Extracted from:

Raspberry Pi: A Quick-Start Guide, 2nd Edition

The Pragmatic Bookshelf

Dallas, Texas • Raleigh, North Carolina

The Pragmatic Programmers

Raspberry Pi

A Quick-Start Guide

Second Edition

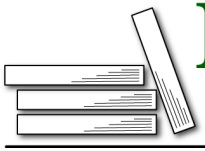
Maik Schmidt

Edited by Jacquelyn Carter

Raspberry Pi: A Quick-Start Guide, 2nd Edition

Maik Schmidt

The Pragmatic Bookshelf
Dallas, Texas • Raleigh, North Carolina



Pragmatic Bookshelf

Our Pragmatic courses, workshops, and other products can help you and your team create better software and have more fun. For more information, as well as the latest Pragmatic titles, please visit us at *http://pragprog.com*.

The team that produced this book includes:

Jacquelyn Carter (editor)
Potomac Indexing, LLC (indexer)
Cathleen Small (copyeditor)
David J Kelly (typesetter)
Janet Furlow (producer)
Ellie Callahan (support)

For international rights, please contact *rights@pragprog.com*.

Printed in the United States of America.
ISBN-13: 978-1-93778-580-2
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—February 2014

Detect Motion with the Pi

Chances are good that you benefit from motion detectors several times a week or even several times a day. They might turn on the lights automatically at dark, or perhaps they turn on the lights when you enter the restroom at work. In this section, you'll learn how these detectors work, and you'll learn how to turn the Pi into a motion detector.

Connect the PIR Sensor to the Pi

Many motion detectors use passive infrared (PIR) sensors.[2] A PIR sensor permanently measures infrared light and notices whenever something in the infrared spectrum changes. This is all you need to detect motion, because nearly every object emits infrared light. That's true for everything in front of your house: the ground, a bicycle, a garbage can, and so on. All of these things emit a fairly constant portion of infrared light, and it doesn't change rapidly. But if a human being or an animal approaches your front door, the sensor will notice a big variation and fire a signal.

The innards of PIR sensors are rather complex, but using them is easy. In the following figure, you can see the Parallax PIR sensor (Rev A) that we'll use in this section's examples. The sensor has a jumper you can use for changing its behavior. Make sure it's in position H; that is, the jumper covers the pin next to the H.[3]

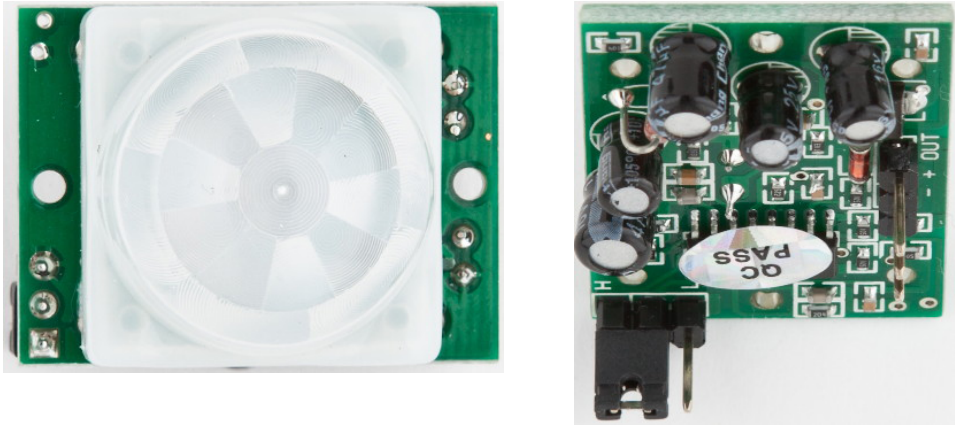


Figure 52—Top and bottom of a passive infrared sensor

2. http://en.wikipedia.org/wiki/Passive_infrared_sensor
3. At http://www.ladyada.net/learn/sensors/pir.html, you'll find an excellent tutorial explaining all the sensor's details.

Also, the sensor has three pins that you need to connect to the Pi using female/male jumper wires. In the following figure, you can see how. Connect the Pi's 5V pin to the sensor's power pin and the Pi's ground pin to the sensor's ground pin. Finally, connect the sensor's signal pin to pin GPIO23 on the Pi. Usually, the pins on the sensor are labeled. When in doubt, look at the sensor's data sheet.

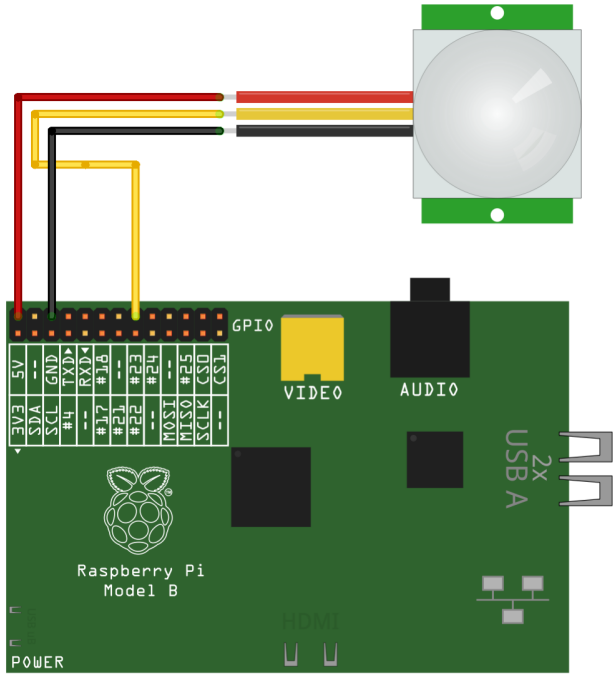


Figure 53—The PIR circuit

All digital PIR sensors work more or less the same way. As long as they don't detect motion, they don't output any current on their signal pin. When they detect motion, they output a high signal—that is, a certain current that you can usually look up in the sensor's data sheet.

