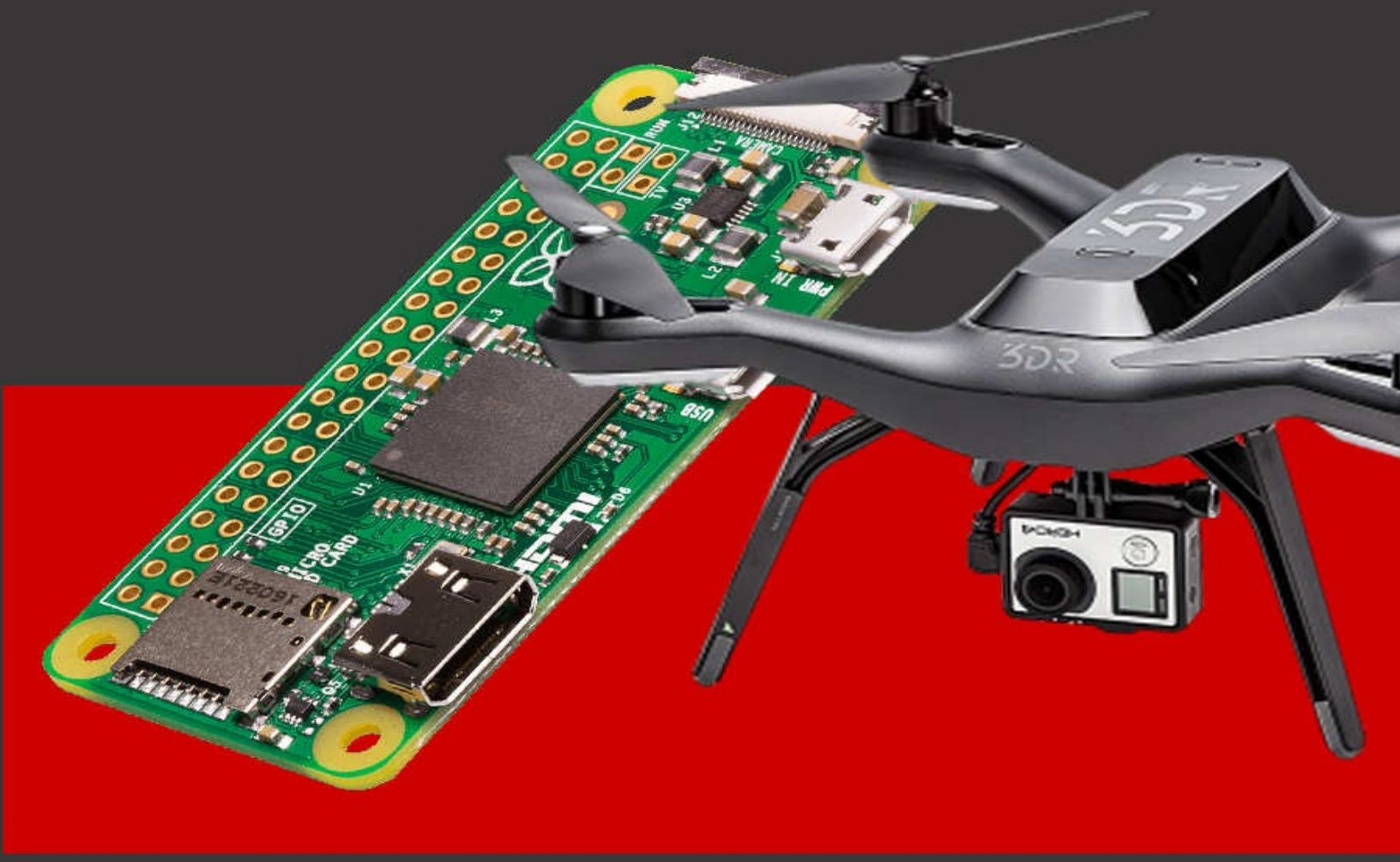
Best Raspberry PI Zero Project For Home Automation Part 2



Home Automation Tutorial

Best Raspberry PI Zero Project For Home Automation Part 2

Index Of Contents

We've seen a lot of DIY home automation projects over the years, but Instructables user electronichamsters shows off one of the most complete systems we've seen yet. With it, you'll be able to monitor just about everything in your house.

Instead of concentrating on the simple things like lights or automated blinds, elecronichamsters goes all in. With his system you can monitor for water leaks, see if the garage door is open, check for new mail, watch for movement, sense for gas, and even see how the dog is doing. The system uses both an Arduino

and a Raspberry Pi alongside a ton of various sensors to monitor the house. You can set up the system to send you alerts when something happens or just monitor everything from your phone.

Here Best Raspberry PI Zero Project For Home Automation Part 2

Home Automation using Raspberry Pi and Nodejs

The Story

Well I moved to a new apartment and right at this time I had heard many things about the popularity of the IoT and the whole home automation stuff. So I thought, why not create an Web application to control my devices at home.

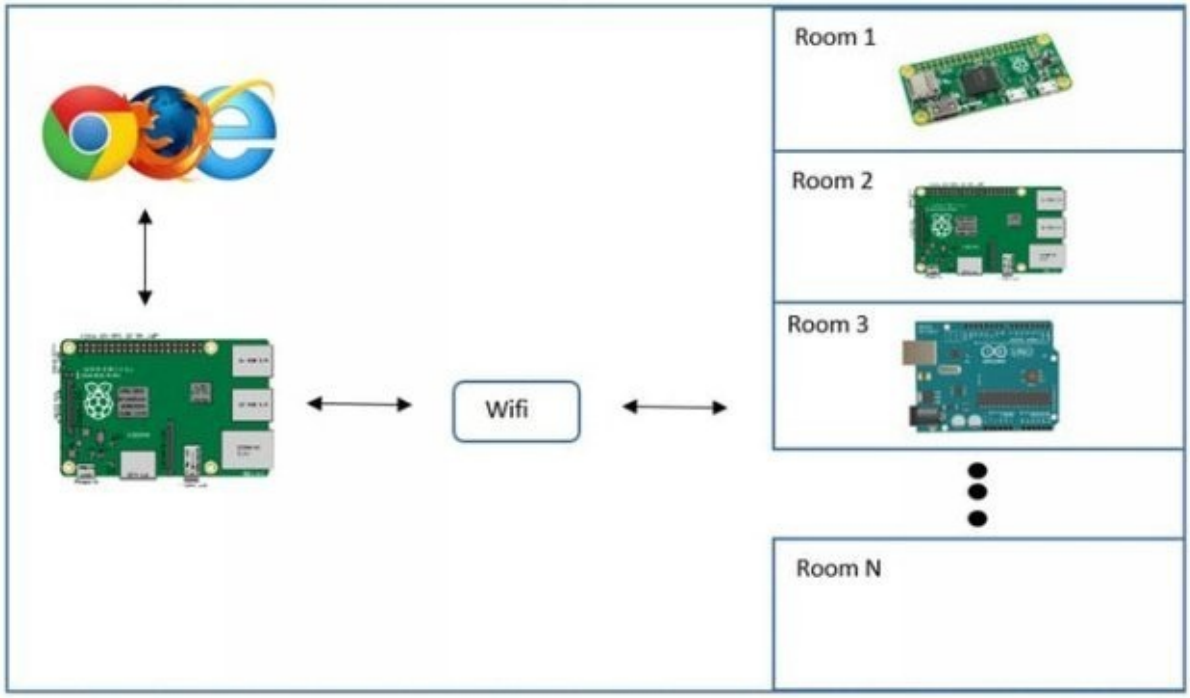
Why a web application?

I had to decide if I want to write my app with C# or with a Web language. I was really intrested to make a real time application, so I decided to develop my app with NodeJs.

First things first

Before starting the project, let me explain the basics first.

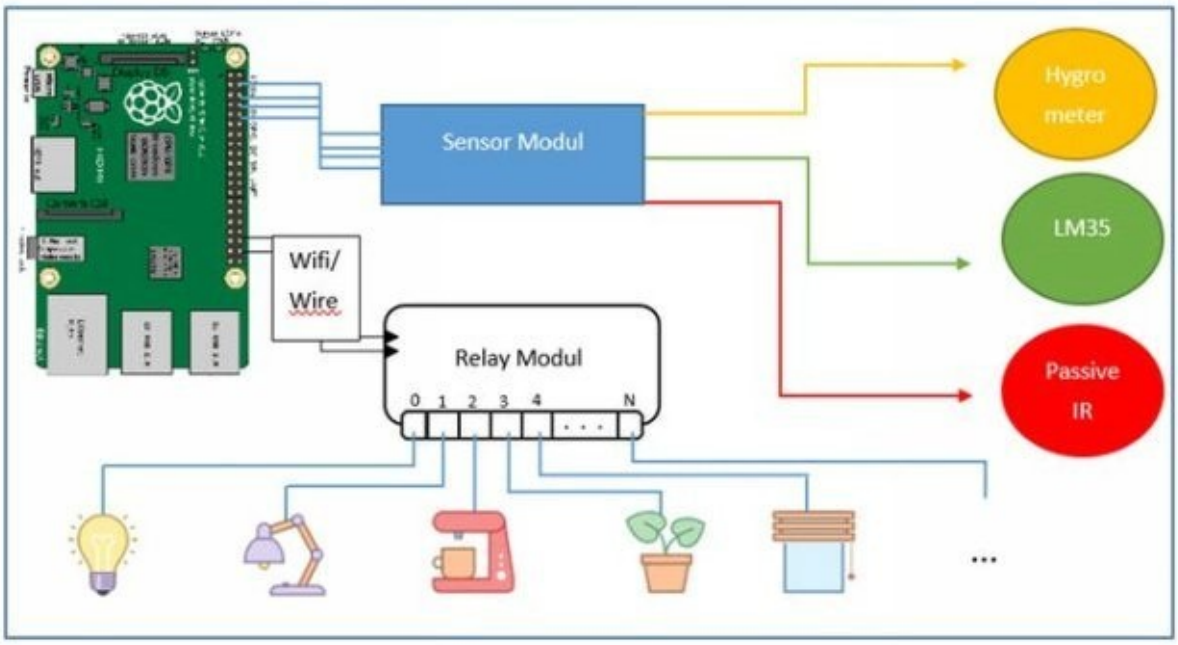
I will use the Raspberry Pi as a master device, which can be controlled via the Browser/Internet. For each device and room, that I want to control, I use another Raspberry or Arduino as slave. The slaves will take command form the master via WiFi or via 433mHz. I will explain this better later in this project.



Device and Room

The Raspberry Pi that is configured as a slave will take some command from the master via some curl commands. How does this works? Well I configured my Raspberry Pi (slave) as webserver and on the webserver I host some PHP files. Now if I open those file via the browser or via an curl command, the Raspberry Pi opens the PHP file and run automatically a Skript with commands.

Now considering room scenario, an Raspberry Pi and Arduino will control devices and reads sensor data. Periodically, each room will have multiple controllable devices (i.e. Light(s), Fan, Wall Socket(s), Coffe machine, etc.), one PassiveIR (to detect human presence in the room), one temperature sensor (LM35 to collect room temperature) and LDR (to detect light intensity near room window).



Device Address Mapping

To identify my devices i will give each device a solid IP Adress and for each command a PHP file.

For example:

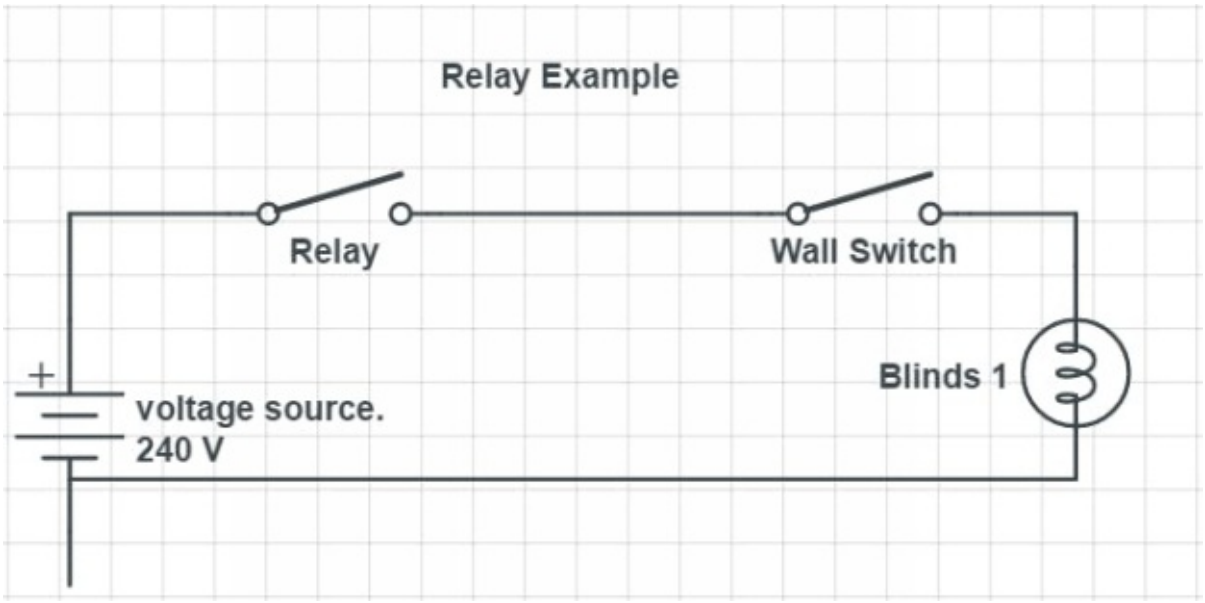
Coffemachine = IPADDRESS/espresso.php
Coffemachine = IPADDRESS/Doppio.php
Plant = IPADDRESS/whaterOnAll.php
Lights = IPADDRESS/LightID/RoomName/ON
Lights = IPADDRESS/LightID/RoomName/OFF

Note: I have also some Hue lights witch will be controlled via Hue API.

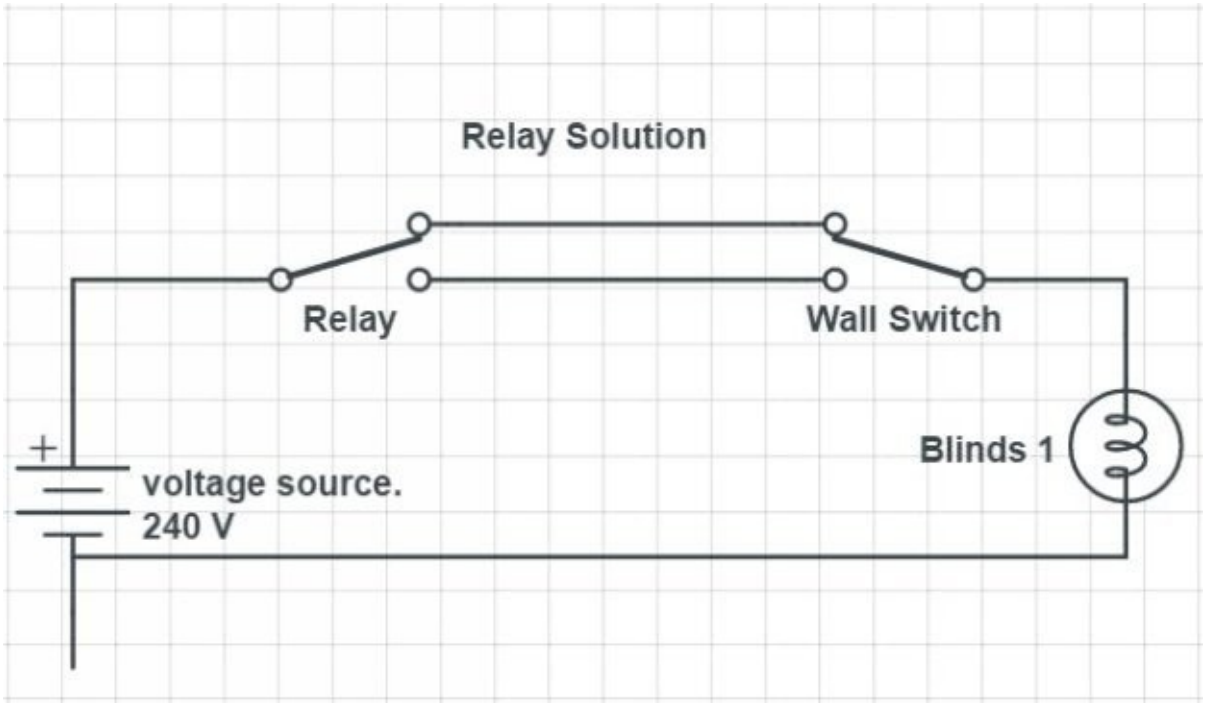
Hue Lights = IPADDRESS/api/{User-ID}/StateAndComands/Value

Basics of relays

The main problem with the Relais is, if you have a device that is connected to the relay, and the relay is set to off, you can't use the device anymore obviously. So the wall switch for the lights gets useless.



To resolve this problem you can either use a DPST relay or use a transistor. I will explain this later again. This is how I did it:



Install NodeJS and Setup the Raspberry

//Requirements
sudo apt-get update
sudo apt-get install xrdp -y
sudo apt-get install git-core git scons build-essential scons libpcre++-dev libboost-dev libboost-program-options-dev libboost-thread-dev libboost-filesystem-dev -y

//Install NodeJS
wget http://nodejs.org/dist/v6.3.0/node-v6.3.0.tar.gz

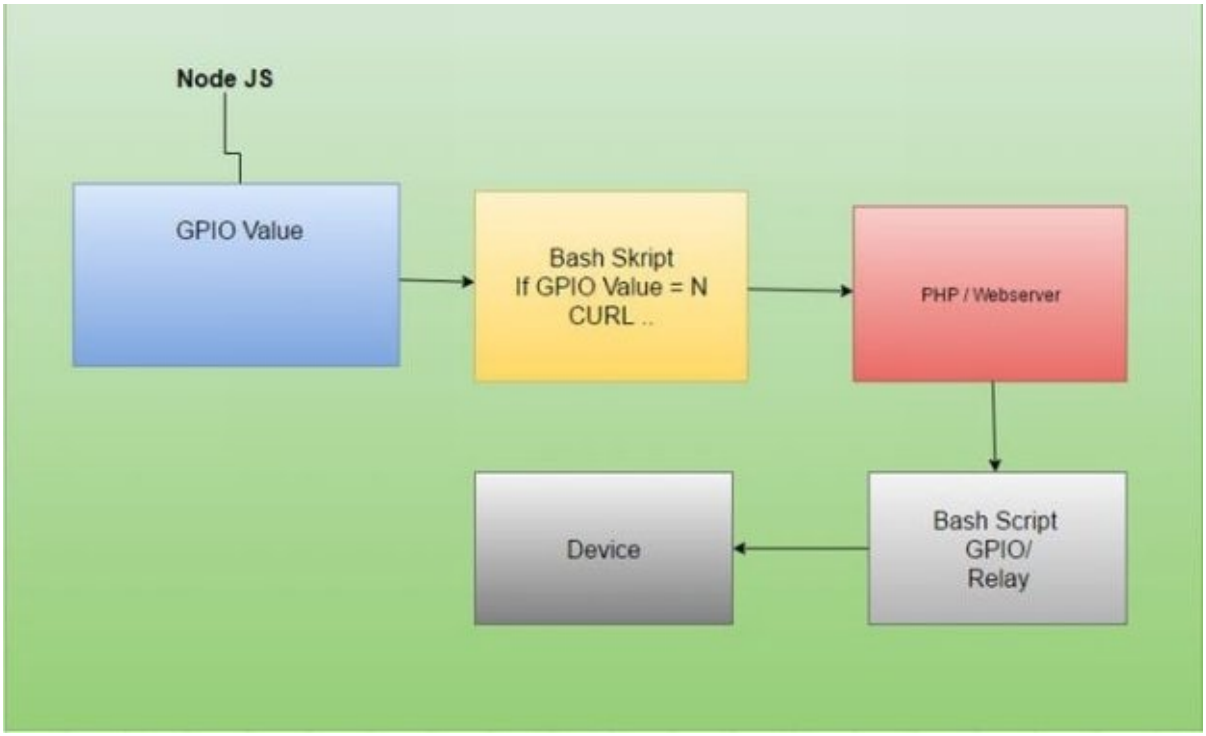
```
tar -zxf node-v6.3.0.tar.gz
cd node-v6.3.0
./configure
make
sudo make install
sudo sh install-node.sh
node --version
npm ---version
sudo node


// others
Sudo apt-get install apache2 -y
Sudo apt-get install proftpd -y
```

Configure NodeJS

```
/* First App with nodeJS by Weslley De Souza*/
/* Variables and Requires */var express = require("express")
, app = express()
, http = require("http").createServer(app)
, bodyParser = require("body-parser")
, io = require("socket.io").listen(http)
, _ = require("underscore");
app.locals.moment = require('moment');
var datetime = require('node-datetime');
var raspi = require('raspi-llio'); //Comment out, to run on Windows PC


/* Server config *///Server's IP address
app.set("ipaddr", "192.168.0.53");
//Server's port number
app.set("port", 8080);
//Specify the views folder
app.set("views", __dirname + "/views");
//View engine is Jade
//app.set("view engine", "jade"); //If you want to code with Jade remove the "//"


//Specify where the static content is
app.use(express.static("public", __dirname + "/public"));
//Tells server to support JSON requests
app.use(bodyParser.json());
/* Server routing */


//Handle route "GET /", as in "http://localhost:8080/"
app.get("/", function(request, response) {
//Render the view called "index" response.render("index");
});
/* Socket.IO events */
io.on("connection", function(socket){
console.log('Client has connectet to the server');
});
//Event on disconet (Is not working)
io.on("disconnect", function(socket){
```

```
console.log('Client has disconnectet !');
});

//Output IP & Port
http.listen(app.get("port"), app.get("ipaddr"), function() {
console.log("Server wurd gestartet und ist unter http://" + app.get("ipaddr") + ":" + app.get("port") + "
erreichbar");
});
```

Scheme of how NodeJs communicates with the other devices:



I don't know the command to send some curl command directly from NodeJs, so I made a little loop script that checks the value of the GPIO pins. So if I write with NodeJs the GPIO pin 1 to value 1, the script run a command for GPIO 1.

For example: `if ($pin1 == 1){ curl -s 192.168.0.60/espresso.php}`

Configure the devices

All Raspberry Pi slaves will be connect via Wireless, and the Wlan Power Saving will be set to Off!

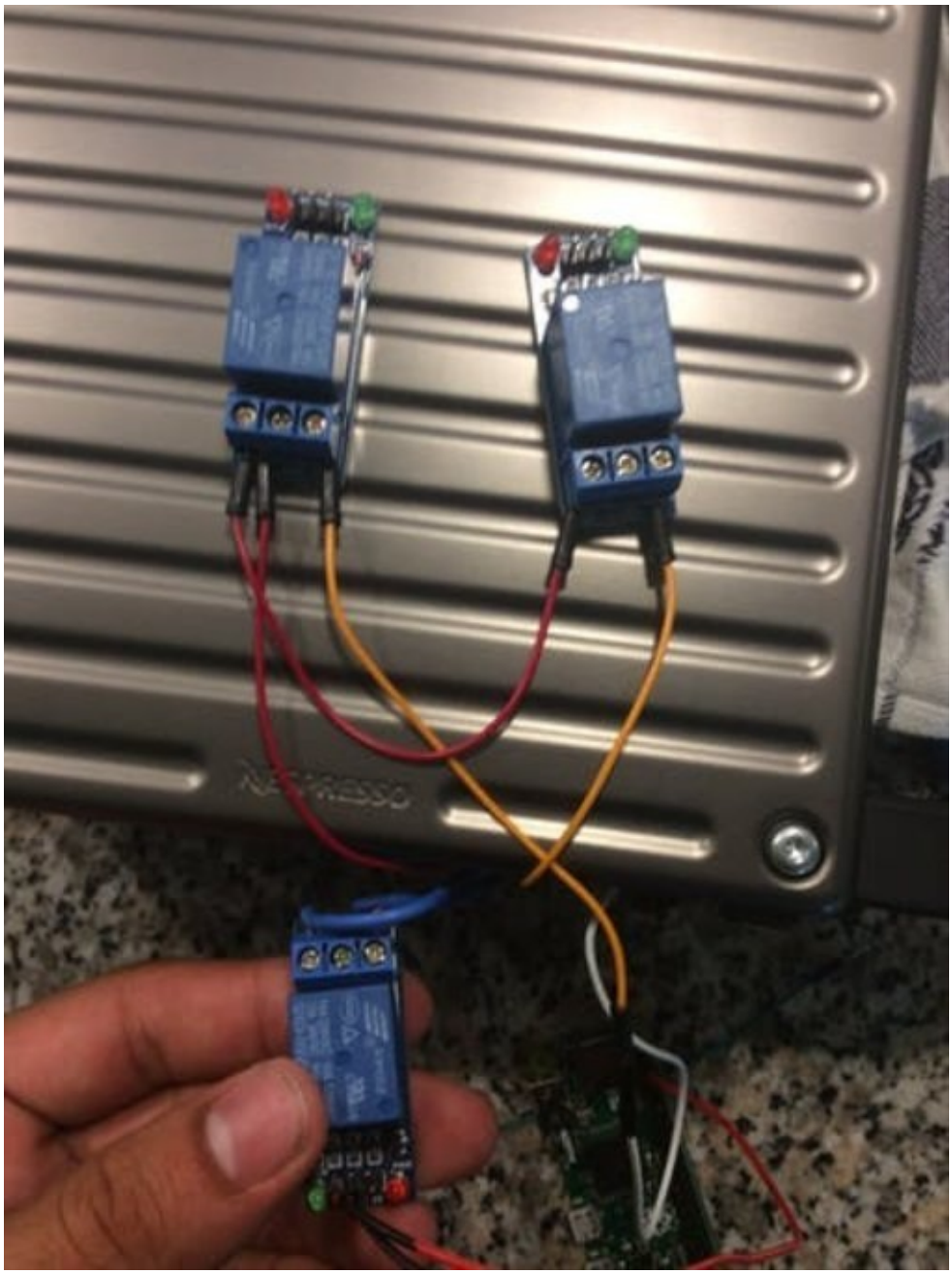
```
sudo nano /etc/network/interfaces
//add line:
wireless-power off
sudo nano /etc/modprobe.d/8192cu.conf
//add line
options 8192cu rtw_power_mgnt=0 rtw_enusbss=0
```

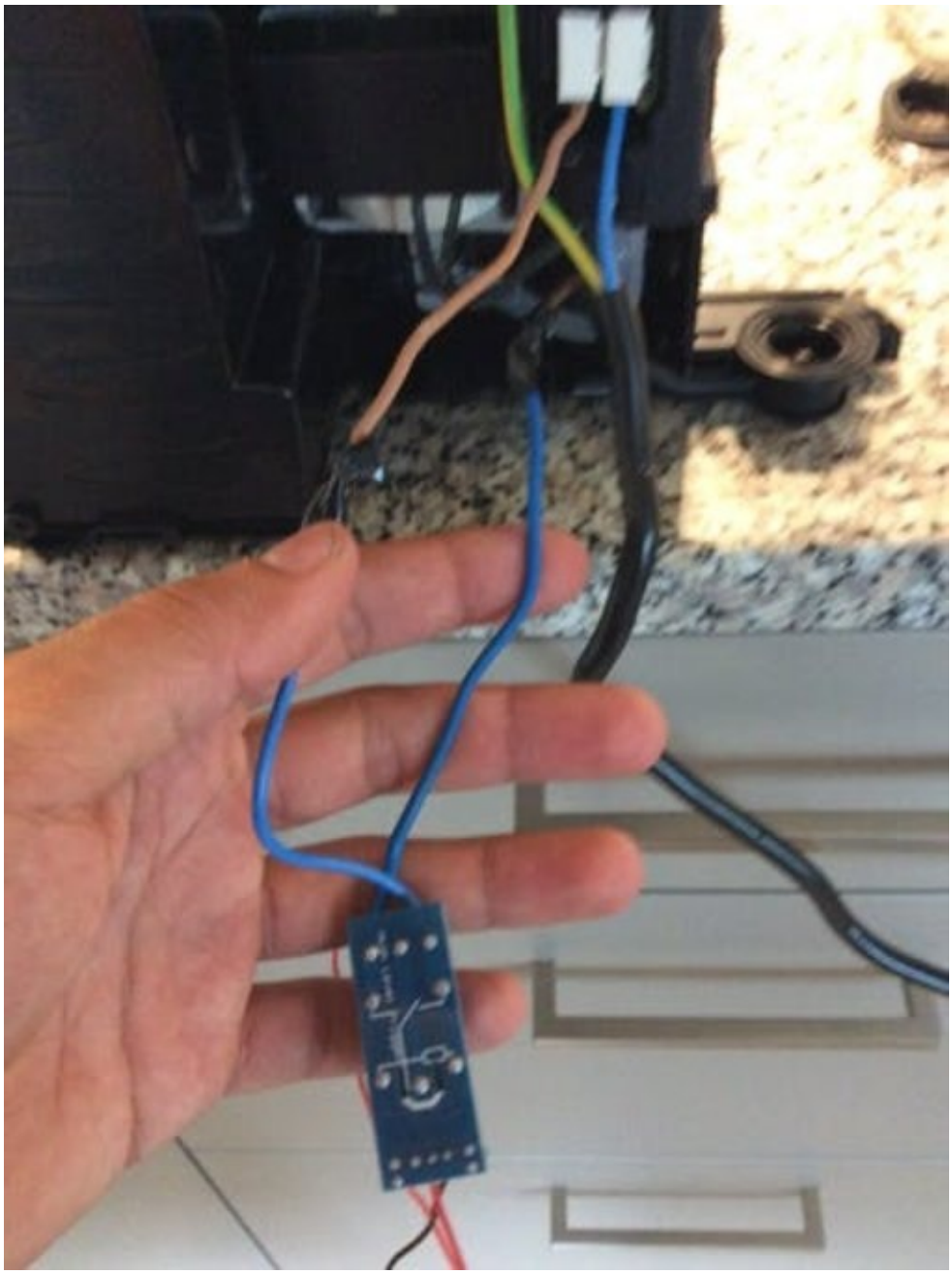
Espresso Machine

Example how to set up a Device to control it. I had at home a Pixie Coffee

machine so I used that for my project.

