LAB MANUAL 2

Interfacing Analog Sensors

Temperature Sensor

A temperature sensor is a device that is designed to measure the degree of hotness or coolness in an object. The working of a temperature meter depends upon the voltage across the diode. The temperature change is directly proportional to the diode's resistance. The cooler the temperature, the lesser will be the resistance, and vice-versa.
The resistance across the diode is measured and converted into readable units of temperature (Fahrenheit, Celsius, Centigrade, etc.) and, displayed in numeric form over readout units. In the geotechnical monitoring field, these temperature sensors are used to measure the internal temperature of structures like bridges, dams, buildings, power plants, etc.

How does a temperature sensor work?

The basic principle of working the temperature sensors is the voltage across the diode terminals. If the voltage increases, the temperature also rises, followed by a voltage drop between the transistor terminals of the base and emitter in a diode.
Besides this, Encardio Rite has a vibrating wire temperature sensor that works on the principle of stress change due to temperature change.

Types of temperature sensors

Temperature sensors are available in various types, shapes, and sizes. The two main types of temperature sensors are:

1. **Contact Type Temperature Sensors**: There are a few temperature meters that measure the degree of hotness or coolness in an object by being in direct contact with it. Such temperature sensors fall under the category of contact type. They can be used to detect solids, liquids, or gases over a wide range of temperatures.
2. **Non-Contact Type Temperature Sensors**: These types of temperature meters are not in direct contact with the object rather, they measure the degree of hotness or coolness through the radiation emitted by the heat source.

The contact and non-contact temperature sensors are further divided into:
1. Thermostats

Thermostat

A thermostat is a contact-type temperature sensor consisting of a bi-metallic strip made up of two dissimilar metals such as aluminum, copper, nickel, or tungsten. The difference in the coefficient of linear expansion of both metals causes them to produce a mechanical bending movement when it’s subjected to heat.

2. Thermistors

Thermistor

Thermistors or thermally sensitive resistors are the ones that change their physical appearance when subjected to a change in temperature. The thermistors are made up of ceramic material such as oxides of nickel, manganese, or cobalt coated in glass which allows them to deform easily.

3. Resistive Temperature Detectors (RTD)

RTD

RTDs are precise temperature sensors that are made up of high-purity conducting metals such as platinum, copper, or nickel wound into a coil. The electrical resistance of an RTD changes similar to that of a thermistor.

4. Thermocouples

Thermocouple

One of the most common temperature sensors includes thermocouples because of their wide temperature operating range, reliability, accuracy, simplicity, and sensitivity. A thermocouple usually consists of two junctions of dissimilar metals, such as copper and constantan that are welded or crimped together. One of these junctions, known as the Cold junction, is kept at a specific temperature while the other one is the measuring junction, known as the Hot junction. n being subjected to temperature, a voltage drop is developed across the junction.

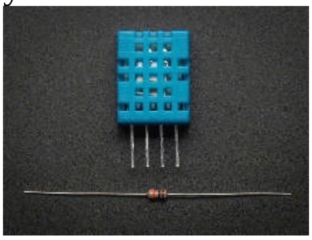
5. Negative Temperature Coefficient (NTC) Thermistor

NTC Thermistor

A thermistor is basically a sensitive temperature sensor that reacts precisely to even minute temperature changes. It provides a huge resistance at very low

temperatures. This means, that as soon as the temperature starts increasing, the resistance starts dropping quickly.
Due to the large resistance change per degree Celsius, even a small temperature change is displayed accurately by the Negative Temperature Coefficient (NTC) Thermistor. Because of this exponential working principle, it requires linearization. They usually work in the range of -50 to 250 °C.

6. DHT11 temperature-humidity sensor



DHT11

The DHT11 is a basic, ultra 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). Its fairly simple to use, but requires careful timing to grab data. You can get new data from it once every 2 seconds.
Comes with a 4.7K or 10K resistor, which you will want to use as a pullup from the data pin to VCC.

Specifications:

1. 3 to 5V power and I/O
2. 2.5mA max current use during conversion (while requesting data)
3. Good for 20-80% humidity readings with 5% accuracy
4. Good for 0-50 °C temperature readings +-2 °C accuracy
5. No more than 1 Hz sampling rate (once every second)
6. Body size 15.5mm x 12mm x 5.5mm
7. 4 pins with 0.1" spacing

DHT Interfacing using GrovePI & Raspberry-PI

Steps-

1. Connect DHT sensor to port D7
2. Connect LCD display to any I2C port
3. Run below code to fetch data from sensor

Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/GrovePI_Codes/9_DHT_Sensor.py

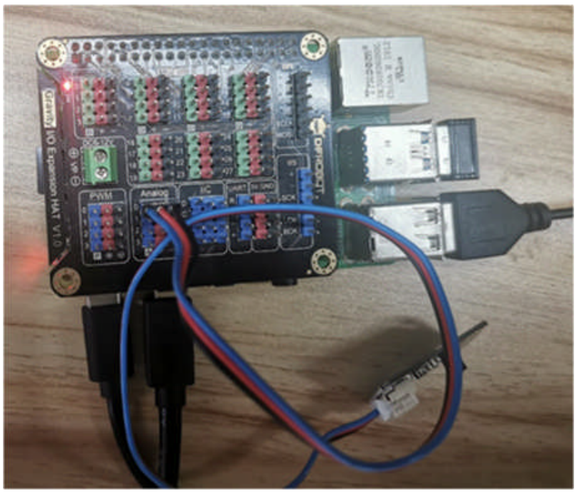
Code-
```
from grovepi import *
from grove_rgb_lcd import *
import time
dht_sensor_port = 7
dht_sensor_type = 0 # 0 for DHT11 and 1 for DHT22
setRGB(0,255,0)
while True:
    [t,h] = dht(dht_sensor_port,dht_sensor_type)
    print(f"Temp:{t} C Humidity:{h}%")
    setText_norefresh(f"Temp:{t} C\nHumidity:{h}%")
    time.sleep(2)
```

Temperature Sensor Interfacing using DFRobot Hat & Raspberry-PI

Based on LM35 semiconductor, LM35 temp sensor of DFRobot produced by National Semiconductor Corporation can be used to detect ambient temperature. It offers a measurement range from －40 °C to 150 °C and a sensitivity of 10 mV/°C. And its output voltage is proportional to the temperature. Moreover, if used in combination with sensor-specific expansion of Arduino board, this sensor can be really easy to achieve interactive effects related to ambient temperature perception. Commonly-used sensors for temperature measurement include thermocouples, platinum resistance, thermal resistance and temperature semiconductor chips. Thermocouples are commonly used in high temperature measurement. Platinum resistance temperature modules are used in measurement of 800 degrees Celsius, while the thermal resistance and semiconductor temperature sensor are suitable for measuring the temperature of 100-200 degrees or below. With good linearity and high sensitivity, the semiconductor temperature sensor is easy to use.

Use LM35 Analog Linear Temperature Sensor on Your Raspberry Pi

- Connect the sensor to the analog pin 0 on the expansion board



- Open Thonny Python IDE to copy the following program into it

Github link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/DFRobot_IoT_Codes/DFRobot_RaspberryPi_Expansion_Board.py

