

# Build a Burglar Alarm System

## Using the Raspberry Pi Computer

PICTURE OF ALARM SYSTEM



**Figure 1: The Raspberry Pi burglar alarm system completed**

## Acknowledgements

This project booklet was funded by ExploreSTEM. <http://www.explorestem.co.uk>

ExploreSTEM provides support to schools and communities within Essex, promoting Science, Technology, Engineering and Mathematics.

The roaming Raspberry Pi kits were funded by the Institution of Engineering and Technology, March 2014, from the IET Education Fund, kindly supported by The IET Essex committee.

<http://faraday.theiet.org>

<http://mycommunity.theiet.org/communities/home/253>

The IET (Institution of Engineering and Technology) is the professional home for engineers and technicians. Through the Education Fund and other programmes, they support and promote STEM activities in schools and communities.

This project booklet was written by David Whale, Thinking Binaries Ltd.

<http://www.thinkingbinaries.com>

Many of the photos in this booklet were kindly supplied by Sukkin Pang.

<http://www.skpang.co.uk>

You can always get the latest version of this booklet from this [github](#) page:

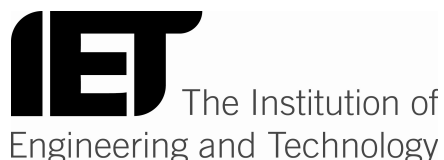
[https://github.com/explorestem/rpi\\_kit/alarm](https://github.com/explorestem/rpi_kit/alarm)

Please report any errors you find in this booklet via the *issues* link on the above github page.

This booklet is Copyright (c) 2014 Thinking Binaries Ltd.

It is licensed to ExploreSTEM Ltd to support the IET funded roaming Raspberry Pi kits in Essex.

If you would like to use this material in your own kits, please contact the copyright holders to ask permission first.



Printed in England by [ ]

Print version **DRAFT** 1 2 3 4 5 6 7 8 9

# 1. Introduction

## 1.1. How long will it take?

This project should take you between 45 minutes and 3 hours to complete, depending on how many of the tasks you decide to try.

## 1.2. What will I build?

In this project, you will build a working burglar alarm system, using a Raspberry Pi computer, some electronic components, and by writing a Python program.

This burglar alarm can be armed and disarmed, and if movement is detected while it is armed, an alarm will sound, just like a real burglar alarm. The alarm can be reset, which will stop the alarm sounding.

PHOTO OF COMPLETED ALARM SYSTEM



**Figure 2: The completed Raspberry Pi Burglar Alarm**

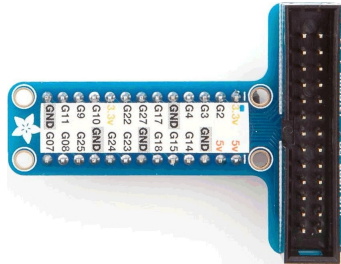
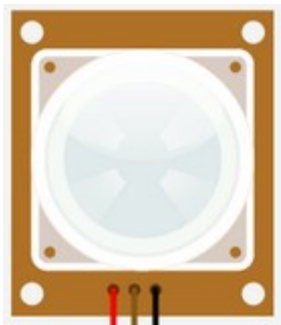
A PIR (Passive Infra-Red) sensor detects movement, which causes the LED (Light Emitting Diode) to flash. The push button, when pressed, will arm, disarm, or reset the alarm system. A powered speaker is used to generate alarm sounds. The Raspberry Pi computer runs programs that you write, that monitor and control the electronics, and these programs, written in the Python programming language, give your alarm system it's individual personality.

## 1.3. Worksheets

If you wish to use this booklet in worksheet form instead, please print pages 6 to 15 as double sided A4 sheets. Each project step has been designed to fit on two sides of A4, and printing from page 6 will create a useful stack of standalone worksheets that you can use in schools and clubs.

## 2. List of Parts

You will need the following parts to complete this project:

- |                            |   |                             |  |
|----------------------------|---|-----------------------------|--|
| <input type="checkbox"/> 1 | <a href="#"><u>PICTURE</u></a>  | <input type="checkbox"/> 1  | <a href="#"><u>PICTURE</u></a>   |
|                            | <b>Raspberry Pi computer</b>  |                             | <b>Powered Speaker</b>   |
|                            | with peripherals  |                             |  |
| <br>                       |   |                             |  |
| <input type="checkbox"/> 1 | <a href="#"><u>PICTURE</u></a>  | <input type="checkbox"/> 1  | <a href="#"><u>PICTURE</u></a>   |
|                            | <b>Breadboard</b>   |                             | <b>3.5mm Audio Cable</b>   |
| <br>                       |   |                             |  |
| <input type="checkbox"/> 1 | <a href="#"><u>PICTURE</u></a>  | <input type="checkbox"/> 1  | <a href="#"><u>PICTURE</u></a>   |
|                            | <b>LED</b>  |                             | <b>330 Ohm Resistor</b>  |
|                            |   |                             | (orange, orange, black)  |
| <br>                       |   |                             |  |
| <input type="checkbox"/> 4 | <a href="#"><u>PICTURE</u></a>  | <input type="checkbox"/> 15 | <a href="#"><u>PICTURE</u></a>   |
|                            | <b>Push Button</b>  |                             | <b>Jumper Cables</b>   |
|                            | (push to make)  |                             | (pin to pin)   |
| <br>                       |   |                             |  |
| <input type="checkbox"/> 1 | <a href="#"><u>PICTURE</u></a>  | <input type="checkbox"/> 1  |  |
|                            | <b>10K Ohm Resistor</b>   |                             | <b>Pi-T-Cobbler</b>  |
|                            | (brown, black, orange)  |                             |  |
| <br>                       |   |                             |  |
| <input type="checkbox"/> 1 |  |                             |  |
|                            | <b>Passive Infra Red Sensor (PIR)</b>   |                             |  |

### 3. Do an experiment!

A PIR sensor (Passive Infra Red) detects small changes in heat in front of it. Heat is not visible to the human eye, it appears in the spectrum as Infra Red, which is slightly below Red in the spectrum. However, you can do this small experiment to prove that it is still there!

Take a TV remote and press the buttons – you can't see the LED at the front, can you? Now get your smart-phone and turn on the camera, shine the TV remote into the camera. What happens?

Cameras on most smart-phones can sense a wider range of the spectrum than the human eye can, which is why you see the LED flashing inside the camera preview screen.

PHOTO OF TV REMOTE SHINING INTO MOBILE CAMERA



**Figure 3: Shining the Infra-Red LED of a TV remote into a smart-phone camera**

TODO PICTURE



**Figure 4: The electromagnetic spectrum**

## 4. Building the Alarm System

### 4.1. Flashing an LED

You are going to build a circuit and a program that flashes an LED repeatedly. This is your first step in any electronic circuit, and it tests that everything is working and that you can connect and control electronics from your computer. The **GPIO** pins on the Raspberry Pi will be used to control the electronics. Once you can flash an LED, you can build anything you could possibly imagine!

