Make Your Own PiDoorbell!

Handout for PyCon, Montreal, April 2014

Rupa Dachere and Akkana Peck, co-presenters, for CodeChix

Parts List:

- Raspberry Pi (you provide) and power supply (preferably 1 amp or better)
- SD card loaded with Raspbian and other required software
- HC-SR04 ultrasonic rangefinder
- Solderless breadboard
- Jumper wires: 4 male-female, 2 male-male
- Resistors: 1 360 Ω (or optionally 2), 1 470 Ω
- LED
- USB to TTL Serial Cable for Rpi, OR ethernet cable (for RPi-laptop communication)

Optional:

- Camera for RPi: either USB webcam or Pi Camera
- USB wi-fi dongle
- cell phone (for text alerts)

Software

Installed on the Raspberry Pi: Raspbian (or comparable distro), RPi.GPIO, git, fswebcam; PiDoorbell repository, installed via: git clone https://github.com/codechix/PiDoorbell

On your laptop: some means of communicating with the RPi. Either a program like screen plus drivers for a USB-serial device (most Linux systems and some Mac systems have this already), or a way of using IP masquerading over an ethernet cable.

First step: conect to the Raspberry Pi

Without a monitor and USB keyboard, there are two ways of talking to a Raspberry Pi: over a serial cable, or over ethernet.

Using a serial cable lets you see boot messages and works even when the Pi has a networking problem. Using ethernet takes more setup, but may be easier on some Macs.

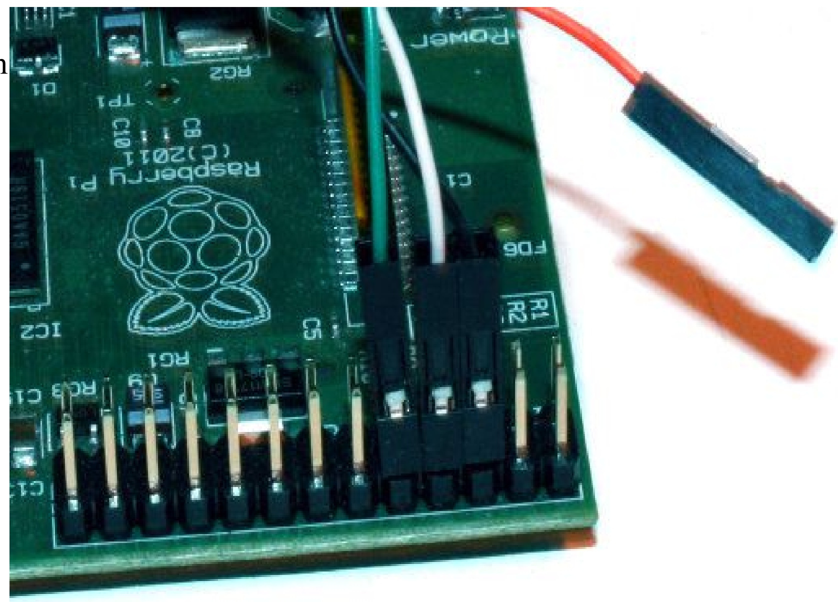
Option 1: Connect with Serial Cable

Linux already comes with a driver that will work for the serial cable.

On Mac, you may need to download and install a driver first. You can get them here: http://sourceforge.net/projects/osx-pl2303/

If using Lion or Mountain Lion or later OS X, try this driver: http://changux.co/osx-installer-to-pl2303-serial-usb-on-osx-lio

Accept all defaults when prompted by the installer.



Now wire up the serial cable as shown. The black wire goes on the third pin from the corner; the white wire goes next to that, and then the green.

DO NOT USE THE RED WIRE!
If you do, the Pi will try to get power from that wire (over the USB) which conflicts with getting its power from its micro-USB power plug. Just leave that wire hanging free.

Once the serial cable is connected to the Pi, plug the USB end into your laptop. Then run this command:

```
Linux: screen /dev/ttyUSB0 115200
```

```
Mac: screen /dev/cu.PL<something> 115200
```

The <*something*> part (on Mac) will vary, so use the Tab key after PL and let it autocomplete. If you have trouble, you might have better luck by running Linux under VMFusion. Make sure you select the "connect to Linux" option when connecting the USB to the Mac.

To exit screen: C-A C-\ (where C- is Control on Linux, Command on Mac).

You can use any other program you prefer for talking to a serial port: minicom, etc. The device ID may be something other than /dev/ttyUSB0, depending on your OS.