Code

Use below code to inherit the Class DFRobot_Expansion_Board. Save below code as DFRobot_RaspberryPi_Expansion_Board.py

```
import time

_PWM_CHAN_COUNT = 4
_ADC_CHAN_COUNT = 4

class DFRobot_Expansion_Board:

  _REG_SLAVE_ADDR = 0x00
  _REG_PID = 0x01
  _REG_VID = 0x02
  _REG_PWM_CONTROL = 0x03
  _REG_PWM_FREQ = 0x04
  _REG_PWM_DUTY1 = 0x06
  _REG_PWM_DUTY2 = 0x08
  _REG_PWM_DUTY3 = 0x0a
  _REG_PWM_DUTY4 = 0x0c
```

```
  _REG_ADC_CTRL = 0x0e
  _REG_ADC_VAL1 = 0x0f
  _REG_ADC_VAL2 = 0x11
  _REG_ADC_VAL3 = 0x13
  _REG_ADC_VAL4 = 0x15

  _REG_DEF_PID = 0xdf
  _REG_DEF_VID = 0x10

  ''' Enum board Analog channels '''
  A0 = 0x00
  A1 = 0x01
  A2 = 0x02
  A3 = 0x03

  ''' Board status '''
  STA_OK = 0x00
  STA_ERR = 0x01
  STA_ERR_DEVICE_NOT_DETECTED = 0x02
  STA_ERR_SOFT_VERSION = 0x03
  STA_ERR_PARAMETER = 0x04

  ''' last operate status, users can use this variable to determine the result of a
function call. '''
  last_operate_status = STA_OK

  ''' Global variables '''
  ALL = 0xffffffff

  def _write_bytes(self, reg, buf):
    pass

  def _read_bytes(self, reg, len):
    pass

  def __init__(self, addr):
    self._addr = addr
    self._is_pwm_enable = False

  def begin(self):
```

```
  '''
    @brief    Board begin
    @return   Board status
  '''
  pid = self._read_bytes(self._REG_PID, 1)
  vid = self._read_bytes(self._REG_VID, 1)
  if self.last_operate_status == self.STA_OK:
    if pid[0] != self._REG_DEF_PID:
      self.last_operate_status = self.STA_ERR_DEVICE_NOT_DETECTED
    elif vid[0] != self._REG_DEF_VID:
      self.last_operate_status = self.STA_ERR_SOFT_VERSION
    else:
      self.set_pwm_disable()
      self.set_pwm_duty(self.ALL, 0)
      self.set_adc_disable()
  return self.last_operate_status

def set_addr(self, addr):
  '''
    @brief    Set board controler address, reboot module to make it effective
    @param address: int    Address to set, range in 1 to 127
  '''
  if addr < 1 or addr > 127:
    self.last_operate_status = self.STA_ERR_PARAMETER
    return
  self._write_bytes(self._REG_SLAVE_ADDR, [addr])

def _parse_id(self, limit, id):
  ld = []
  if isinstance(id, list) == False:
    id = id + 1
    ld.append(id)
  else:
    ld = [i + 1 for i in id]
  if ld == self.ALL:
    return range(1, limit + 1)
  for i in ld:
    if i < 1 or i > limit:
      self.last_operate_status = self.STA_ERR_PARAMETER
      return []
```

```
    return ld

  def set_pwm_enable(self):
    '''
      @brief    Set pwm enable, pwm channel need external power
    '''
    self._write_bytes(self._REG_PWM_CONTROL, [0x01])
    if self.last_operate_status == self.STA_OK:
      self._is_pwm_enable = True
    time.sleep(0.01)

  def set_pwm_disable(self):
    '''
      @brief    Set pwm disable
    '''
    self._write_bytes(self._REG_PWM_CONTROL, [0x00])
    if self.last_operate_status == self.STA_OK:
      self._is_pwm_enable = False
    time.sleep(0.01)

  def set_pwm_frequency(self, freq):
    '''
      @brief    Set pwm frequency
      @param freq: int    Frequency to set, in range 1 - 1000
    '''
    if freq < 1 or freq > 1000:
      self.last_operate_status = self.STA_ERR_PARAMETER
      return
    is_pwm_enable = self._is_pwm_enable
    self.set_pwm_disable()
    self._write_bytes(self._REG_PWM_FREQ, [freq >> 8, freq & 0xff])
    time.sleep(0.01)
    if is_pwm_enable:
      self.set_pwm_enable()

  def set_pwm_duty(self, chan, duty):
    '''
      @brief    Set selected channel duty
      @param chan: list     One or more channels to set, items in range 1 to 4, or chan
= self.ALL
```

```
      @param duty: float    Duty to set, in range 0.0 to 100.0
    '''
    if duty < 0 or duty > 100:
      self.last_operate_status = self.STA_ERR_PARAMETER
      return
    for i in self._parse_id(_PWM_CHAN_COUNT, chan):
      self._write_bytes(self._REG_PWM_DUTY1 + (i - 1) * 2, [int(duty), int((duty
* 10) % 10)])

  def set_adc_enable(self):
    '''
      @brief    Set adc enable
    '''
    self._write_bytes(self._REG_ADC_CTRL, [0x01])

  def set_adc_disable(self):
    '''
      @brief    Set adc disable
    '''
    self._write_bytes(self._REG_ADC_CTRL, [0x00])

  def get_adc_value(self, chan):
    '''
      @brief    Get adc value
      @param chan: int    Channel to get, in range 1 to 4, or self.ALL
      @return :list       List of value
    '''
    for i in self._parse_id(_ADC_CHAN_COUNT, chan):
      rslt = self._read_bytes(self._REG_ADC_VAL1 + (i - 1) * 2, 2)
    return ((rslt[0] << 8) | rslt[1])

  def detecte(self):
    '''
      @brief    If you forget address you had set, use this to detecte them, must have
class instance
      @return   Board list conformed
    '''
    l = []
    back = self._addr
    for i in range(1, 127):
```

```
      self._addr = i
      if self.begin() == self.STA_OK:
        l.append(i)
    for i in range(0, len(l)):
      l[i] = hex(l[i])
    self._addr = back
    self.last_operate_status = self.STA_OK
    return l

class DFRobot_Epansion_Board_Digital_RGB_LED():

  def __init__(self, board):
    '''
      @param board: DFRobot_Expansion_Board   Board instance to operate digital
rgb led, test LED: https://www.dfrobot.com/product-1829.html
                                             Warning: LED must connect to pwm channel,
otherwise may destory Pi IO
    '''
    self._board = board
    self._chan_r = 0
    self._chan_g = 0
    self._chan_b = 0

  def begin(self, chan_r, chan_g, chan_b):
    '''
      @brief    Set digital rgb led color channel, these parameters not repeat
      @param chan_r: int    Set color red channel id, in range 1 to 4
      @param chan_g: int    Set color green channel id, in range 1 to 4
      @param chan_b: int    Set color blue channel id, in range 1 to 4
    '''
    if chan_r == chan_g or chan_r == chan_b or chan_g == chan_b:
      return
    if chan_r < _PWM_CHAN_COUNT and chan_g < _PWM_CHAN_COUNT
and chan_b < _PWM_CHAN_COUNT:
      self._chan_r = chan_r
      self._chan_g = chan_g
      self._chan_b = chan_b
      self._board.set_pwm_enable()
      self._board.set_pwm_frequency(1000)
      self._board.set_pwm_duty(self._board.ALL, 100)
```

```
  def color888(self, r, g, b):
    '''
      @brief    Set LED to true-color
      @param r: int   Color components red
      @param g: int   Color components green
      @param b: int   Color components blue
    '''
    self._board.set_pwm_duty([self._chan_r], 100 - (r & 0xff) * 100 // 255)
    self._board.set_pwm_duty([self._chan_g], 100 - (g & 0xff) * 100 // 255)
    self._board.set_pwm_duty([self._chan_b], 100 - (b & 0xff) * 100 // 255)

  def color24(self, color):
    '''
      @brief    Set LED to 24-bits color
      @param color: int   24-bits color
    '''
    color &= 0xffffff
    self.color888(color >> 16, (color >> 8) & 0xff, color & 0xff)

  def color565(self, color):
    '''
      @brief    Set LED to 16-bits color
      @param color: int   16-bits color
    '''
    color &= 0xffff
    self.color888((color & 0xf800) >> 8, (color & 0x7e0) >> 3, (color & 0x1f) << 3)

class DFRobot_Expansion_Board_Servo():

  def __init__(self, board):
    '''
      @param board: DFRobot_Expansion_Board   Board instance to operate servo,
test servo: https://www.dfrobot.com/product-255.html
                                             Warning: servo must connect to pwm channel,
otherwise may destory Pi IO
    '''
    self._board = board

  def begin(self):
```

```
    '''
      @brief    Board servo begin
    '''
    self._board.set_pwm_enable()
    self._board.set_pwm_frequency(50)
    self._board.set_pwm_duty(self._board.ALL, 0)

  def move(self, id, angle):
    '''
      @brief    Servos move
      @param id: list     One or more servos to set, items in range 1 to 4, or chan =
self.ALL
      @param angle: int   Angle to move, in range 0 to 180
    '''
    if 0 <= angle <= 180:
      self._board.set_pwm_duty(id, (0.5 + (float(angle) / 90.0)) / 20 * 100)

import smbus

class DFRobot_Expansion_Board_IIC(DFRobot_Expansion_Board):

  def __init__(self, bus_id, addr):
    '''
      @param bus_id: int   Which bus to operate
      @oaram addr: int     Board controler address
    '''
    self._bus = smbus.SMBus(bus_id)
    DFRobot_Expansion_Board.__init__(self, addr)

  def _write_bytes(self, reg, buf):
    self.last_operate_status = self.STA_ERR_DEVICE_NOT_DETECTED
    try:
      self._bus.write_i2c_block_data(self._addr, reg, buf)
      self.last_operate_status = self.STA_OK
    except:
      pass

  def _read_bytes(self, reg, len):
    self.last_operate_status = self.STA_ERR_DEVICE_NOT_DETECTED
    try:
```

```
    rslt = self._bus.read_i2c_block_data(self._addr, reg, len)
    self.last_operate_status = self.STA_OK
    return rslt
  except:
    return [0] * len
```

Save below code as *dfadc.py* (library for ADC interface)

Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/DFRobot_IoT_Codes/dfadc.py

Code

```
import time

from DFRobot_RaspberryPi_Expansion_Board import
DFRobot_Expansion_Board_IIC as Board

board = Board(1, 0x10)    # Select i2c bus 1, set address to 0x10

def board_detect():
  l = board.detecte()
  print("Board list conform:")
  print(l)

''' print last operate status, users can use this variable to determine the result of a
function call. '''
def print_board_status():
  if board.last_operate_status == board.STA_OK:
    print("board status: everything ok")
  elif board.last_operate_status == board.STA_ERR:
    print("board status: unexpected error")
  elif board.last_operate_status == board.STA_ERR_DEVICE_NOT_DETECTED:
    print("board status: device not detected")
  elif board.last_operate_status == board.STA_ERR_PARAMETER:
    print("board status: parameter error")
  elif board.last_operate_status == board.STA_ERR_SOFT_VERSION:
```

```
    print("board status: unsupport board framware version")
```