#### 4.1.1. Connect up the circuit

Follow the layout in the diagram below to connect everything together on the **breadboard**.

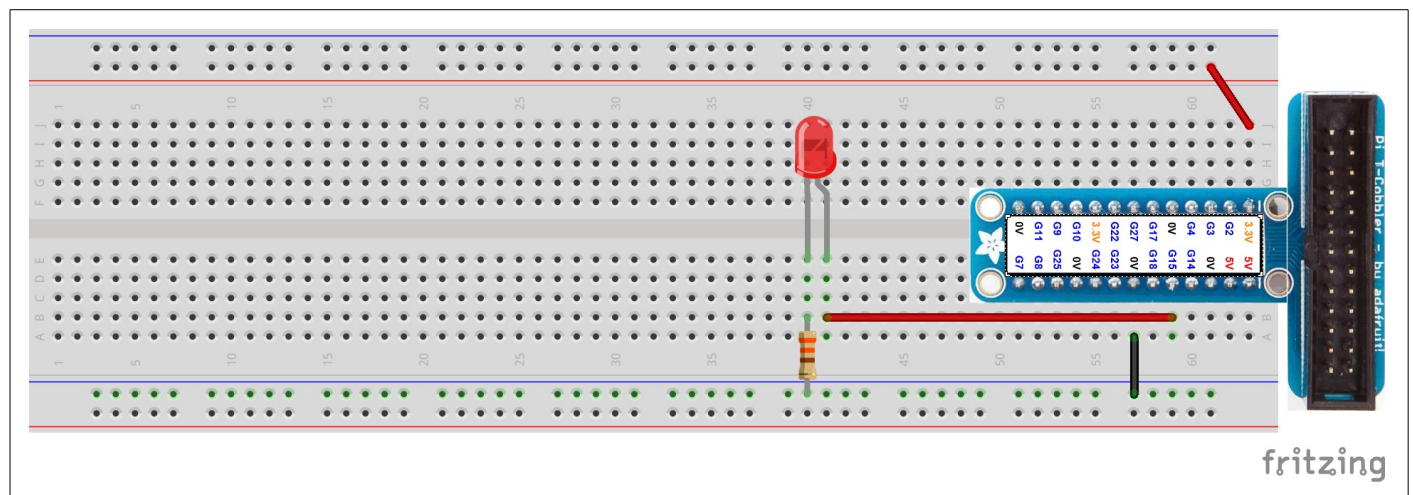


Figure 5: Wiring up an LED to the Raspberry Pi

#### 4.1.2. Write the program

Open the **IDLE** program from the desktop.

Choose FILE → NEW WINDOW from the menu.

Choose FILE → SAVE AS and type in the name `flash.py`

Now type in this program, making sure that you **indent** lines by pressing the TAB key where they are indented in the listing below.

```
import RPi.GPIO as GPIO
import time

LED = 15

GPIO.setmode(GPIO.BCM)
GPIO.setup(LED, GPIO.OUT)

while True:
    GPIO.output(LED, True)
    time.sleep(1)
    GPIO.output(LED, False)
    time.sleep(1)
```

Program 1: flash.py

### 4.1.3. Test the program

Save the program with FILE → SAVE, then open **LXTerminal** from the desktop. Now type this into the LXTerminal window to **run** your program:

```
sudo python flash.py
```

### 4.1.4. Did it Work?

When you run this program, the LED should flash on for 1 second, then off for 1 second. It should do this repeatedly, all the while your program is running. The LED should be quite bright.

If your LED does not flash, check the following:

- ☐ **Is your LED in the right way round?**  
There is a picture in the reference section of how an LED works.
- ☐ **Have you used the correct value of resistor?**  
If you use the 10K Ohm resistor by mistake, the LED might glow very dimly.
- ☐ **Have you correctly understood how a breadboard is wired?**  
There is a photo of the insides of a breadboard in the reference section.
- ☐ **Have you used the correct GPIO number on your Pi-T-Cobbler?**  
The sticker should be lined up correctly, but check it hasn't slipped.
- ☐ **Did you use the GPIO.BCM numbering mode?**  
If you used the GPIO.BOARD mode instead, the wrong pin will be used.

### 4.1.5. Other experiments

- ☐ What happens if you wire two LEDs in parallel (so that the long wires on both are connected to the GPIO pin, and the short wires on both are connected to the resistor). Can you explain what you now see happening?
- ☐ What happens if you wire the two LEDs in series (wire the circuit so that the short wire of the first LED connects to the long wire of the second LED, and the short wire of the second LED connects to the resistor). Can you explain what you now see happening?
- ☐ Ask a friend to challenge you to do something new with your LED, and see if you can make it work!

## 4.2. Adding a button to your circuit

You are going to add a button to your circuit, and modify your program so that when you press the button the LED flashes once. You are building the alarm system in small steps, and testing it as you go along, so that if something breaks, it is easy to understand what has changed and what the problem might be.

### 4.2.1. Wiring the circuit

Modify your circuit by adding the button, the 10K resistor and the extra wires.