Once you have a connection established, you should be able to plug in the Raspberry Pi's power supply and see boot messages.

Option 2: Connect with Ethernet Cable

Connect the two ends of the ethernet cable to the Pi and your laptop.

Configure your laptop's ethernet address to be 192.168.2.1/24:
On Linux, type: `ifconfig eth0 192.168.2.1/24 up`
On Mac, use 'network' settings in 'System Preferences' and configure Manual IP address

Your Pi has been pre-configured with a static IP address on its ethernet port. Note the IP address, then try to log in to your Raspberry Pi using ssh:

```
ssh pi@<your-pi's-IP-address>
```

Of course, you can use an ssh client like putty if you prefer.
You can even port-forward X to run remote windows (ssh -X pi@<ip address>) if you're already familiar with how to do that, but that's beyond the scope of this tutorial.

Either Way: Now Set Up Wi-Fi

The Raspberry Pi uses the file */etc/network/interfaces* to configure networking.
You'll need to edit that file as root (using sudo). You can use any editor you're familiar with, but if you aren't familiar with any on the Pi, pico is probably the easiest to work with.
First make a backup of it:
`sudo cp /etc/network/interfaces /etc/network/interfaces.sav`
Then edit it:
`sudo pico /etc/network/interfaces`
There should be a stanza for eth0 already there. Leave it in place, go to the end of the file (add a blank line if needed), then type one of these, depending on the conference wi-fi we're using:

Open wi-fi (no password):
```
iface wlan0 inet dhcp
    wireless-essid "network-name-goes-here"
    wireless-mode managed
```

WPA/Secure wi-fi, with password:
```
auto wlan0
allow-hotplug wlan0
iface wlan0 inet dhcp
    wpa-ssid “network name”
    wpa-psk “passphrase”
```

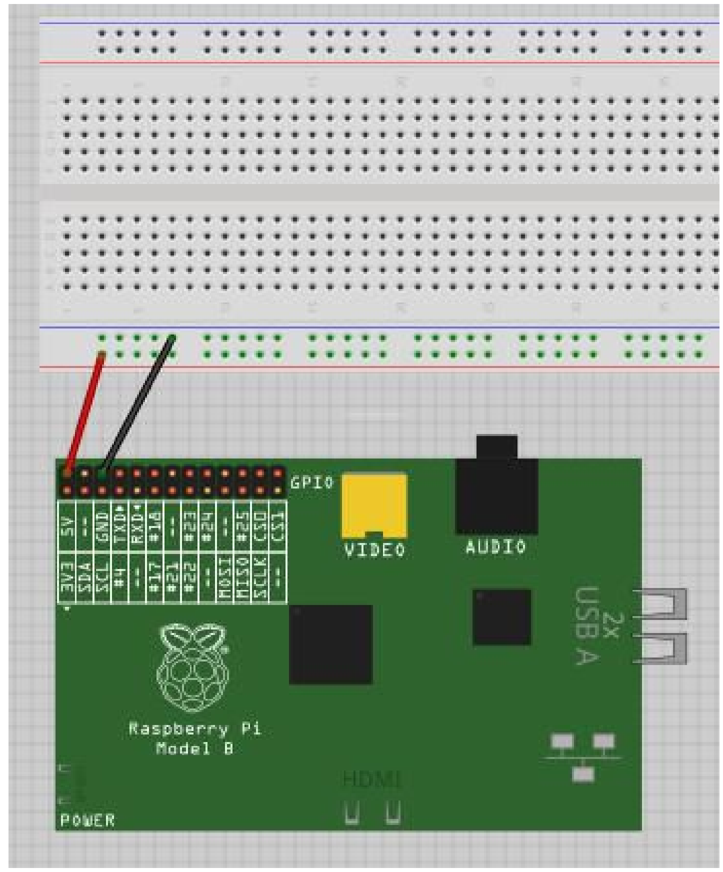
Save the file and exit the editor. Then, back in the Pi's shell, restart networking:
`sudo service networking stop; sudo service networking start`

Now test networking by trying to ping an outsite site:
`ping www.codechix.org`

If that doesn't work, you may need to stop andstart networking again, or possibly even reboot.

Using GPIO: Blinking an LED

1. Wire Power and Ground



You'll be using the Raspberry Pi's GPIO header. The first step is to wire up power and ground to your breadboard.