How to wire





Wire the Relay to the Raspberry. Script example for Doppio:

pi@192.168.0.60's
#!/bin/bash

echo "Doppio"
gpio export 4 in
gpio export 13 in

echo "Machine is on"
#Warming up machine
gpio export 3 out

```
gpio -g write 3 1
sleep 34


echo " Machine is ready"
gpio export 4 out
gpio -g write 4 1
sleep 11.5
gpio -g write 4 0
sleep 0.5
gpio -g write 4 1
sleep 10
gpio -g write 4 0
sleep 0.5
gpio -g write 4 1
sleep 10.9


echo "Doppio serviert"
gpio -g write 4 0
gpio export 4 in
sleep 1
echo "Machine is off"
gpio -g write 3 0
```

Lights Hue API

Server site code example:

```
//------------------------------Connect to Hue Light ------------------------------//var hue = require("node-hue-api"),
HueApi = hue.HueApi,
lightState = hue.lightState;
var displayResult = function(result) {
console.log(result);
};
var displayError = function(err) {
console.error(err);
};
var displayStatus = function(status) {
console.log(JSON.stringify(status, null, 1));
};
var host = "192.168.0.50",
username = "UAW718bnyTbHknwnu1JvGlokRbGoAVIVFxDwG**",
api = new HueApi(host, username),
state = lightState.create();
// Return Hue Online State
io.sockets.on('connection', function (socket) {
socket.on('HUElights', function (connections) {
console.log("Hue Online");
})
//-------------------------Toogle Light on Off----------------------------//
//Name of Button function = DayNight //
io.sockets.on('connection', function (socket) {
socket.on('DayNight', function(DayNight){
```

```
console.log(DayNight);
if (DayNight == false) {
    io.sockets.emit('DayNightFalse');
}
else{
    io.sockets.emit('DayNightTrue');
    api.setLightState(1, state.off())
    .done();
    setTimeout(function () {
    api.setLightState(2, state.off())
    .done();
    },5000)
    setTimeout(function () {
    api.setLightState(3, state.off())
    .done();
    },10000)
}
});
});
//------------------Button Light SLEEPROOM ----------------------------//
socket.on('LightOben', function (LightOben) {
console.log(LightOben);
if(LightOben == 1){
    api.setLightState(3, state.on())
    .done();
    io.sockets.emit('LightObenTrue');
}
else {
    api.setLightState(3, state.off())
    .done();
    io.sockets.emit('LightObenFalse');
}
})
//----------------------Button Light All-----------------------
socket.on('LightAll', function (lightAll) {
console.log(lightAll);
if (lightAll == true) {
    api.setLightState(3, state.on())
    api.setLightState(2, state.on())
    api.setLightState(1, state.on())
    .done();
    io.sockets.emit('LightAllTrue');
}
else {
    api.setLightState(3, state.off())
    api.setLightState(2, state.off())
    api.setLightState(1, state.off())
    .done();
    io.sockets.emit('LightAllFalse');
}
})
});
```

Http site code:

```
//------------------Light Event----CheckBox Sun Moon ------------------------------//
function daynight(id) {
var SunMoon = document.getElementById(id).checked;
socket.emit('DayNight', SunMoon);
}
socket.on('DayNightFalse', function (DayNight) {
console.log("False");
    $('#toggle--daynight').prop('checked', false);
});
socket.on('DayNightTrue', function () {
console.log("True");
$('#toggle--daynight').prop('checked', true);
$('#toogle-modus').css({'opacity': 0.2}, 1000);
setTimeout(function () {
    $('#toggle--daynight').prop('checked', false);
    $('#toogle-modus').removeClass("hidden-here");
    $('#toogle-modus').css({'opacity': 0.7}, 1000);
}, 800)
});



//---------------------------BUTTON----------------------------------------//
function lightOben() {
var LightOben = $('input[name="LightOben"]:checked').length;
console.log(LightOben)
socket.emit('LightOben', LightOben);
}
socket.on('LightObenFalse', function (LightOben) {
console.log("LightOben False");
$('input[name="LightOben"]').prop('checked', false);
document.getElementById('foo').jscolor.hide();
});
socket.on('LightObenTrue', function (LightOben) {
console.log("LightOben True");
$('input[name="LightOben"]').prop('checked', true);
document.getElementById('foo').jscolor.show();
});
//-----------------------------------------------------//
function lightAll(id) {
var LightAll = document.getElementById(id).checked;
console.log(LightAll)
socket.emit('LightAll', LightAll);
}
socket.on('LightAllFalse', function (LightAll) {
console.log("Light All False");
$('#LightAll').prop('checked', false);
});
socket.on('LightAllTrue', function (LightAll) {
console.log("Light All True");
$('#LightAll').prop('checked', true);
```

```
});
```

API for the Train information

Server site:

```
io.sockets.on('connection', function (socket) {
socket.on('SBB', function (connections) {
console.log("Zug- Verbindungen");
//Erste Verbinndung var Abfahrt0 = (connections.connections[0].from.departure).toString().substr(11,5)
var Von0 = (connections.connections[0].from.location.name).toString(1, 8);
var Gleis0 =(connections.connections[0].from.platform);
var Ankunft0=(connections.connections[0].to.arrival.toString().substr(11,5))
//Zweite Verbindung var Abfahrt1 = (connections.connections[1].from.departure).toString().substr(11,5)
var Von1 = (connections.connections[1].from.location.name).toString(1, 8);
var Gleis1 =(connections.connections[1].from.platform);
var Ankunft1=(connections.connections[1].to.arrival.toString().substr(11,5))
//Zweite Verbindung var Abfahrt2 = (connections.connections[2].from.departure).toString().substr(11,5)
var Von2 = (connections.connections[2].from.location.name).toString(1, 8);
var Gleis2 =(connections.connections[2].from.platform);
var Ankunft2=(connections.connections[2].to.arrival.toString().substr(11,5))
//Zweite Verbindung var Abfahrt3 = (connections.connections[3].from.departure).toString().substr(11,5)
var Von3 = (connections.connections[3].from.location.name).toString(1, 8);
var Gleis3 =(connections.connections[3].from.platform);
var Ankunft3=(connections.connections[3].to.arrival.toString().substr(11,5))
switch(Gleis0) {
case "":
    Gleis0="Tram";
    break;
default:
    Gleis0=(connections.connections[0].from.platform);
}
switch(Gleis1) {
case "":
    Gleis1="Tram";
    break;
default:
    Gleis1=(connections.connections[1].from.platform);
}
switch(Gleis2) {
case "":
    Gleis2="Tram";
    break;
default:
    Gleis2=(connections.connections[2].from.platform);
}
switch(Gleis3) {
case "":
    Gleis3="Tram";
    break;
default:
    Gleis3=(connections.connections[3].from.platform);
}
/* Erste Verbinndung */ socket.emit('SBBZEIT0', Abfahrt0);
```

```
socket.emit('SBBGLEIS0', Gleis0);
socket.emit('SBBANKUNFT0', Ankunft0);
socket.emit('SBBZEIT1', Abfahrt1);
socket.emit('SBBGLEIS1', Gleis1);
socket.emit('SBBANKUNFT1', Ankunft1);
socket.emit('SBBZEIT2', Abfahrt2);
socket.emit('SBBGLEIS2', Gleis2);
socket.emit('SBBANKUNFT2', Ankunft2);
socket.emit('SBBZEIT3', Abfahrt3);
socket.emit('SBBGLEIS3', Gleis3);
socket.emit('SBBANKUNFT3', Ankunft3);
```

Http Site:

```
<!--------------------------- Train Information ----------------------------------> <div id="transit" style="cursor:
pointer">
<main id="main-train">
    <!-- <input style="display: none !important;" placeholder="Suchen" x-webkit-speech
autocomplete="off" /> -->
    <div class="tabelle-Zug">
    <table>
    <thead>
    <tr>
    <th scope="col">Gleis</th>
    <th scope="col">Zeit</th>
    <th scope="col">Ankunft</th>
    <th scope="col">Verspätung</th>
    </tr>
    </thead>
    <tbody>
    <tr>
    <td scope="row" data-label="Account"> <span id="SbbGleis0"> Laden..</span></td>
    <td data-label="Amount"> <span id="SbbZeit0"> Laden.. </span></td>
    <td> <span id="SbbAnkunft0"> Laden..</span> </td>
    <td> <span id=""></span> </td>
    </tr>
    <tr>
    <td scope="row" data-label="Account"> <span id="SbbGleis1"> Laden..</span></td>
    <td data-label="Amount"> <span id="SbbZeit1"> Laden.. </span></td>
    <td> <span id="SbbAnkunft1"> Laden..</span> </td>
    <td> <span id="."></span> </td>
    </tr>
    <tr>
    <td scope="row" data-label="Account"> <span id="SbbGleis2"> Laden..</span></td>
    <td data-label="Amount"> <span id="SbbZeit2"> Laden.. </span></td>
    <td> <span id="SbbAnkunft2"> Laden..</span> </td>
    <td> <span id=".."></span> </td>
    </tr>
    <tr>
    <td scope="row" data-label="Account"> <span id="SbbGleis3"> Laden..</span></td>
    <td data-label="Amount"> <span id="SbbZeit3"> Laden.. </span></td>
    <td> <span id="SbbAnkunft3"> Laden..</span> </td>
    <td> <span id="..."></span> </td>
    </tr>
```

```
    </tbody>
    </table>
</div>
</main>
</div>
/* Javascript SBB */
<script type="text/javascript">
var socket = io('http://192.168.0.53:8080');
function onload() {
// ------------------------------------ SBB----------------------------------//
$.getJSON('http://transport.opendata.ch/v1/connections?from=europaplatz+bern&to=bern', function
(connections) {
    socket.emit('SBB', connections);
    console.log(connections.connections[0].from.platform);
});
//
socket.on('SBBZEIT0', function (data) {document.getElementById ("SbbZeit0").innerHTML = data;});
socket.on('SBBGLEIS0', function (data) {document.getElementById ("SbbGleis0").innerHTML = data;});
socket.on('SBBANKUNFT0',function (data) {document.getElementById ("SbbAnkunft0").innerHTML =
data;});
socket.on('SBBZEIT1', function (data) {document.getElementById ("SbbZeit1").innerHTML = data;});
socket.on('SBBGLEIS1', function (data) {document.getElementById ("SbbGleis1").innerHTML = data;});
socket.on('SBBANKUNFT1',function (data) {document.getElementById ("SbbAnkunft1").innerHTML =
data;});
socket.on('SBBZEIT2', function (data) {document.getElementById ("SbbZeit2").innerHTML = data;});
socket.on('SBBGLEIS2', function (data) {document.getElementById ("SbbGleis2").innerHTML = data;});
socket.on('SBBANKUNFT2', function (data) {document.getElementById ("SbbAnkunft2").innerHTML =
data;});
socket.on('SBBZEIT2', function (data) {document.getElementById ("SbbZeit2").innerHTML = data;});
socket.on('SBBGLEIS2', function (data) {document.getElementById ("SbbGleis2").innerHTML = data;});
socket.on('SBBANKUNFT2', function (data) {document.getElementById ("SbbAnkunft2").innerHTML =
data;});
socket.on('SBBZEIT3', function (data) {document.getElementById ("SbbZeit3").innerHTML = data;});
socket.on('SBBGLEIS3', function (data) {document.getElementById ("SbbGleis3").innerHTML = data;});
socket.on('SBBANKUNFT3', function (data) {document.getElementById ("SbbAnkunft3").innerHTML =
data;});
socket.on('SBBZEIT4', function (data) {document.getElementById ("SbbZeit4").innerHTML = data;});
socket.on('SBBGLEIS4', function (data) {document.getElementById ("SbbGleis4").innerHTML = data;});
socket.on('SBBANKUNFT4', function (data) {document.getElementById ("SbbAnkunft4").innerHTML =
data;});
}
```

Pre-Collision Assist with Pedestrian Detection

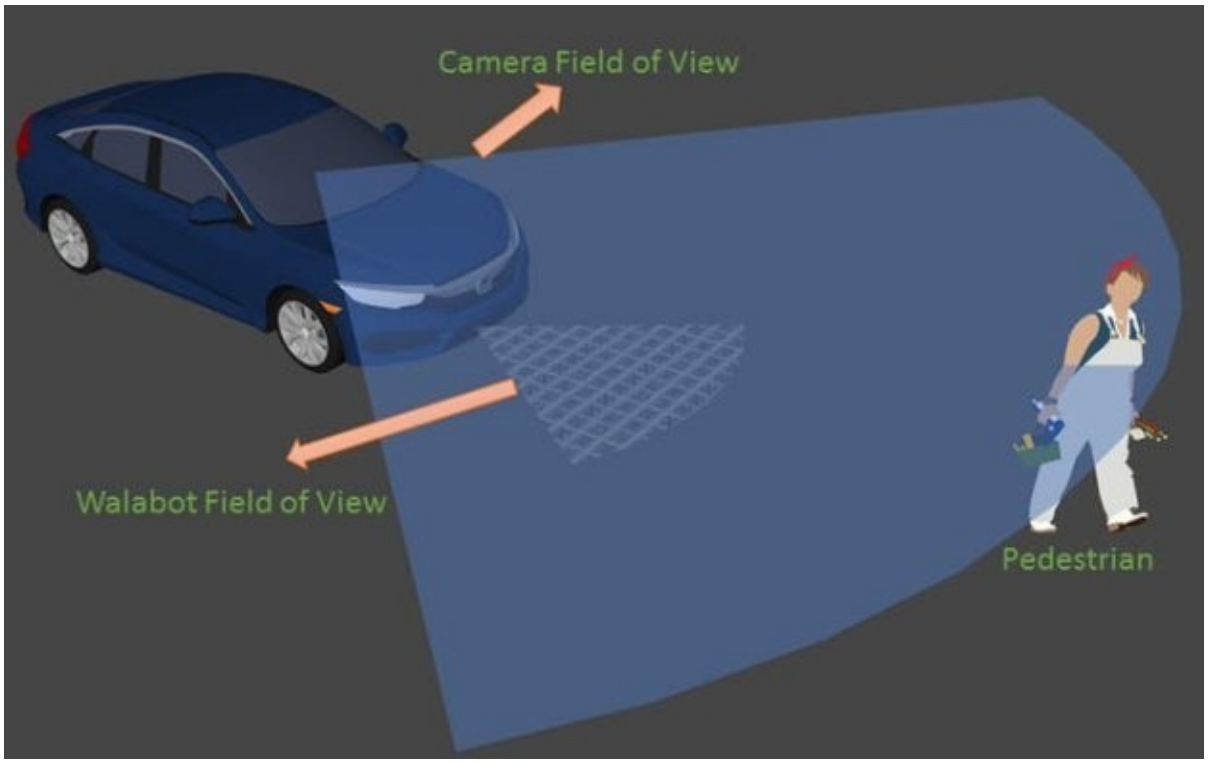
Couple months back I bought a new 10th gen Honda Civic. The trim offered in my region lacked all ADAS features. Being an engineer with Embedded Systems background, I decided to build one myself.

Most ADAS combine camera with RADAR or LIDAR. I however only had camera to start with. It turned out I can do some basic tasks like Lane detection and departure warning but not much else, till the day Walabot arrived. Cameras are great of classification and texture interpretation but they struggle with 3D mapping and motion estimation. For automotive applications, Walabot can be used as a short range RADAR

Let's see how this performs!

Architecture

Integration of Camera and Walabot's data is shown below.



Generally we combine sensory data in a way that the resulting information has less uncertainty than would be possible when these sources were used individually.

Using different sensor types also offer redundancy in situations where one type of sensors would fail. That failure or malfunction can be caused by nature or it can be deliberate e.g jamming a RADAR.

A camera generally has trouble in fog, rain, sun glare and low- light. RADAR on the other hand, lacks the resolution of cameras. but it is great for measuring distances and looking through rain and fog. RADAR and camera can complement each other.

My algorithm raises an alert, if both Camera and Walabot has detected an object of interest in the same spatial coordinates

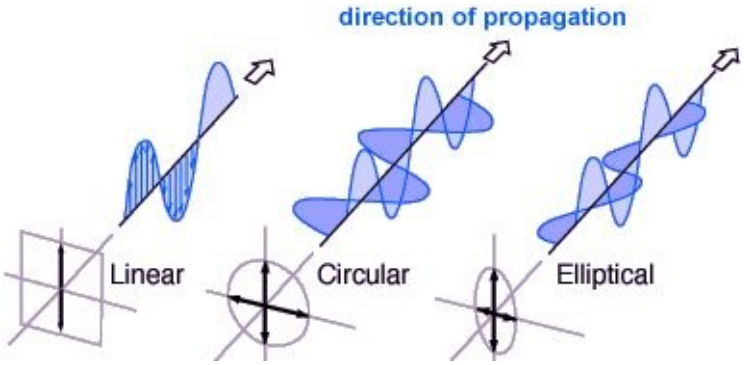
This is high level diagram of how data flows between major components.

RADAR subsystem

Our RADAR is built using Walabot connected to a Raspberry PI v2 and communicating by MQTT over WiFi. I used WiFi hotspot built into the IVI unit. You can use a standalone hotspot but they tend to perform poor in automotive environment.

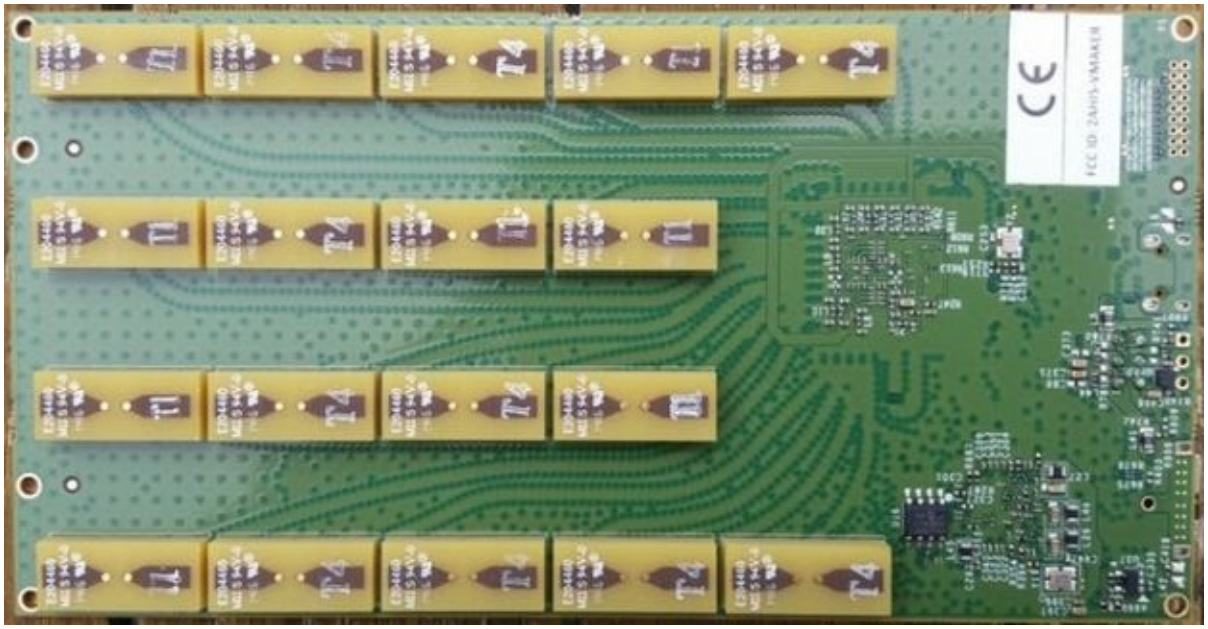
Background

Walabot gives you x-ray like vision but it does not use x-ray. Walabot has an array of linearly polarized broadband antennas that operate in frequency range: 3.3-10.3 GHz for the FCC model and 6.3-8.3 GHz in CE model.



If you are not familiar with Antenna Theory, the term polarization comes from the standing wave notation. In RF case, it means the orientation of Electric and Magnetic Field w.r.t direction of propagation. Linear polarized antennas typically have greater range due to the concentrated emission. I guess this makes sense for the Walabot case since we want to maximize the field of view.

At the heart of the Walabot there is a proprietary Vayyar VYYR2401 System-on-Chip for signal generation and reception. To connect via USB it uses Cypress FX3 controller.



Walabot can be configured in different modes for different use-case. They call it "scan profiles".

- *Short-range*: penetrative scanning inside dielectric materials such as walls.
- *Sensor*: High-resolution images, but slower capture rate. Good for distance scanning:
- *Sensor Narrow*: Lower-resolution images for a fast capture rate. Useful for tracking quick movement.

I choose to use "Sensor" scan profile.

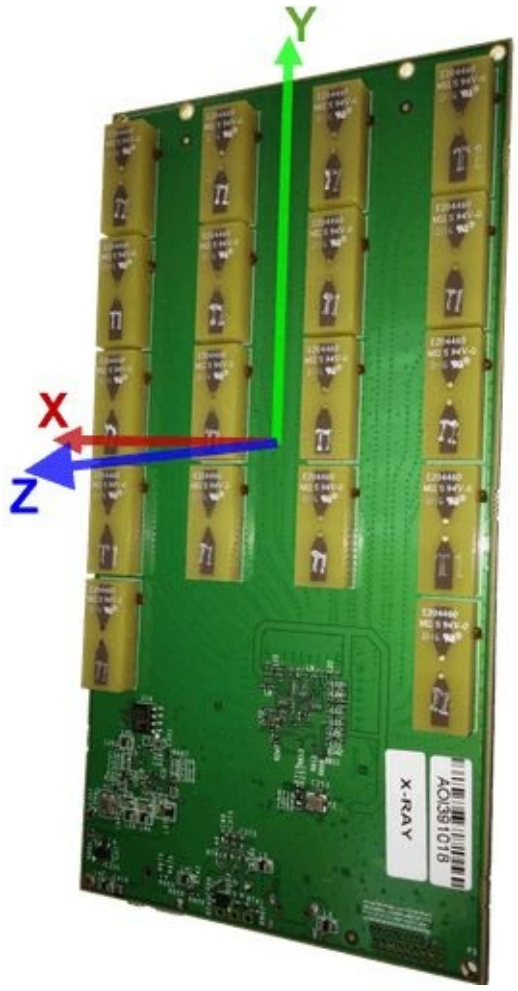
When Sensor profiles is used, Walabot processes and provides image data in a

Spherical coordinates system. Ranges and resolution along radial distance and Theta and Phi angles.

Your application can convert spherical coordinates to Cartesian ones, using the formulae:

$$X = R \cdot \sin\theta$$
$$Y = R \cdot \cos\theta \cdot \sin\phi$$
$$Z = R \cdot \cos\theta \cdot \cos\phi$$

Once you are in Cartesian (X-Y-Z) axes, then question is which side is up? The Walabot's center is the origin, and their positive directions are*:*

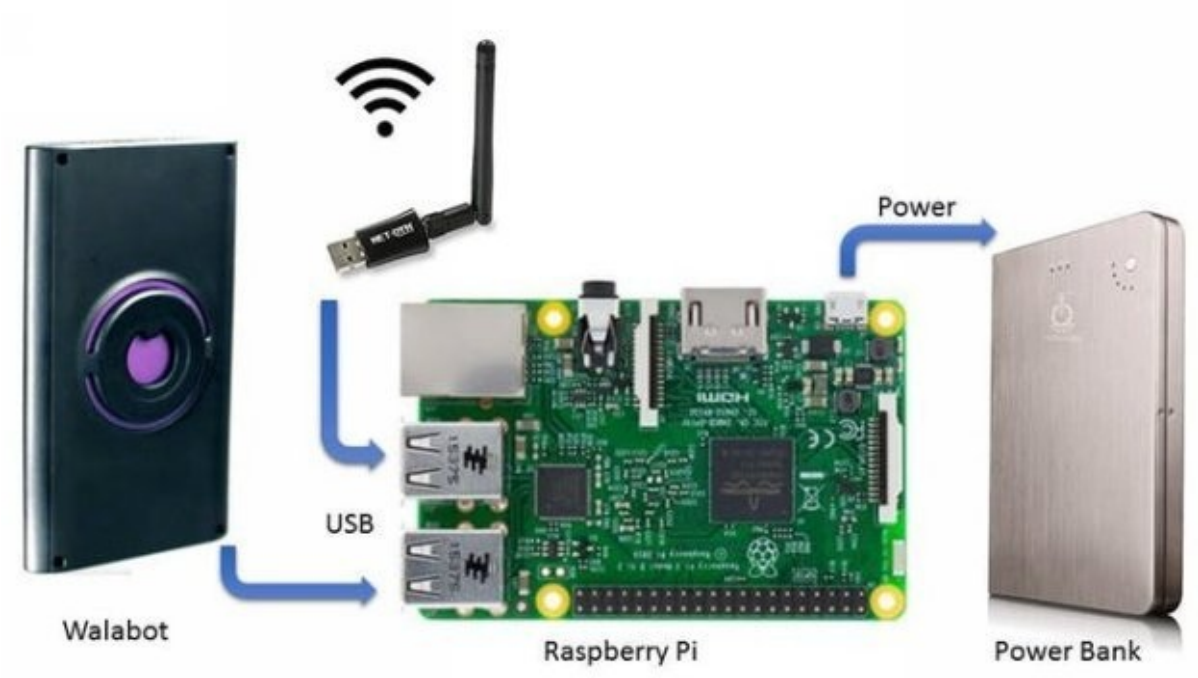


When Walabot is in its plastic case, USB connector side is the bottom.

You may ask the question why Raspberry PI v2? We are limited by Walabot SDK. The SDK is only available for x86 and Raspberry PI. I tried with Raspberry PI Zero first but the CPU was maxing out and I was loosing real-time detection capability. When I traded up to a Raspberry PI v2, things became

smoother. I still think we loose samples but it seems to be workable for my use case.

RADAR Hardware Setup



Wire up everything as shown below. Make sure to use a 2A power source!

Wrap everything in waterproof cover and install in the font of the vehicle.

RADAR Software Setup

- You need a MicroSD Card with Raspbian. Either buy one or program using these instructions.
- Boot Raspberry Pi and install updates. Instructions here.
- WiFi connection instructions are here.
- You need to modify boot arguments to increase USB current. Open a console shell to your RPI and edit like this:

$sudo nano /boot/config.txt

- Add the following

max_usb_current=1
safe_mode_gpio=4

- Reboot
- Open a console shell to your RPI and use these instructions to install Walabot SDK.

$wget https://walabot.com/WalabotInstaller/Latest/walabotSDK_RasbPi.deb
$sudo dpkg -i walabotSDK_RasbPi.deb

- Connect your Walabot via provided USB cable. Run command

$ walabot-diagnose

- It should complete without errors. Power issues frequently cause this to fail.
- Copy script Walabot_RPI /PedestrianDetect.py to any location on your RPI
- Open PedestrianDetect.py in your favourite editor and look for a line that reads "mqttc.connect". Insert address of your MQTT broker. For testing on your desk, you can use "iot.eclipse.org" but the

round trip time will be too big for realtime usecase. You will need to run MQTT broker on your LAN. I choose to do this on my Raspberry Pi Zero. Instructions further down this post.

- Run it like:

```
$python PedestrianDetect.py
```

Eventually you will need to make this autorun on boot up. You can read about all the different ways to do that from here. I choose to go about this by modifying /etc/rc.local.

Walabot input parameters

I used Sensor profile. When Sensor profiles is used, Walabot processes and provides image data in a Spherical coordinates system. Ranges and resolution along radial distance and Theta and Phi angles. This translates to the following Walabot APIs:

- SetArenaR
- SetArenaTheta
- SetArenaPhi

All of these require specifying constants. This is very important, because the fine tuned parameter can make a huge difference in the results you get. The values that worked for me are as follows:

```
minInCm, maxInCm, resInCm = 10, 100, 2
minIndegrees, maxIndegrees, resIndegrees = -20, 20, 10
minPhiInDegrees, maxPhiInDegrees, resPhiInDegrees = -45, 45, 2
```

Your RADAR setup is now done,.

Camera subsystem

Background

There are following challenges with using camera for our use-case:

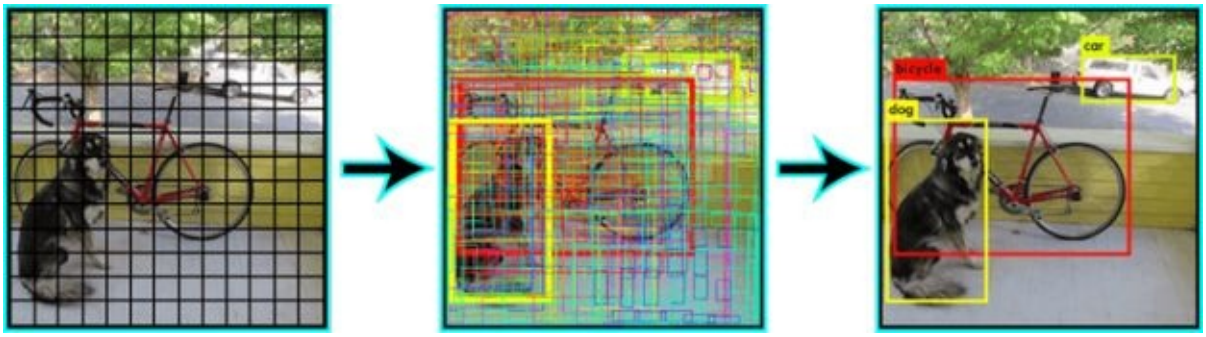
- Various style of clothing in appearance for pedestrians.
- The presence of occluding accessories on cars and persons.
- Different possible articulations
- Frequent occlusion between cars and or pedestrians

I used TensorFlow which was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research. For my application it works on a frame size of 640x480 from camera. I've noticed that it does not work very well if device is rotated, e.g from portrait to landscape.

I have used YOLO generated graphs with TensorFlow because it is able to run on a Android mobile and still able to get decent detections in realtime.

How YOLO works?

Yolo apply a single neural network to the full image. This network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.



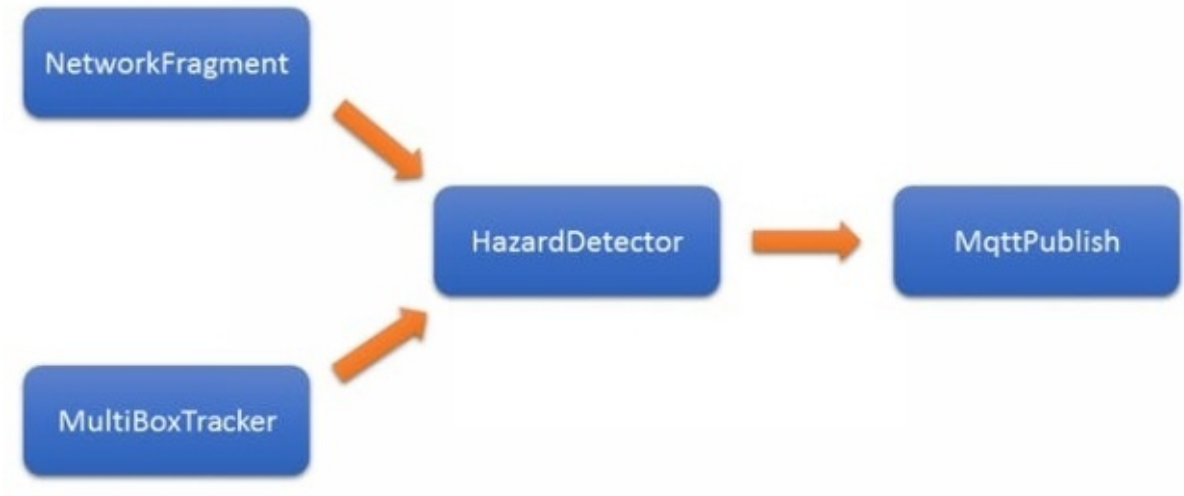
Yolo speed up comes from using a joint training algorithm that allows training object detectors on both detection and classification data. Yolo leverages labeled detection images to learn to precisely localize objects while it uses classification images to increase its vocabulary and robustness.

Block Diagram

The following steps happen on the Android side:

- Image acquisition
- Detecting an object rectangle by Yolo
- Classifying the object rectangle by Yolo
- Tracking this object by TensorFlow's Multi-box tracker
- Comparing Tracked object location with Walabot's reported location.
- Raise alert if both coincide in same spatial coordinates

Here is block diagram for the key classes involved in combining sensor data and generating alert when a hazard is detected.



Hardware setup

You will need a mid-range Android device running at least version 5.0 Lollipop. A decent camera is a plus. I tried on Samsung Galaxy A3 and Huawei P9. Both worked great. You don't need any extra hardware with that, may be except a data cable and a stand to keep the phone upright on the dashboard.

Enable WiFi on your Android device and connect to the same hotspot where your Raspberry Pis are connected.

Keep your AC vents towards the windshield. The Android phone tend to get overheat in hot sunny day.

Software Setup

- Install Android Studio on Ubuntu 16.04 LTS desktop. Instructions are here.
- Clone my projects's git repo.
- Clone DarkFlow.
- Open *DetectorActivity.java* in your favourite editor and look for a line that reads "NetworkFragment.getInstance". Insert address of your MQTT broker. The syntex is *tcp://<ip address>:1883*
- For testing on your desk, you can use "iot.eclipse.org" but the round trip time will be too big for realtime usecase. You will need to run MQTT broker on your LAN. I choose to do this on my Raspberry Pi Zero. Instructions further down this post.
- Follow the build steps mentioned on this page.
- The bazel build will download TensorFlow related data files automatically. However the YOLO graph is not included with TensorFlow and must be manually placed in the assets/ directory.
- Download *tiny-yolo-voc.cfg* and *tiny-yolo-voc.weights* from http://pjreddie.com/darknet/yolo/
- Convert Tiny Yolo via DarkFlow. The command I used:

$./flow --model cfg/tiny-yolo-voc.cfg --load bin/tiny-yolo-voc.weights --savepb --verbalise=True

- Enable developer options on your Android Phone and download and run .apk. The instructions are here.
- Setup and enable MirrorLink on you device to mirror the display to your IVI head unit. Samsung's instructions are here. Please follow your IVI unit documentation. My Honda Civic required me to connect my phone to IVI unit via USB and start E-Link application, then follow on-screen instructions.

Alert Subsystem

Once a hazard is detected, Android TensorFlow application will publish an Alert. This is done on the same MQTT server with topic "walabot/alert". This gives great flexibility on how you choose to react to the alert. You can potentially have multiple devices respond to this if they are all subscribed to the same topic.

I choose to blink red LEDs.