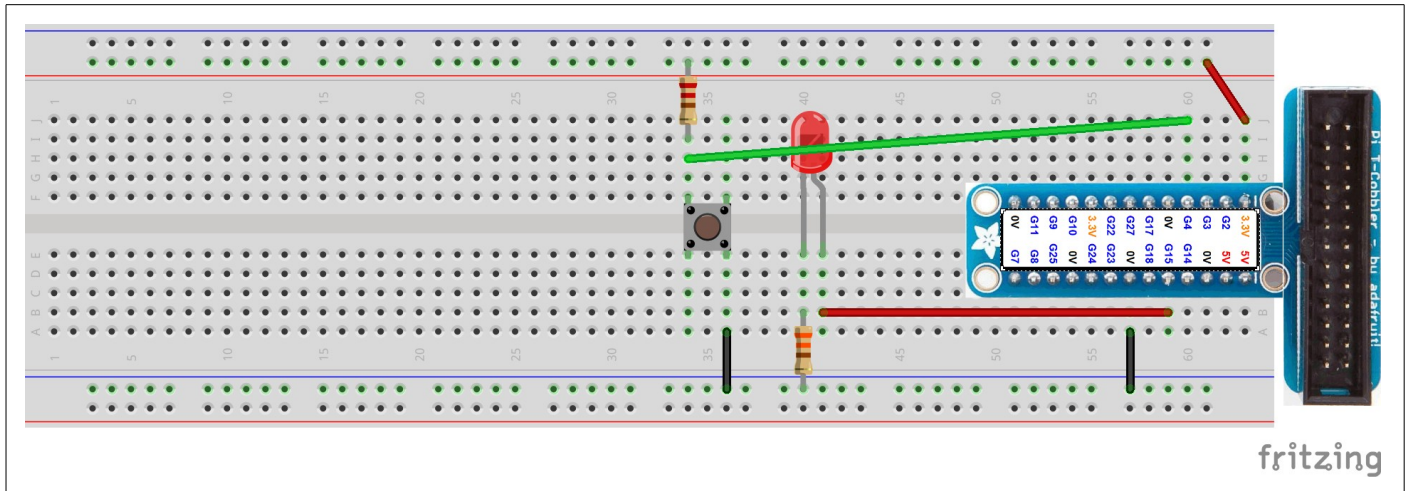


Figure 6: Wiring up the button

### 4.2.2. Writing the program

Use FILE → SAVE AS to save your existing program as `button.py`, and modify it by adding the lines marked in **bold** below. Make sure that you indent the lines correctly by using the TAB key.

```
import RPi.GPIO as GPIO
import time

LED = 15
BUTTON = 4

GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

while True:
    time.sleep(0.1)
    if GPIO.input(BUTTON) == False:
        GPIO.output(LED, True)
        time.sleep(1)
        GPIO.output(LED, False)
        time.sleep(1)
```

Program 2: `button.py`



### 4.2.3. Testing the program

Press **CTRL-C** in the LXTerminal window to stop your existing program from running. Save the program with FILE → SAVE, then type this into the LXTerminal window to run your program:

```
sudo python button.py
```

### 4.2.4. Did it work?

When you run this program, the LED should be off, until you press the button, then the LED should turn on for 1 second and then go off again.

If your LED does not come on and off when you press the button, check the following:

- ☐ **Is your button in the right way round?**  
The pins of the button should be on the top and the bottom of the breadboard (not on the sides). Check also that one of the pins has not broken off. If it has, the two pins on the left are connected together, and the two pins on the right are connected together, so you should be able to rewire your circuit using different pins.
- ☐ **Have you used the correct value of resistor?**  
If you use the 330 Ohm resistor by mistake, sometimes this can cause the computer to think that the button is always pressed.
- ☐ **Have you correctly understood how a breadboard is wired?**  
There is a photo of the insides of a breadboard in the reference section.
- ☐ **Have you used the correct GPIO number on your Pi-T-Cobbler?**  
The sticker should be lined up correctly, but check it hasn't slipped.
- ☐ **Did you use the GPIO.BCM numbering mode?**  
If you used the GPIO.BOARD mode instead, the wrong pin will be used.

### 4.2.5. Other experiments

- ☐ Change your program so that the LED flashes 4 times when you press the button.
- ☐ Change your program so that when the button is released the LED is always flashing once every second. When you press the button the LED should stop flashing and be off.
- ☐ Add a second LED to the breadboard. Change your program so that when the button is released, the first LED is off and the second LED is on. When the button is pressed, the first LED should be on, and the second LED should be off.
- ☐ Challenge a friend to do something new with their button, and help them to fix their program if it does not work!

### 4.3. PIR blinks LED

You are going to add the PIR sensor to your circuit, so that when it detects a movement it flashes the LED. You are still building the alarm system in small steps, and testing it each time you add a new piece of electronics to the circuit, to make sure it still works correctly.

#### 4.3.1. Wiring the circuit

Connect up the PIR to your circuit as shown in the diagram below:

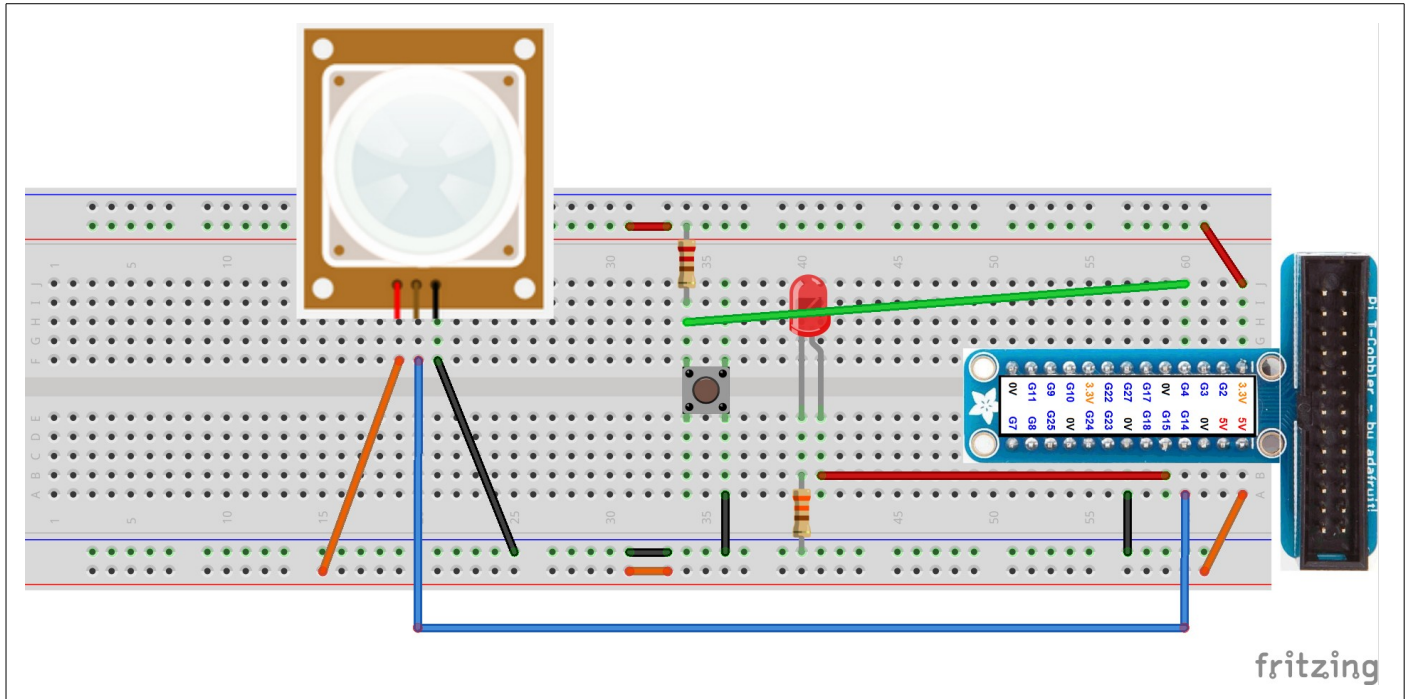


Figure 7: Wiring up the PIR

#### 4.3.2. Writing the program

Use FILE → SAVE AS to save your existing program as `PIR.py`, and modify it by adding the lines marked in **bold** below. Make sure that you indent the lines correctly.

```
import RPi.GPIO as GPIO
import time

LED = 15
PIR = 14
GPIO.setmode(GPIO.BCM)
GPIO.setup(PIR, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

while True:
    time.sleep(0.1)
    if GPIO.input(PIR):
        GPIO.output(LED, True)
        time.sleep(1)
        GPIO.output(LED, False)
        time.sleep(1)
```

Program 3: PIR.py

### 4.3.3. Testing the program

Press CTRL-C in the LXTerminal window to stop your existing program from running. Save the program with FILE → SAVE, then type this into the LXTerminal window to run your program:

```
sudo python PIR.py
```

### 4.3.4. Did it work?

When you run this program, the LED should be off, until you move in front of the PIR then the LED should be on. The LED should go off after a short while. You can try waving at the PIR to make it turn on and off.