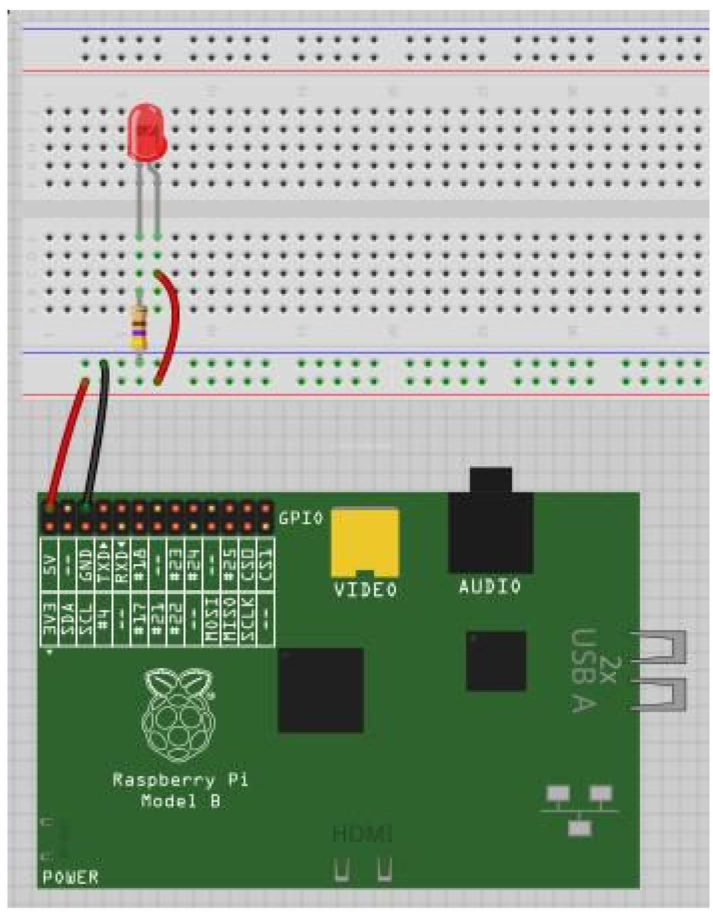
On a solderless breadboard, the strips along the edges of the board – labeled with red and blue – are generally used for power (red) and ground (blue).

On the Raspberry Pi's GPIO header, the pin in the corner gives you 5 V power – the power supply the RP is getting from its power input.

The third pin from the corner, in the outer row, is ground. (There are several other pins that can provide ground, if you need more than one. But since we're using a breadboard that's not a problem.)

Use male-female jumpers the female end of each jumper fits over the Pi's GPIO pin, and the male end plugs into the breadboard. If possible, use red for power and black for ground.

2. Wire up the LED



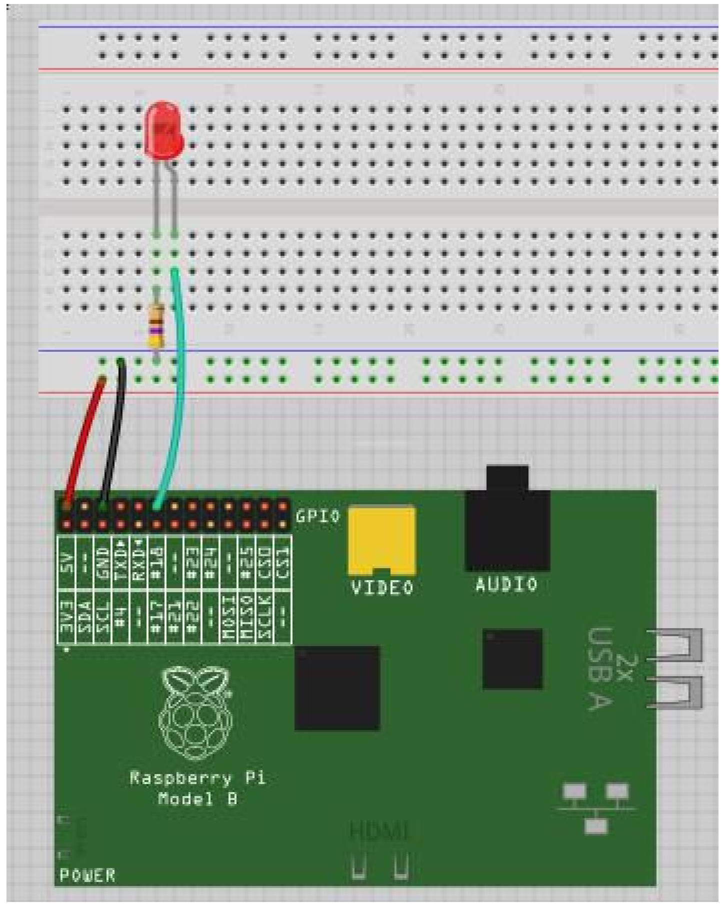
An LED has one leg longer than the other. The longer leg should be connected to power, the shorter one to ground. But you need a small value resistor in the circuit so you don't burn out the LED.

