C. RODRIGUES	TP Internet of Things

TP 2 : Tutorial

Goals:

1. **Learn to manipulate a Raspberry Pi**
2. **Learn to assemble and use different devices with a Raspberry Pi**
3. **Implement simple applications in Python, using sensors and actuators connected to the Raspberry**

I – Tutorial:

Before proceeding to the exercises, you MUST complete the following examples to understand how to assemble and manipulate the devices we are going to use during this course.

1. Servo

A Servomotor, or servo, is a small device that has an output shaft. This shaft can be positioned to specific angular positions by sending the servo a coded signal. As long as the coded signal exists on the input line, the servo will maintain the angular position of the shaft. As the coded signal changes, the angular position of the shaft changes. Servos are extremely useful in practice. They may be used to operate remote-controlled toy cars, robots, or airplanes. Servos are also used in industrial applications, robotics, in-line manufacturing, pharmaceutics, and food services.

Image Source: https://goo.gl/9pLJLz

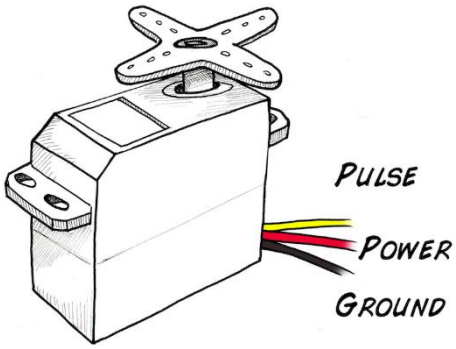
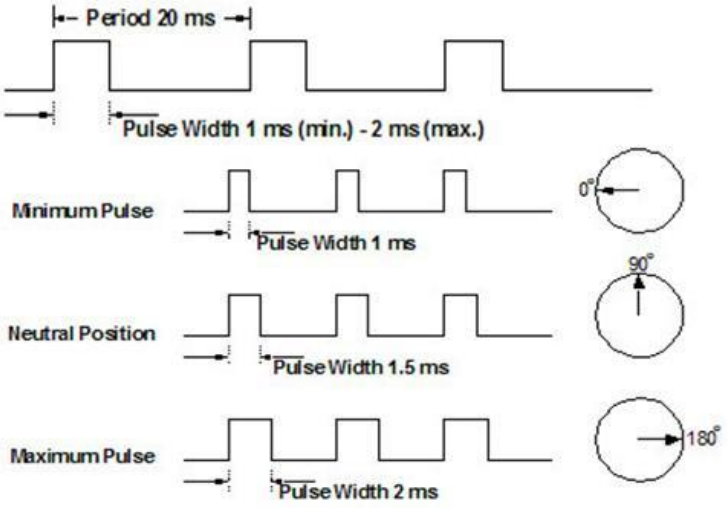


Image Source: https://goo.gl/KxRNZs

How does Servo work?

Servos are controlled by sending an electrical pulse of variable width, or pulse width modulation (PWM), through the control wire. There is a minimum pulse, a maximum pulse, and a repetition rate. A servo motor can usually only turn 90 degrees in either direction for a total of 180 degree movement. The motor's neutral position is defined as the position where the servo has

the same amount of potential rotation in the both the clockwise and counterclockwise direction. The PWM sent to the motor determines position of the shaft and based on the duration of the pulse sent via the control wire; the rotor will turn to the desired position. The servo motor expects to see a pulse every 20 milliseconds (ms) and the length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90-degree position. Shorter than 1.5ms moves it to 0 degrees, and any longer than 1.5ms will turn the servo to 180 degrees, as diagrammed below:



When these servos are commanded to move, they will move to the position and hold that position. If an external force pushes against the servo while the servo is holding a position, the servo will resist from moving out of that position. The maximum amount of force the servo can exert is called the torque rating of the servo. Servos will not hold their position forever though; the position pulse must be repeated to instruct the servo to stay in position.

Controlling a servo from the Raspberry Pi

To control the servo motor from the Raspberry Pi we are going to use the PWM module in RPi.GPIO. The first step is to create the PWM instance associated with the GPIO pin:

```
GPIO.setup(12, GPIO.OUT)
p = GPIO.PWM(12, 50)
```

In the above case we have instantiated the PWM module for the pin number 12 with a frequency of 50Hz. That frequency was selected because the servo motor expects a pulse every 20ms (period), that means 50 pulses per second or Hertz (1 / 50 = 0.02 = 20 ms). Once instantiated the PWM module, to start sending a pulse we do:

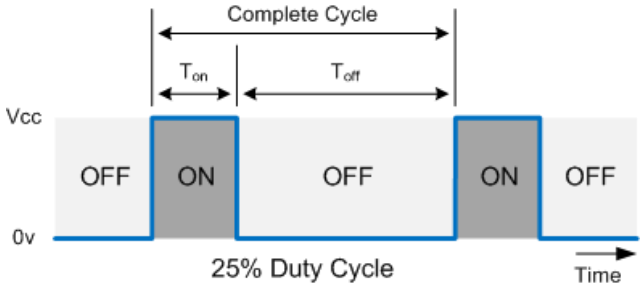
```
p.start(dc)
```

In this case dc is the duty cycle. The duty cycle describes the proportion of on time to the regular interval or period of time. If we want a pulse with an specific length we can calculate the duty cycle as follows:

$$dc = \frac{length}{period}$$

Since the servo uses 20ms cycles, we can calculate the duty cycle of the 3 turns of the servo motor:

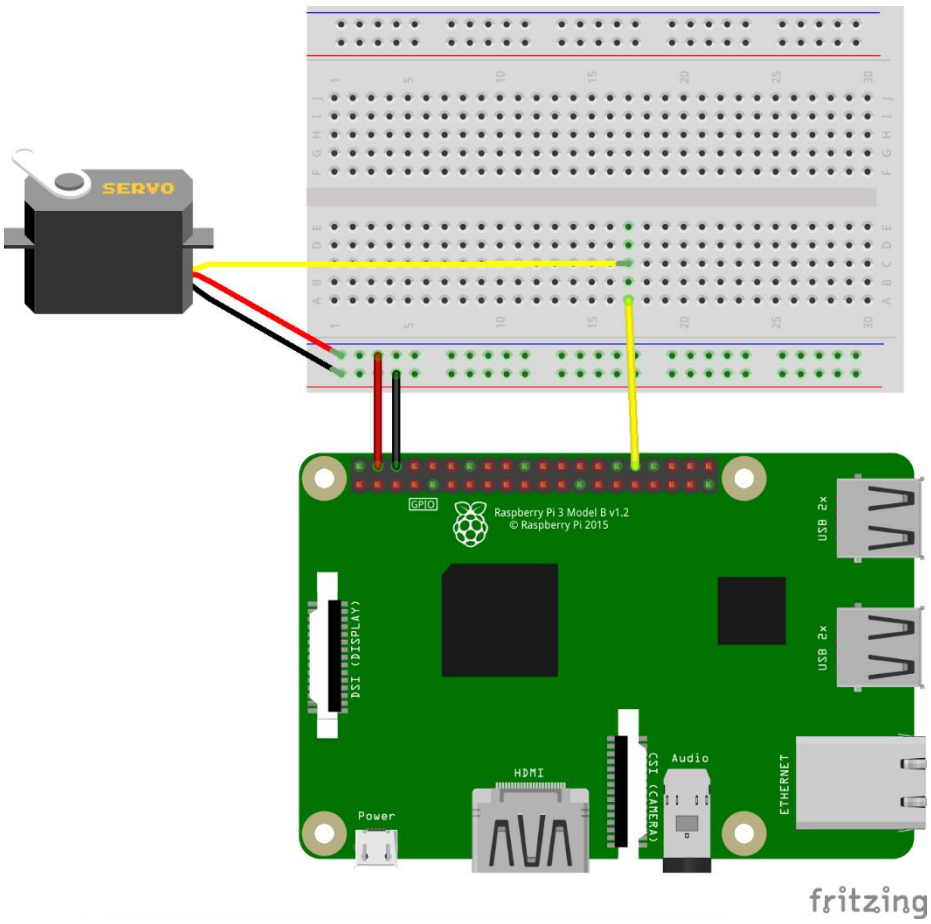
$$dc = \frac{0.5}{20} \times 100 = 2.5\%$$
$$dc = \frac{1.5}{20} \times 100 = 7.5\%$$
$$dc = \frac{2.5}{20} \times 100 = 12.5\%$$



To change the duty cycle we can use:
`p.ChangeDutyCycle(dc)`

and to stop the pulse emission:
`p.stop()`

Example of controlling a servo from the Raspberry Pi



The following diagram shows how to connect the servo to the Raspberry Pi:

The following program will control the servo making it move to its neutral position (90 degrees), wait 1 second and then move to its 0 degrees, wait 1 second and finally move to its 180 degrees. The cycle continues until interrupted:

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(12, GPIO.OUT)
p = GPIO.PWM(12, 50)
p.start(2.5) # initial position
try:
    while True:
        p.ChangeDutyCycle(7.5)  # turn towards 90 degree
        time.sleep(1) # sleep 1 second
        p.ChangeDutyCycle(2.5)  # turn towards 0 degree
        time.sleep(1) # sleep 1 second
        p.ChangeDutyCycle(12.5) # turn towards 180 degree
        time.sleep(1) # sleep 1 second
except KeyboardInterrupt:
    p.stop() # stop the pulse emission
    GPIO.cleanup()
```

2. PIR Motion Sensor

PIR sensors allow you to sense motion, almost always used to detect whether a human has moved in or out of the sensors range. They are small, inexpensive, low-power, easy to use and don't wear out. For that reason, they are commonly found in appliances and gadgets used in homes or businesses. They are often referred to as PIR, "Passive Infrared", "Pyroelectric", or "IR motion" sensors.

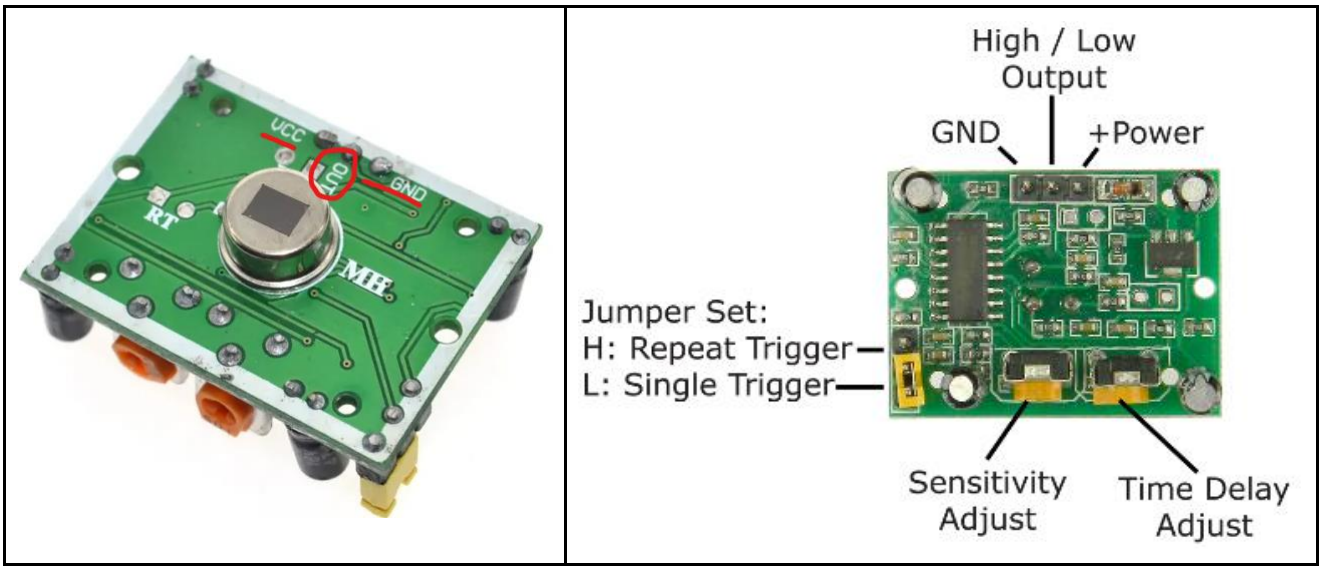
PIRs are basically made of a pyroelectric sensor, which can detect levels of infrared radiation. Everything emits some low-level radiation, and the hotter something is, the more radiation is emitted. The sensor in a motion detector is actually split in two halves. The reason for that is that we are looking to detect motion (change) not average IR levels. The two halves are wired up so that they cancel each other out. If one half sees more or less IR radiation than the other, the output will swing high or low.

For many basic projects or products that need to detect when a person has left or entered the area, or has approached, PIR sensors are great. They are low power and low cost, pretty rugged, have a wide lens range, and are easy to interface with. Note that PIRs won't tell you how many people are around or how close they are to the sensor, the lens is often fixed to a certain sweep and distance and they are also sometimes set off by housepets.

Connecting to a PIR

Most PIR modules have a 3-pin connection at the side or bottom. One pin will be ground, another will be signal and the final one will be power. Power is usually 3-5VDC input but may be as high as 12V. The output signal is digital with high pulse (usually 3.3V) when triggered (motion detected) and low (0V) when idle (no motion detected).

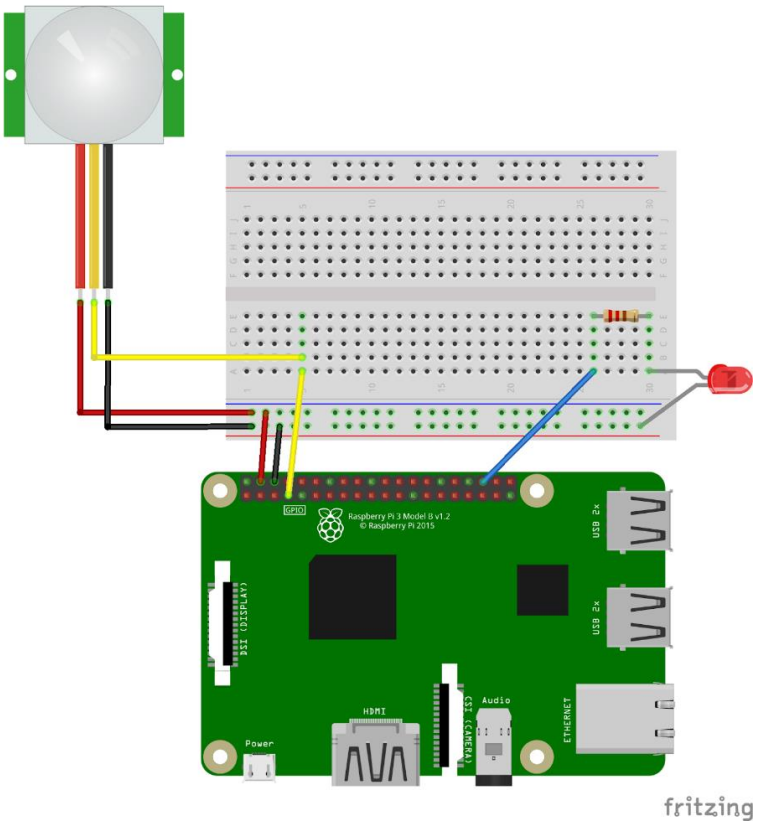
There are three pins on the PIR: they should be labelled Vcc, Gnd, and Out. **These labels are sometimes concealed beneath the Fresnel lens (the white cap), which you can temporarily remove to see the pin labels.**



Example

Connecting PIR sensors to a Raspberry Pi is really simple. The PIR acts as a digital output so all you need to do is listen for the pin to flip high (detected) or low (not detected).

Power the PIR with 5V and connect ground to ground, then connect the output to a GPIO pin. In this example we'll use the GPIO 4. The code is very simple, and it basically just keeps track of whether the input to GPIO 4 is high or low. It prints out a message and turns a LED on when it detects motion; and it turns a LED off when idle.

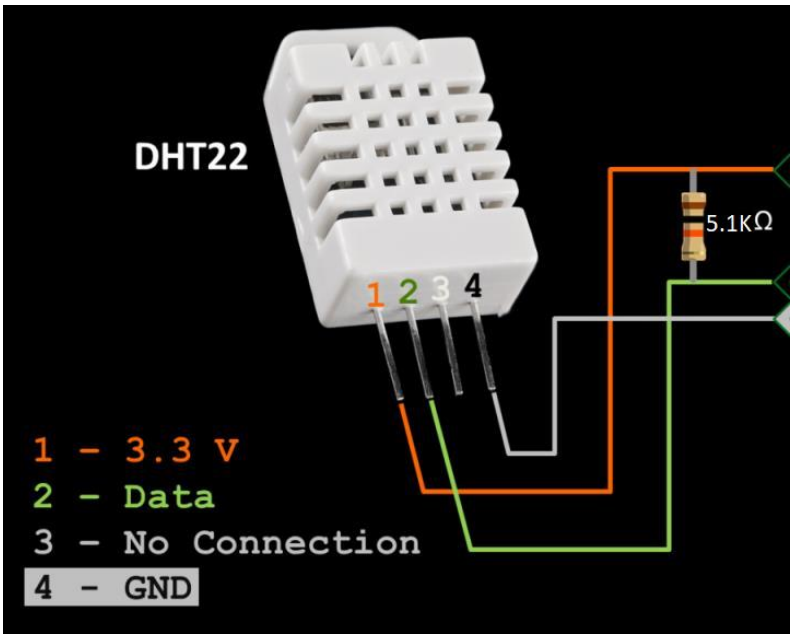