If your LED does not come on and off when you wave at the PIR or move in front of it, check the following:

- ☐ **Is the PIR wired correctly?**

Some PIR sensors have a different arrangement of pins, so check the wiring very carefully. The PIR's that are included in these packs match the diagrams, but if you have bought a different PIR, the pins are often wired differently.

- ☐ **Have you allowed time for the PIR to “settle”**

Some PIR's require a few seconds of nothing moving in front of them, in order to self-calibrate to the environment. Try to arrange for the room to be as still as possible for a few seconds to allow the PIR to calibrate to it's normal environment.

- ☐ **Try the button instead?**

If the PIR does not seem to work, check the rest of your circuit by wiring GPIO 14 over to the button instead, and make sure that your program works with that first. If you are having problems getting your PIR to settle, you can test all of the alarm system by replacing the PIR with a second push-button, in order to complete all the exercises in this booklet.

### 4.3.5. Other experiments

- ☐ Change your program so that when it first runs, it waits for a period of 10 seconds without a single PIR trigger, before entering the main loop. If within that 10 second period the PIR triggers, reset your 10 second timer and try again. This will be useful later on when you test your alarm system in a real house, in order to detect that it is safe to arm the alarm system. Hint: use `time.time()`.

## 4.4. PIR plays a sound

You are going to make your circuit play a sound when it detects a movement. This will make your circuit more like an alarm system that warns you when someone enters your house. You are still adding and testing things in small steps as you go along.

### 4.4.1. Writing the program

Use FILE → SAVE AS to save your existing program as `PIRSound.py`, and modify it by adding the lines marked in **bold** below. Make sure that you indent the lines correctly by using the TAB key.

```
import RPi.GPIO as GPIO
import time
import pygame

pygame.mixer.init()
beep = pygame.mixer.Sound("beep.wav")

LED = 15
BUTTON = 4
PIR = 14

GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON, GPIO.IN)
GPIO.setup(PIR, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

while True:
    time.sleep(0.1)
    if GPIO.input(PIR):
        GPIO.output(LED, True)
        beep.play()
        time.sleep(1)
        GPIO.output(LED, False)
```

#### Program 4: PIRSound.py

### 4.4.2. Testing the program

Press CTRL-C in the LXTerminal window to stop your existing program from running. Save the program with FILE → SAVE, then plug in your speaker and make sure it is turned on. Finally, type this into the LXTerminal window to run your program:

```
sudo python PIRSound.py
```

### 4.4.3. Did it work?

When you run this program, the LED should be off, until you move in front of the PIR then the LED should be on and the speaker should play a beep sound. The LED should go off after a short while. You can try waving at the PIR to make it turn on and off.

If your LED does not come on and off and you don't get a sound when you wave at the PIR or move in front of it, check the following:

☐ **Does the `beep.wav` file exist?**

Check in File manager that there is actually a `beep.wav` file in the same folder as your Python program.

☐ **Does `beep.wav` work at the command line?**

From LXTerminal, type:

```
aplay beep.wav
```

Make sure you hear the sound.

☐ **Is the speaker turned on and the battery charged?**

If you don't hear the sound when you use the `aplay` command, check the speaker is powered and charged up and connected properly. Bluetooth speakers sometimes go into bluetooth mode if the wire to the Raspberry Pi is not correctly connected.

☐ **Is the Raspberry Pi configured to send sound to the speaker?**

If you are using a HDMI lead, or a HDMI to VGA adaptor, the Raspberry Pi might be sending its sound along the HDMI lead to the monitor. You can force the Raspberry Pi to send sound to the audio jack and set full volume by typing these commands into LXTerminal:

```
sudo amixer cset numid=3 1
sudo amixer cset numid=3 100%
```

#### 4.4.4. Other experiments

- ☐ Find some other `.wav` file sounds. The first time the PIR is triggered, play the `beep.wav`. The second time it is triggered, play a different sound. The third time it is triggered, play yet another sound. After 3 sounds have been played, return to the first sound again.
- ☐ Change your program so that in addition to sounds playing when the PIR detects movement, it plays a different sound if you press the push button.
- ☐ Use the free audacity audio editor program on a PC or a Mac, and record you and your friends making funny sounds or saying jokes. Record 10 different sounds, and transfer these as `.wav` files onto your Raspberry Pi. Every time your PIR is triggered, play a random sound from your 10 sounds. Hint: use `random.choice()`.
- ☐ Write a program that times how long between each PIR trigger, and make it display the time in seconds on the screen using `print()`. When you press the button, count down from 5 to 0 on the screen, then start a timer. When a PIR movement is triggered, play a sound indicating you have lost, and show how long you have stood still for. Challenge your friends to stand still in front of the PIR for the longest time possible.

## 4.5. Finishing off the alarm system program

Now that all of your electronics is wired up and working, you are going to finish off the alarm system program. This program will work like a real alarm system: when you press the button, it will change between an `idle` state and an `armed` state. If the PIR detects movement while the alarm is `armed`, it will sound the alarm. Pressing the button while the alarm is sounding will reset the alarm.