Use your 360 Ω resistor for this. It's the one with red and orange bands.

In the middle part of the breadboard (between the power/ground strips), each line of 5 is wired together. So with this wired up, the current flows from the Pi to the red power strip, to the long pin of the LED, through the LED, through the resistor, out to the ground (blue) power strip of the breadboard, and out to the Pi's ground pin.

If your Pi is plugged in and running, the LED should light up as soon as you finish connecting the wires.

3. Connect the LED to a pin you can control



It's not much fun to have an LED always on. So instead of connecting the long pin to the breadboard's power rail, connect it to pin 18 of the Pi's GPIO header.

The pin numbering on the GPIO header doesn't make any sense, so you should always check a reference to figure out which pin is which. And worse, there are two different incompatible ways of numbering the pins. We'll be using the “BCM” numbering system. BCM pin 18 is the 6th pin from the corner on the outer row.

The LED should be off now. Try running the led.py script. You'll have to run it as root: only root can access the Raspberry Pi's GPIO pins.

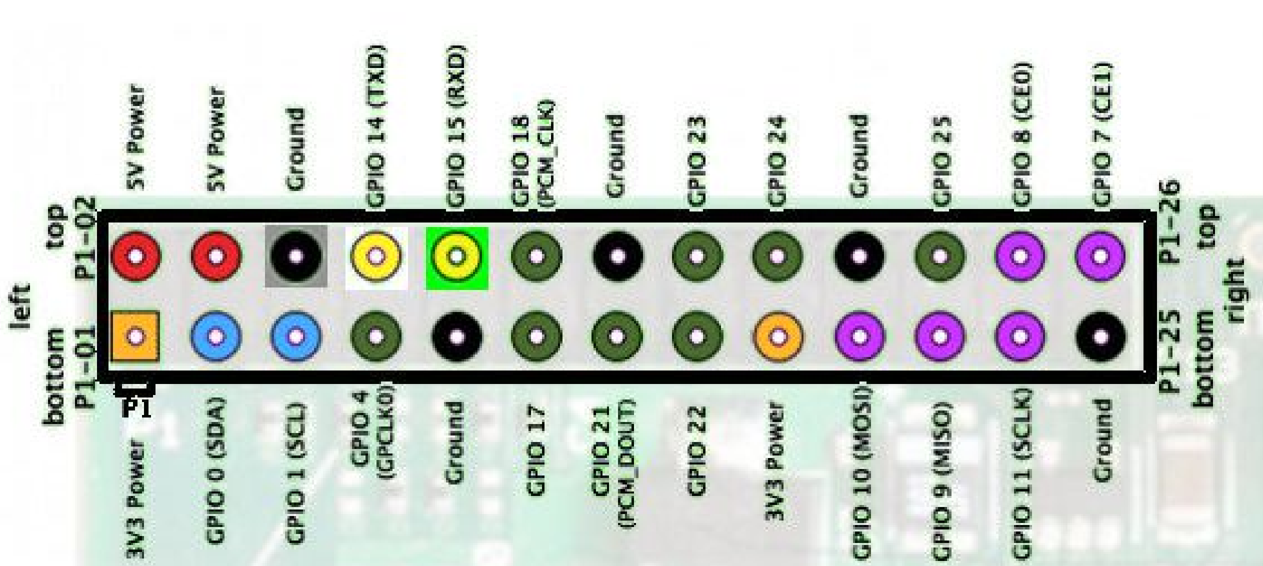
```
sudo python led.py
```

Your LED should start blinking.

You can edit *led.py* and adjust the sleep times.