Hardware Setup

- Solder headers on Pimoroni pHat.

It will fit nicely on a Raspberry Pi Zero.

- Insert a USB WiFi dongle. You may need a micro-B USB to USB A female cable.
- Attach a PowerBank or your Pi Zero or plug into a USB charging ports of your car.

- You Alert subsystem is assembled.

Software Setup

- You need a MicroSD Card with Raspbian. Either buy one or program using these instructions.
- Boot Raspberry Pi and install updates. Instructions here.
- Connect to the WiFi hotspot built into the IVI unit. You can use a standalone hotspot but they tend to perform poor in automotive environment. Instructions are here.
- Take a note of the IP address assigned to this. If you are planning to use this Raspberry Pi Zero as MQTT broker, then you will need this IP.
- Open a console shell to your RPI and use these instructions to install Pimoroni SDK.

$curl https://get.pimoroni.com/scrollphat | bash

- Copy script Unicorn_RPI /MqttAlert.py to any location on your RPI
- Install MQTT broker by following command:

$sudo apt-get install mosquitto

- If you are using a different MQTT server then open MqttAlert.py in your favourite editor and look for a line that reads "client.connect ". Insert address of your MQTT broker.
- Run it like:

$sudo python MqttAlert.py

Eventually you will need to make this autorun on boot up. You can read about all the different ways to do that from here. I choose to go about this by modifying /etc/rc.local.

Alert subsystem is ready!

Disclaimer

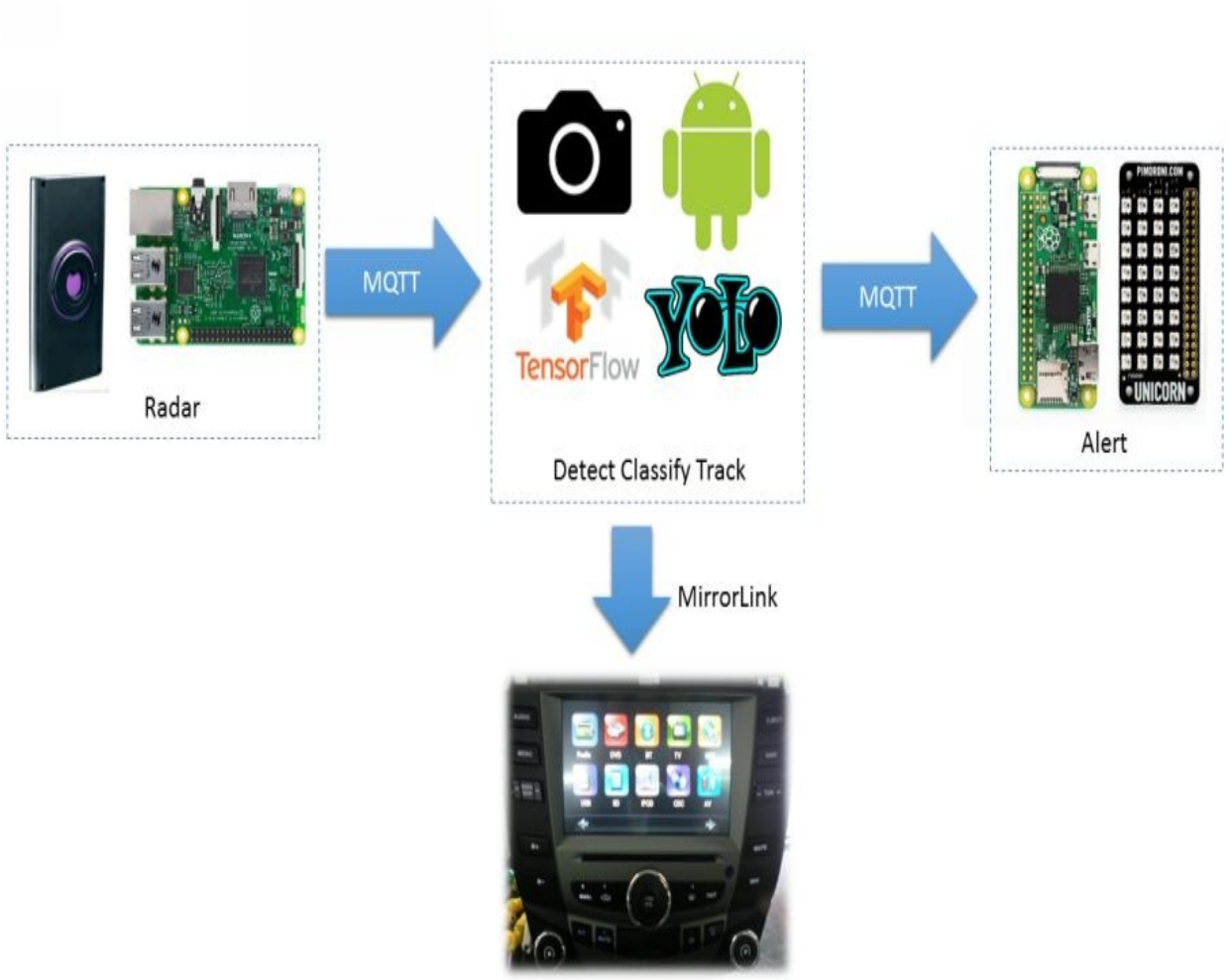
Systems like Pedestrian Detection are not a replacement for an attentive driver.

Walabot does not come in a casing that is suitable for prolonged outdoor use.

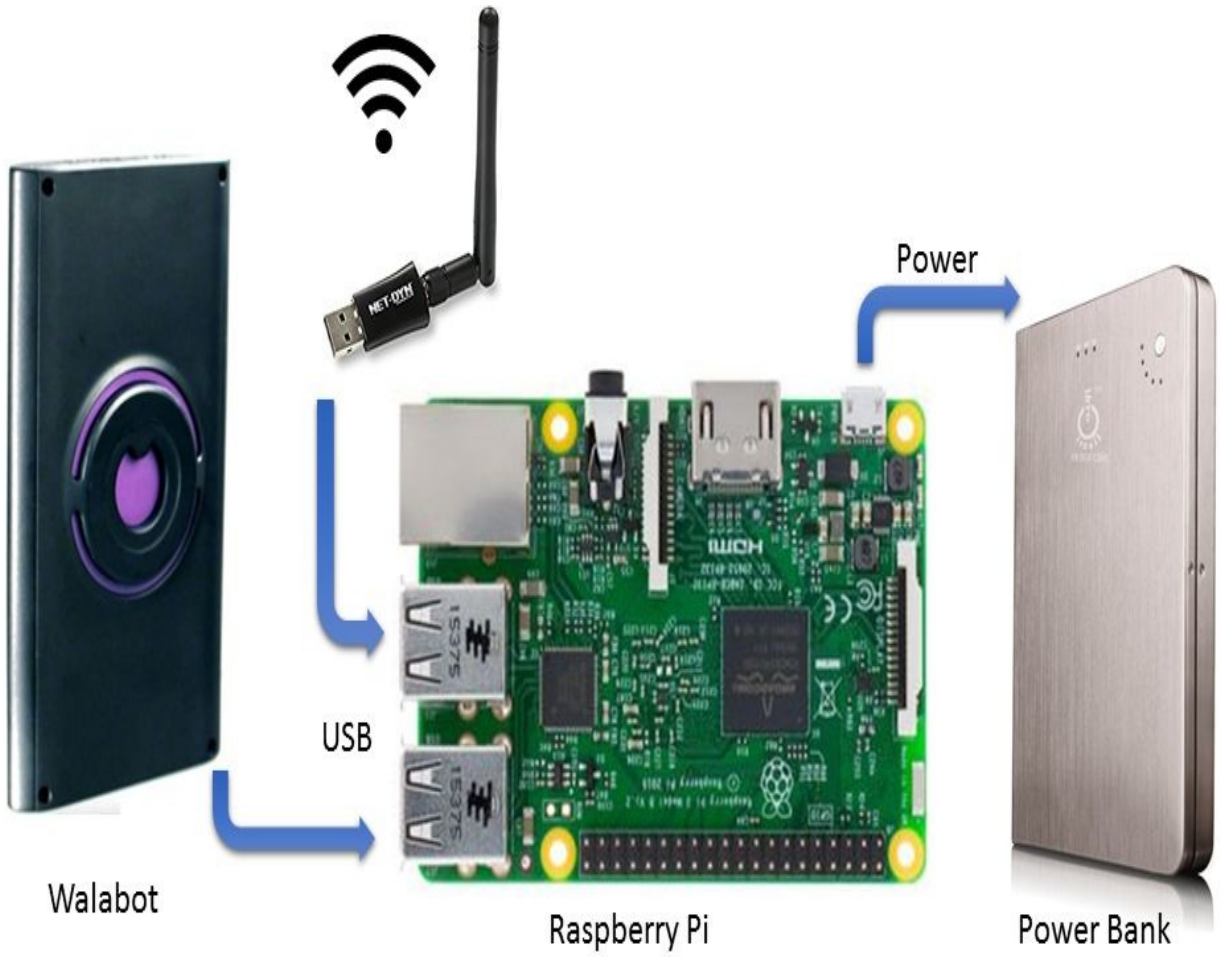
Schematics

Architecture

System components and communication



Walabot RADAR


Power
USB
Walabot
Raspberry Pi
Power Bank

Code

Walabot TensorFlow
Walabot script TensorFlow on Android., a native .so library and a Java JAR Pimoroni Unicorn pHAT scripts MQTT support

Download Sourcode

Trigger Google Assistant on Pi Using Ultrasonic HC-SR04

This project is an extension of the previous project (Custom Wake Word for Google Assistant on Raspberry Pi). The previous project showcased the triggering of Google Assistant on Raspberry Pi using a pushbutton and a custom wake word. This project shows how to trigger google assistant SDK on Pi using an ultrasonic sensor, HC-SR04.

Modified Rasbian Image

I had modified a Raspbian image for the custom wake word to work efficiently which can be found here.

If you are using that image, download the following zip file. It contains all the codes and scripts that are required.

- Install the audio drivers.
- Get the google assistant up and running (Should hardly take 10 mins if using Pi3 or about 45 mins is using Pi Zero as most of the installation has been already done. Only Authentication is required).
- Wire the ultrasonic sensor to Pi as shown in the schematic.
- Start the script to start google assistant and then start the ultrasonic trigger script.

Already using Google Assistant

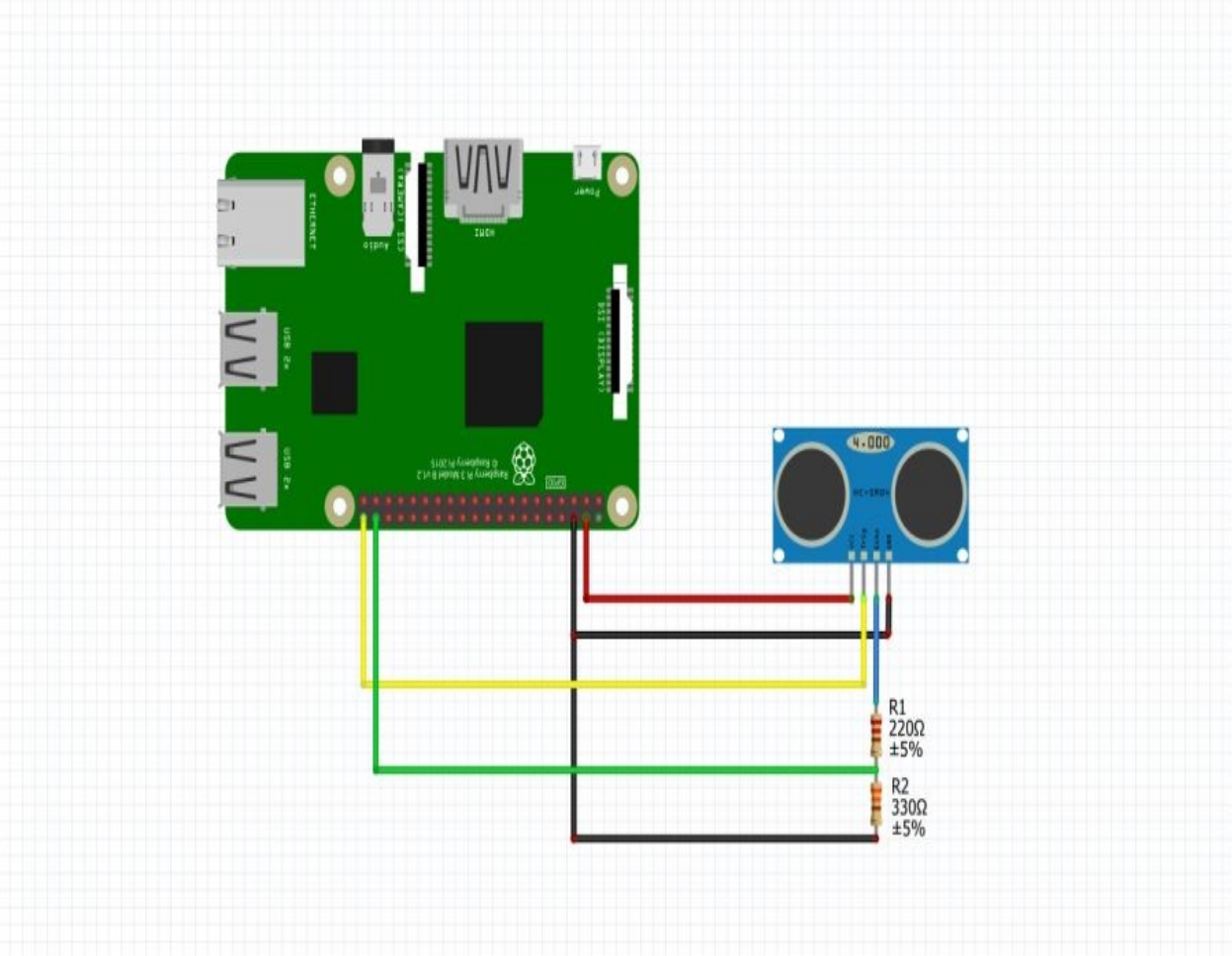
If your Pi is already running a Google Assistant, then download the following zip file https://goo.gl/HWxo6M.

- Extract the contents to any directory.
- Wire the ultrasonic sensor to Pi as shown in the schematic.
- Move into /home/pi/env/lib/python3.4/site-packages/googlesamples/assistant/grpc folder and replace the existing pushtotalk.py file with the pushtotalk.py given in the zip.
- Copy the trigger folder into any directory and update it's location in the HCSR04-Trigger.sh script.
- Start the script to start google assistant and then start the ultrasonic trigger script.
- Make the gassist-start.sh and HCSR04-Trigger.sh executable and use it to start the Google Assistant and ultrasonic trigger respectively.

Schematics

Wiring diagram

Wire the HC-SR04 to Pi as shown



Code

HCSR04.pyPython
Script for using HC-SR04 as trigger

```
#Libraries
import RPi.GPIO as GPIO
import time

#GPIO Mode (BOARD / BCM)
GPIO.setmode(GPIO.BCM)

#set GPIO Pins
GPIO_TRIGGER = 21
GPIO_ECHO = 20
GPIO_GASSIST =17
#set GPIO direction (IN / OUT)
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)
GPIO.setup(GPIO_ECHO, GPIO.IN)
GPIO.setup(GPIO_GASSIST, GPIO.OUT)

def distance():
    # set Trigger to HIGH
    GPIO.output(GPIO_TRIGGER, True)

    # set Trigger after 0.01ms to LOW
    time.sleep(0.00001)
    GPIO.output(GPIO_TRIGGER, False)

    StartTime = time.time()
    StopTime = time.time()

    # save StartTime
    while GPIO.input(GPIO_ECHO) == 0:
    StartTime = time.time()

    # save time of arrival
    while GPIO.input(GPIO_ECHO) == 1:
    StopTime = time.time()

    # time difference between start and arrival
    TimeElapsed = StopTime - StartTime
    # multiply with the sonic speed (34300 cm/s)
    # and divide by 2, because there and back
    distance = (TimeElapsed * 34300) / 2

    return distance
```

```
if __name__ == '__main__':
    try:
    while True:
    dist = distance()
    if distance()<10:
    print ("Trigger Detected at %.1f cm" % dist)
    GPIO.output(GPIO_GASSIST, False)
    time.sleep(0.3)
    GPIO.output(GPIO_GASSIST, True)
    else:
    print ("Waiting For Trigger")
    time.sleep(1)

    # Reset by pressing CTRL + C
    except KeyboardInterrupt:
    print("Measurement stopped by User")
```

Earthquake Pi - Shake and Rattle Your Desk

There are a number of earthquake detector projects for the raspberry pi. These are good for anyone living in an area prone to earthquakes or for those that want to try to detect distant quakes themselves.

This project is not focused on detecting quakes, but uses USGS earthquake data to make an interesting little alerting system. Sure, you can always go to your PC and browse the USGS maps for the latest data, but you are a Maker and you like to make things because you can!

What does it do?

- · It collects the USGS earthquake data from their website for the past 15 minutes.
- · It displays location, magnitude and other data on a small LCD screen
- · It rattles its box and flashes its lights to the magnitude of the quake (Optional)
- · It plays earthquake sounds (Optional)

You set the minimum magnitude level you wish it to alarm on. But note, if you set to 1.0, it will probably go off every few hours! At 3.0, it will usually sound once every day or two.

Options include adding mechanical rattles, earthquake audio sounds and RGB LED bargraph lights. These are not required, but really make the earthquakePi “alive”.

You can use other GPIO pins for controlling other devices. Even interface it with cloud Internet of Things (IOT) services such as IFTTT (http://ifttt.com/) to send messages or blink your lights!

NOTE: With all these options active, this becomes a fairly elaborate wiring & software job and requires you to jump to other setup tutorials as noted in URL links. If you are not familiar with Linux command line and/or wiring GPIO pins, you may wish to try simpler projects first, or skip one or more of the optional features in this project.

Step 1: Hardware Parts

- Raspberry Pi running Raspian and python 2. Any Pi will do. Even a $5 Pi Zero
- Ethernet connector or WiFi dongle for Raspberry.
- A I2C compatible LCD display 20 char x 4 lineSuch as Amazon 2004 I2C LCD or similar
- Breadboard, Pi Breakout cable/connector to breadboard, wire
- A wooden box (such as from craft stores like Michaels)
- 5v power supply for Raspi (2 amp if using all options)

Optional Accessories:

Vibration Motor:

- · A vibrating motor such as from an old battery toothbrush (see below), from a pager, or buy one similar to https://www.adafruit.com/products/1201
- · 220 ohm resistor
- · 1N4001 or similar diode
- · 2N2222 transistor

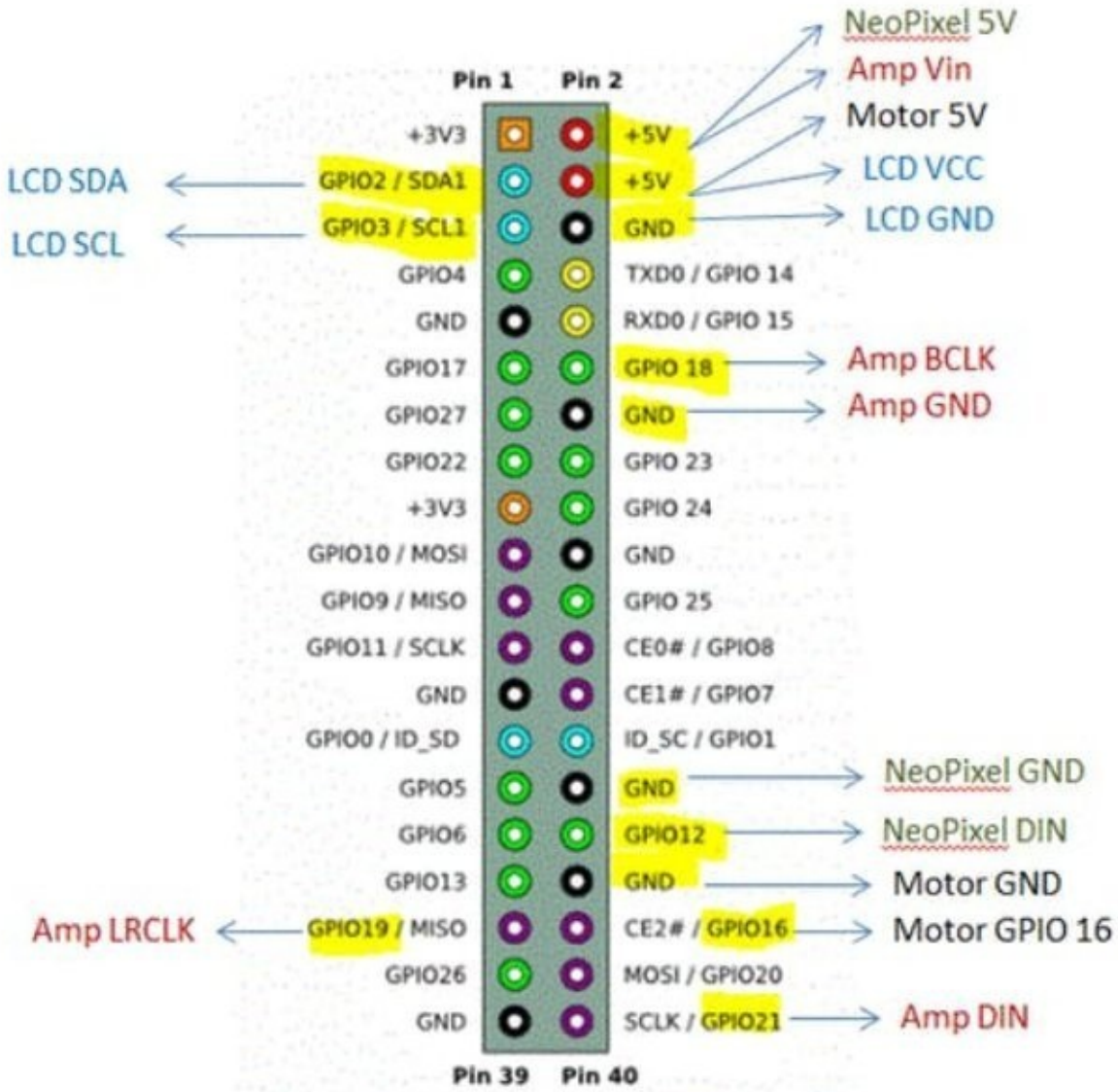
NeoPixel Bargraph:

- · Neopixels LED strip 8x5050 RGB LED NeoPixel Stick from Adafruit: https://www.adafruit.com/products/1426
- · 1N4001 or similar diode

Audio Sound Effects:

- · External speaker (and amp) for Raspi audio.
- · For Raspi Zero, add Adafruit I2S 3W Class D Amplifier Adafruit: https://www.adafruit.com/products/3006
- Small speaker https://www.adafruit.com/products/1314

Step 2: Hardware Installation

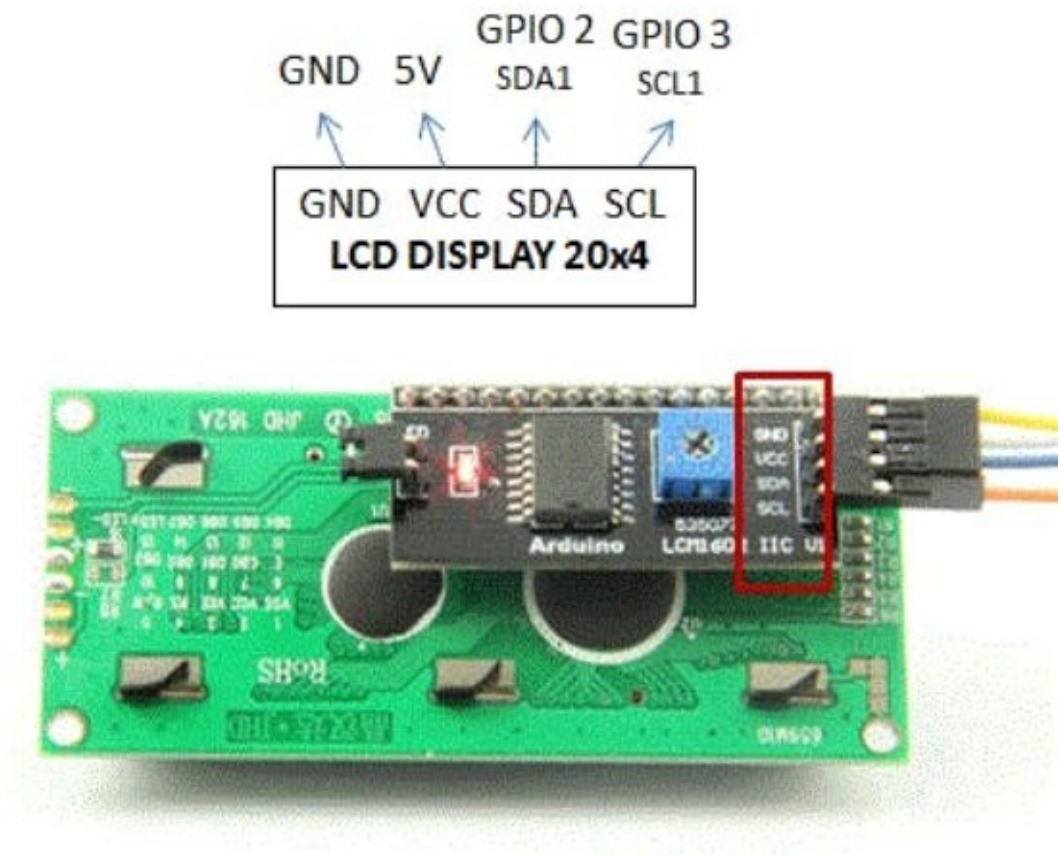


***NOTE**: You should initially set up your Raspberry Pi with Raspian. I used Jessie version, but others should work. This project assumes you can already SSH to the Raspi and connect to the Internet before adding the features below. No HDTV monitor or keyboard is needed (headless operation).*

Internet

As stated above, you need Internet connectivity, either via Ethernet or WiFi. Nothing special is needed for this project (no firewall rule changes, etc.) so just follow existing instructions elsewhere for setting up your Pi.

LCD 20x4 Display



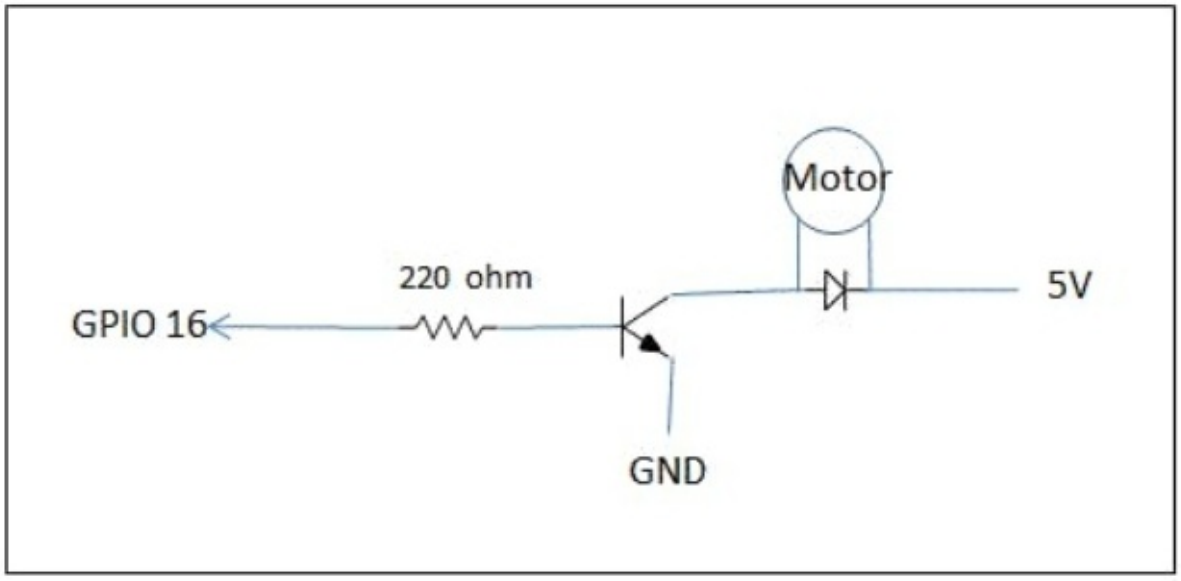
The LCD is the only "required" feature of the Earthquake Pi. But it alone would be rather boring...

For the LCD 20x4 Display, wire the Pi up as shown in photo 2. You need a 20 char x 4 line I2C interface LCD. Not 16x2 or non-I2C types.

- LCD GND -> Raspi GND
- LCD VCC -> Raspi 5V
- LCD SDA -> GPIO 2
- LCD SCL -> GPIO 3

Photo 1 shows pinouts with all options.

Step 3: Optional Accessories - Mechanical



Vibrating Motor

For`the optional vibrating motor, I used an old toothbrush such as the Oral B battery powered brush. Open it up and you will find a tiny “vibrator” motor. Alternately, pick up a vibrating motor from numerous sources.

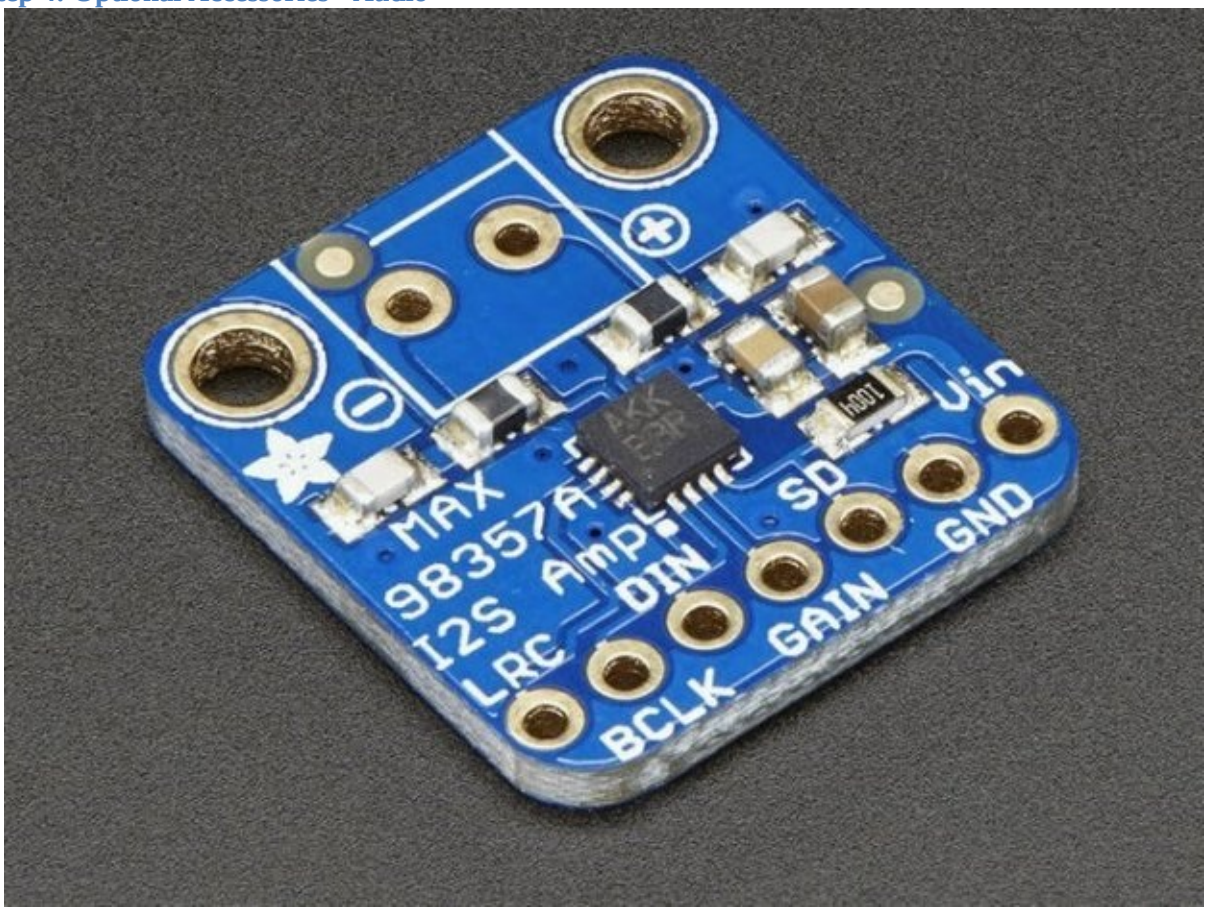
Note that you may need to adjust the settings in the python earthquake.py program to get the “best” vibration desired. The idea is to have it crescendo

quickly and then fade over several seconds, with the volume and length determined by the magnitude of the quake.