### 4.5.1. Writing the program

Use FILE → SAVE AS to save your existing program as `alarm.py`, and modify it by adding the lines marked in **bold** below. Make sure that you indent the lines correctly.

```
import RPi.GPIO as GPIO

import time
import pygame

pygame.mixer.init()
alarm = pygame.mixer.Sound("alarm.wav")
beep = pygame.mixer.Sound("beep.wav")

LED = 15
BUTTON = 4
PIR = 14
state = "idle"

GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON, GPIO.IN)
GPIO.setup(PIR, GPIO.IN)
GPIO.setup(LED, GPIO.OUT)

while True:
    time.sleep(0.5)
    button = not GPIO.input(BUTTON)
    pir = GPIO.input(PIR)
    GPIO.output(LED, pir)

    if state == "idle":
        if button:
            print("Alarm armed")
            beep.play()
            time.sleep(0.5)
            beep.play()
            time.sleep(1)
            state = "armed"

    elif state == "armed":
        if pir:
            print("Alarm tripped")
            alarm.play(loops=-1)
            state = "sounding"
```

```

elif button:
    print("Disabling alarm")
    beep.play()
    time.sleep(1)
    state = "idle"

elif state == "sounding":
    if button:
        print("Resetting alarm")
        alarm.stop()
        time.sleep(0.5)
        beep.play()
        time.sleep(1)
        state = "idle"

```

### Program 5: alarm.py

#### 4.5.2. Testing the program

Press CTRL-C in the LXTerminal window to stop your existing program from running. Save the program with FILE → SAVE, then type this into the LXTerminal window:

```
sudo python alarm.py
```

#### 4.5.3. Did it work?

When you run this program, the LED should be off, until you move in front of the PIR then the LED should be on when it detects a movement.

Press the button and it should beep twice to let you know that the alarm is armed. Wave at the PIR and the alarm should sound. Press the button and it should stop the alarm sound and beep once to let you know that the alarm is disarmed.

If this program does not work, check the following:

- ☐ **Is all of your indentation correct in your Python program?**  
There are places where you have to indent this program by two levels, and the indentation in Python must be perfect for this program to work.
- ☐ **Have you spelled `idle`, `armed` or `sounding` with different case?**  
“Idle” and “idle” are two different strings in Python, if you spell them wrong or with a different case, the `if` statement will not detect the state variable correctly.
- ☐ **Put some extra `print` statements into the program**  
If your program still doesn't work, add some extra `print ( )` statements at points in the program and get it to tell you what it is doing at each step of the program.

#### 4.5.4. Other experiments

- ☐ Add an exit sound, that plays for 10 seconds before finally arming the alarm.
- ☐ Add an entry sound that plays when the alarm is triggered for 10 seconds, and only move into the “sounding” state if the alarm is not reset within that first 10 seconds.



## 5. How it works

### 5.1. Digital and Analogue

The Raspberry Pi is a digital device, working internally using a fixed number of values like a digital clock. It uses two distinct states of an electronic circuit – either on (1) or off (0).

The world around us is analogue, like an analogue clock with hands that tick around the clock face – this means that everything is measured using a range of values. A voltage provided by a battery could be 9 Volts, or 8.9 Volts, or 8.91237 Volts, or anything from a wide range of values – our world is not as simple in reality as just 'on' and 'off'.

### 5.2. GPIO Pins

GPIO pins (General Purpose Input Output) are signals wired directly to the main processor chip of a computer system, that are designed without a specific purpose – it is up to you to decide how to use them. On the Raspberry Pi, the GPIO pins can be 'on' or 'off', and because the Raspberry Pi is a 3.3 Volt device, if you measure the voltage present on a GPIO pin, it will be close to 0 Volts for 'off', and close to 3.3 Volts for 'On'.

The Raspberry Pi has two numbering schemes for GPIO's: they can be numbered according to the pin number on the circuit board connector (`GPIO.BOARD`), or according to the circuit number used by the main processor chip (`GPIO.BCM`). BCM stands for “Broadcom”, which is the company that designed the processor chip used on the Raspberry Pi. Because your program use BCM mode, this means that the numbers you use in your Python programs match up to the GPIO circuit numbers, not the pin numbers on the connector. The Pi-T-Cobbler you use has a sticky label on it that labels the GPIO circuit numbers.

### 5.3. Lighting an LED

In Figure 5 is a circuit that you used to flash an LED from your Python program. A circuit is made between GPIO 15, through the LED, through the 330 Ohm resistor, and through to the GND pin (0 Volts) of the Raspberry Pi. When a digital '1' (on) is applied to GPIO 15, a voltage of about 3.3 Volts appears on the pin – this causes electricity to flow around the circuit, and the LED lights up. When a digital '0' (off) is applied to GPIO 15, there is no longer a voltage (0 Volts), no electricity flows around the circuit, and the LED is dark.

An LED is a Light Emitting Diode, and a Diode is a component that only allows electricity to flow in one direction. The LED has a positive (anode) and negative (cathode) connector, where the positive lead is always the longer one of the two. LED's are sensitive devices, and if you connect them directly to a battery or a computer system, too much electrical current will flow, causing damage to the LED. The 330 Ohm resistor resists the electricity so that damage does not occur. You can use the online calculator in the web links later to see how to calculate the correct value for this resistor.

Program 1 sets up GPIO 15 (connected to the LED) as an output so that the voltage on the pin can be controlled by the computer. The `while True:` is a loop that will loop repeatedly. `GPIO.output(LED, True)` turns the GPIO pin on (it's voltage rises to about 3.3 Volts), causing the LED to light up. `time.sleep(1)` will wait for 1 second, and then the LED is turned off after another 1 second. This makes the LED flash repeatedly.



## 5.4. Sensing a Button Press

Figure 6 shows the addition of a button to your circuit. This button is called a “push to make” button – when you push the button it joins two pieces of metal inside it, and a circuit is made, causing electricity to flow. When you release the button, the metal contacts inside are no longer connected and it breaks the connection, preventing electricity from flowing through it.

The button is connected to GPIO 4. Program 2 adds a `GPIO.setup(BUTTON, GPIO.IN)`, which sets up the GPIO so that it can sense changes in voltage on the pin (rather than `GPIO.OUT` which allows the voltage on the pin to be controlled by the computer).

You learnt earlier that computers are digital devices, and this means that only a 1 (on) or a 0 (off) can be sensed by this GPIO. However, you also learnt that the voltages in your circuit are somewhere around 3.3 Volts. The Raspberry Pi will sense anything that is close to 0 Volts as a 0 (off), and anything that is close to 3.3 Volts as a 1 (on). Voltages somewhere in the middle might be seen as either a 1 or a 0, so it is always important to design circuits that use voltages close to 0V and 3.3 Volts (for the Raspberry Pi, anyway) for them to work.

The button is connected so that when it is pressed, the legs on the left hand side of the button join up electrically to the legs on the right hand side of the button. If you check Figure 6 you will see that the wire from GPIO 4 is on the left hand side of the button. There is a wire connecting the right hand side of the button to GND (0 Volts). So, when the button is pressed, the voltage sensed by GPIO 4 is 0 Volts, and the Raspberry Pi sees this as a 0 (off).

If a GPIO pin is completely disconnected, any stray voltage or interference might be seen by the computer, and this can cause your program to incorrectly think the button has been pressed. In Figure 6 you can see there is a 10K Ohm resistor connected between GPIO 4 and the 3.3 Volt connection. This is called a **pull-up resistor**. This is to prevent those stray voltages from upsetting your circuit.