```
import RPi.GPIO as GPIO
import time

PIR_MOTION = 4
LED = 16

GPIO.setmode(GPIO.BCM)
GPIO.setup(PIR_MOTION, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
GPIO.setup(LED, GPIO.OUT)

GPIO.output(LED, False)
try:
    while True:
        if GPIO.input(PIR_MOTION):
            # When output from motion sensor is HIGH
            print("Motion Detected")
            GPIO.output(LED, True)
        else:
            # When output from motion sensor is LOW
            GPIO.output(LED, False)
except KeyboardInterrupt:
    GPIO.cleanup()
```

3. Temperature Sensor

DHT22

The DHT22 is a basic, low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air and spits out a digital signal on the data pin (no analog input pins needed). It's fairly simple to use but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once **every 2 seconds**, so when using our library, sensor readings can be up to 2 seconds old.



To use this sensor we are going to use Adafruit library that supports supports a variety of sensors.

First of all, some packages have to be installed:

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-openssl git
```

To download and use the library:

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git && cd Adafruit_Python_DHT
sudo python setup.py install
```

This creates a Python library that we can easily integrate into our projects.
If everything went well, we can already read the temperature and humidity. The easiest way is to first use the demo files:

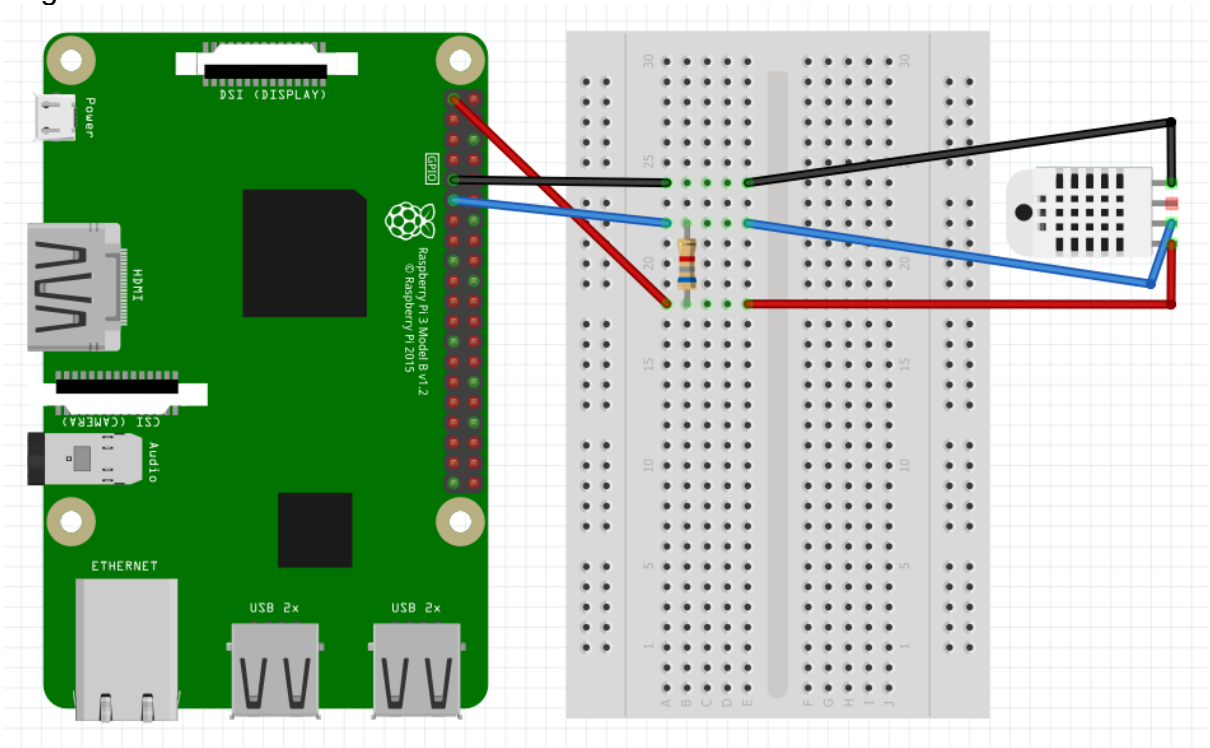
```
cd examples
sudo ./AdafruitDHT.py 22 17
```

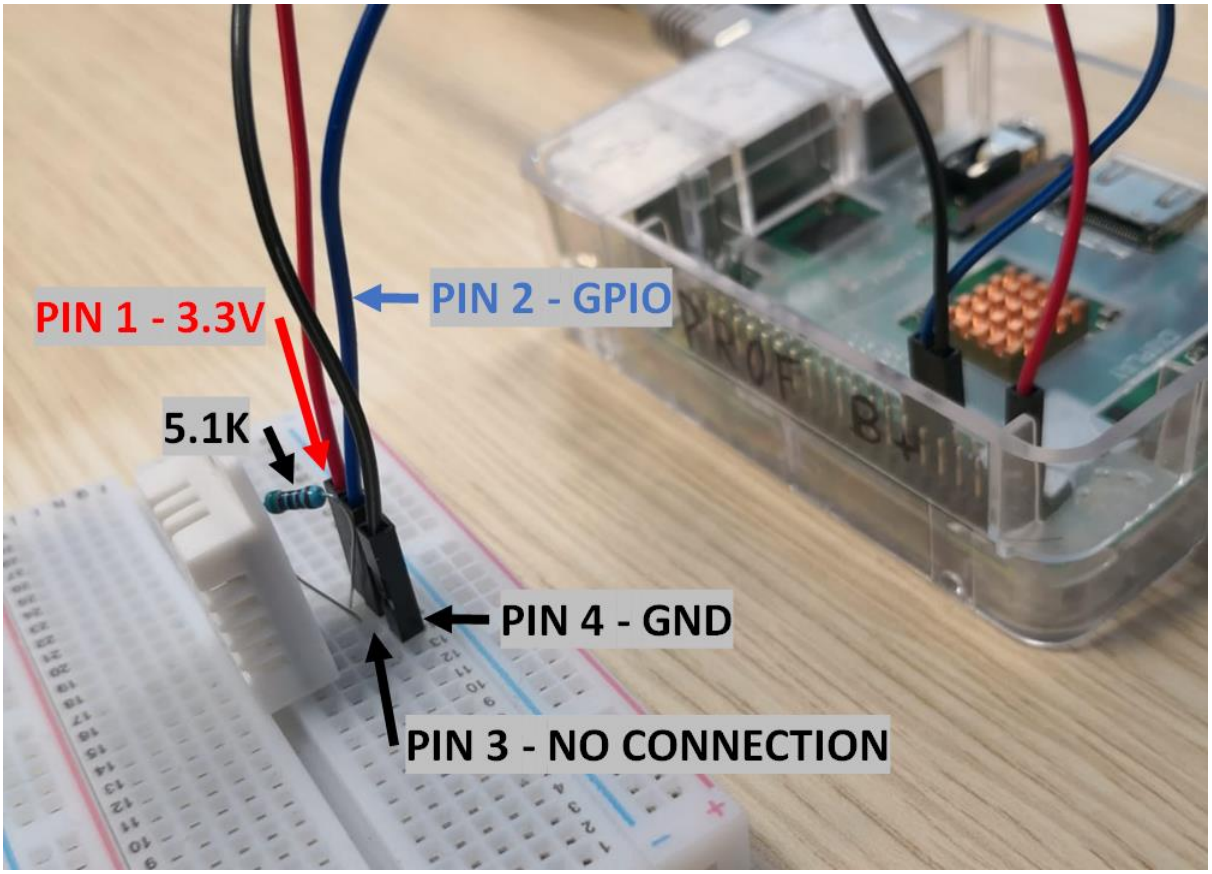
The first parameter (22) indicates which sensor was used (22 for the DHT22) and the second, to which GPIO it is connected **(not the pin number**, but the GPIO number). This produces an output like the following:

```
Temp=24.0*  Humidity=41.0%
```

Attention: The sensors are only ready every two seconds. Be careful not to start a query every second in order to avoid too repeated entries.

To reproduce the schema bellow you need to use a resistor between 4.7K - 10K. For our case we are going to use a **5.1K resistor**.




PIN 1 - 3.3V
PIN 2 - GPIO
5.1K
PIN 4 - GND
PIN 3 - NO CONNECTION

4. Sonar

Sonar is a system to detect objects under water by emitting sound pulses and recognizing or measuring their return after being reflected. A typical sonar used in robotic is the HC-SR04 ultrasonic sensor.

The HC-SR04 ultrasonic sensor uses sonar to measure the distance to an object. It offers excellent range accuracy and stable readings in an easy-to-use package. Its operation is not affected by sunlight or black material like Sharp rangefinders are (soft materials like cloth can be difficult to detect).

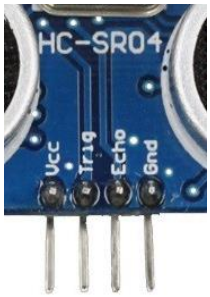
HC-SR04 main technical parameters:
1.Working Voltage : 5V(DC)
2.Static current: Less than 2mA.
3.Output signal: Electric frequency signal, high level 5V, low level 0V.
4.Sensor angle: Not more than 15 degrees.
5.Detection distance: 2cm-450cm.
6.High precision: Up to 0.3cm
7.Input trigger signal: 10us TTL impulse
8.Echo signal : output TTL PWL signal

Obs: *These parameters can change from one manufacturer to another, always looking through the DataSheet*

How does Ultrasonic work?

A basic ultrasonic sensor consists of one or more ultrasonic transmitters (basically speakers), a receiver, and a control circuit. The transmitters emit a high frequency ultrasonic sound, which bounce off any nearby solid objects. Some of that ultrasonic noise is reflected and detected by the receiver on the sensor. That return signal is then processed by the control circuit to calculate the time difference between the signal being transmitted and received. This time can subsequently be used with some math to calculate the distance between the sensor and the reflecting object.

How does HC-SR04 Sensor work?

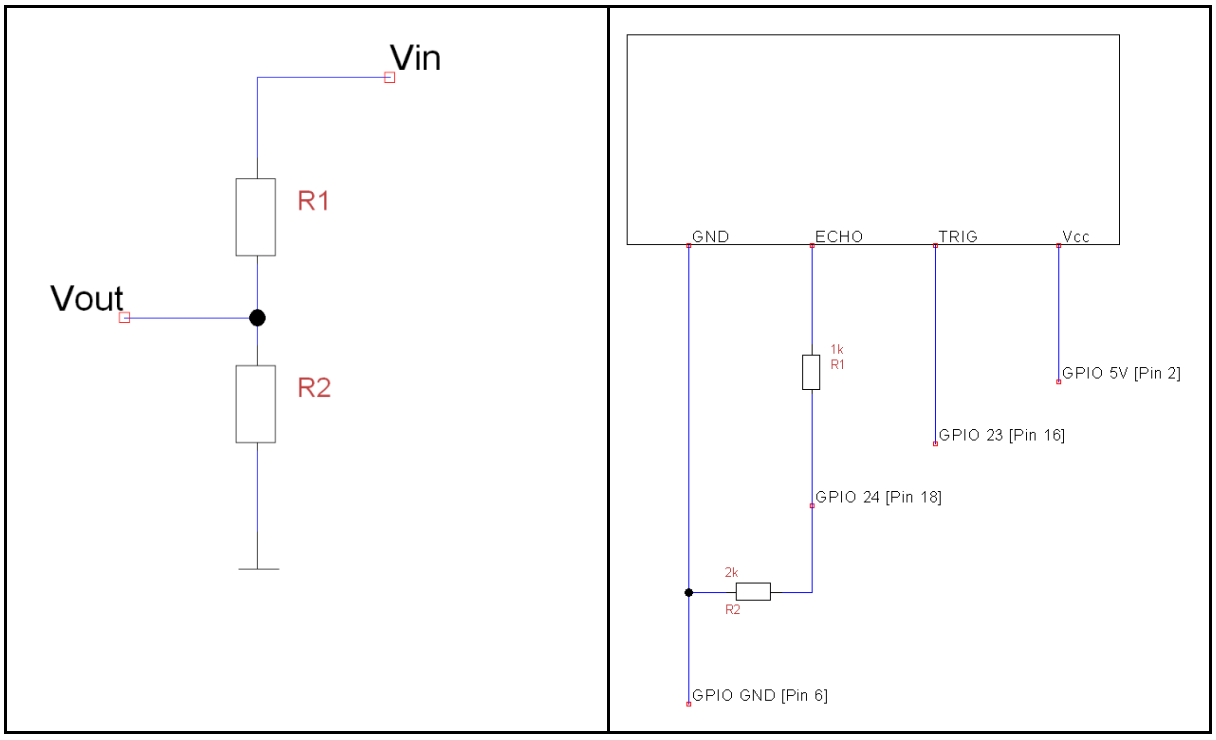


