CamJam EduKit Sensors Worksheet Five

Project Passive Infrared Sensor

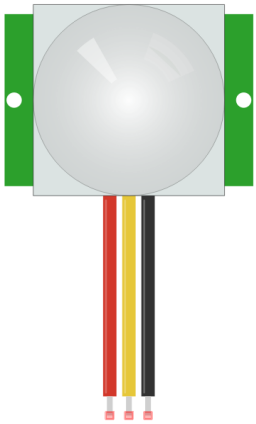
Description In this project, you will learn how to wire and program a passive infrared sensor that detects movement near it.

Equipment Required

- Your Raspberry Pi
- 400 Point Breadboard
- Passive Infrared Sensor
- 6 x m/f jumper wires

The Parts

The Passive Infrared Sensor



The main component of this circuit is itself another circuit board that has a PIR, or Passive Infrared sensor on it. These devices are commonly used in burglar alarms, lights that come on when people approach, and some CCTV cameras.

There are three connectors on the bottom of the PIR, marked VCC, OUT and GND. A 5-volt power supply is applied to VCC pin, with GND pin going to 'ground'. The OUT pin will 'go high' when movement is detected.

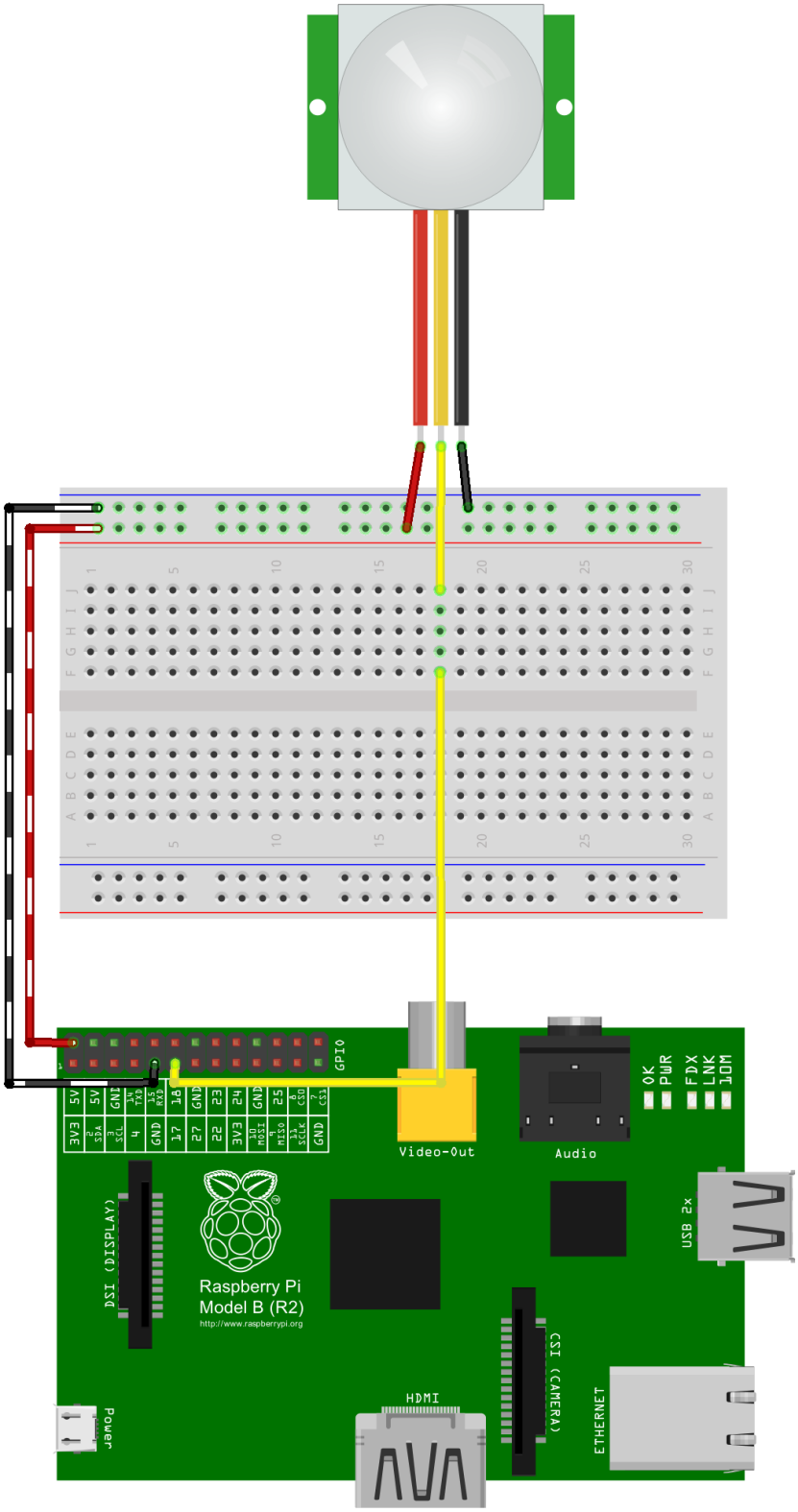
You will notice two 'potentiometers' on the bottom that are used for adjusting the sensitivity (marked Sx) and how long the sensor pin stays high when it senses motion (marked Tx).