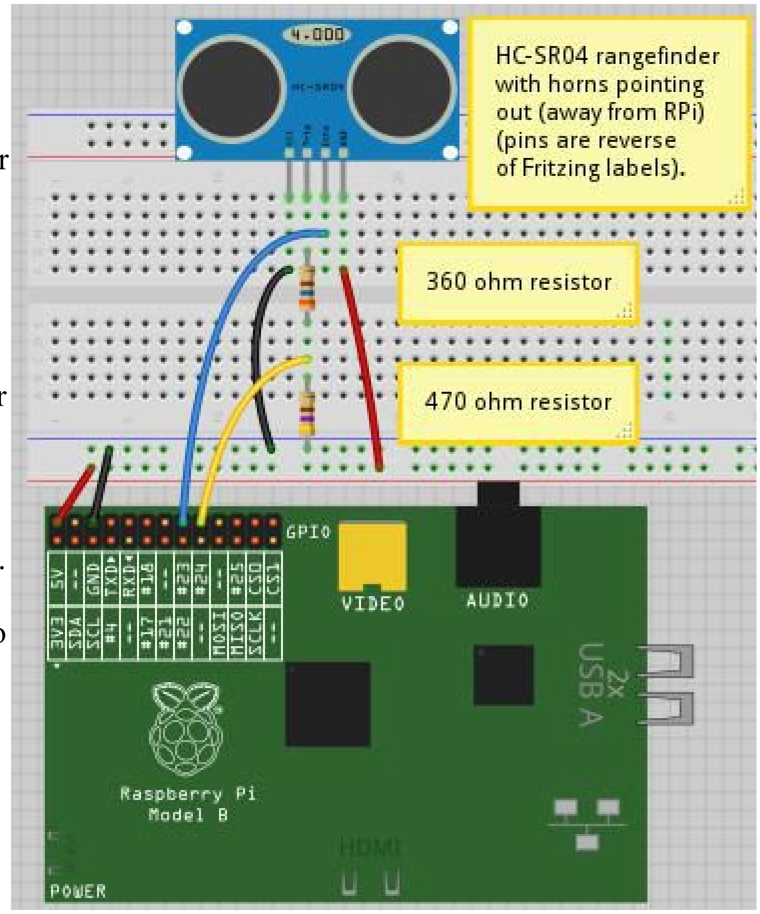
GPIO Pin Diagram

Here's a diagram showing all the pins on the Raspberry Pi's GPIO header.
Warning: some pins may vary between different Pi versions.



Hooking up the sonar rangefinder

The HC-SR04 distance sensor emits a sound pulse, then measures the time it takes for the sound to return. Same principle as bats, dolphins or submarines.



The HC-SR04 has 4 pins. If you point the sonar horns away from you, the pins are, from left to right: Gnd (ground), Echo, Trigger, VCC (5 V power).

We'll connect power and ground, and wire Trigger to BCM pin 23, Echo to pin 24. Those are the 5th and 6th pins from the audio connector end of the GPIO header on the outer row.

There's one snag: the HC-SR04 runs on 5V, while the Raspberry Pi's GPIO pins can only handle 3V (even though it has a 5V power pin). We can compensate for that by using two resistors as a "voltage divider" to convert 5V to a level the Pi can handle.

Remember, your 360 Ω resistor is the one with the orange and blue band. **Try not to get the resistors backwards, and be sure to connect the 470 Ω to ground, not power. Too much voltage on a GPIO pin could potentially damage your Pi!**

Controlling the Rangefinder

To control the HC-SR04, we need to turn the Trigger line high briefly, then turn it low again. The Echo line goes low. When the sound pulse returns, the Echo line goes high. So we just need to time how long it takes to go from low to high.

The HC_SR04.py script does just that. When you have everything wired up, run it as root to see if everything's working:

```
sudo python HC_SR04.py
```

If everything's okay, you'll see it print repeated lines measuring the distance in front of it. Try putting your hand or some other object in front of it and see if the distance changes.

If you don't see repeated printouts, double-check your wiring. It's amazingly easy to find yourself one pin off, or with the Trigger and Echo pins reversed.

Once all the rangefinders are working, we can move on to the real PiDoorbell logic, in *pidoorbell-recognizer-gpio.py.*

PiDoorbell Recognizer

pidoorbell-recognizer-gpio.py is the script that puts everything together.

It needs to be run as root (because it needs access to GPIO). It can take several arguments: `pidoorbell-recognizer-gpio.py -h` will list them all.

For initial testing, run it like this:

```
sudo python pidoorbell-recognizer-gpio.py -i -local
```

-i specifies interactive mode, so you can see verbose messages about what it's doing. -local keeps it from using any web services while you're still getting everything hooked up.

In interactive mode, you should see a steady stream of messages as it reads distances from the rangefinder. Try putting your hand or another object in front of the sensor to trigger it.