The HC-SR04 Ultrasonic sensor has four pins: ground (**GND**), Echo Pulse Output (**ECHO**), Trigger Pulse Input (**TRIG**), and 5V Supply (**Vcc**). Raspberry Pi sends an input signal to TRIG, which triggers the sensor to send an ultrasonic pulse. The pulse waves bounce off any nearby objects and some are reflected back to the sensor. The sensor detects these return waves and measures the time between the trigger and returned pulse, and then sends a 5V signal on the **ECHO** pin.

ECHO will be "low" (0V) until the sensor is triggered when it receives the echo pulse. Once a return pulse has been located, ECHO is set "high" (5V) for the duration of that pulse. Pulse duration is the full time between the sensor outputting an ultrasonic pulse and the return pulse being detected by the sensor receiver. Our Python script must therefore measure the pulse duration and then calculate the distance from this.

IMPORTANT. The sensor output signal (**ECHO**) on the HC-SR04 is rated at 5V. However, the input pin on the Raspberry Pi GPIO is rated at 3.3V. Sending a 5V signal into that unprotected 3.3V input port could damage your GPIO pins. We'll need to use a small voltage divider circuit with two resistors to lower the sensor output voltage to something our Raspberry Pi can handle.

A voltage divider consists of two resistors (R1 and R2) in series connected to an input voltage (Vin), which needs to be reduced to our output voltage (Vout). In our circuit, Vin will be ECHO, which needs to be decreased from 5V to our Vout of 3.3V.



The following circuit and simple equation can be applied to many applications where a voltage needs to be reduced. If you don't want to learn the techy bit, just grab 1 x 1kΩ and 1 x 2kΩ resistor.

$$Vout = Vin \times \frac{R2}{R1 + R2}$$

$$\frac{Vout}{Vin} = \frac{R2}{R1 + R2}$$

Without getting too deep into the math side, we only need to calculate one resistor value, as dividing ratio what matters. We know our input voltage (5V), and our required output voltage (3.3V), and we can use any combination of resistors to achieve the reduction. We used 1kΩ resistors in the circuit as R1.

It would happen as follows:

$$\frac{3.3}{5} = \frac{R2}{1000 + R2}$$

$$0.66 = \frac{R2}{1000 + R2}$$

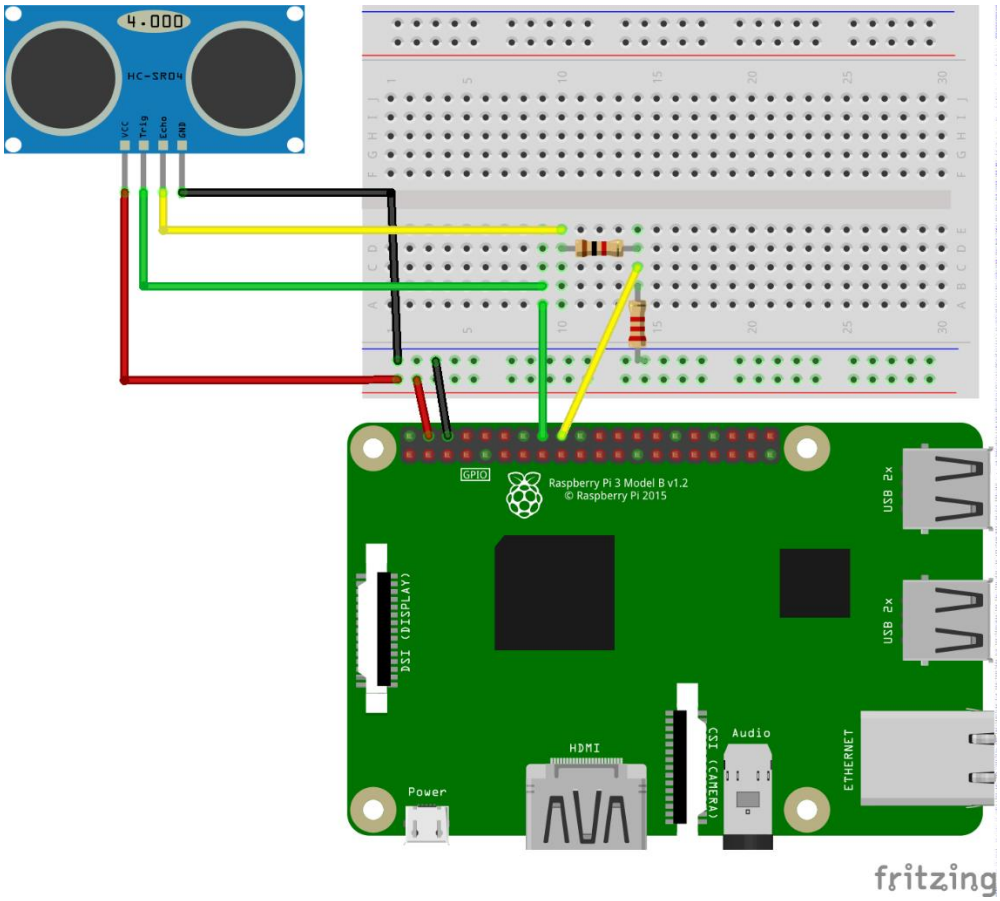
$$0.66(1000 + R2) = R2$$

$$660 + 0.66R2 = R2$$

$$660 = 0.34R2$$

$$1941 = R2$$

So, we'll use a 1kΩ for R1 and a 2kΩ resistor as R2!



Code