You can use either the 5v or 3.3v supply pins on the Pi to run the motor. Most will run ok either way. But you will need to add a simple driver as most motors take too much current to run directly off a GPIO pin, thus the 2N2222 transistor. The 1N4001 diode is needed to keep motor induction spikes off the Raspi.

Connect it to **GPIO pin 16** (default, but can be changed in the python program) as well as a **Ground pin** and a **5V pin** as shown.

Step 4: Optional Accessories - Audio



Audio Earthquake Sound Effect

For audio, plug an amplifier into the audio jack. The default on the Pi is set to send audio automatically to the analog output, but if you have an HDMI monitor plugged in, the sound shifts to the HDMI connector. Since this project doesn't need a monitor, unplug it and reboot. Alternately, use:

```
$ sudo raspi-config
```

Select **Advanced Options -> Audio** and select **Analog out**.

On a Pi Zero, you will need to add external audio, since it's not built-in. I used

the low cost “Adafruit I2S 3W Amplifier breakout MAX98357A”

This requires setup using Adafruit’s excellent tutorial at:

https://learn.adafruit.com/adafruit-max98357-i2s-c...

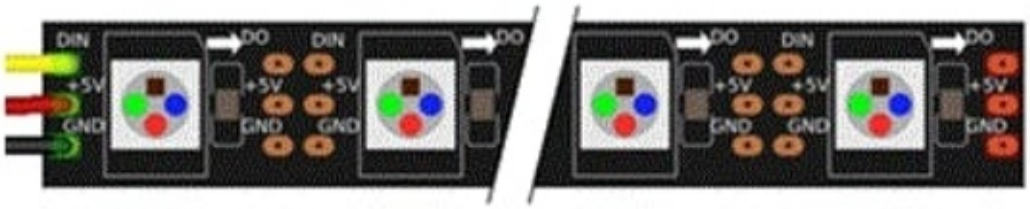
As shown in the tutorial, connect:

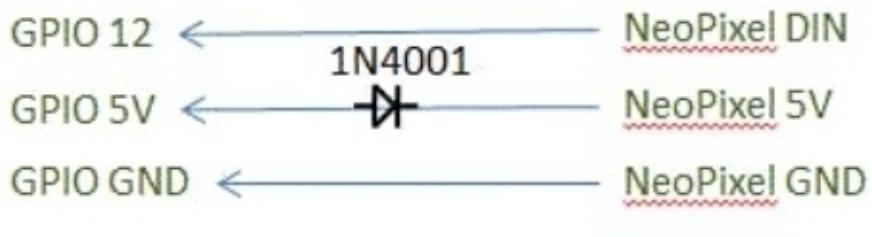
- Amp Vin to Raspi Zero Pi 5V
- Amp GND to Raspi Zero Pi GND
- Amp DIN to Raspi Zero Pi #21
- Amp BCLK to Raspi Zero Pi #18
- Amp LRCLK to Raspi Zero Pi #19

Be sure to edit the files shown in the tutorial and test the audio.

Once you complete the setup tutorial including the audio tests, then sound should be ready.

Step 5: Optional Accessories - LED Bargraph





NeoPixel LED Bargraph

NOTE: *The earthquakepi.py program is designed to only run with the 8x5050 RGB LED NeoPixel Stick from Adafruit.*

Wire the Adafruit NeoPixel as described in the NeoPixels on Raspberry tutorial

https://learn.adafruit.com/neopixels-on-raspberry-...

I was able to use the simpler “**diode wiring**” method, described in the tutorial, since I used only an 8 RGB LED strip.

For the NEOPIXELS LEDs option, you must install the rpi_ws281x Library as detailed in:

https://learn.adafruit.com/neopixels-on-raspberry-...

Important Note!: The default **GPIO Pin 18** in the Neopixels example is **DIFFERENT** from the one used by earthquake.py!

Edit **~/rpi_ws281x/python/examples/strandtest.py** and change **LED_PIN = 18 to LED_PIN = 12**

Wire the NeoPixel DIN to **GPIO 12**.

This is because the GPIO 18 is needed for Raspi Zero sound. BUT, make this change even if you don't use a Raspi Zero. That will make it compatible without modifying earthquake.py.

Be sure you can successfully run the "**strandtest.py**" example in the Neopixels tutorial before continuing with EarthquakePi.

Step 6: Completing the Hardware

POWER REQUIREMENTS:

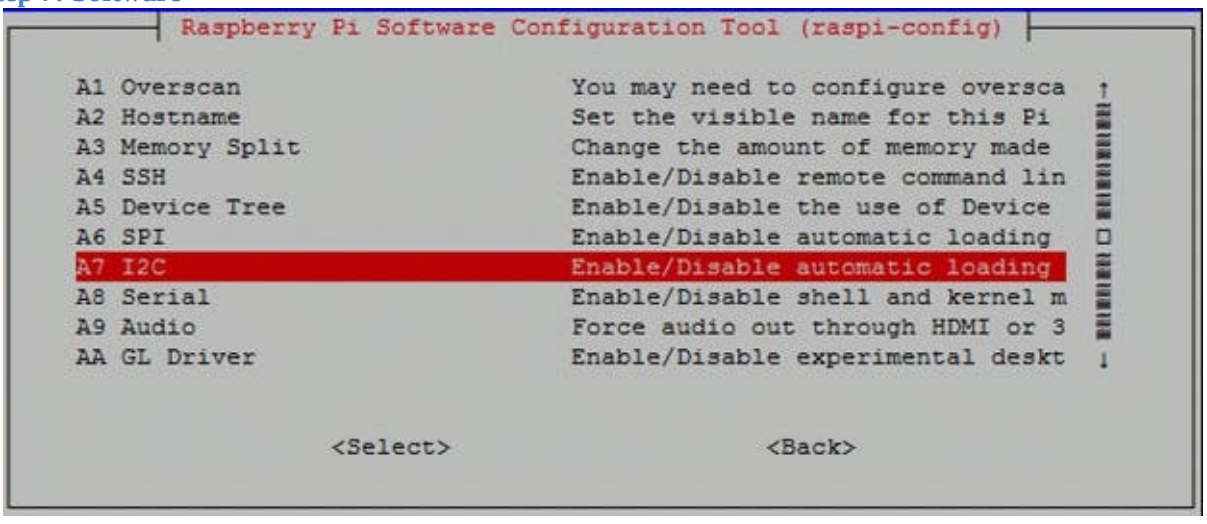
All the options combined can use close to the limit of available current from the Raspberry so if you use different hardware, be mindful to keep total current under 500ma from the Raspi 5V lines.

Wooden Box

Install the system into a box. I used a small 4x7x3 wooden box from Michael's Arts & Crafts. Cut holes for the leads. Hot melt glue the motor down. I used electrical tape with a small piece of wood to angle the LCD display and hold it down on the top of the box. I used double stick foam tape for the RGB LED strip beside the LCD display.

Now for the software...

Step 7: Software



Software Installation

You need to be running Raspian Linux with Python 2.X and be able to ssh to your Raspi. The instructions assume you already have your Pi set up with Raspian, python and connected to the Internet.

Next, download the earthquakepi software from GITHUB: https://github.com/rgrokett/earthquakepi

Install packages:

```
$ cd /home/pi$ sudo apt-get update
$ sudo apt-get install build-essential git
$ sudo apt-get install python-dev python-smbus python-pip
$ sudo pip install RPi.GPIO
$ sudo apt-get install i2c-tools
```

Configure the SPI and I2C bus devices by using:

```
$ sudo raspi-config
```

Select **Advanced Options -> I2C enable**.

Select YES to enable and YES to load by default for both of these.

Select FINISH and Reboot your Pi, if asked, or execute:

```
$ sudo reboot
```

Verify the LCD is detected using:

```
$ sudo i2cdetect -y 1
```

You should see a screen similar to photo 2. If the LCD display is detected, you will see an entry such as **0x20** for Adafruit LCD or **0X27** for Keyestudio LCD.Remember this hex number.

Next install and test the earthquake program:

1. $ cd /home/pi

2. $ git clone https://github.com/rgrokett/earthquakepi.git

Test your LCD Display first. This works with 16x2 or 20x4 LCD displays, but the Earthquake program expects 20x4 only, as it needs the extra lines of text.

1. $ cd ~/earthquakepi

2. $ nano RPi_I2C_driver.py

Edit the LCD address to match what you found above.

```
# LCD AddressADDRESS = 0x27
```

3. $ python lcd.py

The test program will turn on the LCD backlight and then display a series of text and graphics.

If you have a problem, check your wiring, including adjusting the tiny potentiometer on the back of the LCD display for brightness.

Step 8: Optional Audio Software

If you use audio, set the maximum volume on your speaker using:

$ sudo amixer sset PCM,0 95%$ aplay /home/pi/earthquakepi/earthquake.wav

NOTE: Pi Zero & Adafruit MAX DAC doesn't use amixer volume controls and can only handle stereo WAV files (earthquake.wav has been converted to stereo for compatibility.)

Adjust your speaker/amplifier to make this LOUD as this is equivalent to a magnitude 9.0 earthquake! The program will range from barely audible 1.0 quakes upwards to your maximum volume you set. (Again, Pi Zero audio is fixed volume, so set to any level you like!)

Step 9: Test it!

Once you get the expected results for the LCD and audio, next is to try out the earthquake program.

Edit the **earthquakepi.py** file and change any optional features you have installed:

DEBUG = 1

MINMAG = 1.0

AUDIO = 0 or 1

MOTOR = 0 or 1

NEOPIXEL = 0 or 1

Then type:

$ sudo python earthquake.py

An initial DEBUG mode test of the display along with optional LED bargraph, motor and audio should occur. If there is an error or missing software package, its information should print out on your SSH terminal.

Step 10: Operating Earthquake Pi

The program variable MINMAG defaults to magnitude 1.0 or greater which occurs many times per day. You can edit the earthquake.py program to change this higher.

```
############ USER VARIABLESDEBUG = 1 # Debug 0 off, 1 on
LOG = 1 # Log Earthquake data for past 15 min
MINMAG = 1.0 # Minimum magnitude to alert on
AUDIO = 1 # Sound 0 off, 1 on
MOTOR = 1 # Vibrate Motor 0 off, 1 on
MOTORPIN = 16 # GPIO Pin for PWM motor control
NEOPIXEL = 1 # 1 use Neopixel, 0 don't use Neopixel
NEO_BRIGHTNESS = 64 # Set to 0 for darkest and 255 for brightest
## OTHER SETTINGS
```

```
PAUSE = 60 # Display each Earthquake for X seconds
WAV = "/home/pi/earthquakepi/earthquake.wav" # Path to Sound file
DISPLAY = 0 # 0 Turn off LCD at exit, 1 Leave LCD on after exit
########### END OF USER VARIABLES
```

Once tuned, turn off DEBUG mode by editing the program:

DEBUG = **0** debug off

MINMAG = **2.0** or higher quake every few hours.

LOG = 1 prints USGS earthquake data (if any) for the past 15 minutes.

The log will be written to */home/pi/earthquakepi/earth.log* by the cron, below.

Load CRON entries to run the program every 15 minutes between 8:00am and 10:45pm local time daily. This way it will only run during waking hours (adjust as desired!).

```
$ cd ~/earthquakepi
$ crontab pi.cron
```

The @reboot cron just displays a message for a few seconds anytime Raspi is rebooted.

@reboot sudo python /home/pi/earthquakepi/startup.py >/dev/null 2>&1

*0,15,30,45 08-22 * * * sudo python /home/pi/earthquakepi/earthquakepi.py >/home/pi/earthquakepi/earth.log 2>&1*

Be sure your Raspi is set to LOCAL time for the cron to work as expected:

```
$ sudo raspi-config
```

Select **Internationalization Options -> Change Timezone**

NOTE: the LCD time display for earthquakes is always in UTC only, unaffected by local Timezone.

Finally, set up your EarthquakePi on your desk and reboot it. An initial display showing the IP address for the Pi will be shown for a few seconds.

Every 15 minutes you should see the LCD display (but no sounds) even if there are no earthquakes. The display will be blank otherwise.

When a quake occurs above your preset minimum magnitude, the box will rattle, the LED will flash, the LCD display will show the details and the earthquake sound effect will occur. (Assuming you added all of the various options!)

It will probably scare your friends, family, your cat and, if you go to sleep before 11pm and it goes off, it will scare YOU!

Step 11: Troubleshooting

Obviously, if all the options are active, there are potentials for wiring errors or missing software packages. Use the **DEBUG = 1** option and manually run

earthquake.py to display any error messages.

```
$ sudo python earthquake.py
```

In normal operation, if you have LOG = 1 (default), you can look in the log file for the last information.

/home/pi/earthquakepi/earth.log

Typically, issues would be missing software packages (**$ sudo apt-get install {package}**) or a wiring problem, particularly if you had to substitute different components from the above. You may need to cut/paste the error message into Google search to find assistance!

If you have intermittent crashes/reboots/hangs, most likely an insufficient power supply. You MUST have a high quality 5 volt power supply. You CANNOT run this from a PC USB port. A separate supply is needed with at least 1.5 amp or greater. I added **220uf 12v and 22uf 12v capacitors** across the incoming power just to help filter out any noise generated by the motor and LEDs.

NOTE! All the options combined use close to the limit of available current from the Raspberry so if you use different hardware, be mindful to keep total current under 500ma from the Raspi 5V lines.

SHAKE & RATTLE!

Schematics

Download as zip

Download Code Snipet 2

Download Code Snipet 9

GitHub for EarthquakePi
Download Sourcecode

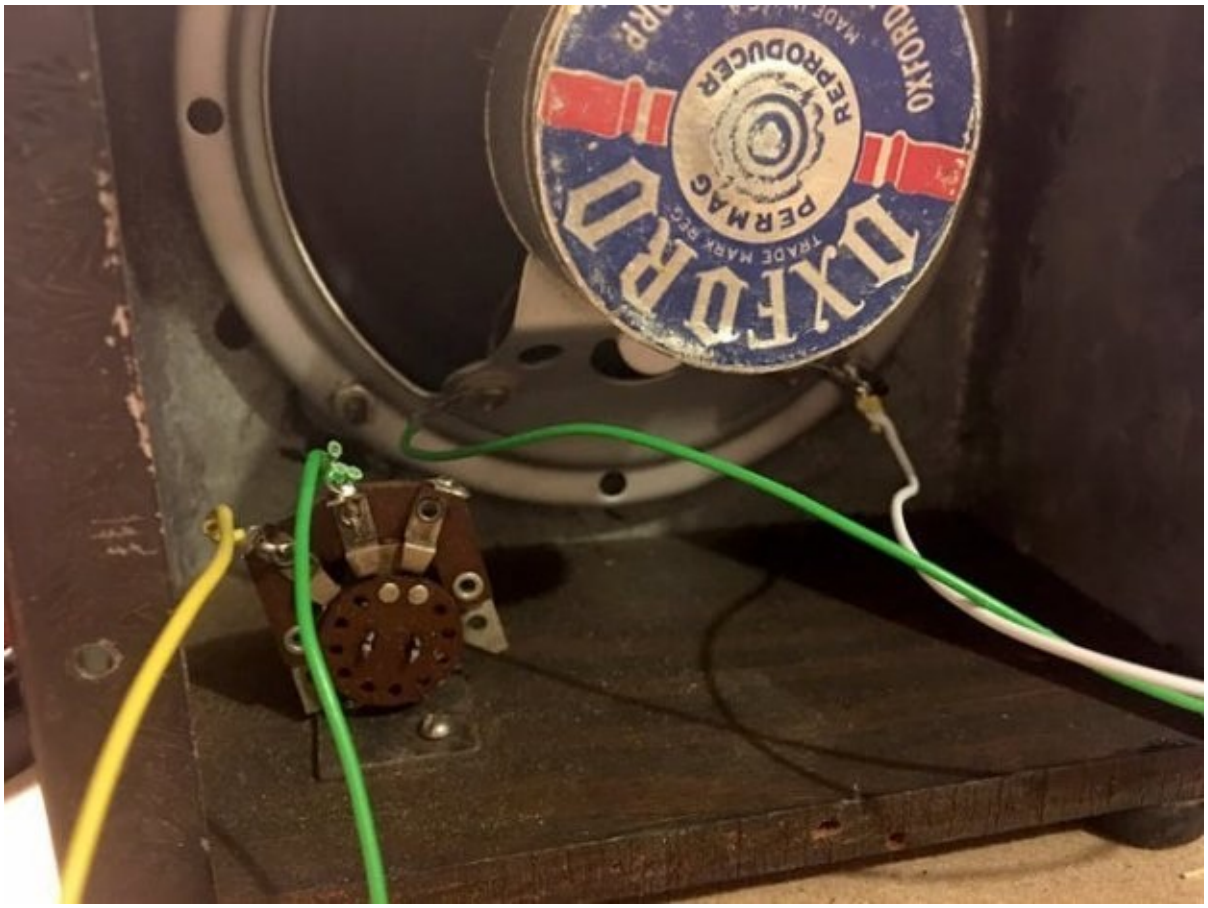
Vintage Intercom Echo

I picked up an old intercom from an antique mall in Austin for about $20. After about a day of thinking of what to do, I landed on installing a Raspberry Pi Zero with Amazon's Alexa voice service installed on it.

What I had

After opening it up, I found that the electronics were incredibly simple. The knob on the front was connected to a 2 position rotary switch. Lift the knob up and it would switch the speaker to a microphone. This is the sort of intercom that would sit on someone's desk.

I decided to wire the speaker to a 1/8 audio jack and the switch directly to the Raspberry Pi Zero GPIO pins.

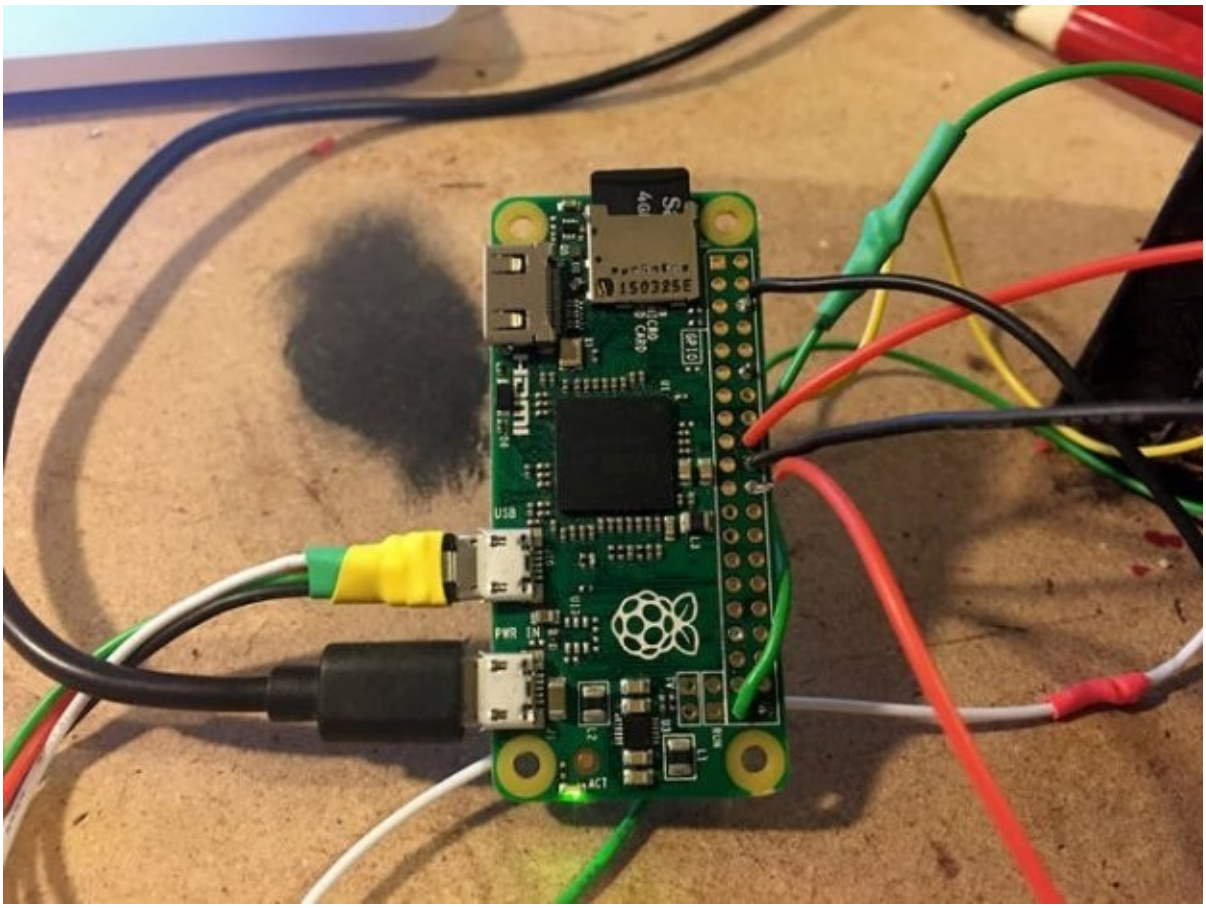


Indicator LEDs

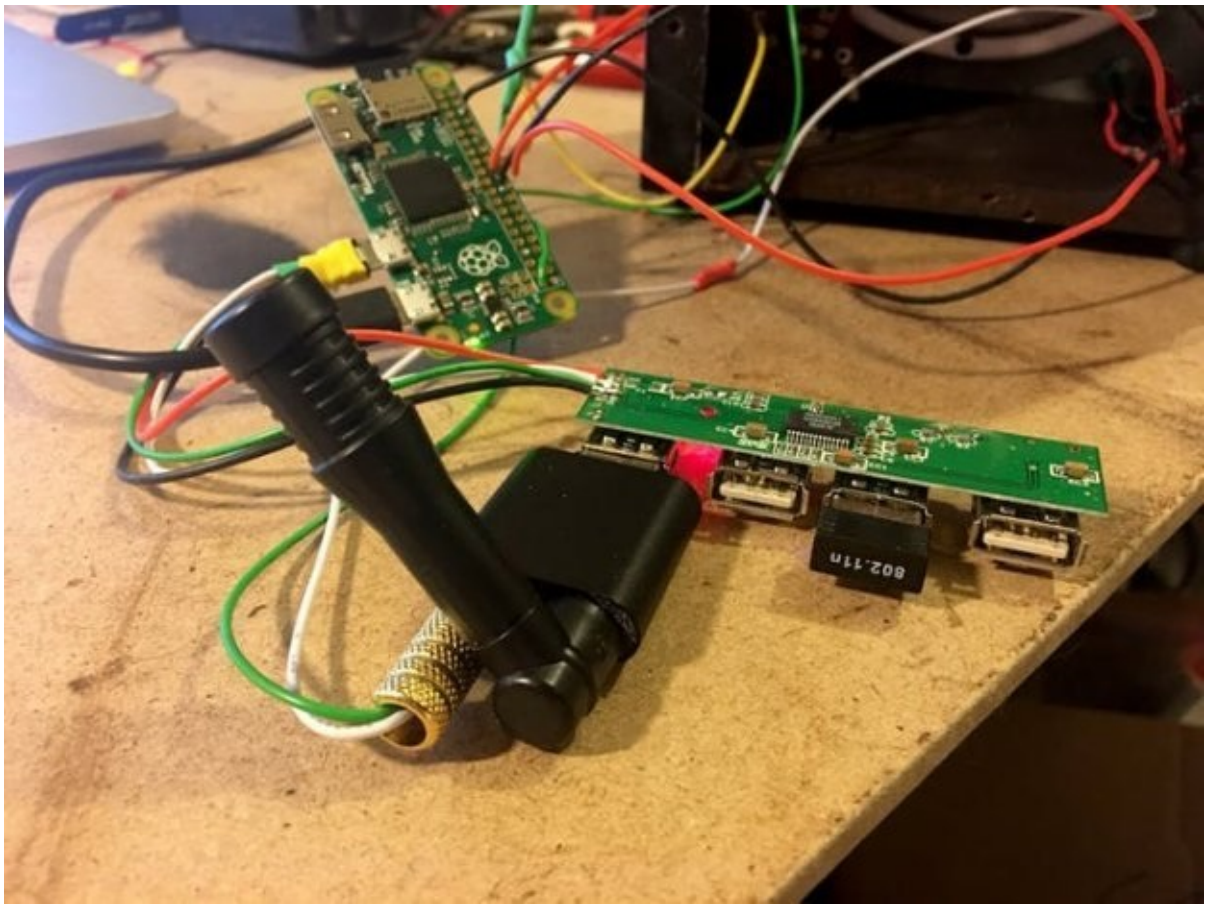
There are two indicator LEDS (green for Alexa listening, red for Alexa responding). I wanted these to be as low profile as possible to keep the entire thing looking like it did when I bought it.

Wiring

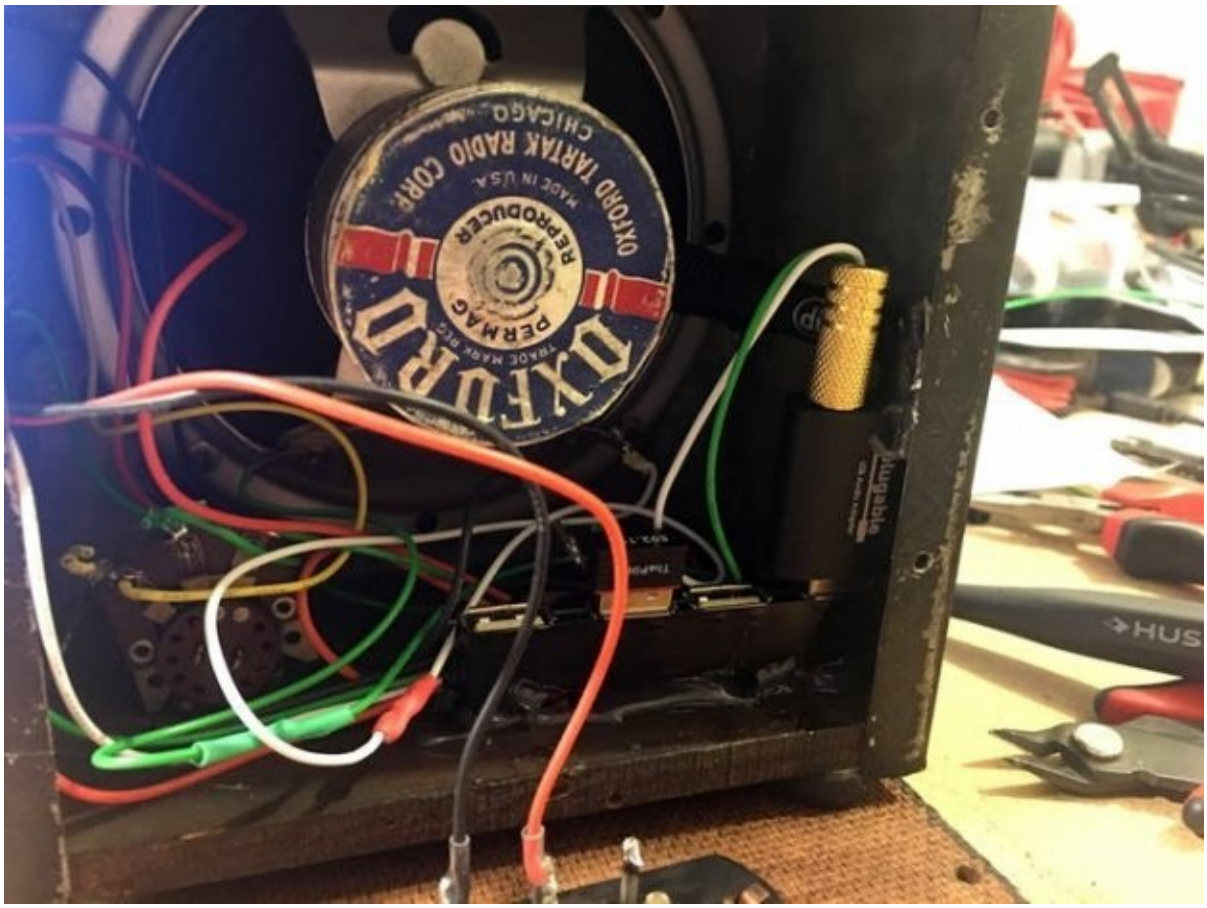
Raspberry Pi Zero wired up. The wiring is pretty simple, LEDs and switch into the GPIO. I modified the USB hub to a USB micro to keep the size to a minimum inside the case.



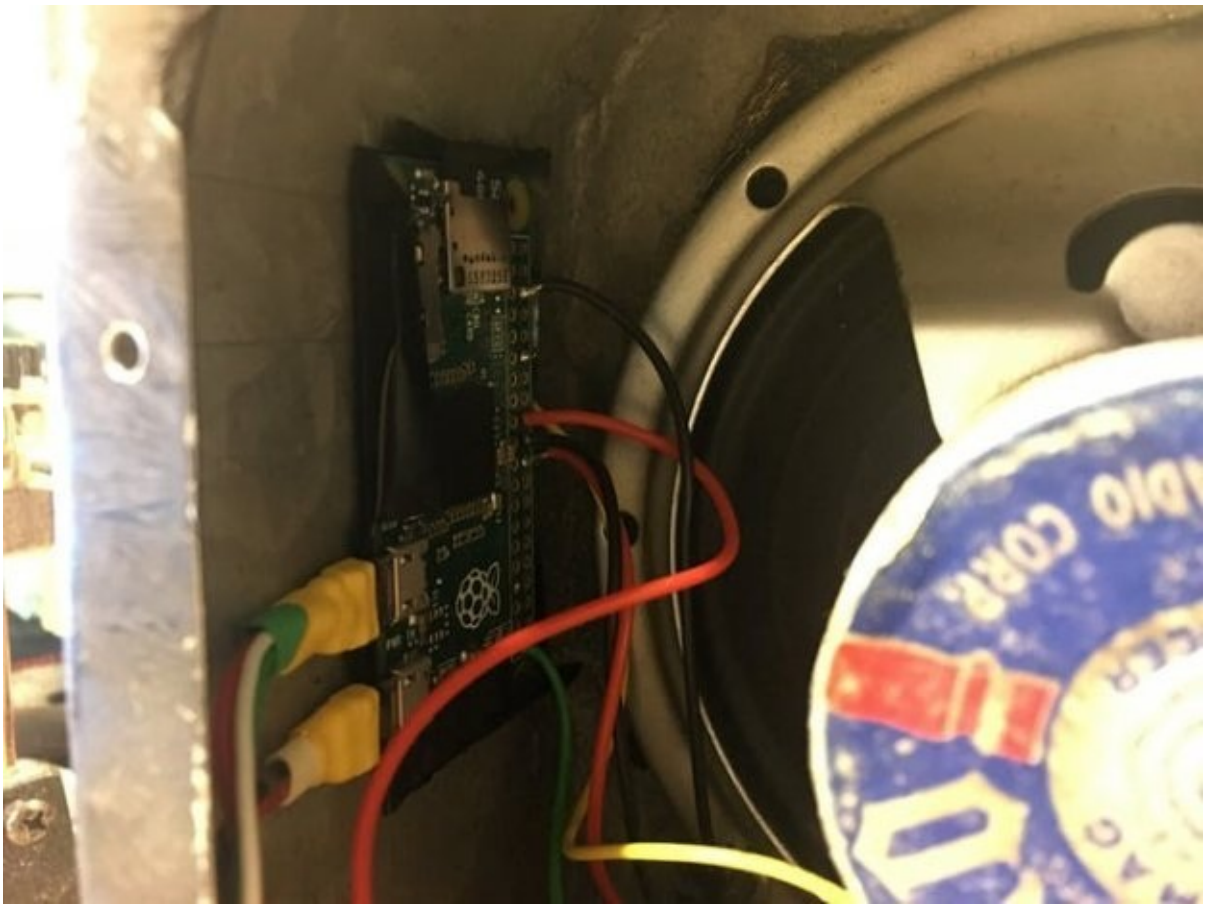
The USB Hub has both a WiFi card and a USB audio card attached (along with a small microphone).



Hot glue is your friend. The USB hub is mounted in the back of the unit and the microphone sits directly behind the speaker. Press the switch and speak towards the speaker and the microphone right behind it will start listening.