Run below code to sense temperature values from LM35 (in same directory of dfadc.py). Save file as “DFR_Linear_temperature_sensor.py”

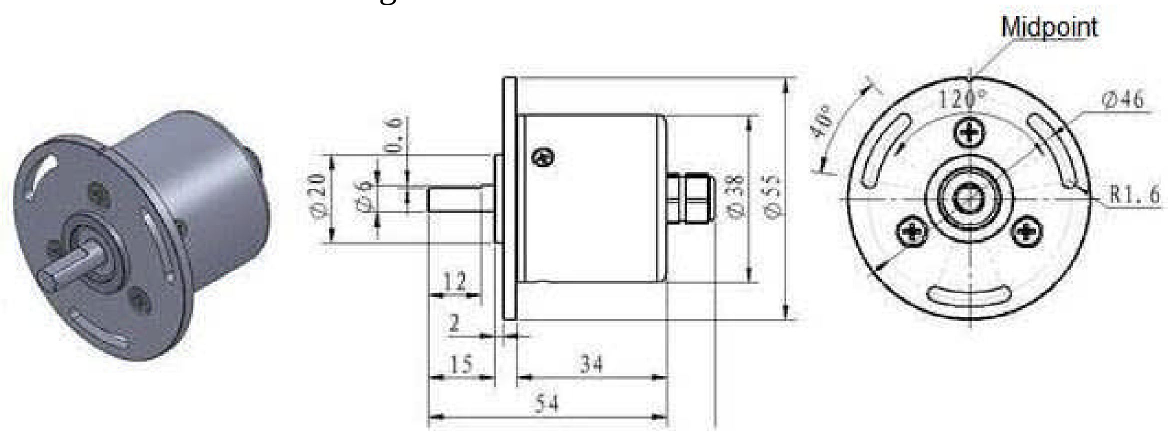
GitHub Link:
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/DFRobot_IoT_Codes/3_DFR_temperature_sensor.py

Code-

```
from dfadc import *

board_detect()

while board.begin() != board.STA_OK:
    print_board_status()
    print("board begin faild")
    time.sleep(2)
print("board begin success")

board.set_adc_enable()

while True:
    val = board.get_adc_value(board.A0)   # A0 channels read
    Temperature = (val/4096)* 3300/10.24
    print("Temperature = %d C" %Temperature)
#    val=val/4096*100
#    print(val)

    time.sleep(2)
```

Analog Rotation Sensor

Angle sensors are used to detect angular movement and are composed of sensitive components, measuring circuits, smart components and interface components. Among them, detecting the value reflecting the angular position is the initial conversion unit of the sensor. There is a hole in the sensor that can match the axis of most devices when installation. When connected to the device, the angle sensor counts every setting revolution of the shaft based on sensor instructions. Because the number of counts is related to the initial position of the angle sensor. So when the angle sensor is initialized, its count value is usually set to 0. Of course, you can still reset the set value range.



Rotary Angle Sensor

Angle Sensor Size

A reference voltage is applied between the fixed terminals which are on the either side of the wiper and the output voltage is taken from this wiper. This configuration forms a voltage divider network and the output voltage is dependent on the position of the slider.

Analog Rotation Sensor Interface with GrovePI & RaspberryPI

Steps-

1. Connect Rotation sensor to Analog port A2
2. Run below code to fetch data from sensor

Github Link

https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/GrovePI_Codes/10_Analog_read_potentiometer.py

Code-

```
import time
import grovepi
from grove_rgb_lcd import *
# Connect the Rotary Angle Sensor to analog port A2
potentiometer = 2

time.sleep(1)
i = 0

while True:
    i = grovepi.analogRead(potentiometer)
    print(i)
    setRGB(i//4,i//4,i//4)
```

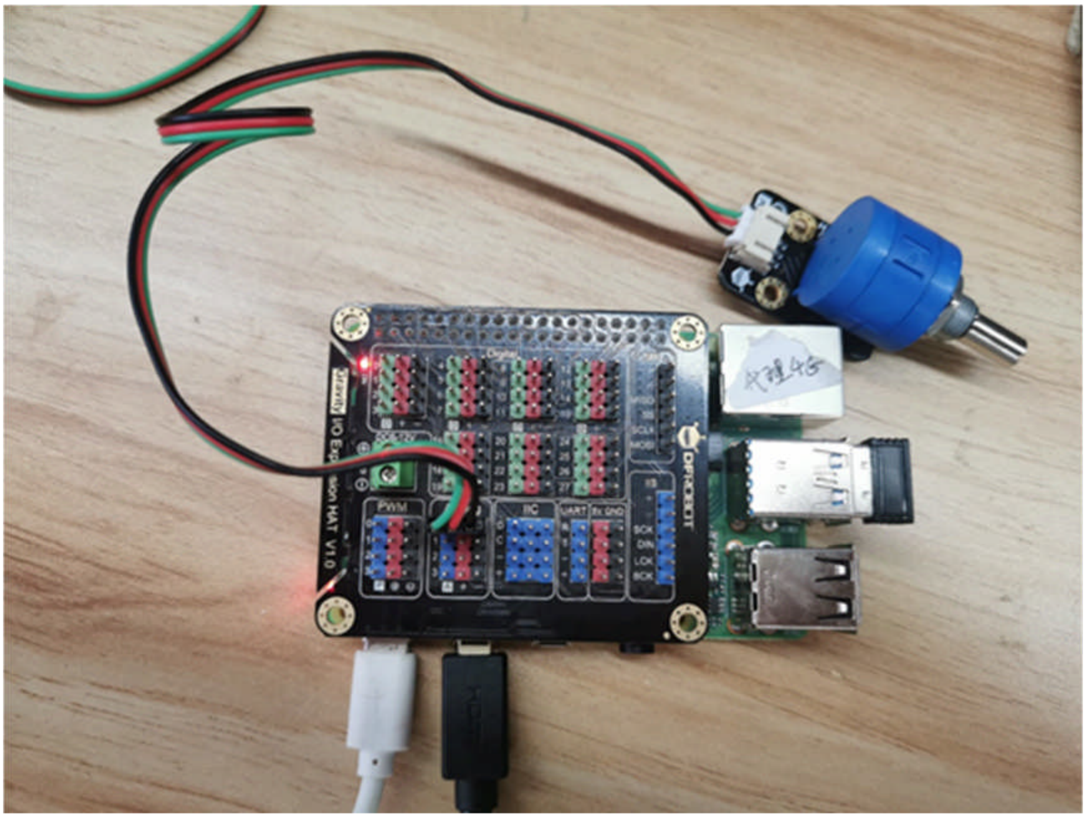
Analog Rotation Sensor Interface with DFRobot Hat & RaspberryPI

As the rotation angle of the ordinary potentiometer is only 300 degrees at most, the accuracy is quite low after distributing the 5V power supply of RaspberryPI to every 1 degree.

So if you want to make a project with precise control of angle or analog quantity, this precision angle sensor is a good choice. Based on a multi-turn precision potentiometer, this sensor can be rotated about 10 turns and subdivide the voltage into 1024 parts. What's more, it can be combined with the sensor expansion board through the 3P connection line, accurately sensing small changes in rotation.

Use Analog Rotation Sensor on Your Raspberry Pi

- Power the Raspberry Pi on and install the Raspberry Pi expansion board correctly
- Connect the sensor to the analog port 0 on the expansion board