```
import RPi.GPIO as
GPIO
import time

GPIO.setmode(GPIO.BCM)

TRIG = 23
ECHO = 24

print ("Distance Measurement In Progress")
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)

GPIO.output(TRIG, False)
print("Waiting For Sensor To Settle")
time.sleep(2)

try:
    while True:
        GPIO.output(TRIG, True)
        time.sleep(0.00001)
        GPIO.output(TRIG, False)
        pulse_start = 0
        pulse_end = 0

        while GPIO.input(ECHO)==0:
            pulse_start = time.time()

        while GPIO.input(ECHO)==1:
            pulse_end = time.time()

        pulse_duration = pulse_end - pulse_start
        distance = pulse_duration * 17150
        distance = round(distance, 2)
        print("Distance:" + str(distance)+ "cm")

except KeyboardInterrupt:
    GPIO.cleanup()
```

Explanation of the Code

The Ultrasonic sensor output (TRIG) will output low (0V) unless it's been triggered in which case it will output 5V (3.3V with our voltage divider!). We therefore need to set one GPIO pin as an output, to trigger the sensor, and one as an input to detect the ECHO voltage change.

First, import the Python GPIO library and our time library so we make our Pi wait between steps and set our GPIO pin numbering.

```
import RPi.GPIO as GPIO

import time

GPIO.setmode(GPIO.BCM)
```

Next, we need to name our input and output pins, so that we can refer to them later in our Python code. We'll name our output pin (which triggers the sensor) GPIO 23 [Pin 16] as TRIG, and our input pin (which reads the return signal from the sensor) GPIO 24 [Pin 18] as ECHO.

```
TRIG = 23

ECHO = 24
```

Next, set your two GPIO ports as either inputs or outputs as previously defined.

```
GPIO.setup(TRIG,GPIO.OUT)

GPIO.setup(ECHO,GPIO.IN)
```

Then, ensure that the Trigger pin is set low, and give the sensor a second to settle.

```
GPIO.output(TRIG, False)

print "Waiting For Sensor To Settle"

time.sleep(2)
```

The HC-SR04 sensor requires a short 10uS pulse to trigger the module, which will cause the sensor to start the ranging program (8 ultrasound bursts at 40 kHz) in order to obtain an echo response. So, to create our trigger pulse, we set out trigger pin high to 10uS then set it low again.

```
GPIO.output(TRIG, True)

time.sleep(0.00001)

GPIO.output(TRIG, False)
```

Now that we've sent our pulse signal, we need to listen to our input pin, which is connected to ECHO. The sensor sets ECHO to high for the amount of time it takes for the pulse to go and come back, so our code needs to measure the amount of time that the ECHO pin stays high. We use the "while" string to ensure that each signal timestamp is recorded in the correct order.

The time.time() function will record the latest timestamp for a given condition. For example, if a pin goes from low to high, and we're recording the low condition using the time.time() function, the recorded timestamp will be the latest time at which that pin was low.

Our first step must be to record the last low timestamp for ECHO (pulse_start) e.g. just before the return signal is received and the pin goes high.