You could use a wire here to directly connect GPIO 4 to the 3.3 Volt connection for when the switch is released, but this would cause the circuit to completely short together the 3.3 Volt and 0 Volt connections on the Raspberry Pi when you press the button, and it would reset the computer. A Resistor is used so that a tiny amount of electricity flows into the GPIO 4 pin and fools the computer into thinking it is connected to the 3.3 Volt connector.

A high value resistor (10K Ohm is quite a high value) is used so that your circuit does not use a lot of current. If the resistor value is too low, too much electrical current will flow and your computer will reset. If the value is too high, the GPIO 4 pin will not sense that the button is released and your program will make the wrong decisions. You can read all about how to calculate this pullup resistor in the web links in the reference section.

## 5.5. Preventing button bounces

Your button is a mechanical device, and when you press and release it, the mechanical connection inside “bounces” between off and on and off again. This “noise” can be misinterpreted by your Python program as multiple button presses, and your program will make the wrong choices. In Program 2, you can see that `time.sleep(0.1)` is used – this delays for a tenth of a second (a short time) to prevent this electrical noise from confusing your program. If this delay is too short, your program will sense multiple button presses. If this delay is too long, it will slow your program down and your user will get frustrated.

## 5.6. Detecting movement with the PIR

The PIR sensor works in a similar way to the experiment with the TV remote and the camera – small changes in infra red (heat) that is emitted by the human body are measured by the PIR. Normally the PIR output pin is at 0 Volts when there is no movement. When enough change is sensed, the output pin changes to 3.3 Volts to indicate that a movement has occurred. Program 3 senses changes in the GPIO pin voltage.

## 5.7. Playing a sound

Program 4 adds sound playback to your alarm system. It uses the `pygame` module, which has a built in sound player. `import pygame` is needed to gain access to this extra module of computer code. `pygame.mixer.init()` initialises the sound mixer, which is what you will use to play sounds. `beep = pygame.mixer.Sound("beep.wav")` reads the sound from the file `beep.wav` into the memory of your Python program so that it can be played on demand. Finally, when the PIR senses movement, `beep.play()` will play the sound through the speakers of the Raspberry Pi. The sound will play once.

## 5.8. Tracking the state of the alarm system

Program 5 adds a variable called 'state', which is used to track which state the Python program is in at any one time. When the alarm system changes between different states, this variable is changed so that the program knows what to do next.

At the start of the `while True:` loop, the button GPIO is read into a variable called 'button'. Because `False` means the button is pressed and `True` means it is released, using `not` here will reverse this so that the button variable is always `True` when the button is pressed, and this will make the later parts of the program easier to read. The PIR GPIO is read into the variable 'pir', and the LED always shows whether the PIR has detected movement or not. Using variables here makes the program easier to read.

“idle” means that the alarm is not set, and in this state the program waits for the button to be pressed, plays a double beep sound, and then moves into the “armed” state. But any PIR movement in the “idle” state will be ignored, even though the LED will flash.

“armed” means that the alarm system will sound if it detects a movement. If the `pir` variable is `True`, movement is detected and the alarm sound is played (`loops=-1` plays it repeatedly). Otherwise, if the button is pressed, a single beep is sounded so your user knows it is disarmed, and then the program moves back to the “idle” state.

“sounding” means that the alarm is sounding. The only thing possible in this state is for the button to be pressed, which then stops the alarm sound playing, plays a single beep to let your user know it is now disabled, and then moves back to the “idle” state again.

The extra `time.sleep()`'s in this program try to prevent the alarm system arming and disarming repeatedly if your user doesn't remove their finger from the button quick enough.

## 6. What you have learnt

- ☐ That Infra-Red can be sensed with a PIR to detect movement
- ☐ How to build a circuit on a breadboard
- ☐ How to connect a circuit board to a Raspberry Pi computer
- ☐ How to write a program to flash an LED
- ☐ How to write a program to sense that a button has been pressed
- ☐ How to connect a PIR and detect external movement
- ☐ Using the GPIO pins of a Raspberry Pi to sense inputs and control outputs
- ☐ Writing programs with time delays in them
- ☐ Writing and testing Python programs
- ☐ Writing a program with a number of different states (called a “state machine”)

## 7. Jargon

**Breadboard** – A solder-less breadboard is used by design engineers to build prototypes of electronic circuits. Components are pushed into the holes in the breadboard, and metal strips inside the breadboard connect them together. Components can be removed and replaced in different positions to change the circuit.

**CTRL-C** – Pressing this sequence of keys on the keyboard while a Python program is running, will stop the program.

**GPIO** – (General Purpose Input Output) – A type of pin connected to the central processor of a computer system, that can be used for any purpose, normally for sensing inputs or controlling outputs.

**IDLE** – The Integrated Development Environment that you use to write Python programs. This is a useful program editor, because it has coloured highlighting and a good copy/paste facility.

**Indent** – The spaces at the left hand edge of a program are called the indent. Python uses indents to group together lines of Python code that belong together; for example, all the code that will run as part of a while loop is indented by one level.

**Infra-Red** – A part of the electromagnetic spectrum that is just outside the visible range of the human eye. It is called Infra Red because it is slightly below the colour red.

**LED** – (Light Emitting Diode) – A diode is a component that only lets electricity flow one way. An LED is a type of diode that also lights up when you pass an electric current through it the right way.

**LXTerminal** – A window on the Raspberry Pi that provides access to a terminal prompt. From this prompt, you can type commands that are run by the computer.

**PIR** – (Passive Infra Red) – a type of sensor that detects changes in infra red (heat) and allows your programs to detect movement.

**Python** – A programming language for computers. You type in instructions in the editor, store these instructions in a file, and run the program. It is this program that gives a new personality to your computer.

**Pull-up Resistor** – A resistor that is used to connect part of the circuit to a high voltage (in this case 3.3 Volts). This is usually used with a button to make sure that when it is released, a voltage is still present on the GPIO pin.

**Resistor** – A small electronic component that resists the flow of electricity in a circuit. It is often used to limit the amount of electric current that flows through a LED.

**Run** – When you run a program, it means that the instructions stored in a file are followed one by one by the computer.

**State** – A computer system may be in one of many different states. Each state defines what can happen while in that state, and how the system moves out of that state.

**sudo** – (Substitute User and DO) – This command tells the operating system to run the command to the right of it as a super user, which gives the program full permission to access the hardware of the computer.