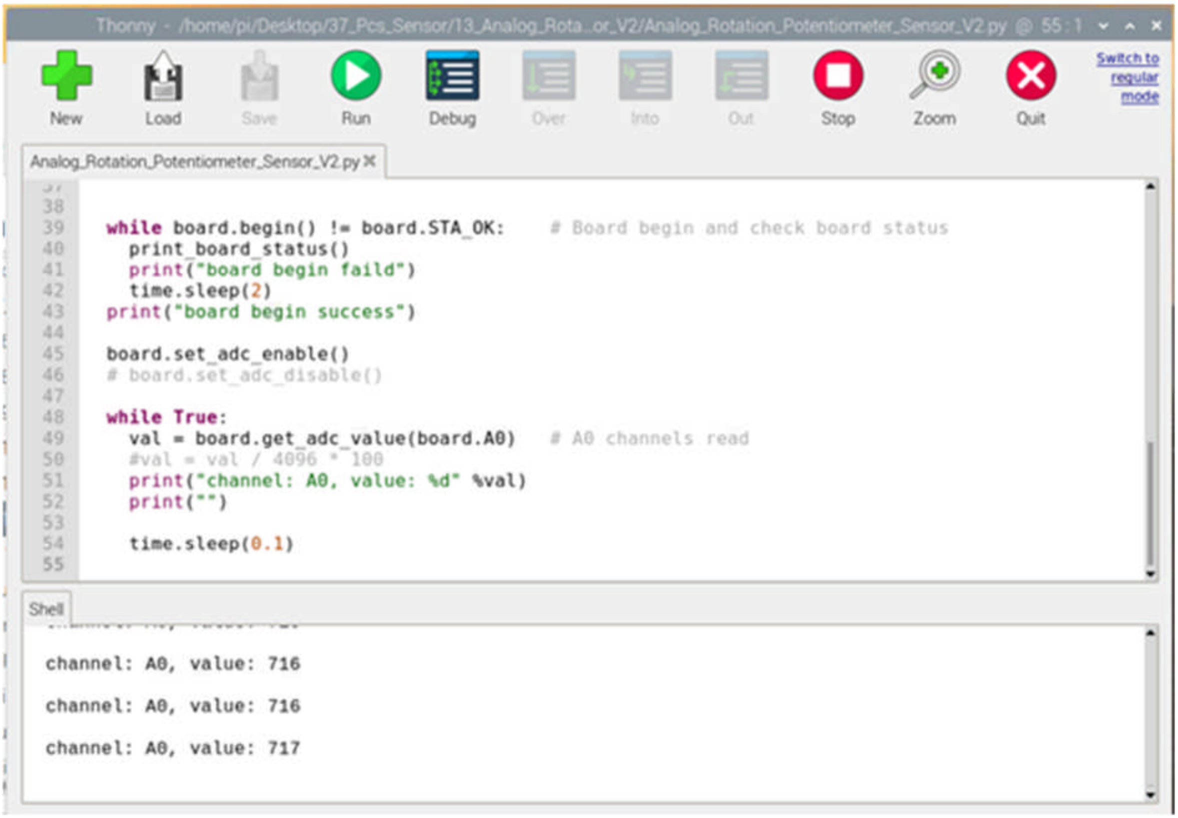


Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/DFRobot_IoT_Codes/4_DFR_rotation_angle_sensor.py

Code-

```
from dfadc import *

board_detect()    # If you forget address you had set, use this to detected them, must have class instance


while board.begin() != board.STA_OK:    # Board begin and check board status
   print_board_status()
   print("board begin faild")
   time.sleep(2)
print("board begin success")

board.set_adc_enable()
```

```
while True:
    val = board.get_adc_value(board.A0)   # A0 channels read
    #val = board.get_adc_value(board.A1)   # A1 channels read
    #val = board.get_adc_value(board.A2)   # A2 channels read
    #val = board.get_adc_value(board.A3)   # A3 channels read
    print("channel: A0, value: %d" %val)
    print("")

    time.sleep(2)
```



Joy-Stick Sensor

The JoyStick produced by DFRobot is made with original high-quality metal PS2 rocker potentiometer. With (X, Y) 2 axis analog output and (Z) 1 button digital output, it can maintain good contact and mechanical properties no matter how you torture it. The 3 signals are respectively connected to the RaspberryPI sensor expansion board through the 3P line, occupying only 3 ports for its control.

Features of its new version:

- Operating Voltage: 3.3 V/5 V, suitable for more 3.3 V controllers.
- Standard Size: The distance between two mounting holes with a diameter of 3mm is several times that of 5mm
- Easy to Identify: The two analog output interfaces are marked with S, while the digital interface is marked with D
- High-quality connectors, resistant to repeated plugging and unplugging
- Immersion gold technology, not only improving the quality of PCB board, but also being with golden fonts.

Precautions

The layout of the new version of the analog sensor port has the following two improvements. When using this sensor on the IO expansion board, you may need to adjust the layout of the connector. For your convenience, we will make more improvements, so stay tuned.

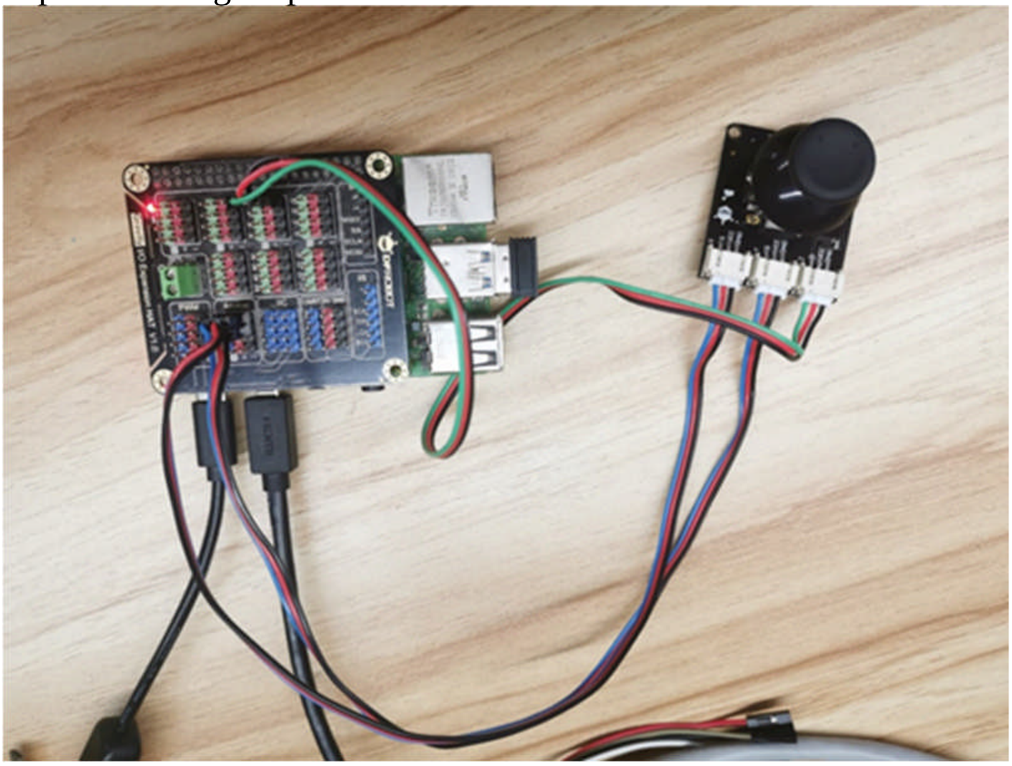
Joystick Sensor

Interfacing Joystick with DFRobot hat & RaspberryPI

Steps to use JoyStick on Your Raspberry Pi

- Power the Raspberry Pi on and install the Raspberry Pi expansion board correctly

- Connect the X-axis output port of the JoyStick to the analog port 0 on the expansion board, the Y-axis output port to the analog port 1, and the Z-axis output to the digital port 8

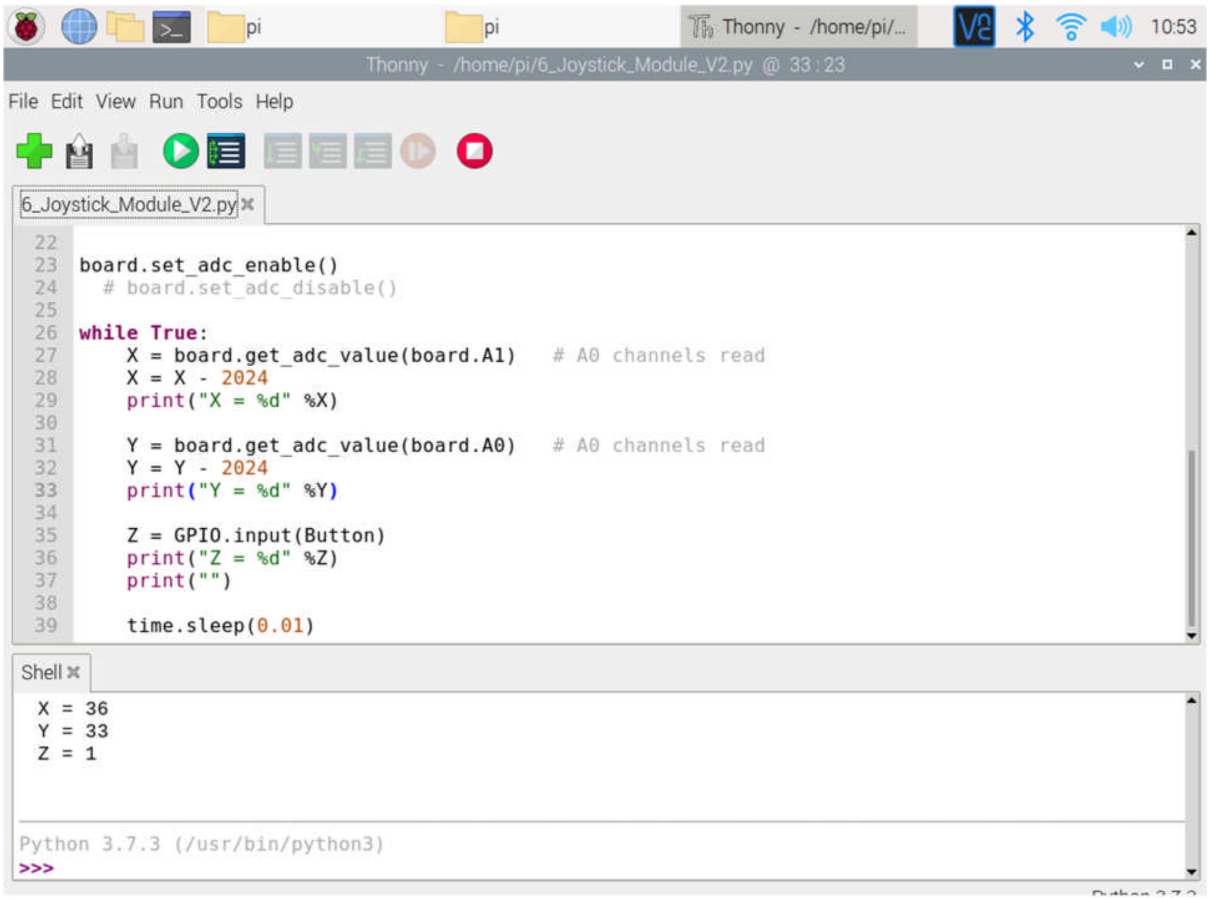


Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/DFRobot_IoT_Codes/6_DFR_Joystick.py