```
while GPIO.input(ECHO)==0:

  pulse_start = time.time()
```

Once a signal is received, the value changes from low (0) to high (1), and the signal will remain high for the duration of the echo pulse. We also need the last high timestamp for ECHO (pulse_end).

```
while GPIO.input(ECHO)==1:

  pulse_end = time.time()
```

We can now calculate the difference between the two recorded timestamps, and the duration of pulse (pulse_duration).

```
pulse_duration = pulse_end - pulse_start
```

With the time it takes for the signal to travel to an object and back, we can calculate the distance using the following formula:

$$Speed = \frac{Distance}{Time}$$

The speed of sound varies according to where it is travelling through and the temperature. However, some clever physicists have calculated the speed of sound at sea level so we'll take our baseline as the 343m/s. Do not measure distance through water and make sure you're using the right speed of sound.

We also need to divide our time by two because what we've calculated above is actually the time it takes for the ultrasonic pulse to travel the distance to the object and back. We only want the distance to the object and we can simplify the calculation to be completed in our Python script as follows:

$$34300 = \frac{Distance}{Time/2}$$

$$17150 = \frac{Distance}{Time}$$

$$17150 \times Time = Distance$$

We can plug this calculation into our Python script:

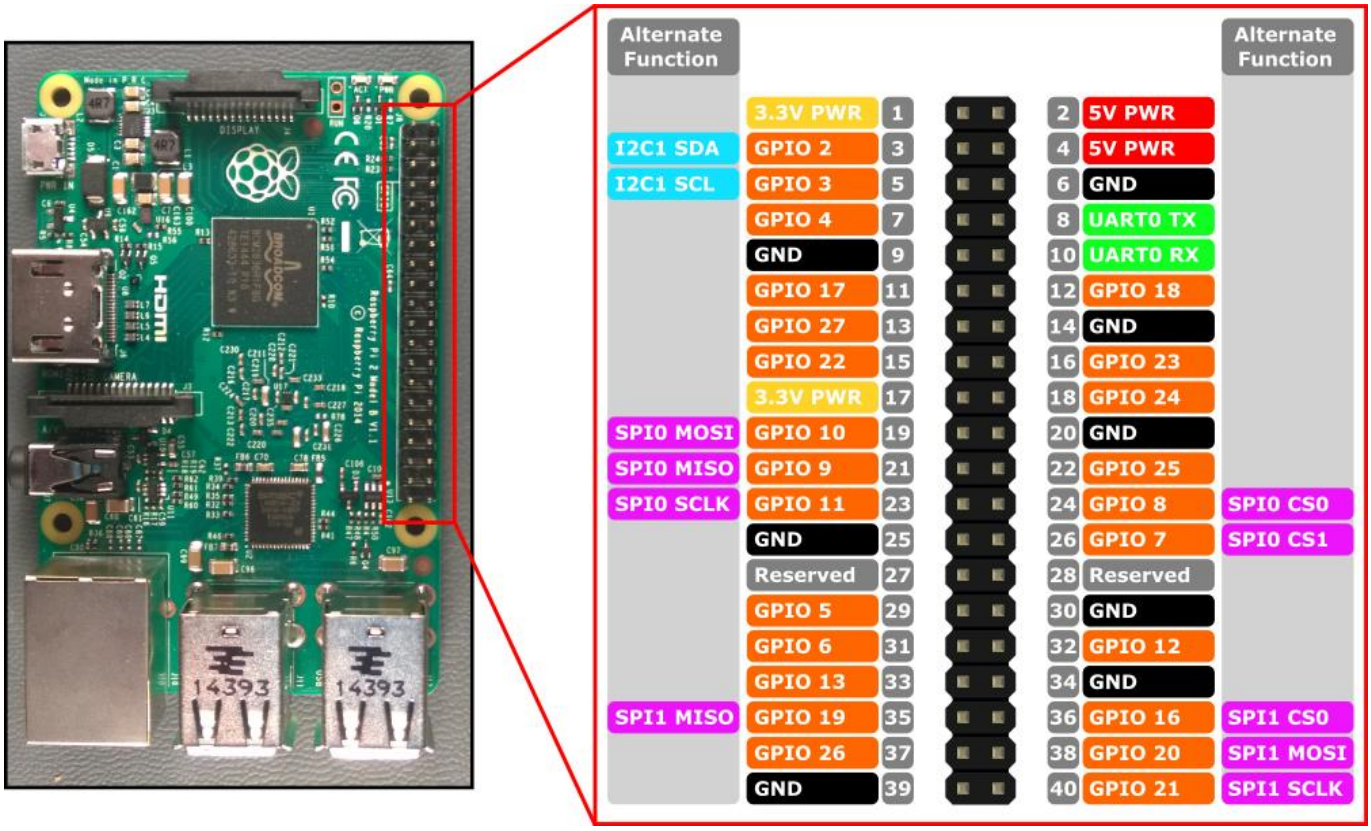
```
distance = pulse_duration x 17150
```

Now we need to round our distance to 2 decimal places (for neatness!)

```
distance = round(distance, 2)
```

Then, we print the distance. The below command will print the word "Distance:" followed by the distance variable, and by the unit "cm"

```
print "Distance:",distance,"cm"
```



For more information about Raspberry Pi GPIO ports, check:

https://pinout.xyz