## 8. Ideas to try

1. **Add a disarm code:** use a `raw_input()` statement to ask the user for a 4 digit code number when they press the button to disarm the alarm. Only if the code is correct, disable the alarm.
2. **Add an arm code:** use a `raw_input()` statement to ask the user for a 4 digit code to arm the alarm when they press the button. Only arm the alarm if the code is correct.
3. **Use Minecraft to build a house:** use geo-fencing to detect when your player enters the house, and this will trigger the alarm. Have a red block and a green block outside of the house, that when you hit them, they arm and disarm the alarm. Make the alarm play sounds on the speaker as before.
4. **Add a reset code:** if the alarm is sounding, use a `raw_input()` statement to ask the user for a different 4 digit code to reset the alarm when they press the button. Only disarm the alarm if the code is correct.
5. **Add an arming delay:** this will make it easier to get out of the house when you arm the alarm. Add an extra step `arming`, that beeps 10 times before arming the alarm. This gives you time to get out of the house before the alarm can be triggered.
6. **Add an entry delay:** this will make it easier to get into the house when the alarm is armed. Add an extra step `triggered` that beeps once per second, for up to 20 seconds.
7. **Add hardware buttons for code entry:** add 3 more buttons to the circuit. Write some code in the `sounding` state so that you have to press the 4 buttons in a specific sequence to disarm the alarm.
8. **Add a data log feature:** every time the alarm is armed, disarmed, or triggered, make the program write the date and time and a small message to a text file on the computer. You can use this feature to check what has happened while you have been away from your house or room for a long time.
9. **Add more sensors:** use more PIR sensors and link them up to your program. When the alarm is armed, any sensor that senses movement should trigger the alarm. Put `print()` statements in your program to show which sensor has triggered the alarm.
10. **Add zones to your multi-sensor alarm system:** Nominate one of the PIR sensors as the entry door. If the alarm is triggered by the entry door, allow the entry delay process to take place. However, if any other sensor triggers the alarm system, sound the alarm.
11. **In your minecraft house, use zoning in each room:** if the entry hall sensor is triggered, it should enter `triggered` state for 20 seconds, and allow you to disarm the alarm by hitting the green block. If you don't hit the green block in time, the alarm sounds. All other rooms are in a different zone, and they move to `sounding` immediately they are triggered when the alarm is armed.
12. **Link up your alarm system to an Internet of Things service:** present each zone and the alarm armed state as shared data for other applications to use. Link the arm/disarm buttons to the same service so that they can be remotely controlled.

## 9. Reference Information

### 9.1. Python Reference

<b>Printing to the screen</b>	<b>Looping forever</b>
<pre>print("hello")</pre>	<pre>while True:     # do something here</pre>
<b>Setting up GPIO Pins</b>	<b>Reading and writing GPIO pins</b>
<pre>import RPi.GPIO as GPIO GPIO.setmode(GPIO.BCM) LED = 15 GPIO.setup(LED, GPIO.OUT) BUTTON = 4 GPIO.setup(BUTTON, GPIO.IN)</pre>	<pre># read a button b = GPIO.input(BUTTON)  # write to an LED GPIO.output(LED, True)</pre>
<b>if</b>	<b>if, else, elif</b>
<pre>if a == 0:     print("it's zero") if a &gt; 0:     print("it's bigger than zero") if message == "yes":     print("you said YES!")</pre>	<pre>if a == 0:     print("zero") elif a == 1:     print("one") else:     print("something else")</pre>
<b>Time Delays</b>	<b>Loading a sound</b>
<pre>import time time.sleep(1) # one second time.sleep(0.5) # half a second</pre>	<pre>import pygame pygame.mixer.init() alarm=pygame.mixer.Sound("al.wav")</pre>
<b>Playing sounds</b>	<b>Looping and stopping sounds</b>
<pre>alarm.play() # play once</pre>	<pre>alarm.play(loops=-1) # play forever alarm.stop() # stop a looping sound</pre>

## 9.2. GPIO Pins

If you connect directly to the pins on the Raspberry Pi, here is a useful pin map diagram.