Code-

```
import time
import RPi.GPIO as GPIO
import atexit

Button=8

atexit.register(GPIO.cleanup)
GPIO.setmode(GPIO.BCM)
GPIO.setup(Button,GPIO.IN)
```

```
from dfadc import *

board_detect()    # If you forget address you had set, use this to detected them,
must have class instance


while board.begin() != board.STA_OK:    # Board begin and check board status
    print_board_status()
    print("board begin faild")
time.sleep(2)
print("board begin success")

board.set_adc_enable()
  # board.set_adc_disable()

while True:
    X = board.get_adc_value(board.A1)   # A0 channels read
    X = X - 2024
    print("X = %d" %X)

    Y = board.get_adc_value(board.A0)   # A0 channels read
    Y = Y - 2024
    print("Y = %d" %Y)

    Z = GPIO.input(Button)
    print("Z = %d" %Z)
    print("")

    time.sleep(0.01)
```



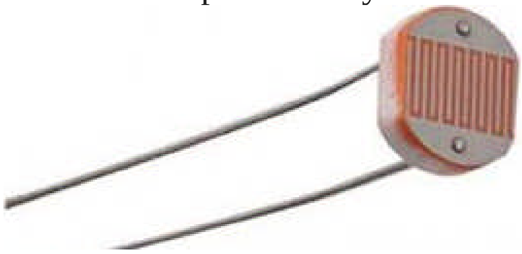
Analog Light Sensor

Analog Light Sensor

The controlling of street lights, make a light sensor circuit outdoor lights, a few in-door home appliances, and so on are usually maintained and operated manually on several occasions. This is not only risky but also results in wastage of power with the negligence of personnel or unusual circumstances in controlling these electrical ap-pliances on and off. Hence, can utilize the light sensor circuit for automatic switching of the loads based on daylight's intensity by using a light sensor.

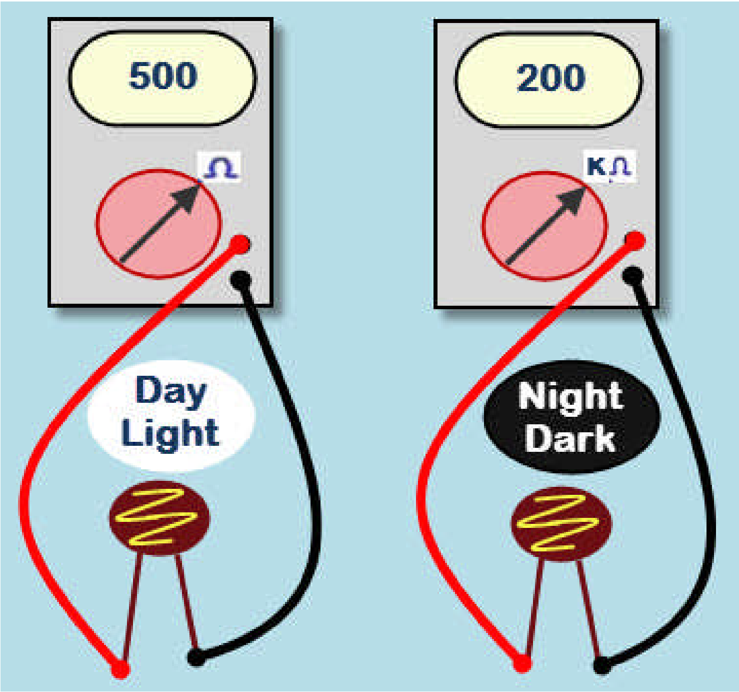
There are different types of light sensors available such as photoresistors, photodiodes, photovoltaic cells, phototubes, photomultiplier tubes, phototransistors, charge coupled devices, and so on. But, LDR (Light Dependent

Resistor or photoresistor) is used as a light sensor in this light sensor circuit. These LDR sensors are passive and doesn't produce any electrical energy.



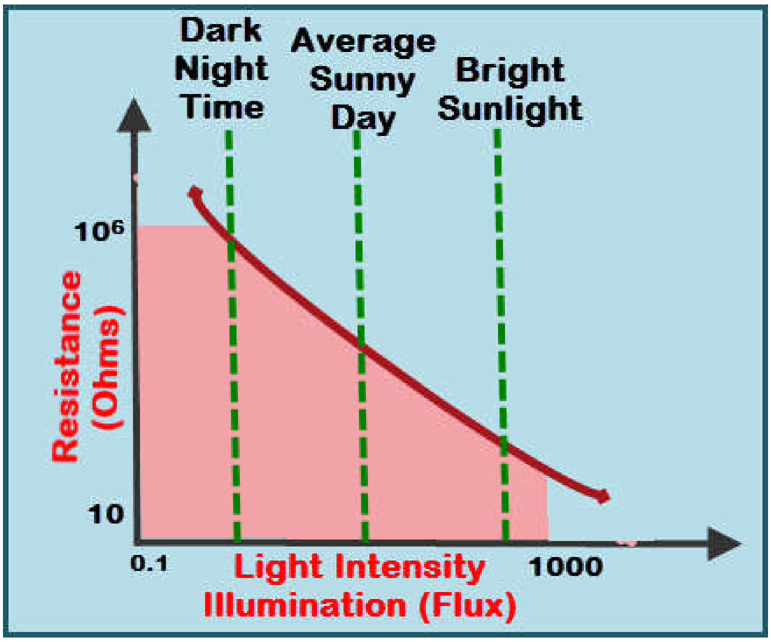
LDR Light Sensor

But, the resistance of the LDR changes with the change in the (light illuminated on the LDR) daylight intensity. LDR sensor is rugged in nature, hence can be used even in dirty and rough external environments. Hence, LDR is preferable compared to other light sensors as it can be used even in the outdoor lighting of homes and in automatic street lights as well.



LDR Resistance Variation with Variation in Light Intensity

Light Dependent Resistor is a variable resistor that is controlled by light intensity. LDRs are made of high resistance semiconductor material, Cadmium Sulphide that exhibits photoconductivity.



Light Intensity vs LDR Resistance

During night time (when the light illuminated on LDR decreases), the LDR exhibits a very high resistance of around a few MΩ (Mega Ohms). During daytime, (when the light is illuminated on LDR), resistance of LDR decreases to around a few 100Ω (hundred Ohms). Hence, the resistance of LDR is inversely proportional to the light illuminated on LDR.

Interfacing Analog Light Sensor Using GrovePI & RaspberryPI

Steps-

1. Connect Light Sensor to Analog port A0
2. Connect Led to Digital port D5.
3. Connect LCD to any of I2C ports
4. Run below code to realize smart street light deployment

Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/GrovePI_Codes/11_Analog_Light_Sensor.py

Code-

```
import time
import grovepi
from grove_rgb_lcd import *
```

```
# Connect the Grove Light Sensor to analog port A0
# SIG,NC,VCC,GND
light_sensor = 0

# Connect the LED to digital port D4
# SIG,NC,VCC,GND
led = 5

# Turn on LED once sensor exceeds threshold resistance
threshold = 10

grovepi.pinMode(light_sensor,"INPUT")
grovepi.pinMode(led,"OUTPUT")

while True:
    try:
        # Get sensor value
        sensor_value = grovepi.analogRead(light_sensor)
        print(sensor_value)

        # Calculate resistance of sensor in K
        resistance = (float)(1023 - sensor_value) * 10 / (sensor_value + 0.001)
        setRGB(255-10*sensor_value,255-10*sensor_value,255-10*sensor_value)

        if sensor_value > threshold:
            # Send HIGH to switch on LED
            grovepi.digitalWrite(led,1)
            setText_norefresh("Turn off Smart Light")
        else:
            # Send LOW to switch off LED
            grovepi.digitalWrite(led,0)
            setText_norefresh("Turn on Smart Light")

        print("sensor_value = %d resistance = %.2f" %(sensor_value,  resistance))
        time.sleep(.5)

    except IOError:
        print ("Error")
```

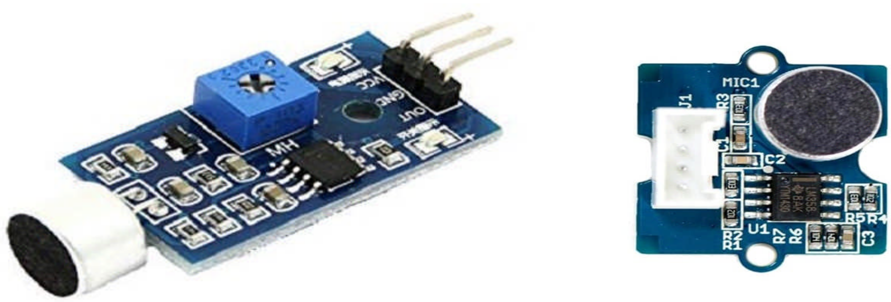
Analog Sound Sensor

The sound sensor is one type of module used to notice the sound. Generally, this module is used to detect the intensity of sound. The applications of this module mainly include switch, security, as well as monitoring. The accuracy of this sensor can be changed for the ease of usage.