By default, the script considers anything from 0 to 30 inches to be within its target range, and an object has to remain there for at least 5 seconds before it reacts. These values are set near the beginning of the script and are easy to change.

Taking photos or video of your visitors

The recognizer can take a photo if you have a camera on your Raspberry Pi. That can be either a USB webcam that works with *fswebcam* and *ffmpeg*, or the Raspberry Pi camera module attached to your Pi.

Ffmpeg and fswebcam are optional software, not installed by default, so they need to be installed:

```
apt-get install fswebcam libv4l-0 v4l-utils
```

You may also need these packages:

```
apt-get install uvcvideo uvccapture
```

The Pi camera module doesn't use the normal Linux video framework, so it needs its own programs: *raspistill* and *raspivid* (which should already be installed as part of package libraspberrypi-bin), or, ideally, a Python module called *picamera* which isn't installed by default:

```
apt-get install python-picamera
```

By default, the recognizer will try to take a still photo using fswebcam if it finds a USB camera, or using the pi camera module if one is installed. With a USB camera, you can shoot a 10-second video instead (video isn't supported yet for the pi camera) by passing `-pic_mode 1` as an argument to *pidoorbell-recognizer-gpio.py*.

These photos should be created in the dropbox-pidoorbell subdirectory of your source directory (where you're running the script). You may need to create that directory.

Adding Dropbox

Open an account at *https://www.dropbox.com/home*

Get your API Key and Secret at: [*https://www.dropbox.com/developers/apply?cont=/developers/apps*](https://www.dropbox.com/developers/apply?cont=/developers/apps)

You can use an App Name like *pidoorbell-yourname*, or whatever you like.

Run *dropbox_uploader.sh*. You will be asked your app key and secret.

Now test the uploader:

```
$ ./dropbox_uploader.sh
# App key:
<copy here from the website>
# App secret:
<copy here from the website>
# Permission type, App folder or Full Dropbox [a/f]: a
> Token request... OK
```

Then open the following URL in your browser, and allow Dropbox Uploader to access your DropBox folder:

https://www2.dropbox.com/1/oauth/authorize?oauth_token=8NGlM1Cj8205nC9q

(this is a sample - you will receive a different URL for authentication when the script runs.)

AddingTwilio

Open an account at *https://www.twilio.com*

Your account SID and TOKEN are in your Account Settings

```
     ACCOUNT SID <>
     AUTH TOKEN <>
```

Update *sms_auth_info.py* file with twilio, dropbox, and twitter account info. It should look like:

```
# Twilio authentication credentials
# The values should be in double quotes, i.e., "xyz"

account_sid = <Your Twilio account secure ID>
auth_token = <Your Twilio authentication token>

#Twitter authentication credentials
# The values should be in singl quotes, i.e., 'xyz'

twitter_auth_key = <Your Twitter authentication key>
twitter_auth_secret = <Your Twitter authentication secret>
twitter_access_key = <Your Twitter access key>
twitter_access_secret = <Your Twitter access secret>

# Dropbox app key and secret from the Dropbox dev site
# The values should be in single quotes, i.e., 'xyz'

DB_APP_KEY = <Your Dropbox application key>
DB_APP_SECRET = <Your Dropbox application secret>
DB_ACCESS_TYPE = 'app_folder'

# going to store the access info in this file
# Update the following file with the Dropbox token
# as the first and only line in the file

DB_TOKENS = 'dropbox_token.txt'
```

Putting It All Together

With Dropbox and Twilio all set up, run the PiDoorbell code:

pi@raspberrypi ~ $ sudo python pidoorbell-recognizer-gpio.py -i

Then put your hand or an object in front of the sensor – you should see messages like:

****** DETECTED AN OBJECT AT --   9.44145122999 -- INCHES ******

Then it should take a photo (if you have a camera), upload it to Dropbox, and send you an SMS!

Appendix: References and Thanks

This handout, all the code, slides, and other materials for this workshop are available on GitHub:

https://github.com/codechix/PiDoorbell

This tutorial is run by members of Bay Area CodeChix, a nonprofit organization for promoting women software developers: http://CodeChix.org

A big thank-you to our volunteer T/As, without which we could not have done this tutorial: Deepa Karnad Dhurka, Lyz Krumbach, Serpil Bayraktar and Stuart.

And thanks to the Python Foundation, which provided a grant for the hardware kits as well as lots of help with logistics, and having this great conference in the first place!