The RPi Zero is mounted on the side of the case.



Conclusions and showing the love

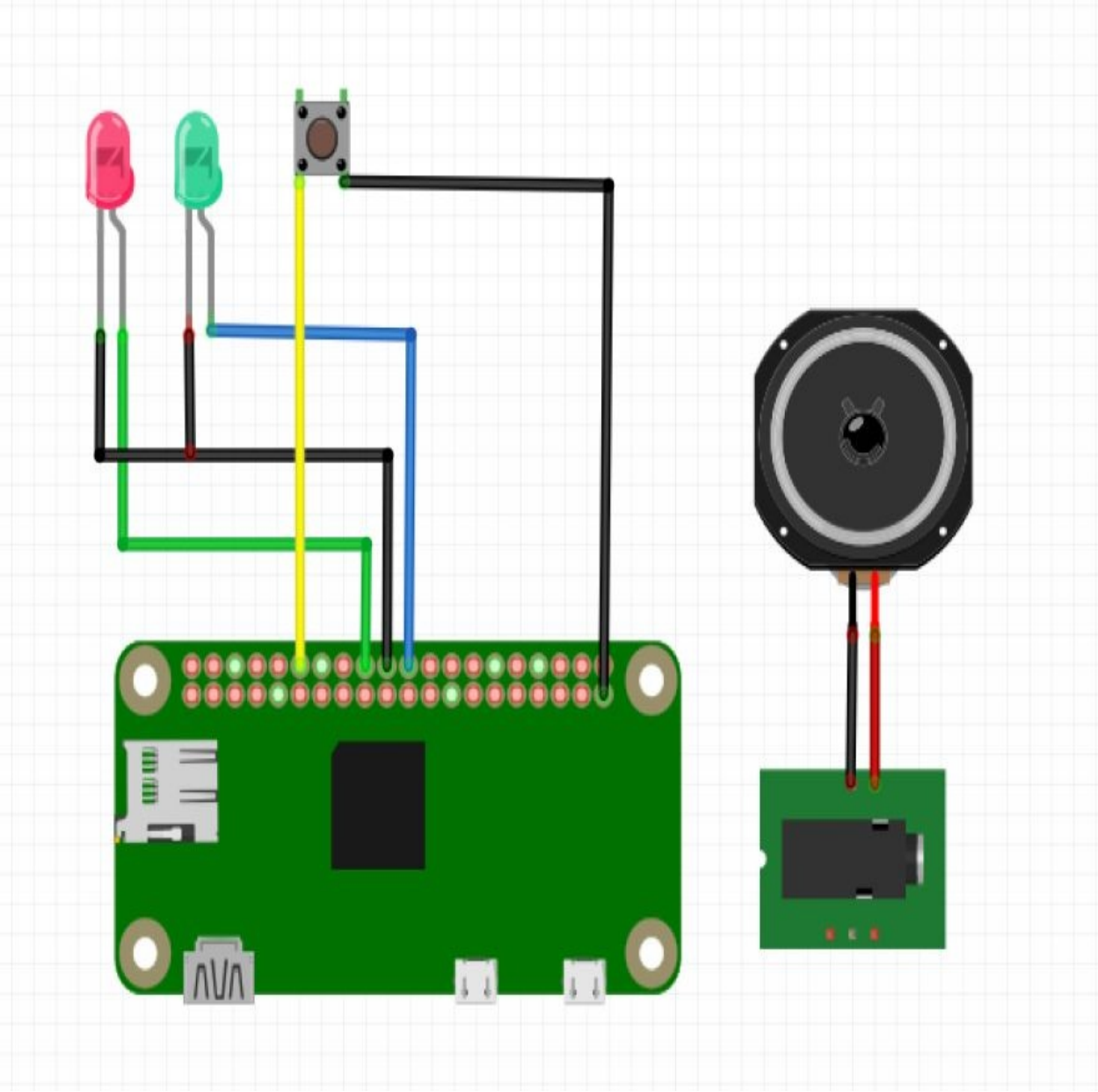
Overall I'm extremely happy with how this build came together. It only took about a day to put it together and most of that time was spent messing around with the programming.

It even got some love from the Raspberry Pi Foundation.

Schematics

Wiring Diagram

Here is the basic wiring diagram. The speaker it wired to a 3.5mm jack and plugged into a USB Sound Card.



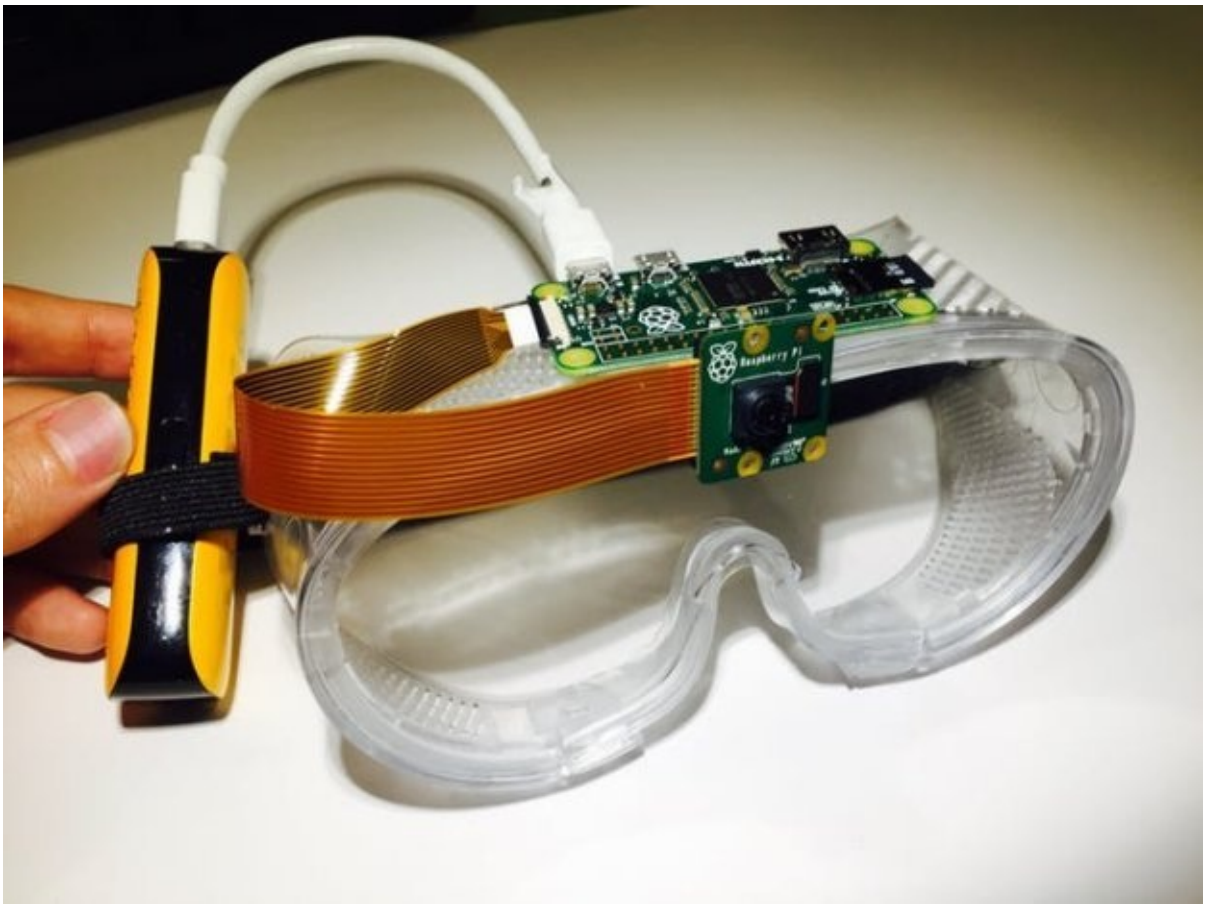
Source Code

[NavaSpirit - Alexa Pi]

Raspberry Pi + Alexa Voice Service

FabDoc - Version Control Tool for Makers

We are all facing a lot of work of documentation beyond making, especially when we would like to share and collaboratively develop "hardware" projects.



There are too many version control system (VCS) tools for "software" projects so that we could fortunately join any large-scale software projects, such as Linux, if they are open source. But what happen to "hardware" projects? Can we enlarge the scale of these projects with VCS tools?

There are too many physical details (materials, machines, parameters...) which should be well-documented before cooperating, but unfortunately cannot be easily digitized into documentation or VCS.

We are trying to hack the way of documenting for all of makers' projects, simply with Raspberry Pi, camera module, and hacked Google.

Details

We want to build a hand-free and easy-to-use documentation tool and platform.

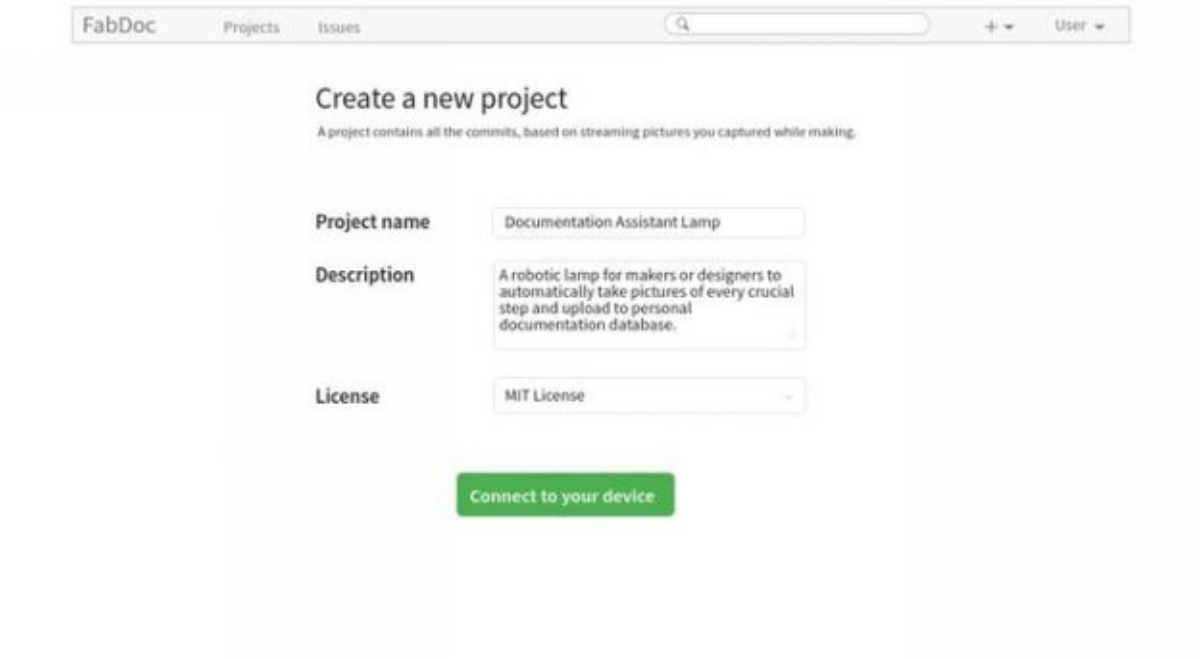
Though currently the way we make our documentation is step-by-step, the real

building and design process will suffer many dead ends. On the other hand, it seems more like a tree with branches if we want to record every detail while making. However, the trade-off is that making means our hands are busy. We would be distracted if the work has to be suspended to take pictures.

What if the system can remain the branches and each dead end you've tried? This is not only for sharing (documented) projects, but also helping you document for your team or yourself.

So, here is how we hack it, the main concept is: "To capture time-lapse pictures as pre-commits".

Before Making



- Create a new project with project name and description

- Generate QR code token
- Login by scanning QR code token with Raspberry Pi Zero
- Start to stream time-lapse pictures to your computer through WebSocket

While Making

- Leave your computer alone
- Happy making

After Making

- Select key frames of time-lapse pictures as "**commit**" in the browser
- Tag physical materials in the frames and add more details
- Upload to the platform

Ongoing

There are two main features we are working on:

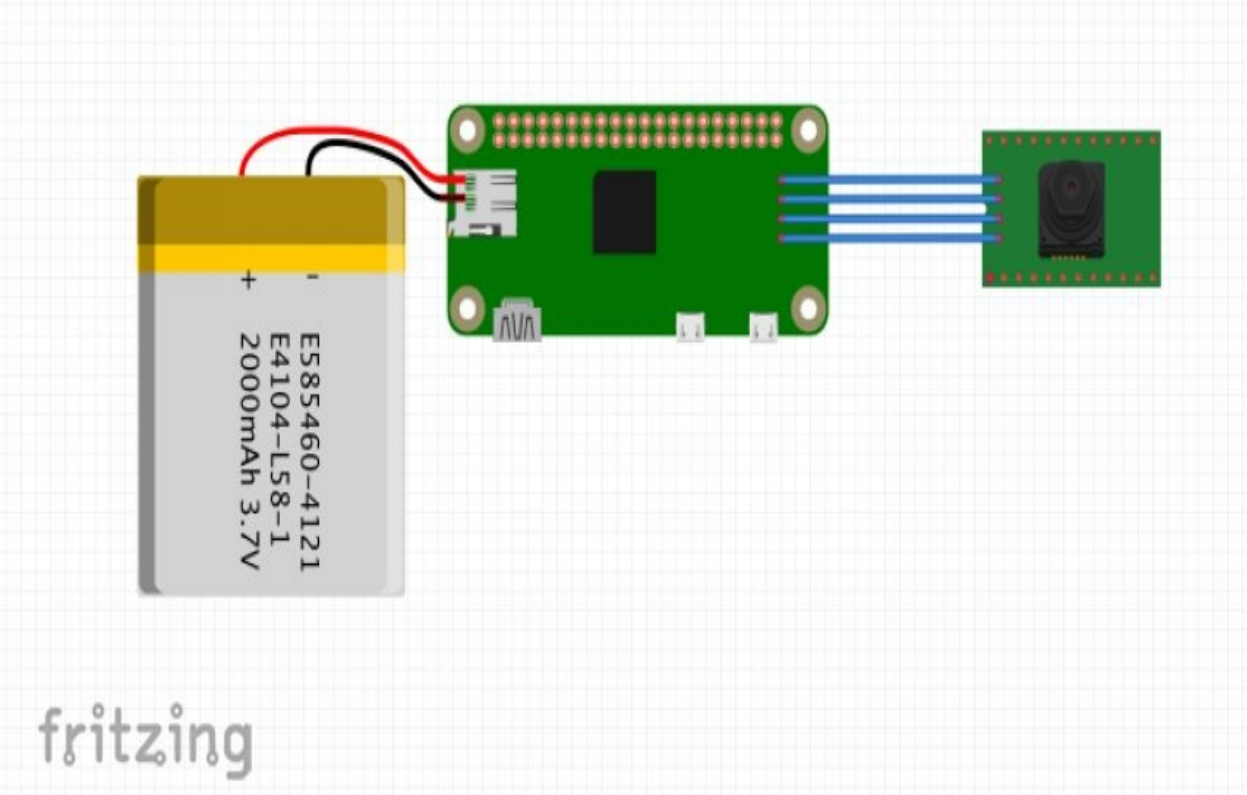
- Device - python-client https://github.com/FablabTaipei/FabDoc-python-client
- Server - console https://github.com/FablabTaipei/FabDoc-console

Future Features

- Add IMU as input on Raspberry Pi, in order to capture more than time-lapse pictures, such as customized time video, by head motion. FabDoc-python-client FabDoc-console

Schematics

FabDoc-v1



Download SourceCode

Connecting Google Home to Raspberry Pi Projects

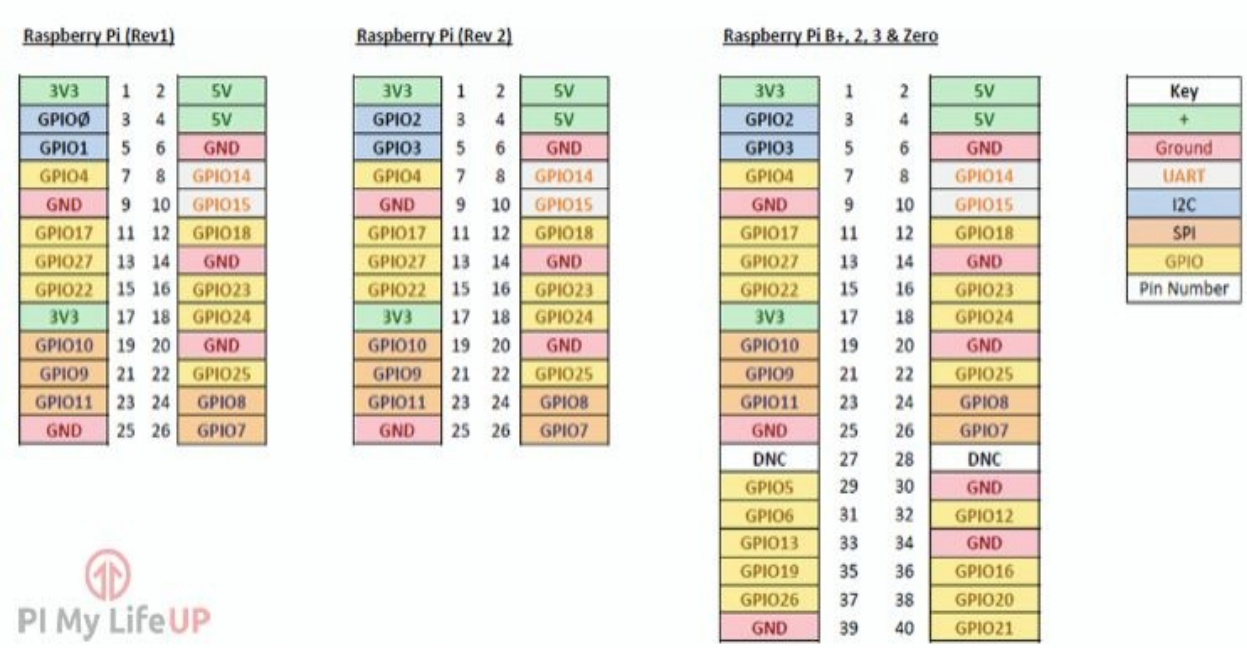
This all started back in the summer, while I was attending a programming bootcamp. When I managed to pick up a Raspberry Pi Zero, I wanted to take on an ambitious project. I gutted a power strip and replaced the switches with relays, and then created a React.js view for its frontend.

However, the vision for my project wasn't complete until I got my Google Home. Plenty of people have gotten their projects working with the Amazon Echo, but I wanted to try working with Google's assistant. Until now, I've only seen one story of someone controlling their fireplace, so I thought I'd show you my own approach.

Kyle Peacock is a web developer currently looking for work in the San Francisco Bay Area. He specializes in full-stack JavaScript development and loves playing with new technology.

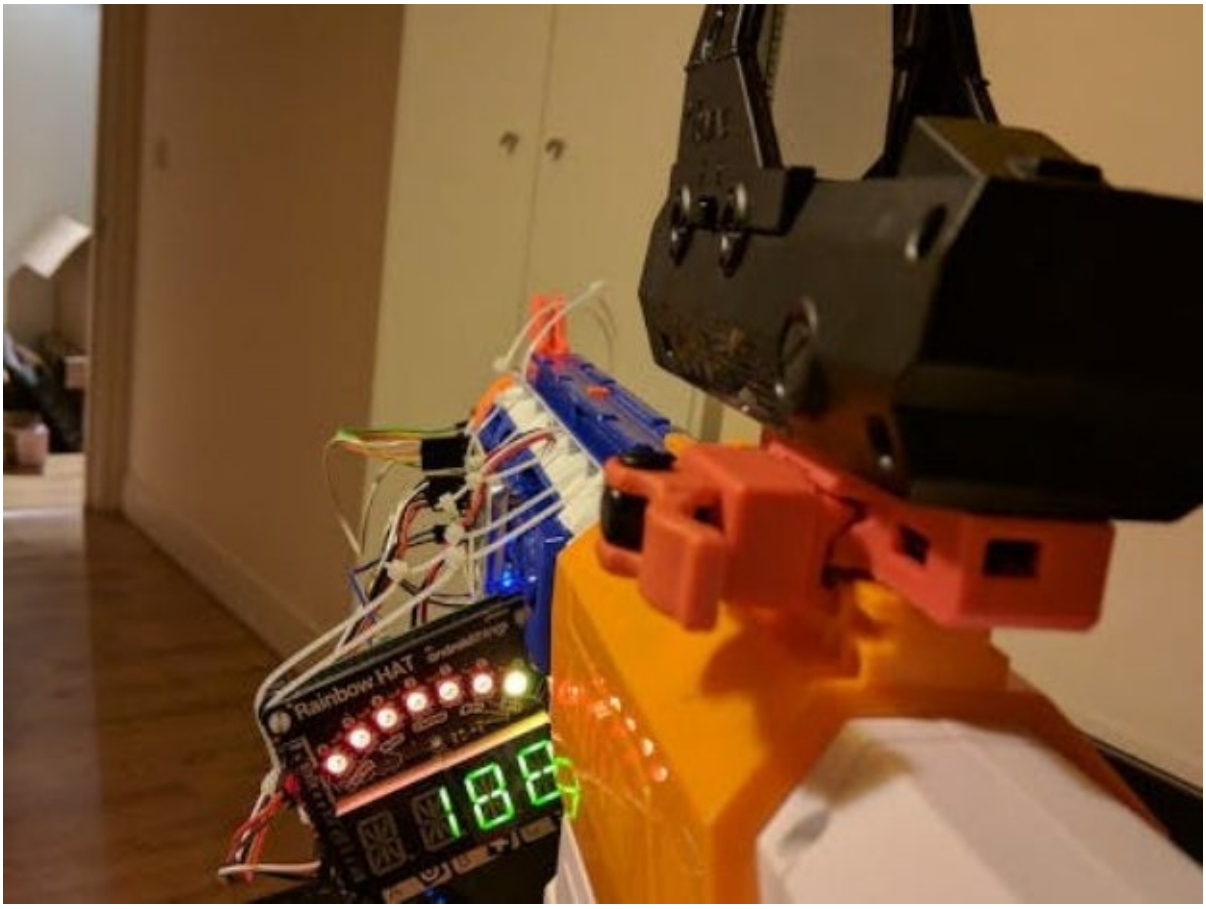
Schematics

GPIO reference
How to choose your GPIO pins for the relay



Download Sourcecode

Nerf Gun Ammo Counter / Range Finder



Because ammo counters are cool and range finders are also pretty cool and Nerf is cool – mix them together and its ice cold.

I had the idea for this a while back but have only now just decided to build, with the release of the awesome Rainbow Hat from Pimoroni.

This would give me the display needed for the range in CM and a clear display for the ammo counter itself.

Key Goals.

- Make a device that can measure distance from the barrel of a Nerf gun.
- That can also count down remaining ammo in a clear way.
- Make it modular.

Parts.

- Rainbow Hat
- Raspberry Pi Zero
- Resistor Kit
- Proximity Sensor

- Jumper Wires
- Pi Zero Header
- Header
- Ultrasonic Sensor
- NERF N-STRIKE ELITE RETALIATOR
- NERF MODULUS RECON MKII BLASTER (Optional)
- Wire
- Veroboard
- Blu-Tak
- Sugru
- Cable Ties
- Powerboost 500
- LiPo Battery 1000mAh
- Pi Zero Case
- Micro USB Cable
- Veroboard cutter
- SD Card
- USB Hub

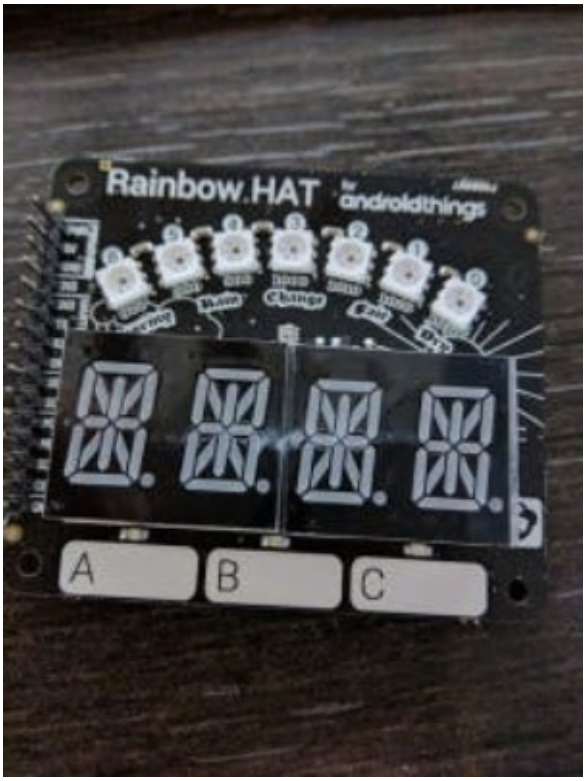
Beginning.

First its best to wire up the battery and the Powerboost 500, for instructions on this refer to my previous project.

Next job is to solder the Pi Zero 40 pin header onto the Pi itself – see here for an introduction into soldering.

Next you'll want to put the Pi Zero together with the Case purchased from above, this is pretty simple, you can then put the Rainbow Hat on top.

The hat and the proximity sensor



Next grab the veroboard, snap the male header from above into 2 parts of nine and solder them into either side of the board as such, with the top resistor being a 1k and the lower one being a 2k, this will attach to the GND pin and form the

voltage divider as per the tutorial on modmypi:

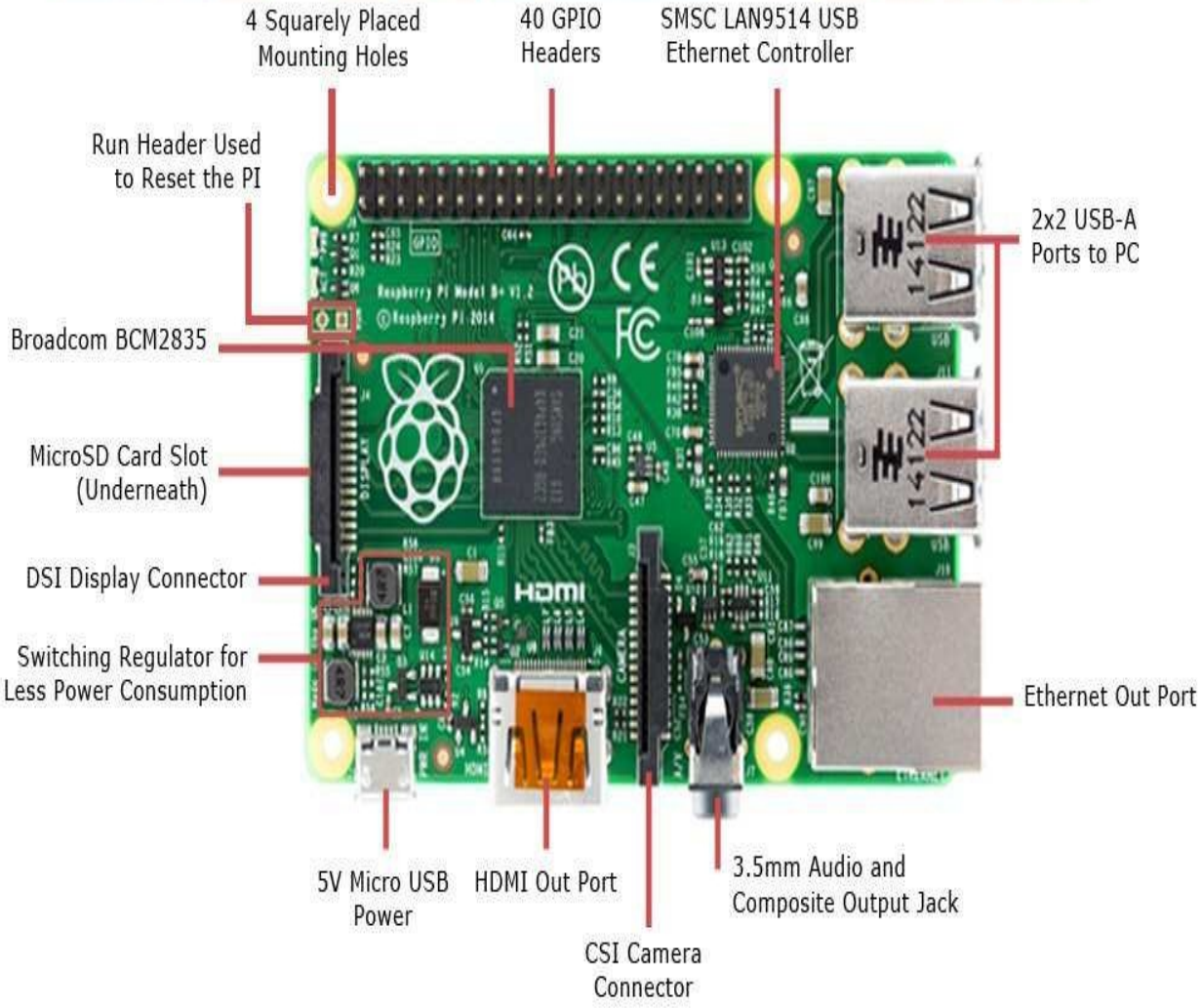
You will then want to drill the gap between the resistor in the area marked below so that it will function correctly, using the veroboard cutter listed above:

Once both ends have headers and the resistors are soldered in attach the jumper cables to the sensors and plug the cables into the headers on the side of the resistors making sure that the wires are as such:

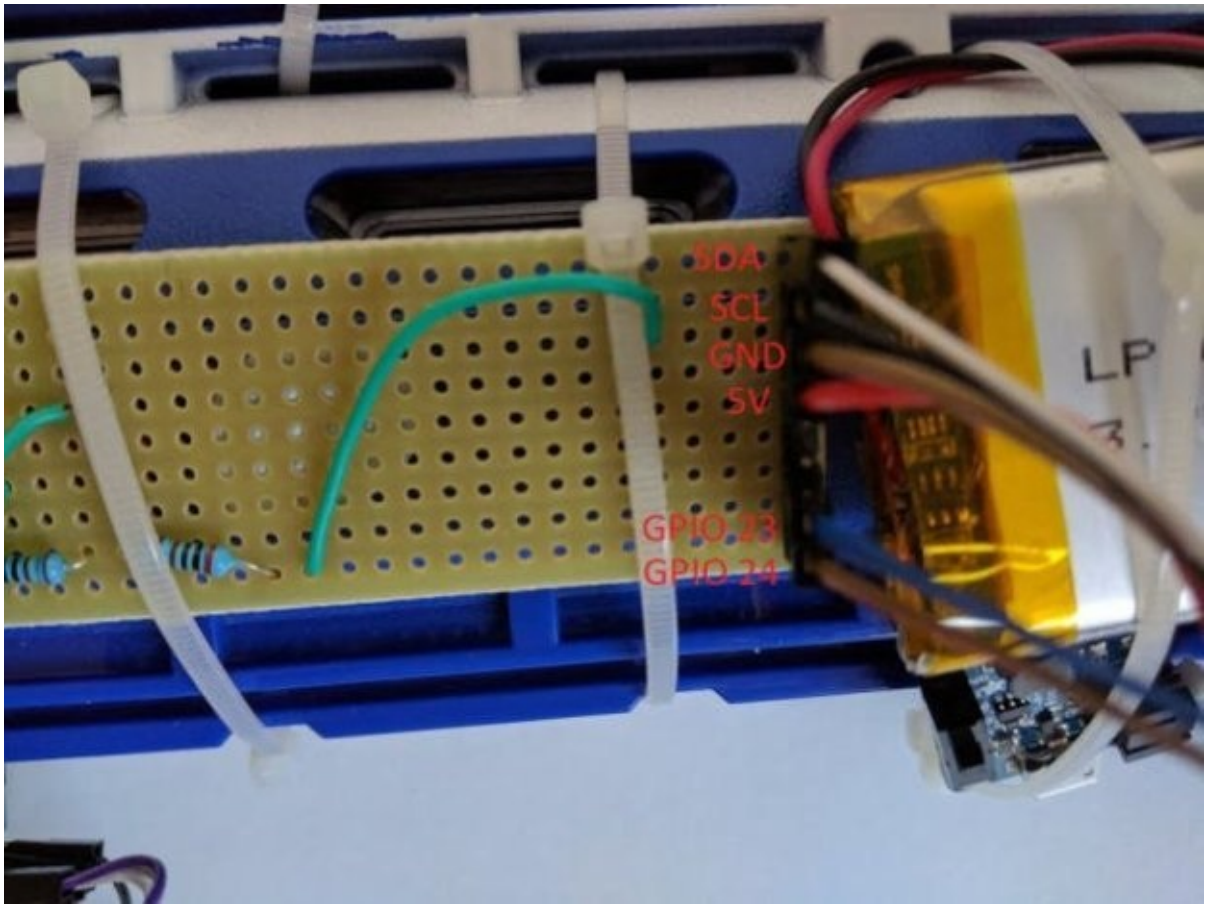
Now to save Pins, we will wire GND and VCC to share the same lines on both sensors, solder in wires as such:

While the hat has a number of breakout pins on it it does not have pinouts for GPIO 23 and 24 which are needed for the ultrasonic range sensor as per the above tutorial so I soldered in direct to the pins using this pinout (soldered from the underside is easiest).