- This sensor employs a microphone to provide input to buffer, peak detector and an amplifier. This sensor notices a sound, & processes an o/p voltage signal to a microcontroller. After that, it executes required processing.
- This sensor is capable to determine noise levels within DB’s or decibels at 3 kHz 6 kHz frequencies approximately wherever the human ear is sensitive. In smartphones, there is an android application namely decibel meter used to measure the sound level.

Sound Sensor Pin Configuration

This sensor includes three pins which include the following.



<u>Sound Sensor</u>

- Pin1 (VCC): 3.3V DC to 5V DC
- Pin2 (GND): This is aground pin
- Pin3 (DO): This is an output pin

Working Principle

The sound sensor working principle is simple and very easy. It works like a human ear. The sound sensor module consists of a small circuit board that is a microphone of 50 Hz-10 kHz and operates with the sensor detector module for detection. Other external processing circuitry components convert sound waves into electrical signals.

Another important hardware component is the high precision comparator LM393N. This device is mandatory to digitize the electrical signal to the digital output D0. To adjust the sensitivity of the digital output D0, the sound sensor module contains the built-in potentiometer. The sound sensor contains a microphone called a condenser microphone with 2 charged plates- one is a diaphragm and the other is a backplate. These plates seem like a capacitor. If the sound signals (claps, snaps, knocking, alarms) or audio signals travel through the air and strike the microphone's diaphragm, then the distance between the 2 charged plates changes due to the vibration of the diaphragm.

Therefore this change in the capacitance between the plates generates the output electrical signal. This output signal is proportional to the input sound signal received by the microphone. Finally, the output signal is amplified by the amplifier and digitized to determine the intensity of the incoming sound signal.

Interfacing Analog Sound Sensor with GrovePI & RaspberryPI

Steps-

1. Connect Sound sensor to Analog port A0
2. Connect led to Digital port D5
3. Connect LCD to any of I2C ports
4. Run below code to control light on behalf on sound signal threshold

Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/GrovePI_Codes/Sound_sensor_interface.py

Code

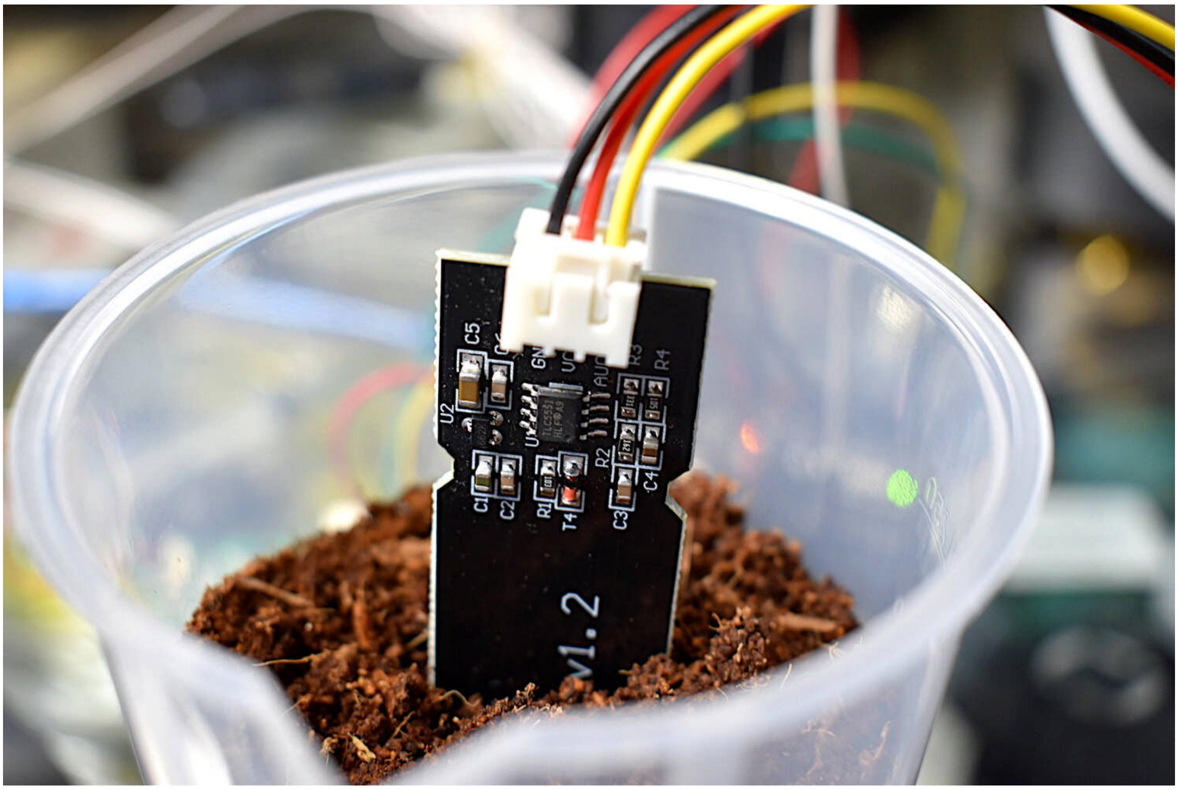
```
import time
import grovepi
from grove_rgb_lcd import *
# Connect the Grove Sound Sensor to analog port A0
```

```
# SIG,NC,VCC,GND
sound_sensor = 0

# Connect the Grove LED to digital port D5
# SIG,NC,VCC,GND
led = 5

grovepi.pinMode(sound_sensor,"INPUT")
grovepi.pinMode(led,"OUTPUT")

# The threshold to turn the led on 400.00 * 5 / 1024 = 1.95v
threshold_value = 600

while True:
    try:
        # Read the sound level
        sensor_value = grovepi.analogRead(sound_sensor)
        print(sensor_value)
        setRGB(sensor_value//2,sensor_value//2,sensor_value//2)

        # If loud, illuminate LED, otherwise dim
        if sensor_value > threshold_value:
            grovepi.digitalWrite(led,1)
        else:
            grovepi.digitalWrite(led,0)

        print("sensor_value = %d" %sensor_value)
        time.sleep(.5)

    except IOError:
        print ("Error")
```

Analog Capacitive Soil Moisture Sensor

Traditional soil moisture sensors are prone to corrosion with a limited lifespan regardless of measures taken. This capacitive soil moisture sensor features no exposed plating and uses capacitive sensing to detect soil moisture. The result is a much more robust sensor without corrosion worries.

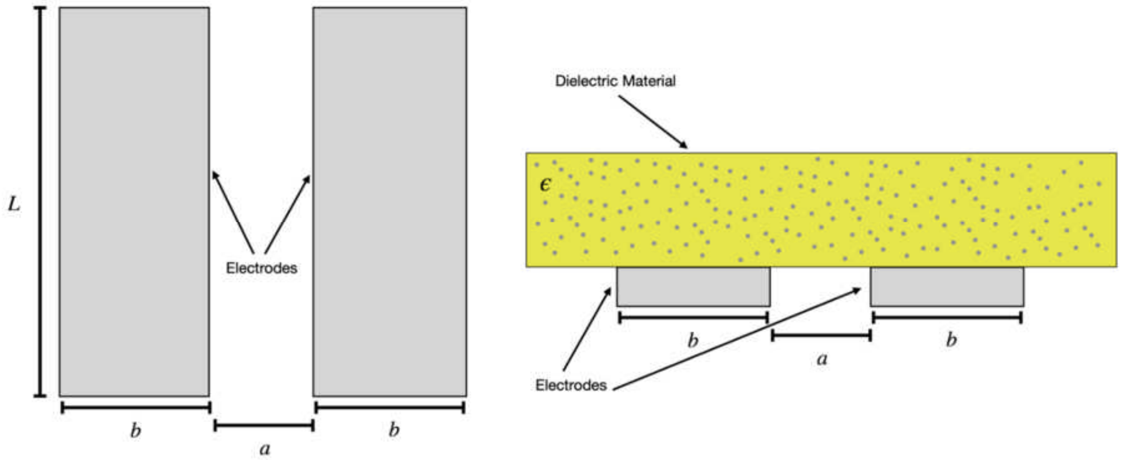
One technique is to use a gravimetric technique to calibrate capacitive-type electromagnetic soil moisture sensors. Capacitive soil moisture sensors exploit the dielectric contrast between water and soil, where dry soils have a relative permittivity between 2-6 and water has a value of roughly 80. Accurate measurement of soil water content is essential for applications in agronomy and botany - where the under- and over-watering of soil can result in ineffective or wasted resources. With water occupying up to 60% of certain soils by volume, depending on the specific porosity of the soil, calibration must be carried out in every environment to ensure accurate prediction of water content. An Embedded device can be used to read the analog signal from the capacitive sensor, which can be calibrated to volumetric soil moisture content via gravimetric methods (using volume and weight of dry and wet soil). Alternatively there are autocalibration approaches



Soil Moisture Sensor

How Does a Capacitive Moisture Sensor Work?