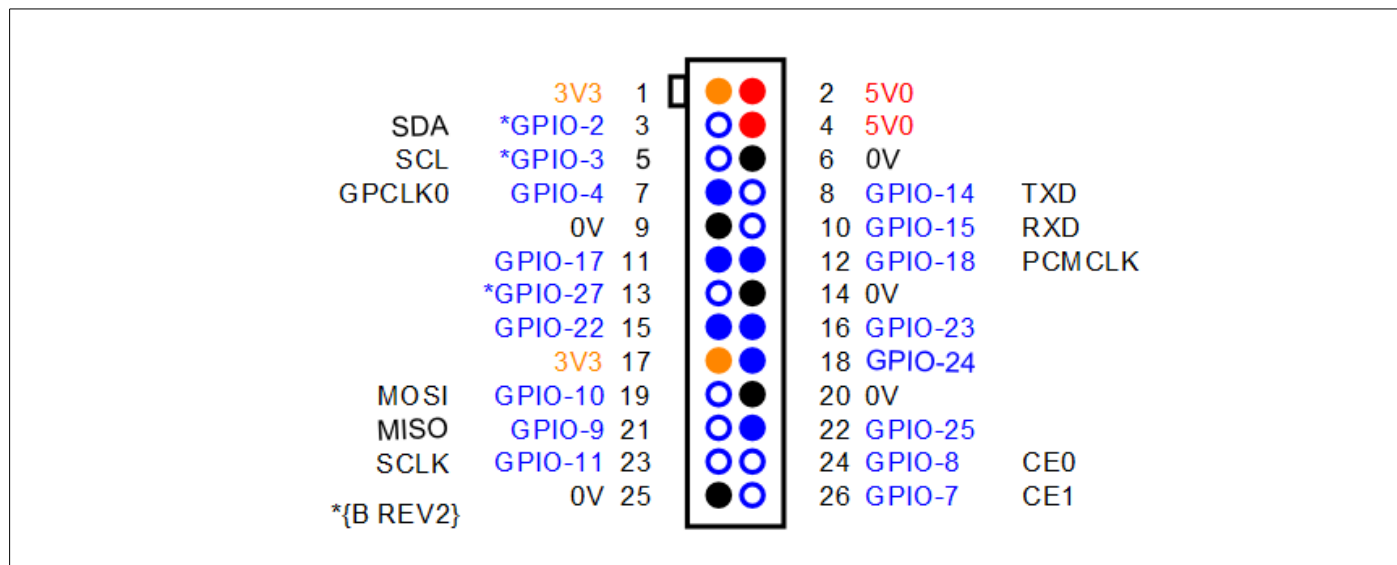


Figure 8 : Wiring of the GPIO connector on the Raspberry Pi model B

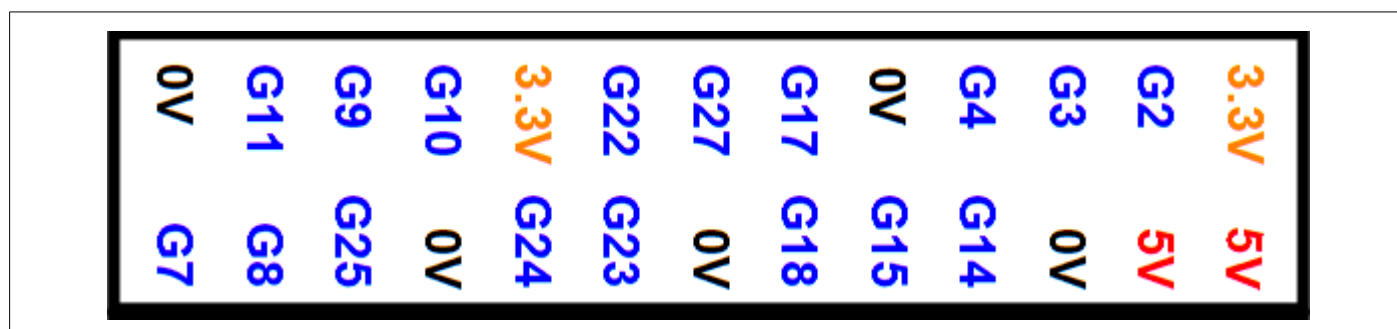


Figure 9: Wiring of the Pi-T-Cobbler pins

## 9.3. Component Wiring

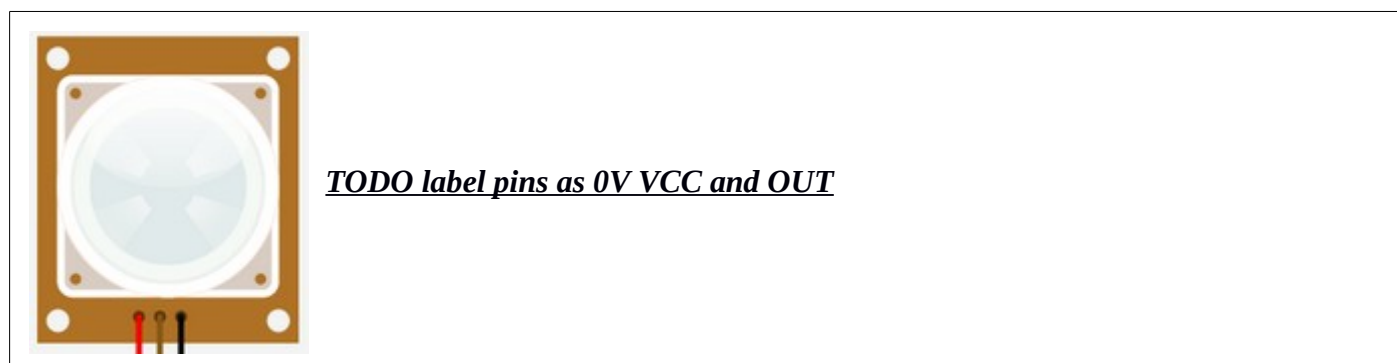
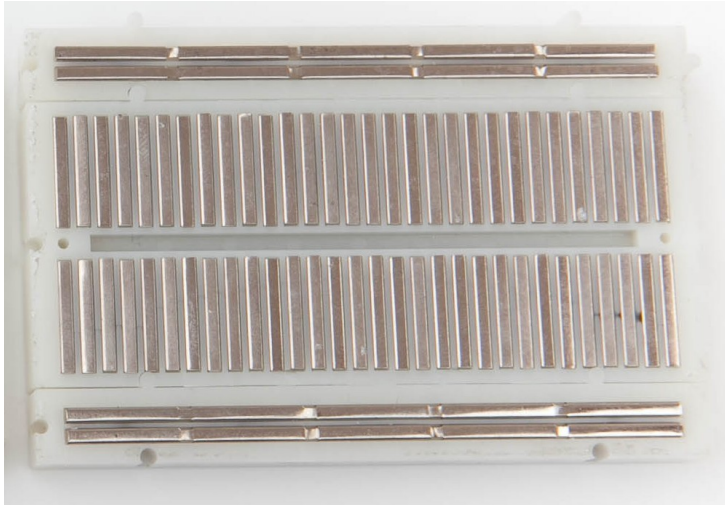


Figure 10: The PIR and how it is wired.



**PICTURE**

**Figure 11: A LED has an anode (positive) and cathode (negative).**



**Figure 12: The insides of a breadboard, showing how the strips are connected.**

**PICTURE OF BUTTON, with schematic of how it is wired?**

**Figure 13: A push-button, and how it is wired internally.**



## 10. Links to further Information

### 10.1. Where to buy the components

You can buy the PIR from here:

[http://4tronix.co.uk/store/index.php?rt=product/product&keyword=pir&product\\_id=135](http://4tronix.co.uk/store/index.php?rt=product/product&keyword=pir&product_id=135)

Other components can be purchased from:

<http://www.maplin.co.uk>

<http://www.rapidonline.com/>

<http://www.skpang.co.uk>

<http://shop.pimoroni.com/>

<http://www.4tronix.co.uk/store/>

### 10.2. Websites

Understanding the sudo command	<a href="http://blog.whaleygeek.co.uk/sudo-on-raspberry-pi-why-and-why-you-need-to-ask-kids-too/">http://blog.whaleygeek.co.uk/sudo-on-raspberry-pi-why-and-why-you-need-to-ask-kids-too/</a>
Pygame	<a href="http://pygame.org/news.html">http://pygame.org/news.html</a>
Raspberry pi learning	<a href="http://www.raspberrypi.org/resources/learn/">http://www.raspberrypi.org/resources/learn/</a>
Python reference	<a href="https://www.python.org/">https://www.python.org/</a>
Code academy	<a href="http://www.codecademy.com/">http://www.codecademy.com/</a>
Code avengers	<a href="http://www.codeavengers.com/">http://www.codeavengers.com/</a>
Alarm system products	<i>(some google search)</i>
How a PIR works	<i>(some google search)</i>
Infra red sensing	<i>(some google search)</i>
Resistor colour codes	<i>(some google search)</i>
Calculating an LED current limiting resistor	<i>(some google search)</i>
Calculating pullup resistor for button	<i>(some google search)</i>
Fritzing diagram editor	<a href="http://fritzing.org">http://fritzing.org</a>

### 10.3. Books

#### **Adventures in Raspberry Pi**

Carrie Anne Philbin, Wiley, December 2013, ISBN 978-1-118-75125-1

#### **Learning Python on the Raspberry Pi**

Alex Bradbury and Ben Everard, Wiley, March 2014, ISBN 978-1-118-71705-9

#### **Programming the Raspberry Pi, Getting started with Python**

Simon Monk, Jan 2013, ISBN 978-0071807838

#### **Adventures in Minecraft**

Martin O'Hanlon and David Whale, Wiley, November 2014, ISBN 978-1-118-94691-6