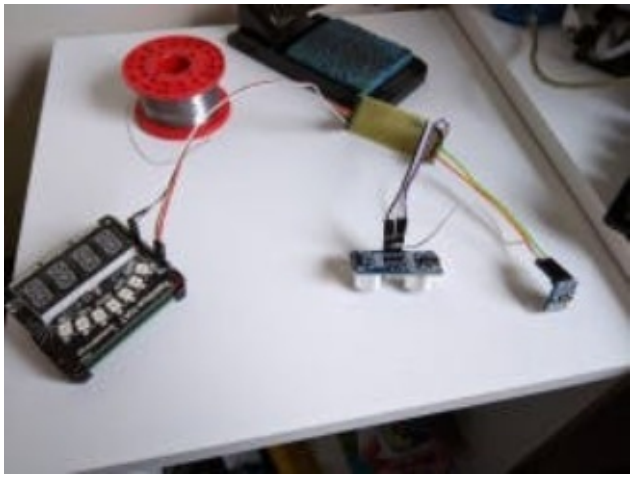
GPIO Pinout Diagram



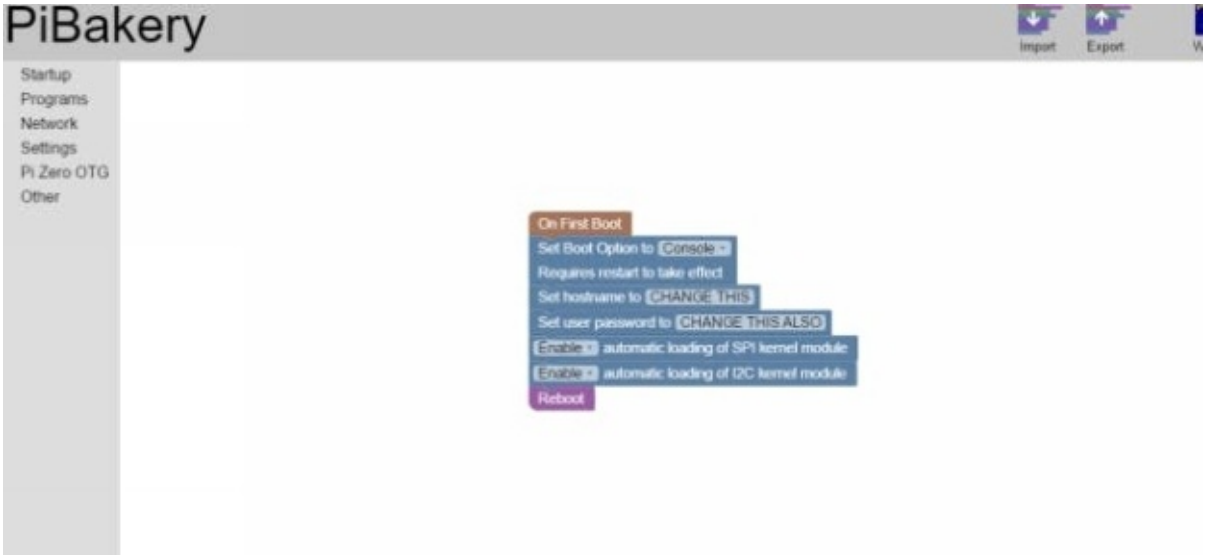
The top 4 wires there go to the breakouts on the Rainbow Hat from the Proximity Sensor, the bottom 2 go to the pins direct on the Pi mentioned above:



When all is connected up it should look something like this:



Now go to the awesome PiBakery and configure this and burn to an SD Card:



Hook the Pi up to Ethernet using a USB hub from the parts section with a KB/Mouse put the SD in and power it up!

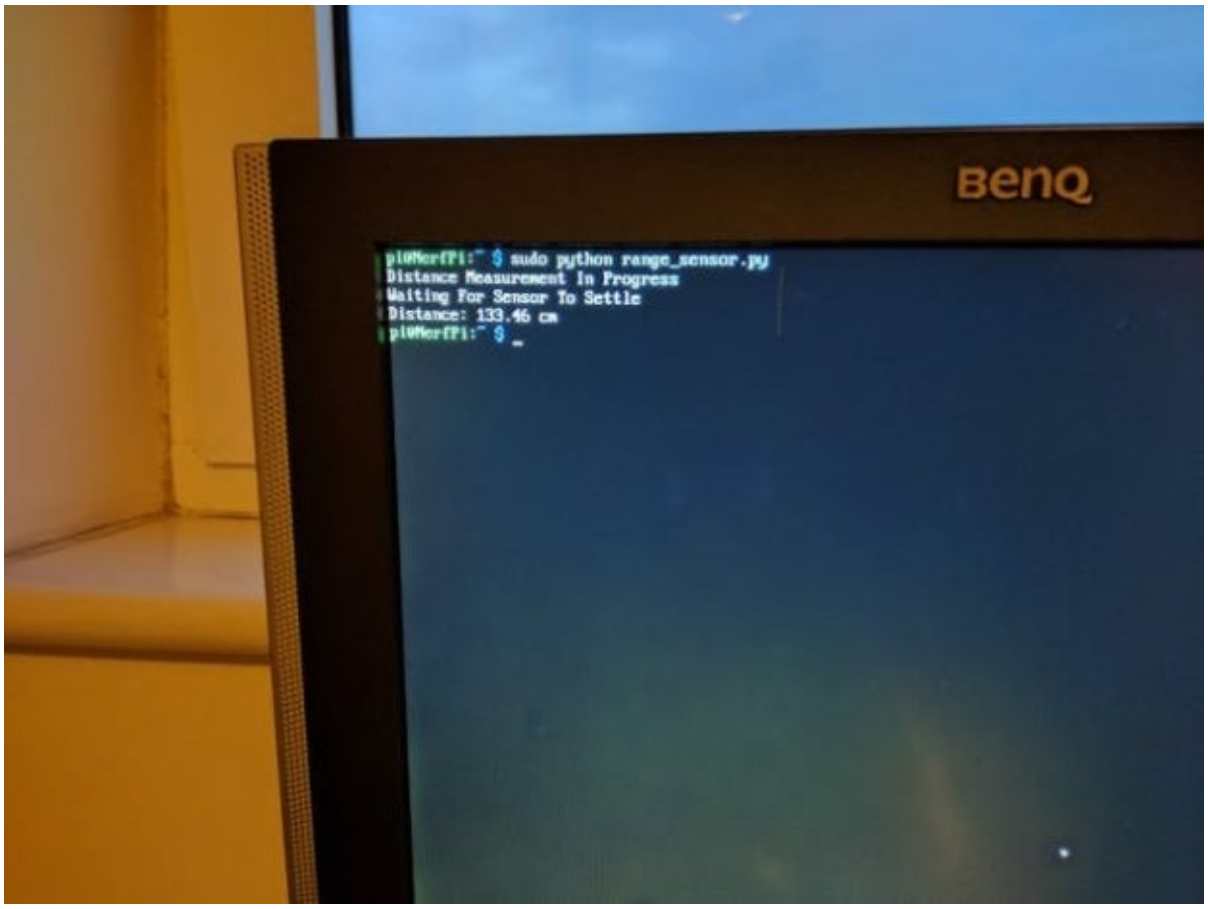
Once it has configured and rebooted, log in and run through the following tutorials/installations:

- Ultrasonic range sensor
- Proximity sensor

Now go to your IDE of choice and download the following test scripts:

- For the Ultrasonic range sensor
- For the proximity sensor

Run the range finder python code and point it somewhere within a few meters:



Now load the proximity sensor test and move the sensor around and it will output the ambient light and distance.

Functionality.

The range finder is simple enough, I want it to run and simply measure the distance in CM from the barrel every couple of seconds or so, nothing complex in the functionality here.

As for the ammo counter, I pondered ways to get it to read the ammo in the magazine dynamically but I could not think of a way to get this done – if someone has an idea for accomplishing this it would be awesome – please let me know if you manage to improve this project.

In lieu of this function I came up with the idea that the ammo per magazine being used should be set manually and then ‘loaded’, as shots are fired this will count down eventually to zero, then when changing magazine the user hits the ‘reload’ button on the hat and the ammo per magazine amount is loaded again.

When manually changing the ammo per magazine by another button on the hat it will show the ammo selection for a second and then return to the current ammo residing in the inserted magazine – there will be a video linked below that will

show this clearer.

Code me up.

Now the sensors are checked to be working we can get onto the code for the shot detector (make sure the following scripts go into your home folder):

Code for the range finder can be found under attachments.

Code for the ammo counter is again under attachments.

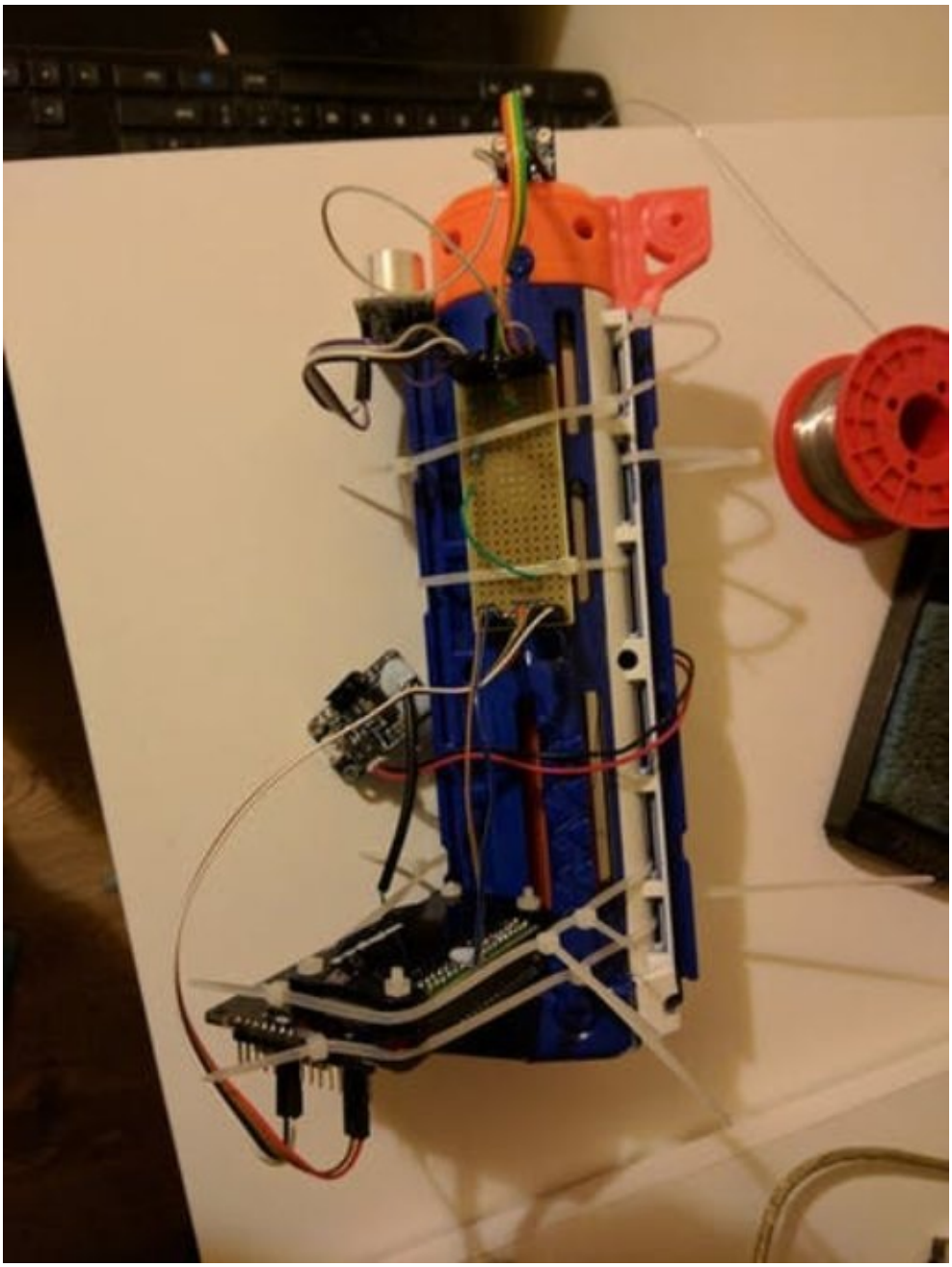
Now we need to configure /etc/rc.local to launch the scripts on boot, also found under the attachments of this project.

Save, reboot and watch as the Rainbow Hat lights up!

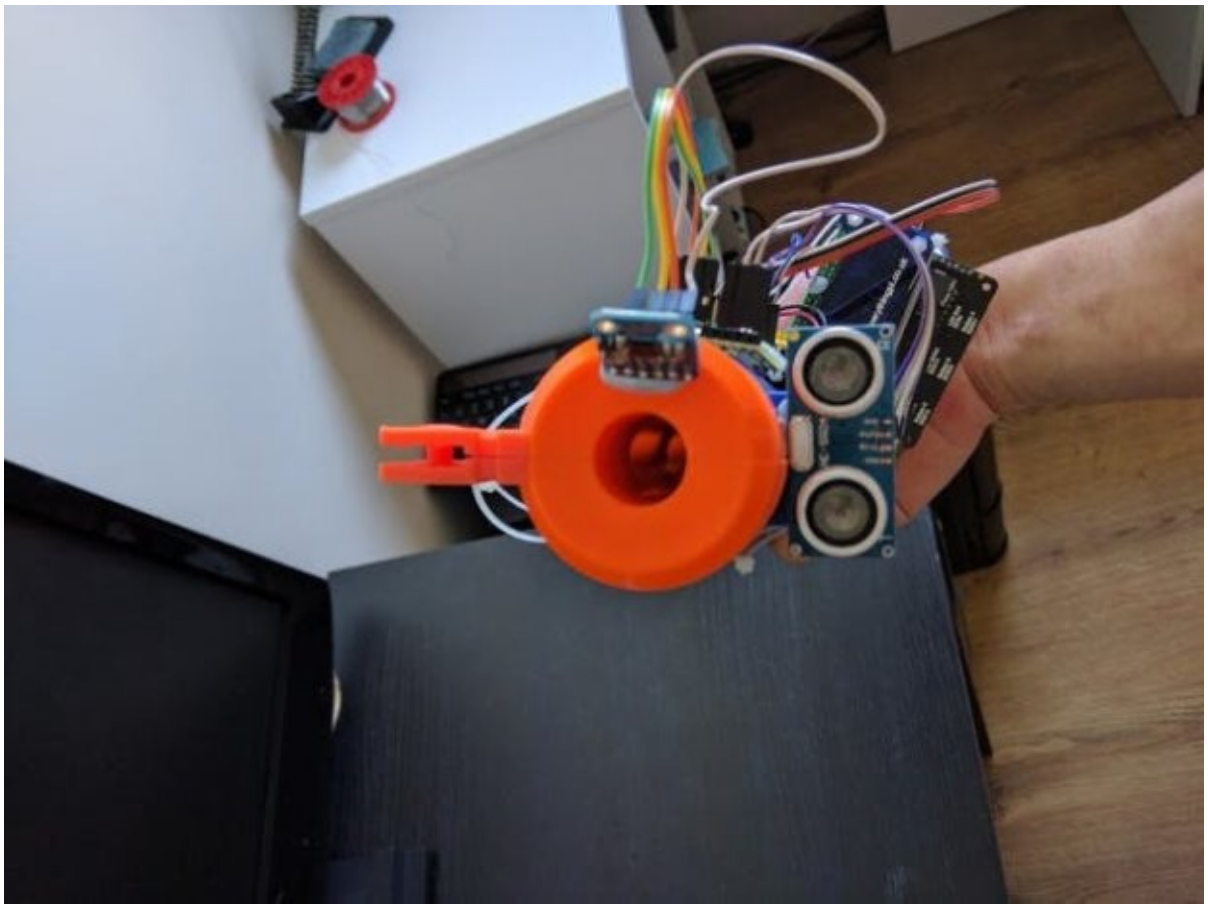
Cable tied.

Now its time to attach it all to the modulus-compatible barrel of the Retaliator, being that the electronics will be on the barrel only this will mean it can be connected or disconnected to or from any modulus Nerf gun.

I ran cable ties through the tiny slits just below the top rail:

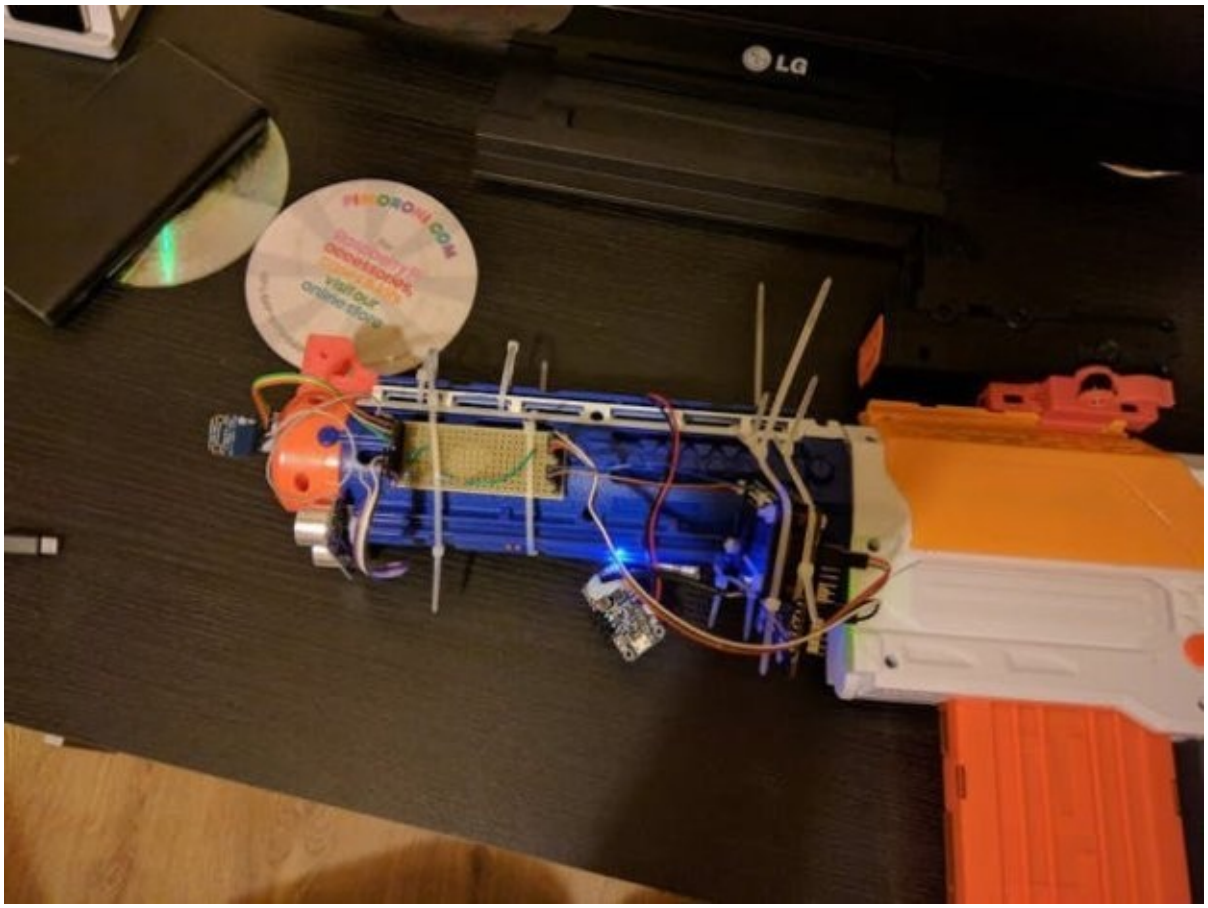


I used blu-tak to hole the PCB, the ultrasonic sensor and the proximity sensor in place (for temporary positioning/testing, Sugru can be used later for a more permanent solution):



The proximity sensor needs to be angled as such so it does not pick up on the proximity of the end of the barrel itself, but will still pick up shots as they leave the barrel.

Wrap the cable ties around the veroboard, the Powerboost, the battery and the Raspberry Pi/Hat assembly itself and pull the ties tight – it is probably best to tighten this one at at a time using hands to hold them in place (4 cable ties were enough to reach around the barrel for each loop):



Now cut off the loose ends of the cable ties and check the stability of the construction and attach to the Nerf gun of choice (mine being the Recon MKII):

Operations.

Here is a YouTube video showing the device in action.

In the video you can see the following:

- Me switching the device on and the range sensor eventually measuring the range to the door in the background.
- I then set the number of rounds per magazine being used to 2.
- I then load the Nerf up and fire off a round and the counter responds accordingly.
- I then hit the ‘reload’ button to simulate a new mag with 1 Nerf dart in it.
- Next part of the video I switch the device on and show it actively measuring the distance to the door as I get closer / further away from it.
- I fire off the 2 darts in the current mag and put in a new mag with 6 rounds, as the default mag size was set to 6 when I hit reload the LED’s reflect this.
- I then fire off the remaining 6 rounds and the LED’s count down 1 per shot.

Conclusions.

So it works quite nicely I think and the aesthetic design lands somewhere between Half-Life 2 and Metro Last Light, with some Titanfall sprinkled on.

The next logical step would be to find a way to actively and automatically measure the number of rounds in an inserted magazine and reset accordingly; I am hoping that someone has found or will find a solution to this and improve my design or use this as a jumping point for an entirely better project – please do let me know I would love to see this idea be improved upon as mine is rather, rustic.

Let me know if I've missed anything in this guide or if you want some more info and I shall provide it!

CODE

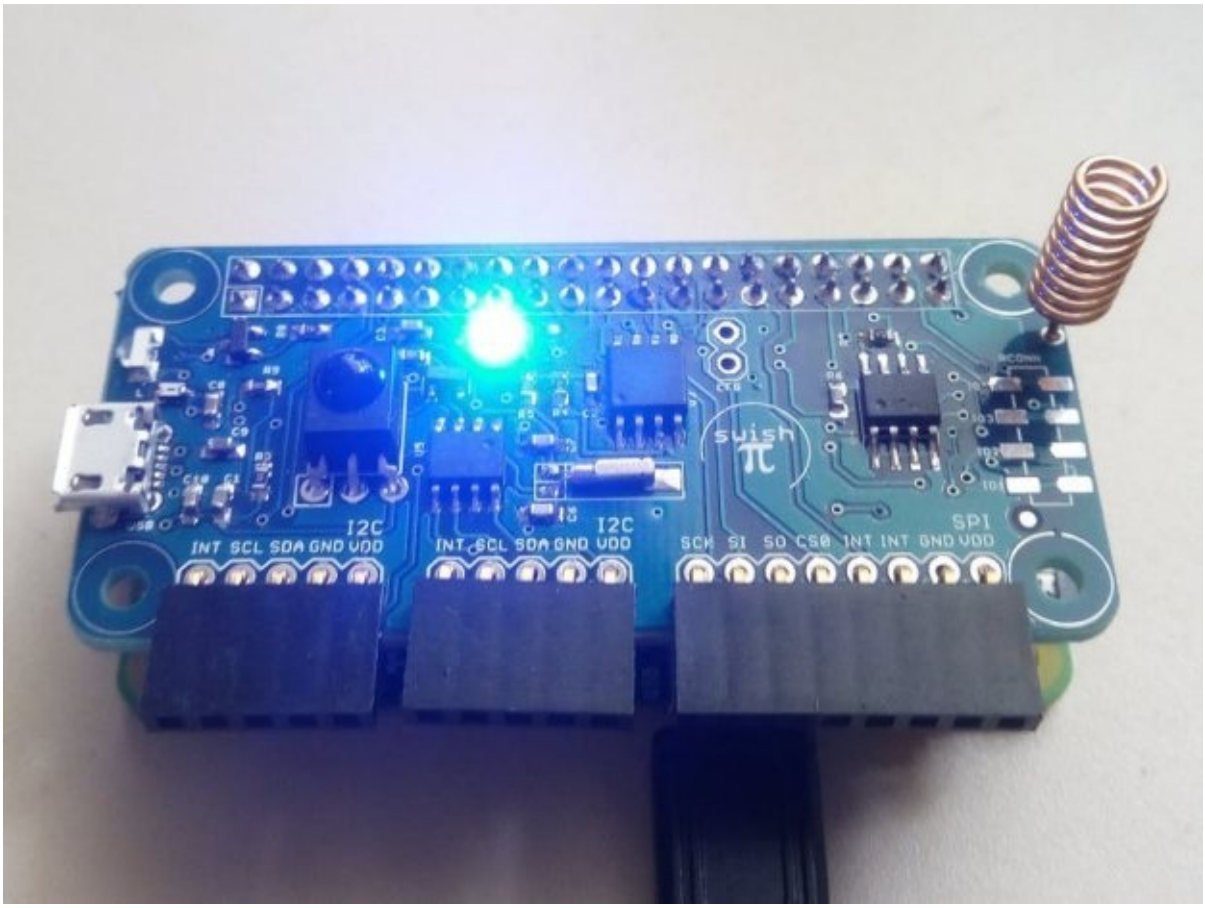
Ammo Counter Code

Range Finder Code

rc.local

Github

Raspberry Pi RGB Console with SwishPi pHAT



In this project I will show how to build an RGB LED console app that can be controlled from the local network or anywhere via the Internet. This allows the user to turn on or off the individual RGB LEDs via a web-browser app. The application will use the Python Flask framework and the SwishPi pHAT together with a Raspberry Pi Zero. The SwishPi pHAT is a modular shield that includes a number of sensor and interfaces such as medium range radio, serial to USB login, non-volatile memory, real time clock and sensor headers.

RGB app.

Let's get started building a Flask app to control the RGB led. The easy route is to clone the Git repository link below and run the example by issuing:

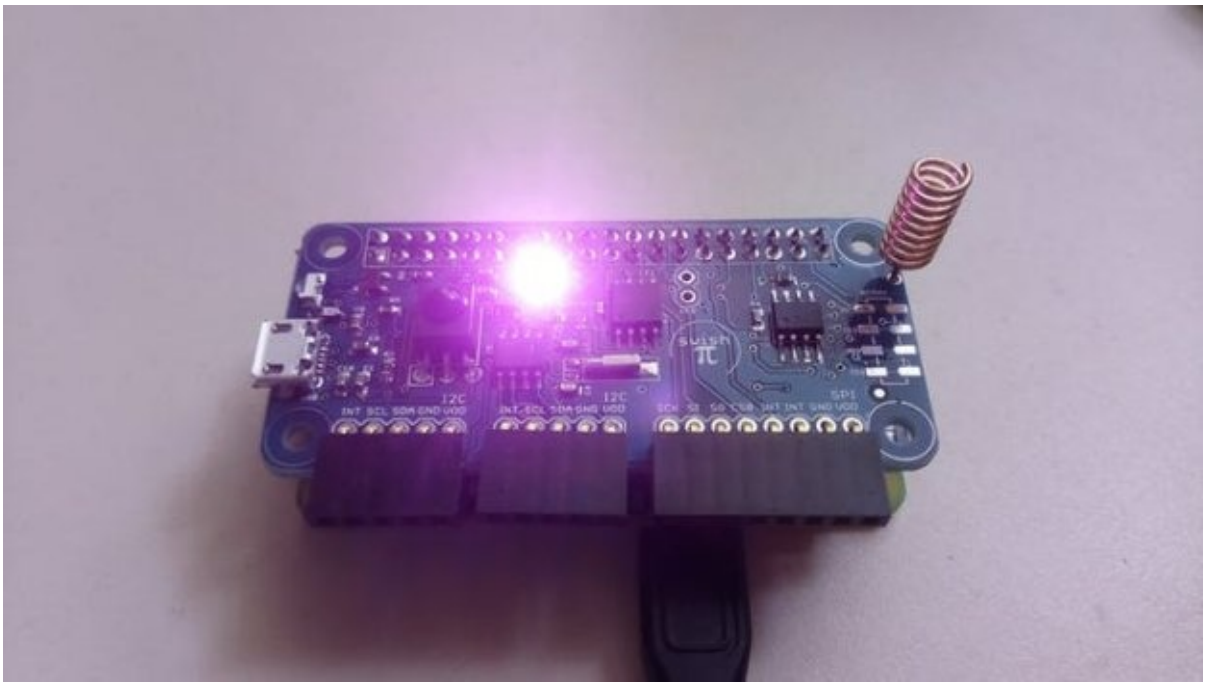
```
sudo python app.py
```

The pre-requisite to this is having Python Flask installed. By default Flask is not installed so you will have to install it issuing:

```
sudo pip install flask flask-basicauth flask-lesscss
```

Below you can see the result of what we are trying to achieve. In this particular

instance the blue and red LED's are turned on.



Python uses the RPi. GPIO library for controlling the GPIO (general purpose input/output) peripherals. This library is very versatile since it offers support for configuring the pins in different modes and it also handles interrupts. The module comes pre-installed on recent versions of Raspbian. If the module is not installed the easiest way is to install it by issuing the command:

```
sudo apt-get -y install python-rpi.gpio
```

The library uses two configuration schemes. The BCM numbering scheme for the pins follows the SOC pin numbering layout. So in this case, pin 24 of the header is named GPIO18.

RGB SPA

Let's take a look at the SPA (Single Page App). It's purpose is to turn on or off the RGB LEDs from the client browser side. This allows any device on the network to turn the RGB LEDs on or off.

First, we setup the pins directions. Then, we create individual functions that turn on or off the LEDs. Finally, we create an API by creating one route for the RGB app.

A POST request is used to parse the button Name and Status from the client side. Depending on the button status each RGB is either turned on or off.

```
@app.route("/RGB" ,methods=['POST'])
def RGB():
if request.method == 'POST':
button = str(request.json['name'])
```

```
bstatus = str(request.json['status'])
```

To beautify the app, JQuery and bootstrap are used. This keeps the amount of Javascript code to a minimum.

The button callbacks have been included, on the html page under the template folder. Every time a button is pushed on the client side a POST request happens, which gets processed from the Flask server.

Below you can see the RGB LED panel with the blue LED turned on.

This app uses the RGB LEDs at full power. A slightly more sophisticated example would make use of PWM to modulate the intensity of each individual LED. That's a project for next time.

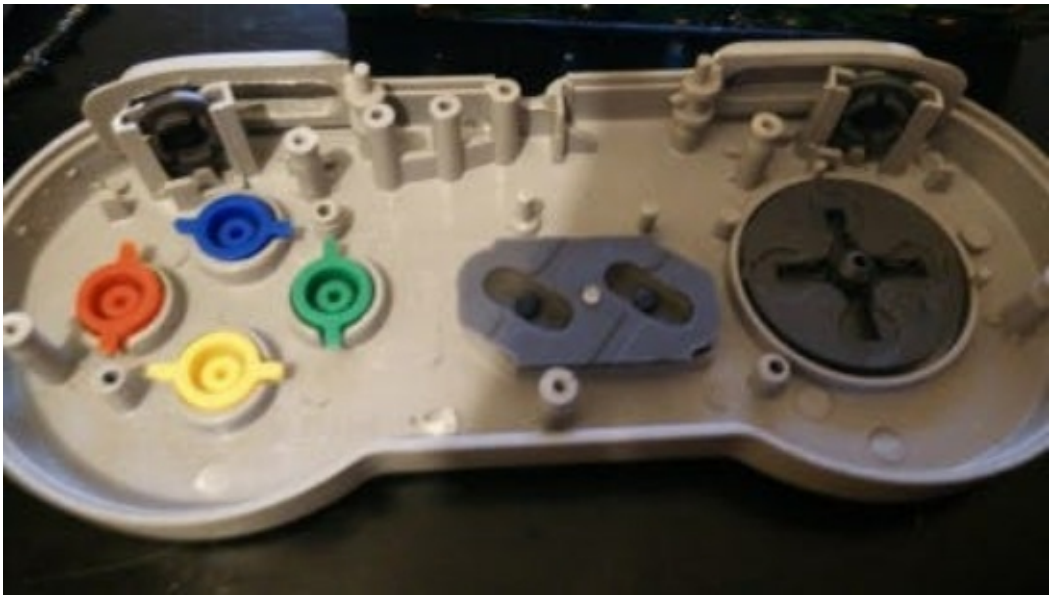
Code

SwishPi pHAT schematics

SwishPi pHAT RGB App

Raspberry Pi Zero Controller Console

With the release of the Pi Zero, everybody has rushed to see how they can use this relatively powerful computer with such a small profile. I saw a few versions of people putting the Pi inside game controllers to host the console from within itself so I thought I would give this a go. A friend was able lend me a real SNES controller, so it became the victim of this project! Most of the similar projects online used USB controllers so I would be doing something a little different here.