To make the PIR more sensitive, turn the Sx potentiometer clockwise with a small screwdriver. To start with, you should set it to the middle.

You may want to experiment with the Tx potentiometer once you have written the code. However, to start with you should turn it all the way anti-clockwise to make the PIR report movement for the shortest time.

You can use Blu-Tack to fix the PIR to a surface if the wires try to turn it over. You may want to protect the sides with a tube or put it in a box so that it does not detect movement from other people in the room.

Building the Circuit



The diagram on the left shows how to connect the PIR sensor. You may leave the buzzer and LED in place from the previous circuit as this will be used in the next worksheet to build an alarm. It is not shown here for clarity.

The PIR circuit is much simpler than the other circuits, mainly because the sensor contains a large amount of its own circuitry.

Power is supplied from the 5v pin, and not the 3.3v that the other circuits use.

Use three jumper wires to connect the PIR pins to the breadboard. The power input pin is marked 'VCC', the negative marked with 'GND', and the sensor pin with 'OUT'. A second jumper wire connects pin 17 to the breadboard.

Code

Open the IDLE3 editor and type in the following code:

```
# CamJam Edukit 2 - Sensors
# Worksheet 5 - Movement

# Import Python header files
import RPi.GPIO as GPIO
import time

# Set the GPIO naming convention
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Set a variable to hold the GPIO Pin identity
PinPIR = 17

print("PIR Module Test (CTRL-C to exit)")

# Set pin as input
GPIO.setup(PinPIR, GPIO.IN)

# Variables to hold the current and last states
Current_State  = 0
Previous_State = 0

try:
    print("Waiting for PIR to settle ...")
    # Loop until PIR output is 0
    while GPIO.input(PinPIR)==1:
        Current_State = 0

    print("  Ready")
    # Loop until users quits with CTRL-C
    while True:
        # Read PIR state
        Current_State = GPIO.input(PinPIR)

        # If the PIR is triggered
        if Current_State==1 and Previous_State==0:
            print("  Motion detected!")
            # Record previous state
            Previous_State=1

        # If the PIR has returned to ready state
        elif Current_State==0 and Previous_State==1:
            print("  Ready")
            Previous_State=0

        # Wait for 10 milliseconds
        time.sleep(0.01)

except KeyboardInterrupt:
    print("  Quit")

    # Reset GPIO settings
```

```
    GPIO.cleanup()
```

Once complete, save the file as 5-PIR.py in the EduKitSensors directory.

Running the Code

Select the Run Module menu option, under the Run menu item. Alternatively, you can just press the F5 key.

When the PIR detects movement, it will print 'Motion detected!' on the screen once and once only. If the movement stops it will return to the steady state.

How the Code Works

The code above introduces a few concepts that may not have been used in the previous worksheets. Let's take a look at some parts of the code that you may not be familiar with. The whole code is not repeated in full below, just the parts that are of interest.

```
PinPIR = 17
```

A variable, `PinPIR`, is being used to store the pin number of the PIR sensor pin. This allows you to change which pin is used in only one place in the code, and makes it easier to code by not having to remember the pin number, just the pin name you have given it.

```
try:
```

The main code is contained within a `try. except` construct. The code within the `try` will continue to be run until the `KeyboardInterrupt` keys are pressed. This is a special key combination that is defined within Python that will interrupt a program when pressed. For the Raspberry Pi, this is 'Ctrl + c', which is pressing the Ctrl key down and pressing the 'c' key.

```
    while GPIO.input(PinPIR)==1:
        Current_State = 0
```

In the first `while` loop after the `try`, the code first waits until the PIR does not see any movement. The `Current_State` variable is set to 0, indicating no movement.

```
    while True:
```

The code then enters an 'eternal' loop; `while True:` means that the loop will always run unless the interrupt keys are pressed.

```
        Current_State = GPIO.input(PinPIR)
```

The `Current_State` is then set to the value of the input pin. If there is no movement, this will be `0`. If there is movement, this will be `1`.

```
        # If the PIR is triggered
        if Current_State==1 and Previous_State==0:
            print("  Motion detected!")
            # Record previous state
            Previous_State=1
```

If the PIR has been triggered, but on the last check it was not, then you will be notified by the message "Motion detected!". The 'previous state' will then be set to show that motion has been detected.

```
        elif Current_State==0 and Previous_State==1:
            print =("  Ready")
            Previous_State=0
```

If the current state shows that there is no movement, but the previous state shows that there was movement, then you will be notified that everything is still around the sensor with the message "Ready".

```
        # Wait for 10 milliseconds
        time.sleep(0.01)
```

The code then sleeps for 0.01 of a second. This is here to stop the code from continuously flipping between seeing movement and not seeing movement.

```
except KeyboardInterrupt:
    print("  Quit")

    # Reset GPIO settings
    GPIO.cleanup()
```

If the interrupt keys are pressed (Ctrl+c), the program will end, but before it does, the GPIO pins will be reset to their default state.