WARNING AGAIN! Never attach a sensor that outputs more than 3.3V directly to the Pi. It will damage your Pi!

Control a PIR Sensor

Now that the wiring is finished, we have to control the sensor using some software. For many programming tasks on the Pi, the Python programming

language[4] is a good choice. It's easy to learn, and the RPi library[5] has many convenient functions for controlling the Pi's GPIO pins. Raspbian already comes with Python, but you have to install RPi:

```
pi@raspberry:~$ sudo apt-get update
pi@raspberry:~$ sudo apt-get install python-dev python-rpi.gpio
```

That's all the preparation we need, and now we can define a new Python class for working with a PIR sensor:

Sensors/pir.py

```
Line 1  import RPi.GPIO as GPIO
    2  class PassiveInfraredSensor:
    3    def __init__(self, pin):
    4      self.pin = pin
    5      GPIO.setmode(GPIO.BCM)
    6      GPIO.setup(self.pin, GPIO.IN)
    7
    8    def motion_detected(self):
    9      return GPIO.input(self.pin)
```

Even if you've never worked with Python before, you should be able to understand most of the code. Before we dissect the code line by line, you should know that Python treats whitespace differently from most other programming languages. Instead of creating code blocks using curly braces ({ }) or keywords such as begin and end, Python uses indentation. It doesn't matter whether you use spaces or tabs to indent a block of code, but you have to be consistent. That is, if you've indented the first line of a block using four spaces, you have to indent the next line using four spaces, too.

In the first line we import RPi's GPIO functions, so we can use them in our own code. Then we define a new class named PassiveInfraredSensor. Whenever we create a new PassiveInfraredSensor object, Python will call the __init__() function. This function expects two arguments: the newly created object (self) and the number of the pin to which we've connected the sensor (pin). Python will initialize the first argument automatically.

In line 4, we store the pin number in the current object, and after that we set the numbering scheme for the pins using the GPIO.setmode() function. We pass it the value GPIO.BCM, so the RPi library interprets pin numbers using the Broadcom definition. (See the top and bottom rows in Figure 46, *The Pi's GPIO pins*, on page ?.) In our case, the pin number is 23 because we have connected the sensor's signal pin to pin GPIO23 on the Pi.

4. http://www.python.org/
5. http://code.google.com/p/raspberry-gpio-python/

Alternatively, we can set the mode to GPIO.BOARD. In this case, RPi interprets pin numbers as they are defined on the Raspberry Pi board, and we'd have to use 16 instead with our current setup. (See the two rows in Figure 46, *The Pi's GPIO pins*, on page ?, beginning with Pin.) Finally, we turn the pin to which the sensor is connected into an input pin by calling GPIO.setup() in line 6.

Then we define a method named motion_detected(). It calls GPIO.input(), passing it our signal pin number. Depending on the method call's result, it returns True if the sensor has detected a motion and False otherwise. Again, Python sets the self argument automatically for us.

Our PassiveInfraredSensor class is complete now, so let's use it to build a motion detector. The following program periodically checks whether the PIR sensor has detected motion. It prints a message if someone moves, and it also prints a message if nobody has moved for more than two seconds.

Sensors/pir_demo.py

```
Line 1 from pir import *
   - import time
   -
   - PIR_PIN = 23
   5 pir = PassiveInfraredSensor(PIR_PIN)
   -
   - last_state = False
   - while True:
   -     if (pir.motion_detected() == True):
  10         if (last_state == False):
   -             print "Movement detected"
   -             last_state = True
   -     else:
   -         if (last_state == True):
  15             print "No movement detected"
   -             time.sleep(2)
   -             last_state = False;
```

In the first two lines, we import the PassiveInfraredSensor class and Python's functions for manipulating dates and times. Then we define a constant named PIR_PIN and set it to the number of our signal pin. We use the constant in the following line when we create a new PassiveInfraredSensor object for the first time.

The detection algorithm starts in line 7. We store the last state of the PIR sensor in a variable named last_state. Then we start an endless loop and check to see whether the PIR sensor has currently detected a motion. If yes, we check whether this is a new motion or whether we have detected it before. If it is new, we print the message "Movement detected."

If the PIR sensor hasn't detected a motion, we check whether it has previously detected a motion. If it has, we print the message "No movement detected" and wait for two seconds until we start again. Overall, we make sure that we print a message only if the state has actually changed. This prevents our motion detector from looking a bit nervous.

Now run the program, move a little in front of the PIR sensor, and stand still from time to time.

```
pi@raspberry:~$ sudo python pir_demo.py
Movement detected
No movement detected
```

That's all you need to turn the Pi into a motion detector. Printing a message isn't very spectacular, but you can easily improve the program so that it sends an email or switches on a light. That way, you could turn your Pi into a burglar alarm or make it the basis of a home-automation system.

Attaching most digital sensors to the Pi is easy as long as their output voltage matches the Pi's specifications and as long as they don't depend on an accurate timing. Working with analog sensors can be more complicated, but in the next section you'll see that it isn't rocket science, either.