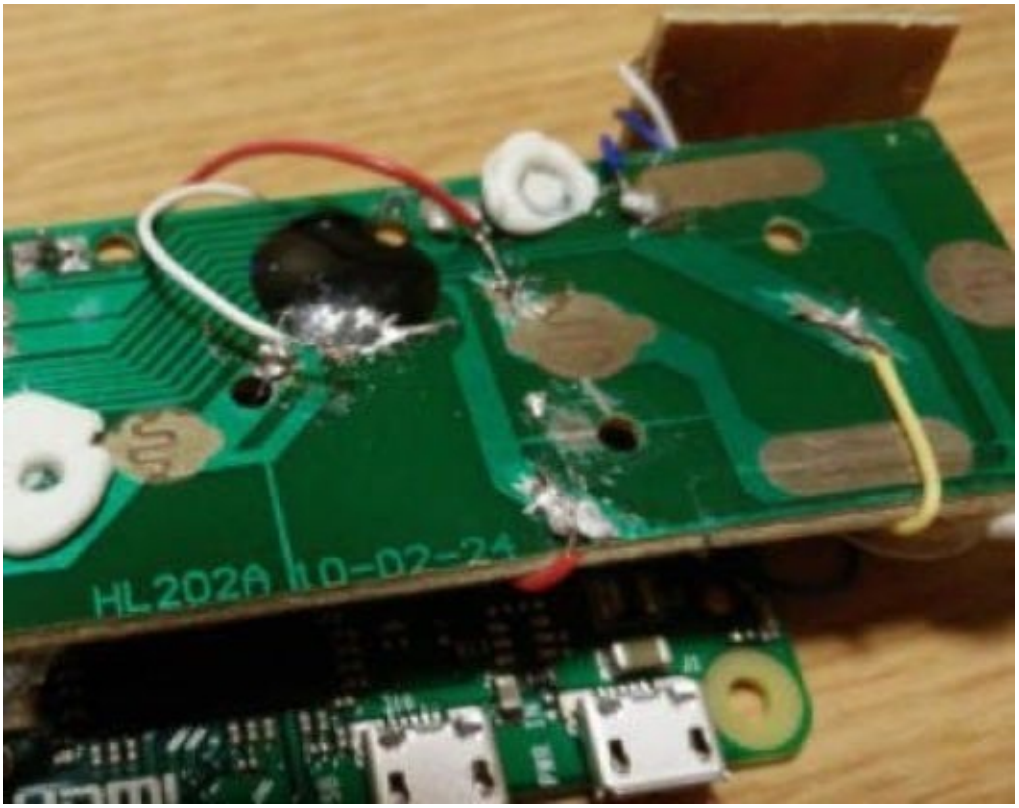
My initial plan was to just cut the cable for the controller, solder the 5 wires to the pi and use the snesdev module that is a part of the Retropie operating system. For some reason that I still don’t know, I couldn’t get that to work for me so I had to figure out a different way around. I tried using a Teensy board to interface the controller to USB and plug that into the pi, which worked, but I couldn’t manage to fit all the boards in the case.

The final, and most time consuming plan was to figure out a way to wire each individual button to the Pi which created a lot of work and headache but if I could get the software right, would almost certainly work.

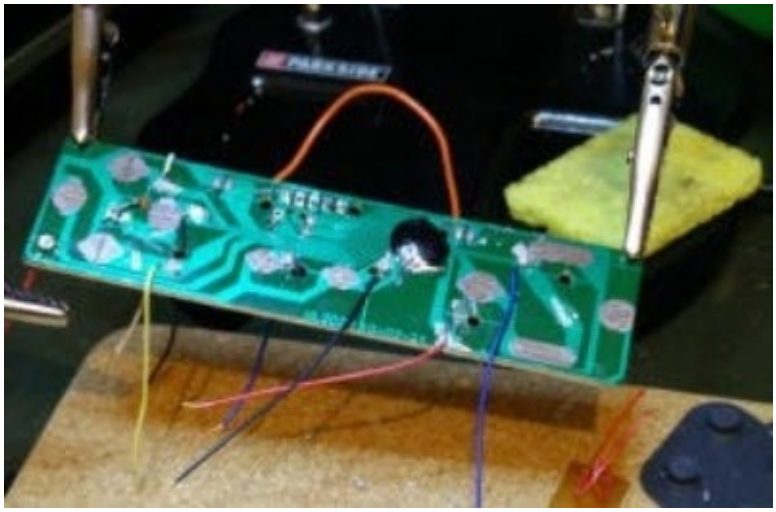
Here I will explain how I went about this:

Hardware

The first step was to remove the PCB from the case and see how the buttons worked. Each button has a common ground contact and an individual Vcc contact that are connected when the button is pressed down. For this I would need to scrape away the PCB to expose the metal for each button.



Next step was to solder the wires on to the new contacts that I had exposed. This proved the most time consuming part as the PCB board was resisting the solder quite a bit but eventually all of the wires complied.



Once I had the PCB board covered, next I had to solder the other end of the wires to the unpopulated GPIO ports taking note of which button went to which port. Soldering to the unpopulated ports is not only far easier than soldering to pre-soldered pins, it is almost essential for this project as the Pi probably would fit in the case if they were present

Now with all of the soldering done, my apparatus was ready to put into the case but a few structural adjustments were required with the Dremel. I cut openings for the cables at the bottom of the controller just big enough to fit the HDMI and the power cables.

Finally, I had to create some makeshift spacers to push the board down enough so the button presses were always registered. Once that was done, and the case was screwed together, that was the hardware element to my project complete.

Software

So the first thing to download is the latest RetroPie image from the RetroPie website.

Once I had that downloaded and written onto the SD card I found the C file that would be the secret to getting this project to work. The source code is available at this Adafruit repository. Just download this folder into your home folder edit the file appropriately to match the pins that you have soldered to on the Pi. Then once you run the makefile and reboot, your buttons should be matched up. There's plenty of instruction included on the Adafruit Github page for that part of the job.

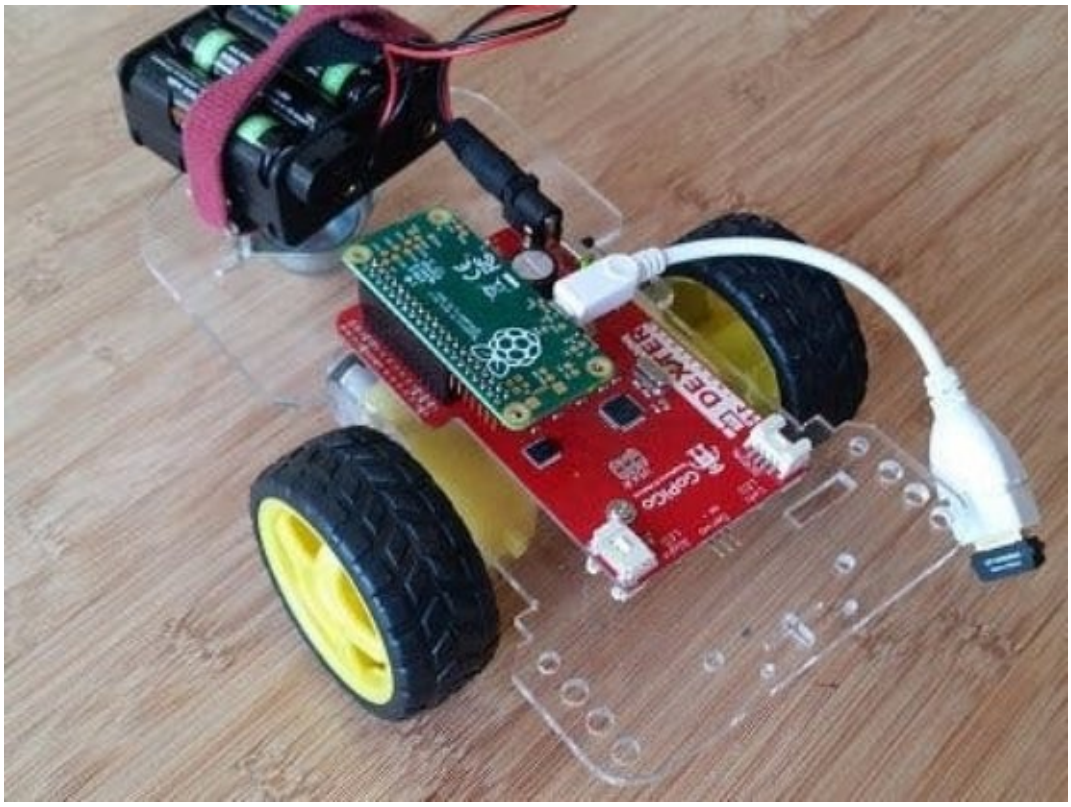
So if that all worked, you are pretty much done. The button presses will now be recognised as keyboard strokes and you configure the controller like you would any controller. Before you screw together your case make sure to upload your ROMs to the SD card and then away you go.

Im very glad I saw this project to the end because I ended up learning a lot more than I initially expected. It took a few times to revisit over a period of time but I eventually cracked it. If you have any questions, don't hesitate to contact me

about the project.

Download SourceCode

GoPiGo with the Raspberry Pi Zero



Step 1: Setup

Parts Needed to Make the Raspberry Pi Zero into a Robot!

Before we begin, you should note that you'll need the Cinch Operating System. This is an easy-to-use operating system for the Raspberry Pi, specially designed for mobile robotics. Cinch works by broadcasting a wifi Access Point from the wifi dongle, allowing you to connect over wifi from a laptop, a phone, or a tablet. You can read more about Cinch here, you can download it for free here,

and you can find instructions for getting Cinch it on your microsd card here. You can also purchase Cinch, preloaded on an SD Card from our website here.

For this setup, we will need the following:

- GoPiGo Raspberry Pi Robot
- Raspberry Pi Zero
- MicroSD Card with Cinch
- OTG USB Cable
- Dexter Industries Wifi Dongle

The Raspberry Pi Zero Slides into the GoPiGo, it should be placed so that it hangs over the red GoPiGo board.

The Raspberry Pi Zero Seated on the GoPiGo

Insert the SD Card into the Raspberry Pi Zero (caution: make sure the power is off!). Finally, insert the OTG USB cable into the Raspberry Pi Zero port labeled “USB” and insert the Wifi Dongle into the USB cable.

After assembly, your GoPiGo should look something like below!

Fully Assembled GoPiGo and Raspberry Pi Zero.

Step 2: Connect

Connecting with Cinch is easy! First power up the GoPiGo by connecting the battery pack and switching the power button on the GoPiGo to “On”.

After approximately a minute, a new wifi network will appear, called “dex”. Connect your device to this network.

Finally, in a browser, type in “. ”. This should connect you to the Raspbian for Robots Front page.

Entrance Screen on Cinch.

From here, you can login to VNC or SSH.

The standard password is "***robots1234***".

Step 3: Run the Robot

The fastest way to try out the GoPiGo is to run the demo. If you click the Scratch icon and select the GoPiGo as your robot, you will see a button to test the hardware. If you click this, the motors will run back and forth.

You can also run the following in the command line:

sudo python Desktop/GoPiGo/Software/Python/basic_test_all.py

That’s all! Three simple and quick steps to turning your Raspberry Pi Zero into a Raspberry Pi Robot!

Schematics

GoPiGo Setup
Source Code

Holiday Pi Movie Camera

Overview

When I was a kid, my sister and I used to have a lot of fun with our family's 8mm home movie camera. Everything about the overall experience, even the deferred gratification of waiting for the film to be developed, was irreplaceably magical. The live preview on camcorder screens (and then later mobile phone screens) dispelled that forever.

Well, perhaps not.

A while back, my wife and I found an old Mansfield Holiday Zoom movie camera on Yerdle. The mechanical parts were still fully operational, but the lens was cracked, so even if it were possible to obtain and develop film for it, that was a nonstarter.

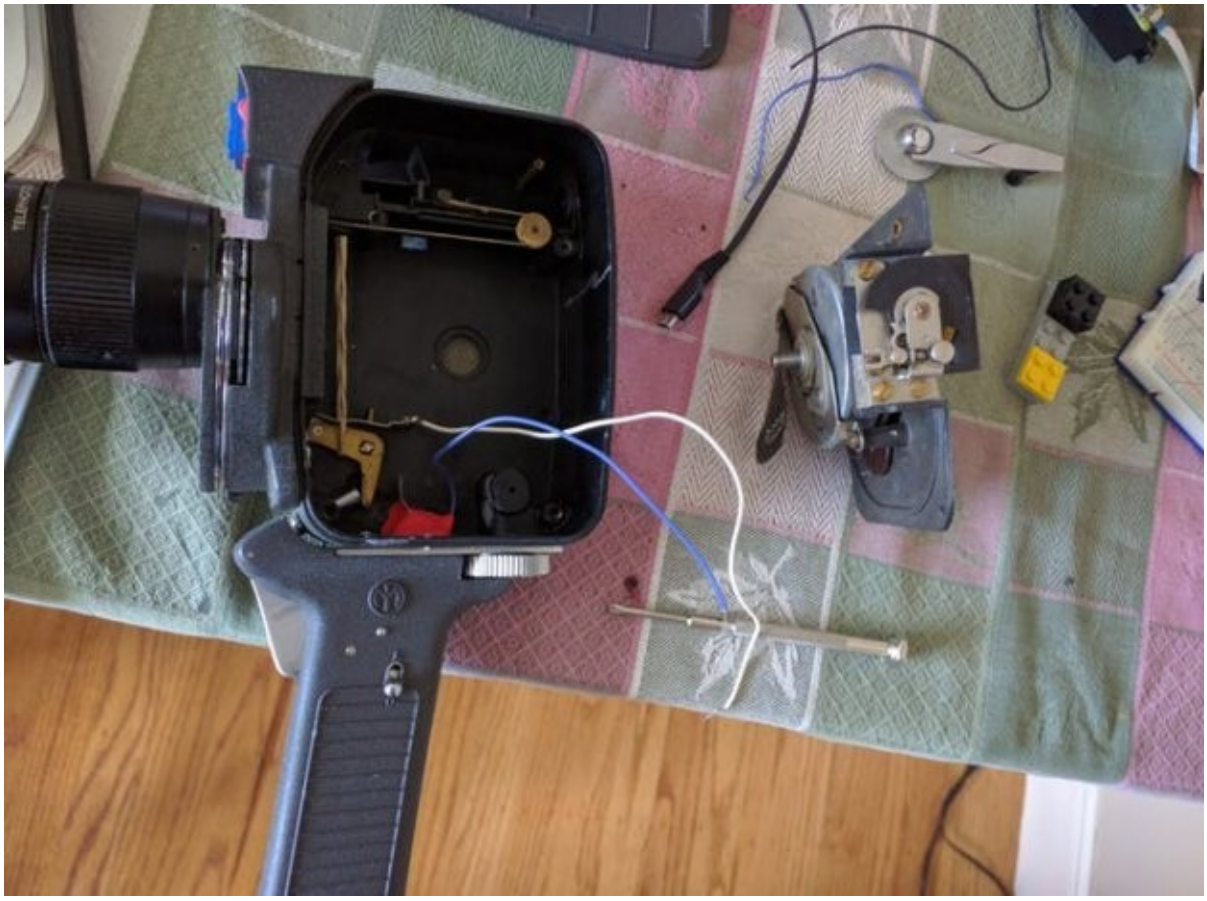
What we did

So, time to replace the lens and film with a Raspberry Pi Zero and Raspberry Pi camera, while still preserving the overall experience:

To do so, I had to remove the big heavy spring to make room inside for the components. This was a shame, because winding the spring and listening to it whir would have been a major bonus for nostalgia factor, but oh well.

Controlling Recording

Next step was to attach wires to the trigger mechanism to control recording start/stop:



(You can see the disemboweled spring mechanism on the right.)

With the spring removed, the trigger would no longer automatically stop recording when released, so I used a rubber band to restore this functionality. Inside, the trigger connects to a bronze hinge, so I soldered one wire (the white one) to that. Using a multimeter, I discovered that all of the metal parts of the camera were connected, so I couldn't detect the "open" state of the trigger by connecting a wire to the other side of the gap (the silver part at the lower left inside). Instead, I used sugru to fasten a random piece of metal at the bottom (with no conductivity to any other part), and then soldered a wire to that. Now when the trigger is pressed, the hinge makes contact with the extra part; when the trigger is released, the circuit is open.

Pin Connections

Then it's as easy as connecting one wire to the Pi's 3.3v contact, and the other wire to one of the GPIO pins (I chose BCM 16). When the circuit is closed, the GPIO will go high; it will be low at all other times.

I attached the Raspberry Pi camera via Sugru and Lego at the top of the camera (being careful not to block the viewfinder). Then I threw in power (via a USB

phone charger battery) and WiFi (via a dongle). After booting the Pi Zero, the camera can be started via a custom script, and then once filming is complete, the results can be "developed" via another script and then downloaded via SCP.

Both scripts are available in this Github repository. Tip of the hat to this unrelated project, from which I adapted the Python code.

For the full retro experience, this project works quite well for digital post-processing. Here are some example results, starring our dog!

It was fun being able to walk around filming, with everything self-contained. However, there are a few improvements I'd like to work on in order to make the experience more substantial:

- Figure out a good way to completely close the camera compartment; the removed spring also had a hook used by the door, so without that, the camera operator has to hold the compartment door shut with a spare finger, or use something like a clamp or rubber band.
- Mount the camera internally; in order to do this, I would need to smash out the broken lens. Even better would be to use a camera with an intact lens, and figure out how to deal with the focus; this could produce some very authentic analog retro effects. Or maybe none of this would work at all.
- Shoehorn the spring contraption back in there. Seems impossible, but maybe with a bigger camera.

Code

Holiday Pi Python Code

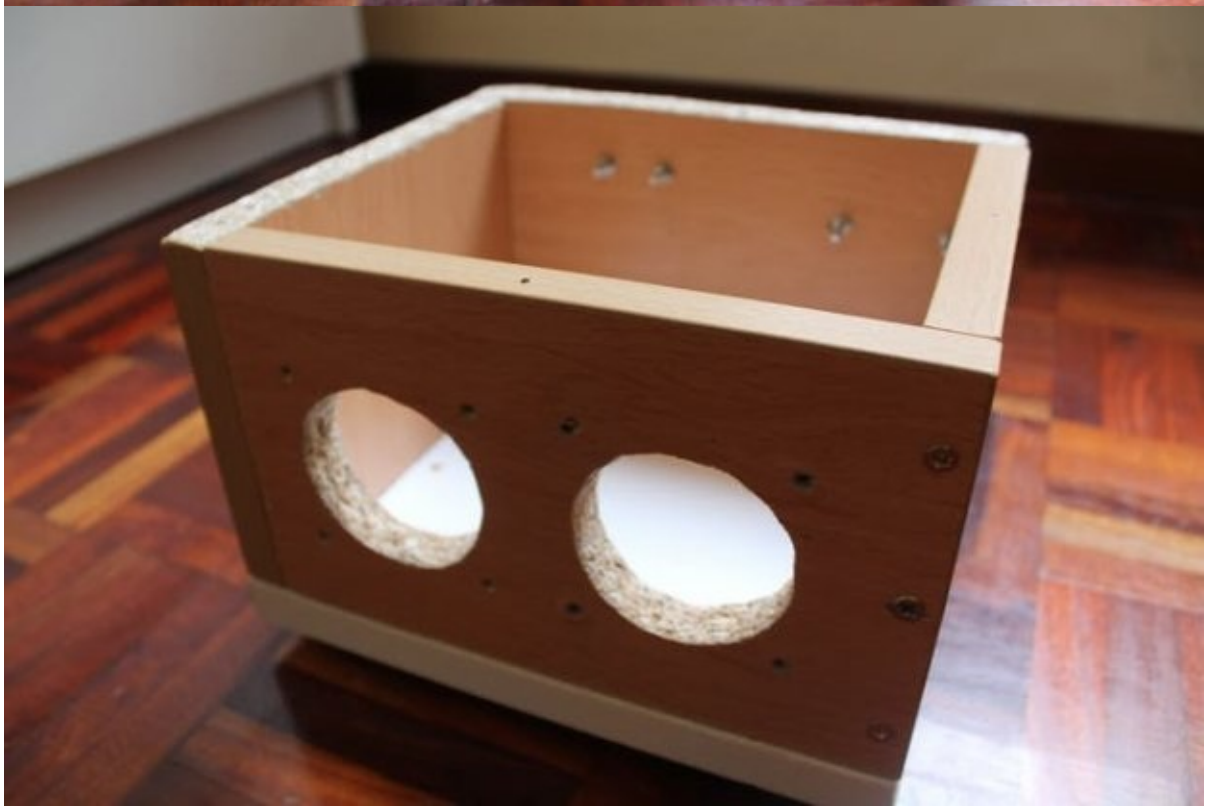
Pi Outlet Relay Control using Cayenne

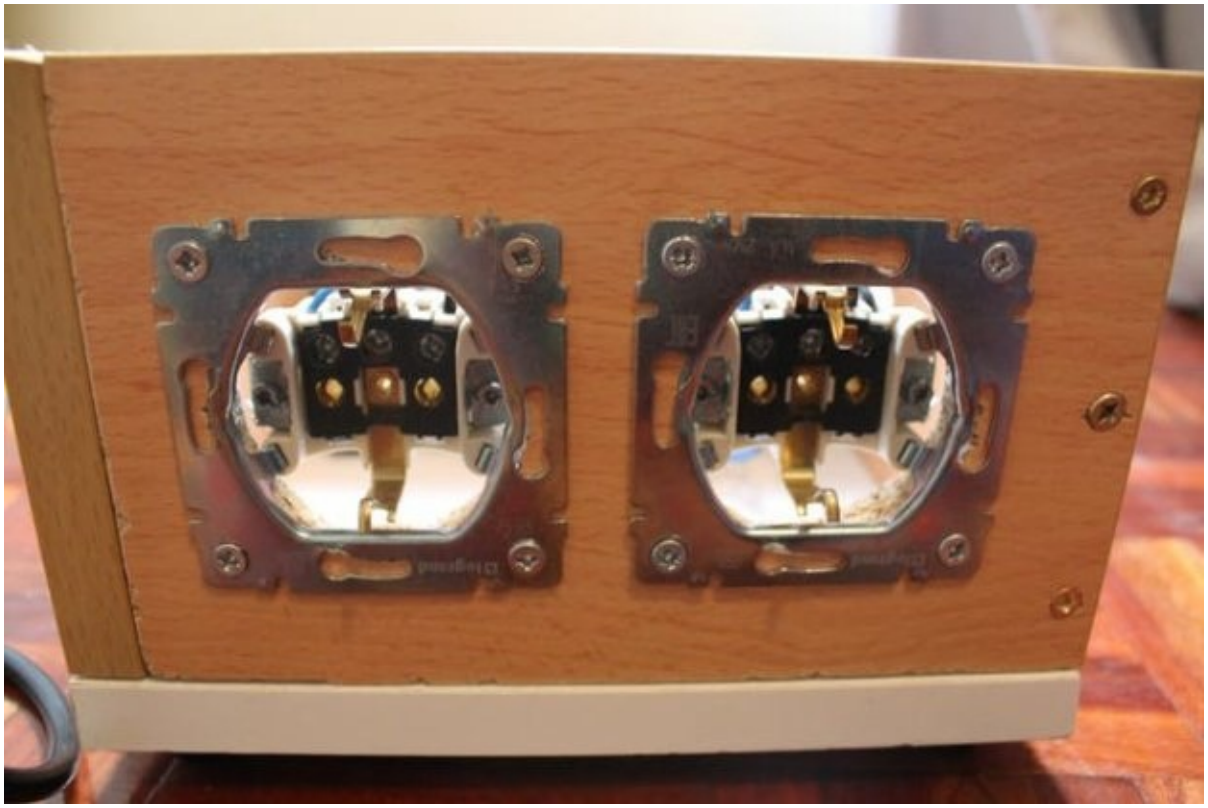
Here i'm describing how to build a box where you can plug devices, and with the aid of a Raspberry PI and Cayenne, you can control anything you like from anywhere.

Please let me know if this (or my english) is not clear enough so I can edit it.

The enclosure was built only using reclyced materials I found around.

- Assemble all the frames except the top one with screws.
- Drill two holes where you will fix the outlets
- Fix the Outlets
- Pimp it as you like xD (I added some supports on the bottom and a puller on the top)

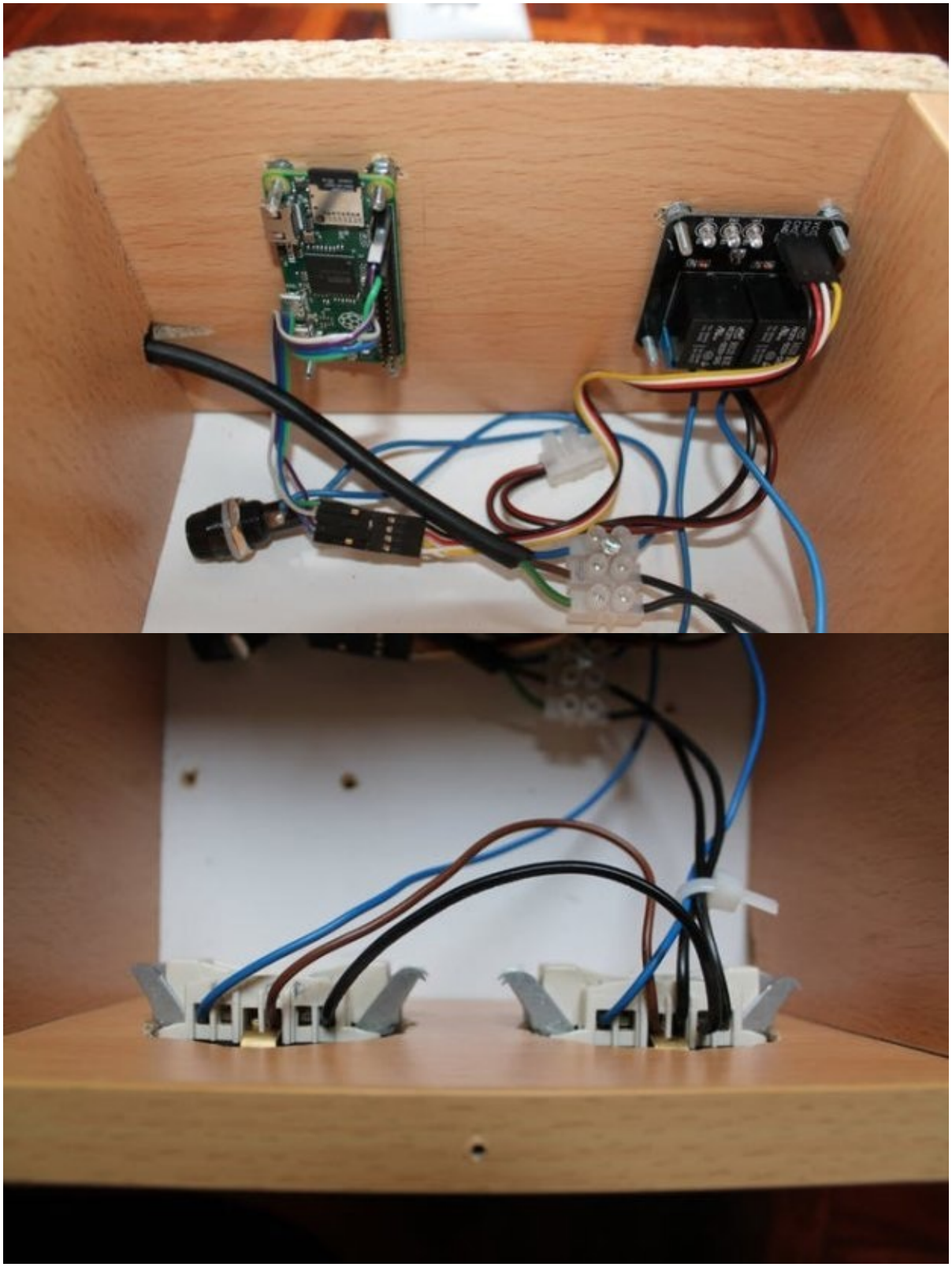




Follow the diagram attached in order to wire everything as it should.

I recommend some extra caution here since the relay will be used to turn on and off devices that uses 230V AC.

Don't forget to check everything before you plug it to the wall, and if you have any questions or doubts please message me before you try it!



Here is where this project gets different from the ones you already saw using Outlets/Relays/Microcontrollers.

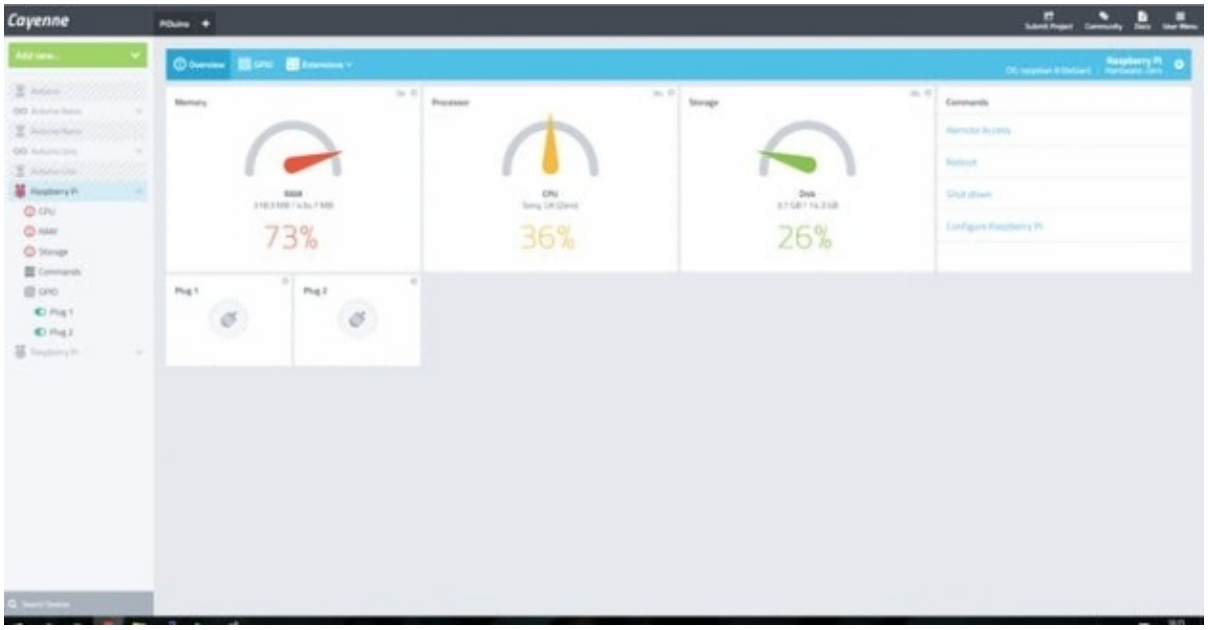
I used the Cayenne Dashboard application so I can control the status of my Relays anywhere I am.

A very simple example that I made, was using an Arduino and a DHT11.

I had the Arduino sending a temperature value to my Cayenne Dashboard, and everytime it reached a certain threshold value, it triggered an event to turn my Heater that was connected to my box to "On".

Pretty simple and amazing since you don't need to code anything!

Please go to www.cayenne-mydevices.com so you can get it running on your PI.



See it Working!