Simply stated, a capacitor stores electrical charge. The electrical component known as a capacitor consist of three pieces. A positive plate, a negative plate and the space in-between the plates, known as the dielectric.

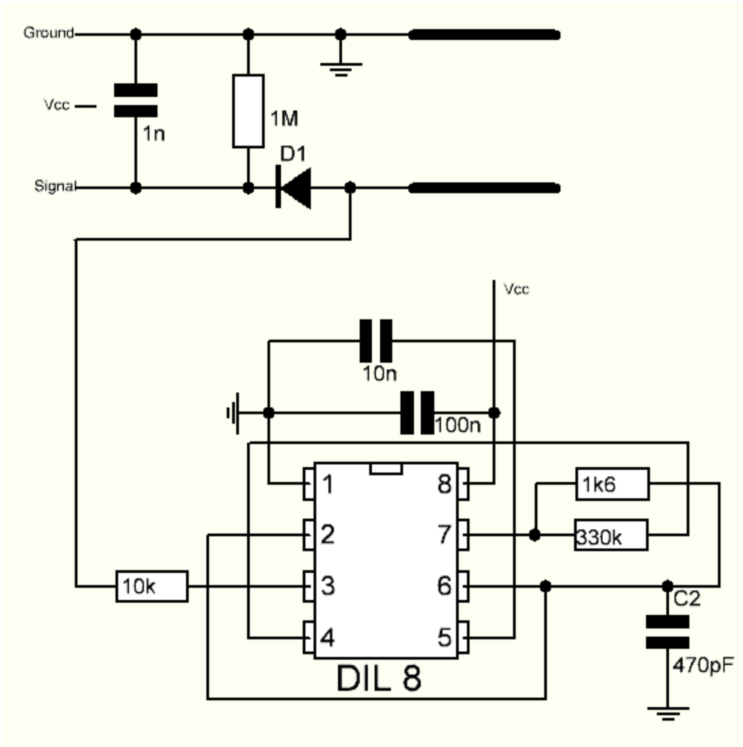


<u>Working of Soil Moisture Sensor</u>

The physical form and construction of practical capacitors vary widely and many capacitor types are in common use. Most capacitors contain at least two electrical conductors often in the form of metallic plates or surfaces separated by a dielectric medium.

A capacitive moisture sensor works by measuring the changes in capacitance caused by the changes in the dielectric. It does not measure moisture directly (pure water does not conduct electricity well), instead it measures the ions that are dissolved in the moisture These ions and their concentration can be affected by a number of factors, for example adding fertilizer for instance will decrease the resistance of the soil. Capacitive measuring basically measures the dielectric that is formed by the soil and the water is the most important factor that affects the dielectric.

Capacitive measuring has some advantages, It not only avoids corrosion of the probe but also gives a better reading of the moisture content of the soil as opposed to using a resistive soil moisture sensor. Since the contacts (the plus plate and the minus plate of the capacitor) are not exposed to the soil, there is no corrosion of the sensor itself.



Hardware schematic for capacitive soil moisture sensor

Soil and sensor form a capacitor where the capacitance varies according to the water content present in the soil. The capacitance is converted into voltage level basically from 1.2V to 3.0V maximum.

There is a fixed frequency oscillator that is built with a 555 Timer IC. The square wave generated is then fed to the sensor like a capacitor. To a square wave signal that capacitor, however, has a certain reactance, or for argument's sake a resistance that forms a voltage divider with a pure ohm type resistor (the 10k one on pin 3). The greater is the soil moisture, the higher the capacitance of the sensor. Consequently, there is a smaller reactance to the square wave, thus lowering the voltage on the signal line. The voltage on the Analog signal pin can be measured by an analog pin on the RaspberryPI which represents the humidity in soil.

Interfacing Soil Moisture Capactive Sensor with GrovePI & RaspberryPI

Interfacing Soil Moisture Capactivit Sensor with DFRobot Hat & RaspberryPI. This is a simple moisture sensor that can be used to detect soil moisture. When the soil is short of water, its output value will decrease, otherwise it will increase. This kind of sensor is mainly used to measure the relative water content of soil, do soil moisture monitoring, agricultural irrigation and forestry protection.

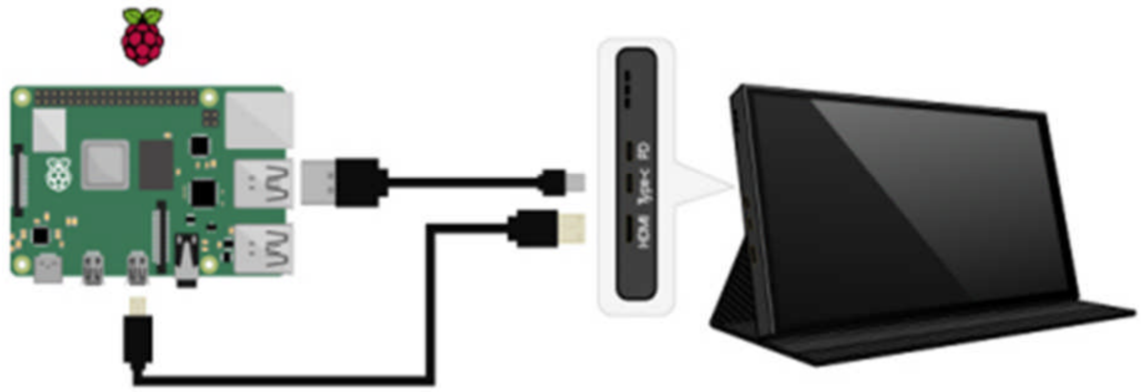
Hardware

1. Gravity: 37 Pcs Sensor Set
2. Raspberry Pi 4 Model B
3. IO Expansion HAT for Raspberry Pi 4B/3B+
4. 8GB + SanDisk Class10 SD/MicroSD Memory Card
5. 5V@3A USB Power Supply

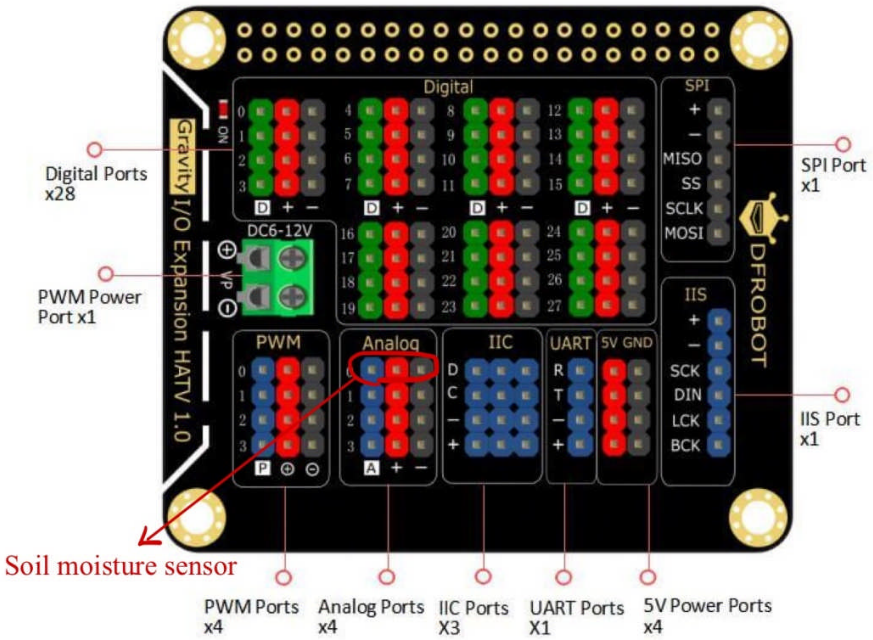
Learning Contents

Connection

- Connect the Raspberry Pi correctly to devices such as the screen, power, keyboard and mouse.