Schematics

Circuit diagram

See Through Buildings With a Drone

This drone can show where people/things are located within a building (or anywhere) by displaying an image with different colors based on density. It uses a Walabot, Raspberry Pi, and multiple other micro-controllers that send information to a ground-station. This is a very complex and time consuming project but I'm sure if you have some type of experience with building drones and python, then you should be able to complete it. If you have a questions, just leave them in the comments and I will try to answer them. Let's begin!

Questions:

- **Why would it be good to see through buildings with a drone?** My reason was for firefighting and/or SAR, the drone (under legal privileges) will be able to fly above a burning building and show where people or pets are trapped.
- **How does it work?** It works by using a Walabot, a 16-segment radar array that can "see" through solid objects. The code I'm using displays objects by their density. An animal shows up more red on the screen because of the higher water content in their body making them more dense.
- **Are you intruding in my privacy?** I know with the way people look at drones nowadays, they believe that they are just for spying on people. I understand that there are people that are inappropriate and should be punished, but most people flying drones are doing everything legal and appropriately. This drone will be only used for good purposes so you don't have to send me messages about how you think I'm breaking the law.

Problems:

- **Frame Rate:** The Raspberry Pi Zero isn't powerful enough for the Walabot image processing, so the frame rate is really slow. Suggestion: If you have the room, which I didn't, you should use a larger Raspberry Pi (like the Pi 3).

- **Incorrect Display:** If there is something like a refrigerator, it will display it just like a person, this could be a problem in some cases. Luckily, if there is a person, they still will show up so it's not too bad to work with.
- **Interference:** The Walabot sometimes does pick up interference from the motors, luckily it's very rare.
- **Battery Life:** The drone is really heavy, so the battery life is really bad. Unfortunately, this cannot be changed.

Step 1: Gather All Your Materials

You are going to need to get a lot of parts for this build; I have listed them below. (Be advised that if you choose different parts, like a different frame, you might need alternative parts.)

I ordered most of my parts from GearBest. They all worked and came in a reasonable time.

- Main Frame - I bought mine from eBay with an built in PDB
- APM with compass, GPS, and Power Supply
- Walabot Pro
- Li-Po Battery (3-4Cell 5000mAh)
- Wire
- 6xESC's (30A)
- 3xCounter Clockwise Motors
- 3xClockwise Motors
- CCW and CW Props (You may want spares)
- 5.8GHZ FPV TX/RX
- Raspberry Pi Zero
- Screws for motors or anything else you might need them for
- APM Telemetry

- 2.4GHZ 7-channel min TX/RX
- Heat Shrink

Tools:

- Soldering Iron
- Solder
- Screw Drivers
- Allen Wrenches
- Pliers
- You probably will need other tools but it all depends how you build it

Step 2: Build the Frame

Build your frame first. If your frame has an on-board power supply, you

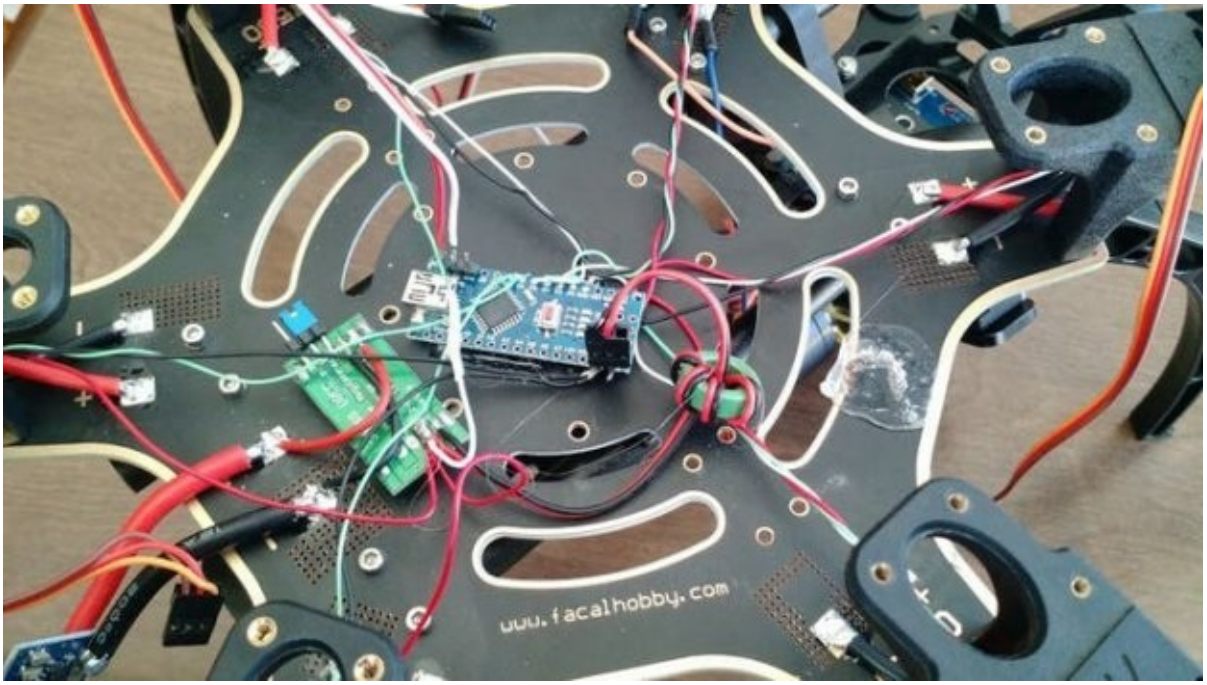
probably don't want to put the top cover on yet. Your frame should come with build instructions, however you probably won't need them.

PHOTO CREDITS: "HKG Store" - Where I bought the frame from.

Step 3: Mount and Solder the Motors and ESCs

Mount each motor to each arm (**MOTOR ORDER MATTERS: FOLLOW THE PICTURE PROVIDED**). Solder the ESC to the PDB and solder the motor to the ESC. Before putting heat-shrink on the wires, make sure the motor is rotating in the right direction by connecting the ESC to a receiver and testing the direction. Most motors also have threads for CW or CCW movements, make sure it all matches up.

Step 4: Optional, Install the LEDs



I used an Arduino Nano and NeoPixels for the LEDs. Connect everything to a UBEC or BEC (**DO NOT CONNECT TO THE FC, IT DRAWS TOO MUCH AND WILL FRY IT**). You can find a project for building the LED setup on my website here. I used electrical tape to hold the LEDs on the arms.

Step 5: Install the FC and Its Add-Ons

I used a vibration absorbing mount to hold the APM on the frame. Everything else can be mounted however you would like, vibration won't affect it. The compass (which is attached to the GPS) should be above the motors to prevent motor interference, I did this by using a GPS mount attached to my 3D printed dome cover. Use the wiring diagrams I provided for understanding the correct connections.

I RECOMMEND YOU FOLLOW THE ARDUPILOT SETUP GUIDE IF YOU NEED MORE INFO ON SETUP INSTRUCTIONS.

*NOTE: It is a good idea to keep the antenna away from any area with higher power running through it for best operation.

Step 6: Test Out the Drone and Configure It

You can now attach the props and battery and test the drone. It is normal for it to not fly good at first; this is because you must configure the PIDs. If you are using telemetry, this is easy: land your drone and change the PIDs (pictured), then takeoff and test it again until it flies how you like.

This can be a time consuming process; you can use auto-tune but I have never used it.

You should have already completed the settings in the initial setup tab

Step 7: Setup the Walabot and Raspberry Pi

You will need at least an 8Gb flash drive and Raspbian installed on it. Plug it into the HDMI (if you have another Raspberry Pi you can configure the stuff on it first, then switch the SD card to the Pi Zero) and wait for the GUI to start. Open the web browser and download the Walabot SDK for Raspberry Pi. Once it is installed, you will need to add the following lines to the *boot/config.txt* file to support the Walabot.

```
safe_mode_gpio=4
max_usb_current=1
```

Reboot the device now.

Once it reboots, we will test out the "walabotdrone.py" script [DOWNLOAD THE FILE] in the GUI (it doesn't matter where the file is located).

**HINT: You will want to make it so it starts the script automatically. Unfortunately, I cannot show how to do this because there are so many different OS versions for Raspberry Pi and they are all different.

Step 8: Attach the Walabot and Raspberry Pi to the Drone

I drilled 4 holes in the Walabot case and attached standoffs. Then on top of the standoffs, I attached the Raspberry Pi Zero. I used a transformer case as the mount for the drone. It stuck well to the Walabot magnet. I used hot glue and super glue to ensure the mount wouldn't come undone. Make sure you have the 5V, GND, and TV pins in place before you put everything together, it will make things easier for you.

WARNING: Be careful with the Walabot, you probably would be better to use gloves when handling it.

Step 9: Test Out the Walabot

Without flying the drone test out the Walabot and video transmitter. When it calibrates, point it towards the sky so there is nothing there, this will make it so the Walabot will detect everything when scanning. Make sure it works how you like and change the settings (yes, you still can use a keyboard and mouse) until it works how you like.

My results: The first image is just "static". The bottom where the colors are is my roof that I used for testing. The second picture is when I walked into the room where the Walabot was facing, you can see me as the red blob in the middle (it was facing downward).

Watch the video for a better visualization.

Step 10: Dome/Cover

I 3D-printed a dome for the hexacopter. I got the original design from thingiverse, but had to make some changes. The original was too small and I couldn't mount my GPS so that is why I edited the design. You can find the new design on my website.

WARNING: MAKE SURE THE DOME DOESN'T TOUCH THE FC (APM) - it could case vibrations resulting in a crash.

Step 11: Test Flight!

Do your first test flight. The PIDs may need to be re-tuned for the extra weight. Test it out first by walking under it; there should be a red blob. Now, **IF IT IS SAFE**, fly over a building and have somebody inside; see if you can see them. If you cannot, change some of your settings or re-calibrate it until they show up as a red blob. *Note: there will be a lot of rubbish because of other things in the building, you will need to go by your best judgement to configure it.

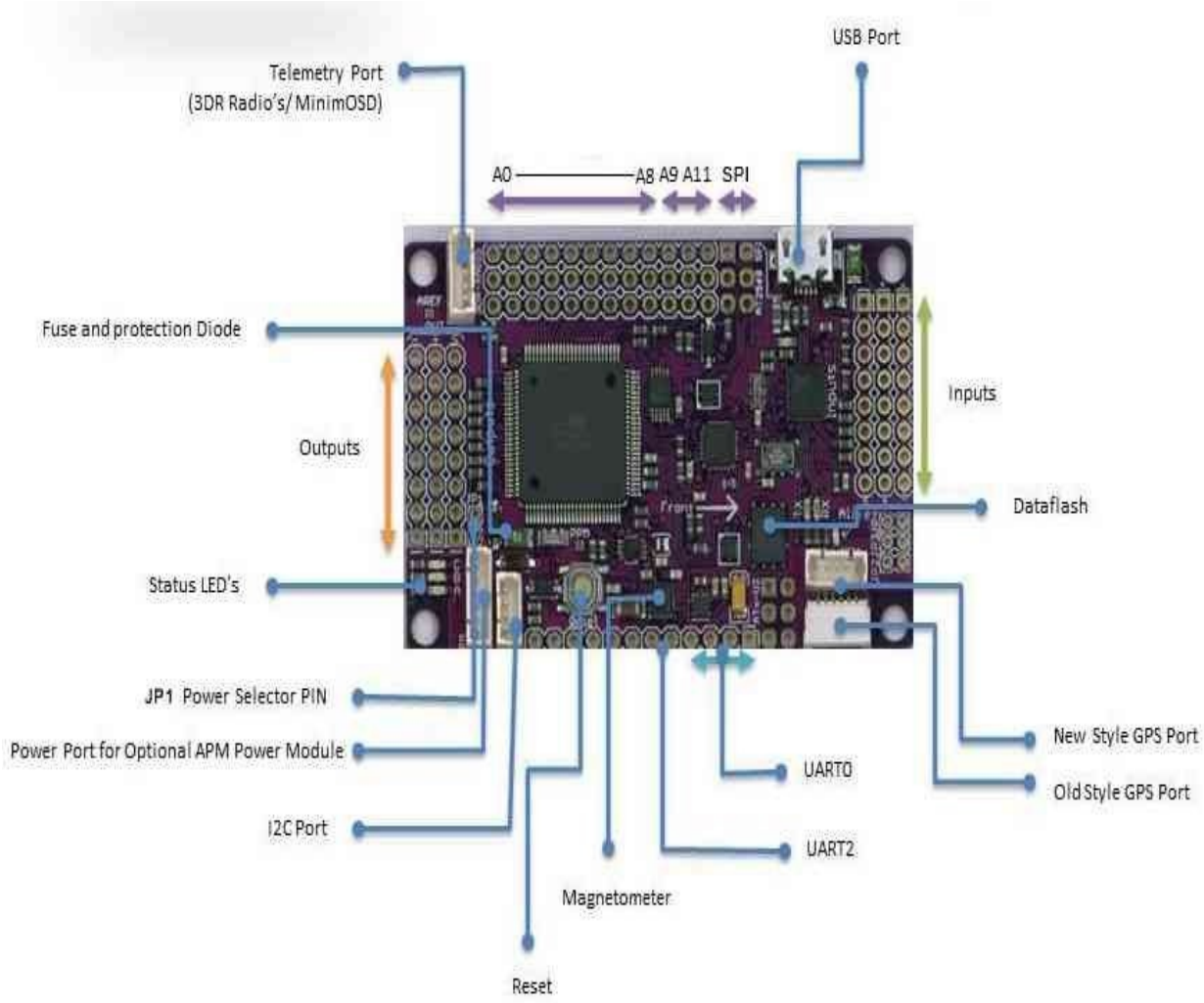
Custom parts and enclosures

Thingiverse
http://www.thingiverse.com/thing:973468

CAD file on thingiverse.com

Schematics

Sample diagram showing wiring



Code

Drone Walabot CodePython

```
from __future__ import print_function
from sys import platform
from os import system
from imp import load_source
from os.path import join
import time
modulePath = join('/usr', 'share', 'walabot', 'python', 'WalabotAPI.py')
wlbt = load_source('WalabotAPI', modulePath)
wlbt.Init()
try: # for Python 2
    import Tkinter as tk
except ImportError: # for Python 3
    import tkinter as tk
try: # for Python 2
    range = xrange
except NameError:
    pass



COLORS = [
    "000083", "000087", "00008B", "00008F", "000093", "000097", "00009B",
    "00009F", "0000A3", "0000A7", "0000AB", "0000AF", "0000B3", "0000B7",
    "0000BB", "0000BF", "0000C3", "0000C7", "0000CB", "0000CF", "0000D3",
    "0000D7", "0000DB", "0000DF", "0000E3", "0000E7", "0000EB", "0000EF",
    "0000F3", "0000F7", "0000FB", "0000FF", "0003FF", "0007FF", "000BFF",
    "000FFF", "0013FF", "0017FF", "001BFF", "001FFF", "0023FF", "0027FF",
    "002BFF", "002FFF", "0033FF", "0037FF", "003BFF", "003FFF", "0043FF",
    "0047FF", "004BFF", "004FFF", "0053FF", "0057FF", "005BFF", "005FFF",
    "0063FF", "0067FF", "006BFF", "006FFF", "0073FF", "0077FF", "007BFF",
    "007FFF", "0083FF", "0087FF", "008BFF", "008FFF", "0093FF", "0097FF",
    "009BFF", "009FFF", "00A3FF", "00A7FF", "00ABFF", "00AFFF", "00B3FF",
    "00B7FF", "00BBFF", "00BFFF", "00C3FF", "00C7FF", "00CBFF", "00CFFF",
    "00D3FF", "00D7FF", "00DBFF", "00DFFF", "00E3FF", "00E7FF", "00EBFF",
    "00EFFF", "00F3FF", "00F7FF", "00FBFF", "00FFFF", "03FFFB", "07FFF7",
    "0BFFF3", "0FFFEF", "13FFEB", "17FFE7", "1BFFE3", "1FFFDF", "23FFDB",
    "27FFD7", "2BFFD3", "2FFFCF", "33FFCB", "37FFC7", "3BFFC3", "3FFFBF",
    "43FFBB", "47FFB7", "4BFFB3", "4FFFAF", "53FFAB", "57FFA7", "5BFFA3",
    "5FFF9F", "63FF9B", "67FF97", "6BFF93", "6FFF8F", "73FF8B", "77FF87",
    "7BFF83", "7FFF7F", "83FF7B", "87FF77", "8BFF73", "8FFF6F", "93FF6B",
    "97FF67", "9BFF63", "9FFF5F", "A3FF5B", "A7FF57", "ABFF53", "AFFF4F",
    "B3FF4B", "B7FF47", "BBFF43", "BFFF3F", "C3FF3B", "C7FF37", "CBFF33",
    "CFFF2F", "D3FF2B", "D7FF27", "DBFF23", "DFFF1F", "E3FF1B", "E7FF17",
    "EBFF13", "EFFF0F", "F3FF0B", "F7FF07", "FBFF03", "FFFF00", "FFFB00",
    "FFF700", "FFF300", "FFEF00", "FFEB00", "FFE700", "FFE300", "FFDF00",
    "FFDB00", "FFD700", "FFD300", "FFCF00", "FFCB00", "FFC700", "FFC300",
```

```
    "FFBF00", "FFBB00", "FFB700", "FFB300", "FFAF00", "FFAB00", "FFA700",
    "FFA300", "FF9F00", "FF9B00", "FF9700", "FF9300", "FF8F00", "FF8B00",
    "FF8700", "FF8300", "FF7F00", "FF7B00", "FF7700", "FF7300", "FF6F00",
    "FF6B00", "FF6700", "FF6300", "FF5F00", "FF5B00", "FF5700", "FF5300",
    "FF4F00", "FF4B00", "FF4700", "FF4300", "FF3F00", "FF3B00", "FF3700",
    "FF3300", "FF2F00", "FF2B00", "FF2700", "FF2300", "FF1F00", "FF1B00",
    "FF1700", "FF1300", "FF0F00", "FF0B00", "FF0700", "FF0300", "FF0000",
    "FB0000", "F70000", "F30000", "EF0000", "EB0000", "E70000", "E30000",
    "DF0000", "DB0000", "D70000", "D30000", "CF0000", "CB0000", "C70000",
    "C30000", "BF0000", "BB0000", "B70000", "B30000", "AF0000", "AB0000",
    "A70000", "A30000", "9F0000", "9B0000", "970000", "930000", "8F0000",
    "8B0000", "870000", "830000", "7F0000"]

APP_X, APP_Y = 0, 0 # location of top-left corner of window
CANVAS_LENGTH = 650 # in pixels



class RawImageApp(tk.Frame):
    """ Main app class.
    """

    def __init__(self, master):
    """ Init the GUI components and the Walabot API.
    """
    tk.Frame.__init__(self, master)
    self.canvasPanel = CanvasPanel(self)
    self.wlbtPanel = WalabotPanel(self)
    self.ctrlPanel = ControlPanel(self)
    self.canvasPanel.pack(side=tk.RIGHT, anchor=tk.NE)
    self.wlbtPanel.pack(side=tk.TOP, anchor=tk.W, fill=tk.BOTH, pady=10)
    self.ctrlPanel.pack(side=tk.TOP, anchor=tk.W, fill=tk.BOTH, pady=10)
    self.wlbt = Walabot()
    self.initAppLoop()



    def initAppLoop(self):
    if self.wlbt.isConnected():
    self.ctrlPanel.statusVar.set('STATUS_CONNECTED')
    self.update_idletasks()
    params = self.wlbtPanel.getParams()
    self.wlbt.setParams(*params)
    self.wlbtPanel.setParams(*self.wlbt.getArenaParams())
    if not params[4]: # equals: if not mtiMode
    self.ctrlPanel.statusVar.set('STATUS_CALIBRATING')
    self.update_idletasks()
    self.wlbt.calibrate()
    self.lenOfPhi, self.lenOfR = self.wlbt.getRawImageSliceDimensions()
    self.canvasPanel.setGrid(self.lenOfPhi, self.lenOfR)
    self.wlbtPanel.changeEntriesState('disabled')
    self.loop()
```

```
    else:
    self.ctrlPanel.statusVar.set('STATUS_DISCONNECTED')

    def loop(self):
    self.ctrlPanel.statusVar.set('STATUS_SCANNING')
    rawImage = self.wlbt.triggerAndGetRawImageSlice()
    self.canvasPanel.update(rawImage, self.lenOfPhi, self.lenOfR)
    self.ctrlPanel.fpsVar.set(self.wlbt.getFps())
    self.cyclesId = self.after_idle(self.loop)


class WalabotPanel(tk.LabelFrame):

    class WalabotParameter(tk.Frame):
    """ The frame that sets each Walabot parameter line.
    """

    def __init__(self, master, varVal, minVal, maxVal, defaultVal):
    """ Init the Labels (parameter name, min/max value) and entry.
    """
    tk.Frame.__init__(self, master)
    tk.Label(self, text=varVal).pack(side=tk.LEFT, padx=(0, 5), pady=1)
    self.minVal, self.maxVal = minVal, maxVal
    self.var = tk.StringVar()
    self.var.set(defaultVal)
    self.entry = tk.Entry(self, width=7, textvariable=self.var)
    self.entry.pack(side=tk.LEFT)
    self.var.trace("w", lambda a, b, c, var=self.var: self.validate())
    txt = "[{}, {}]".format(minVal, maxVal)
    tk.Label(self, text=txt).pack(side=tk.LEFT, padx=(5, 20), pady=1)

    def validate(self):
    """ Checks that the entered value is a valid number and between
    the min/max values. Change the font color of the value to red
    if False, else to black (normal).
    """
    num = self.var.get()
    try:
    num = float(num)
    if num < self.minVal or num > self.maxVal:
    self.entry.config(fg='#'+COLORS[235])
    return
    self.entry.config(fg='gray1')
    except ValueError:
    self.entry.config(fg='#'+COLORS[235])
    return

    def get(self):
    """ Returns the entry value as a float.
    """
```

```
return float(self.var.get())

def set(self, value):
""" Sets the entry value according to a given one.
"""
self.var.set(value)

def changeState(self, state):
""" Change the entry state according to a given one.
"""
self.entry.configure(state=state)

class WalabotParameterMTI(tk.Frame):
""" The frame that control the Walabot MTI parameter line.
"""

def __init__(self, master):
""" Init the MTI line (label, radiobuttons).
"""
tk.Frame.__init__(self, master)
tk.Label(self, text="MTI ").pack(side=tk.LEFT)
self.mtiVar = tk.IntVar()
self.mtiVar.set(0)
self.true = tk.Radiobutton(
self, text="True", variable=self.mtiVar, value=2)
self.false = tk.Radiobutton(
self, text="False", variable=self.mtiVar, value=0)
self.true.pack(side=tk.LEFT)
self.false.pack(side=tk.LEFT)

def get(self):
""" Returns the value of the pressed radiobutton.
"""
return self.mtiVar.get()

def set(self, value):
""" Sets the pressed radiobutton according to a given value.
"""
self.mtiVar.set(value)

def changeState(self, state):
""" Change the state of the radiobuttons according to a given one.
"""
self.true.configure(state=state)
self.false.configure(state=state)

def __init__(self, master):
tk.LabelFrame.__init__(self, master, text='Drone-Walabot Config (By FunguyPro)')
self.rMin = self.WalabotParameter(self, 'R Min', 1, 1000, 10.0)
self.rMax = self.WalabotParameter(self, 'R Max', 1, 1000, 500.0)
```

```
        self.rRes = self.WalabotParameter(self, 'R Res', 0.1, 10, 2.0)
        self.tMin = self.WalabotParameter(self, 'Theta Min', -90, 90, -45.0)
        self.tMax = self.WalabotParameter(self, 'Theta Max', -90, 90, 45.0)
        self.tRes = self.WalabotParameter(self, 'Theta Res', 0.1, 10, 10.0)
        self.pMin = self.WalabotParameter(self, 'Phi Min', -90, 90, -45.0)
        self.pMax = self.WalabotParameter(self, 'Phi Max', -90, 90, 45.0)
        self.pRes = self.WalabotParameter(self, 'Phi Res', 0.1, 10, 2.0)
        self.thld = self.WalabotParameter(self, 'Threshold', 0.1, 100, 15.0)
        self.mti = self.WalabotParameterMTI(self)
        self.parameters = (
        self.rMin, self.rMax, self.rRes, self.tMin, self.tMax, self.tRes,
        self.pMin, self.pMax, self.pRes, self.thld, self.mti)
        for param in self.parameters:
        param.pack(anchor=tk.W)

        def getParams(self):
        rParams = (self.rMin.get(), self.rMax.get(), self.rRes.get())
        tParams = (self.tMin.get(), self.tMax.get(), self.tRes.get())
        pParams = (self.pMin.get(), self.pMax.get(), self.pRes.get())
        thldParam, mtiParam = self.thld.get(), self.mti.get()
        return rParams, tParams, pParams, thldParam, mtiParam

        def setParams(self, rParams, thetaParams, phiParams, threshold):
        self.rMin.set(rParams[0])
        self.rMax.set(rParams[1])
        self.rRes.set(rParams[2])
        self.tMin.set(thetaParams[0])
        self.tMax.set(thetaParams[1])
        self.tRes.set(thetaParams[2])
        self.pMin.set(phiParams[0])
        self.pMax.set(phiParams[1])
        self.pRes.set(phiParams[2])
        self.thld.set(threshold)



        def changeEntriesState(self, state):
        for param in self.parameters:
        param.changeState(state)



class ControlPanel(tk.LabelFrame):
    """ This class is designed to control the control area of the app.
    """


    def __init__(self, master):
    """ Initialize the buttons and the data labels.
    """
    tk.LabelFrame.__init__(self, master, text='Control Panel')
    self.buttonsFrame = tk.Frame(self)
    self.runButton, self.stopButton = self.setButtons(self.buttonsFrame)
```

```
self.statusFrame = tk.Frame(self)
self.statusVar = self.setVar(self.statusFrame, 'APP_STATUS', '')
self.errorFrame = tk.Frame(self)
self.errorVar = self.setVar(self.errorFrame, 'EXCEPTION', '')
self.fpsFrame = tk.Frame(self)
self.fpsVar = self.setVar(self.fpsFrame, 'FRAME_RATE', 'N/A')
self.buttonsFrame.grid(row=0, column=0, sticky=tk.W)
self.statusFrame.grid(row=1, columnspan=2, sticky=tk.W)
self.errorFrame.grid(row=2, columnspan=2, sticky=tk.W)
self.fpsFrame.grid(row=3, columnspan=2, sticky=tk.W)



def setButtons(self, frame):
""" Initialize the 'Start' and 'Stop' buttons.
"""
runButton = tk.Button(frame, text='Start', command=self.start)
stopButton = tk.Button(frame, text='Stop', command=self.stop)
runButton.grid(row=0, column=0)
stopButton.grid(row=0, column=1)

return runButton, stopButton



def setVar(self, frame, varText, default):
""" Initialize the data frames.
"""
strVar = tk.StringVar()
strVar.set(default)
tk.Label(frame, text=(varText).ljust(12)).grid(row=0, column=0)
tk.Label(frame, textvariable=strVar).grid(row=0, column=1)
return strVar

def start(self):
""" Applied when 'Start' button is pressed. Starts the Walabot and
the app cycles.
"""


self.master.initAppLoop()

def stop(self):
""" Applied when 'Stop' button in pressed. Stops the Walabot and the
app cycles.
"""
if hasattr(self.master, 'cyclesId'):
self.master.after_cancel(self.master.cyclesId)
self.master.wlbtPanel.changeEntriesState('normal')
self.master.canvasPanel.reset()
self.statusVar.set('STATUS_IDLE')
```

```
class CanvasPanel(tk.LabelFrame):
    """ This class is designed to control the canvas area of the app.
    """

    def __init__(self, master):
    """ Initialize the label-frame and canvas.
    """
    tk.LabelFrame.__init__(self, master, text='Raw Image Slice: R / Phi')
    self.canvas = tk.Canvas(
    self, width=CANVAS_LENGTH, height=CANVAS_LENGTH)
    self.canvas.pack()
    self.canvas.configure(background='#'+COLORS[0])



    def setGrid(self, sizeX, sizeY):
    """ Set the canvas components (rectangles), given the size of the axes.
    Arguments:
    sizeX Number of cells in Phi axis.
    sizeY Number of cells in R axis.
    """
    recHeight, recWidth = CANVAS_LENGTH/sizeX, CANVAS_LENGTH/sizeY
    self.cells = [[
    self.canvas.create_rectangle(
    recWidth*col, recHeight*row,
    recWidth*(col+1), recHeight*(row+1),
    width=0)
    for col in range(sizeY)] for row in range(sizeX)]

    def update(self, rawImage, lenOfPhi, lenOfR):
    """ Updates the canvas cells colors acorrding to a given rawImage
    matrix and it's dimensions.
    Arguments:
    rawImage A 2D matrix contains the current rawImage slice.
    lenOfPhi Number of cells in Phi axis.
    lenOfR Number of cells in R axis.
    """
    for i in range(lenOfPhi):
    for j in range(lenOfR):
    self.canvas.itemconfigure(
    self.cells[lenOfPhi-i-1][j],
    fill='#'+COLORS[rawImage[i][j]])

    def reset(self):
    """ Deletes all the canvas components (colored rectangles).
    """
    self.canvas.delete('all')
```

```
class Walabot:
    """ Control the Walabot using the Walabot API.
    """

    def __init__(self):
    """ Init the Walabot API.
    """
    self.wlbt = wlbt
    self.wlbt.Init()
    self.wlbt.SetSettingsFolder()

    def isConnected(self):
    """ Try to connect the Walabot device. Return True/False accordingly.
    """
    try:
    self.wlbt.ConnectAny()
    except self.wlbt.WalabotError as err:
    if err.code == 19: # "WALABOT_INSTRUMENT_NOT_FOUND"
    return False
    else:
    raise err
    return True

    def setParams(self, r, theta, phi, threshold, mti):
    """ Set the arena Parameters according given ones.
    """
    self.wlbt.SetProfile(self.wlbt.PROF_SENSOR)
    self.wlbt.SetArenaR(*r)
    self.wlbt.SetArenaTheta(*theta)
    self.wlbt.SetArenaPhi(*phi)
    self.wlbt.SetThreshold(threshold)
    self.wlbt.SetDynamicImageFilter(mti)
    self.wlbt.Start()

    def getArenaParams(self):
    """ Returns the Walabot parameters from the Walabot SDK.
    Returns:
    params rParams, thetaParams, phiParams, threshold as
    given from the Walabot SDK.
    """
    rParams = self.wlbt.GetArenaR()
    thetaParams = self.wlbt.GetArenaTheta()
    phiParams = self.wlbt.GetArenaPhi()
    threshold = self.wlbt.GetThreshold()
    return rParams, thetaParams, phiParams, threshold

    def calibrate(self):
    """ Calibrates the Walabot.
    """
```

```
        self.wlbt.StartCalibration()
        while self.wlbt.GetStatus()[0] == self.wlbt.STATUS_CALIBRATING:
        self.wlbt.Trigger()

        def getRawImageSliceDimensions(self):
        """ Returns the dimensions of the rawImage 2D list given from the
        Walabot SDK.
        Returns:
        lenOfPhi Num of cells in Phi axis.
        lenOfR Num of cells in Theta axis.
        """
        return self.wlbt.GetRawImageSlice()[1:3]

        def triggerAndGetRawImageSlice(self):
        """ Returns the rawImage given from the Walabot SDK.
        Returns:
        rawImage A rawImage list as described in the Walabot docs.
        """
        self.wlbt.Trigger()
        return self.wlbt.GetRawImageSlice()[0]

        def getFps(self):
        """ Returns the Walabot current fps as given from the Walabot SDK.
        Returns:
        fpsVar Number of frames per seconds.
        """
        return int(self.wlbt.GetAdvancedParameter('FrameRate'))



def rawImage():
    """ Main app function. Init the main app class, configure the window
    and start the mainloop.
    """
    root = tk.Tk()
    root.title('FunguyPro - Drone Walabot')
    RawImageApp(root).pack(side=tk.TOP, fill=tk.BOTH, expand=True)
    root.geometry("+{}+{}".format(APP_X, APP_Y)) # set window location
    root.update()
    root.minsize(width=root.winfo_reqwidth(), height=root.winfo_reqheight())
    root.mainloop()



if __name__ == '__main__':
    rawImage()
```

To Be Continued To Part 3...