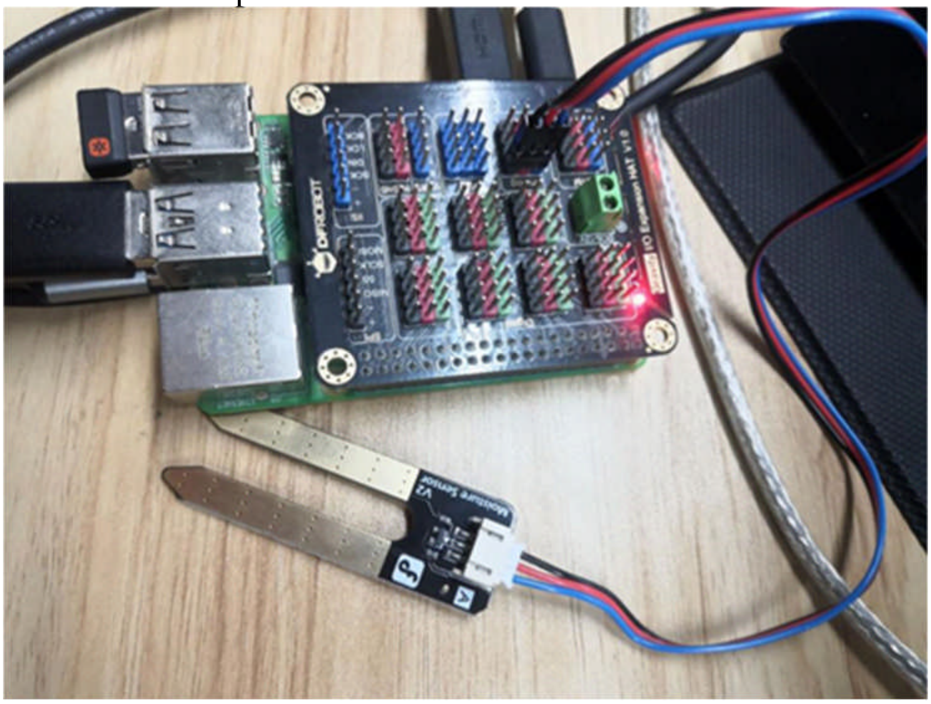


- Connect the sensor to analog port A0 on Raspberry Pi expansion board. The wiring diagram is as follows.

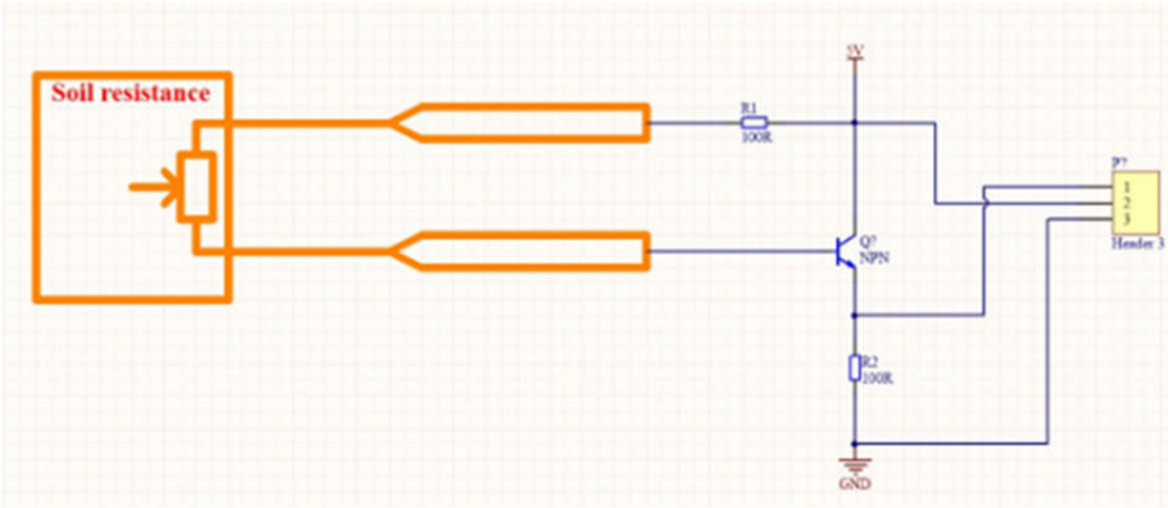


DFRobot Hat

- Find components and connect



Schematic and operating principle



Operational principle of DFR Moisture Sensor

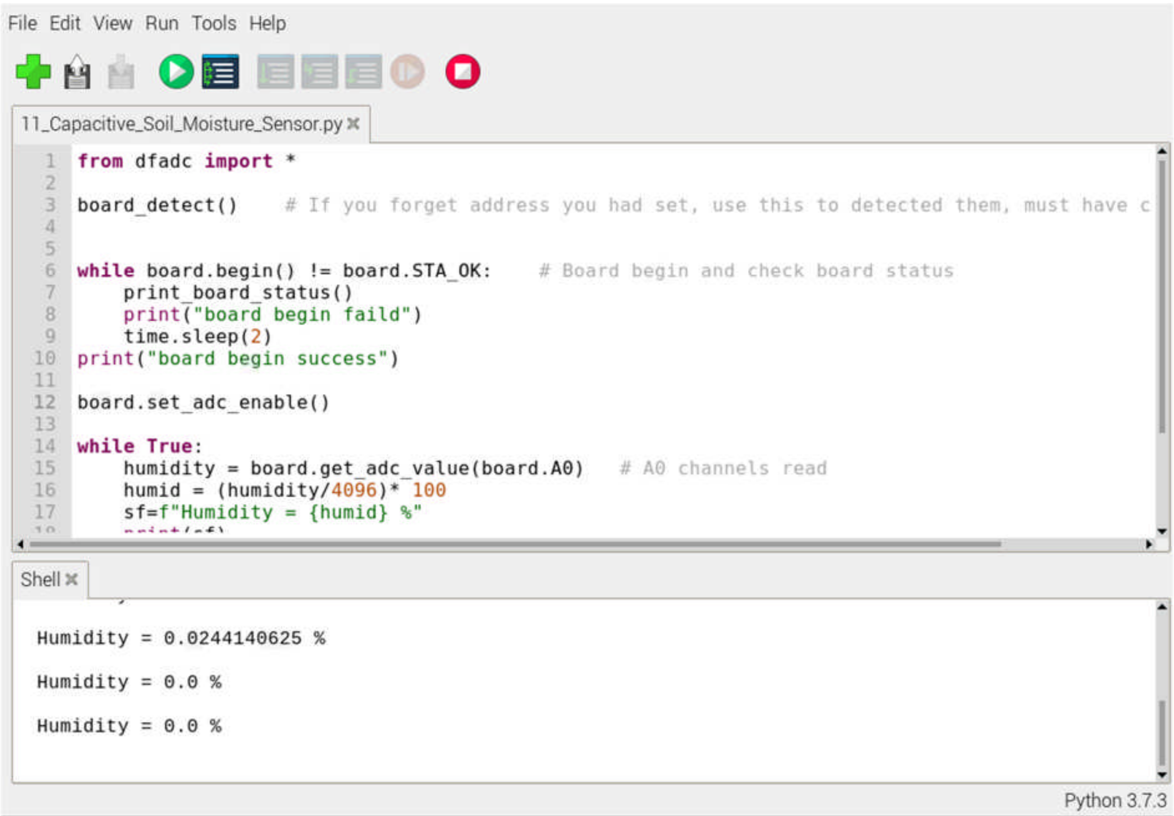
The soil moisture sensor judges the soil moisture content by its water level. As shown in the figure, when the soil moisture sensor probe is suspended in the air, the base of the triode is open, and the output of the triode is 0. When it is inserted into the soil, the resistance value of the soil is different due to the different moisture content. Then the base of the triode will provide a variable conduction current. The conduction current from the collector to the emitter of the triode is controlled by the base, converted into a voltage after the pull-down resistor of the emitter.

Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/DFRobot_IoT_Codes/12_DFR_Soil_Moisture_Sensor.py

Code-

```
from dfadc import *

board_detect()    # If you forget address you had set, use this to detected them,
must have class instance


while board.begin() != board.STA_OK:    # Board begin and check board status
    print_board_status()
```

```
    print("board begin faild")
    time.sleep(2)
print("board begin success")

board.set_adc_enable()

while True:
    humidity = board.get_adc_value(board.A0)   # A0 channels read
    humid = (humidity/4096)* 100
    sf=f"Humidity = {humid} %"
    print(sf)
    print("")

    time.sleep(2)
```



Ultrasonic Sensor

In industrial applications, an ultrasonic detection used to detect hidden tracks, discontinuities in metals, composites, plastics, ceramics, and for water level detection. For this purpose, the laws of physics which are indicating the propagation of sound waves through solid materials have been used since ultrasonic sensors using sound instead of light for detection.
Ultrasonic sensors work by emitting sound waves at a frequency which is too high for humans to hear.

Ultrasonic Sensor

An above image shows the **HC-SR-04 ultrasonic sensor** which has a transmitter, receiver. The pin configuration is,

- VCC - +5 V supply
- TRIG – Trigger input of the sensor. Microcontroller applies 10 us trigger pulse to the HC-SR04 ultrasonic module.
- ECHO–Echo output of the sensor. Microcontroller reads/monitors this pin to detect the obstacle or to find the distance.
- **GND** – Ground

Sound is a mechanical wave travelling through the mediums, which may be a solid, or liquid or gas. Sound waves can travel through the mediums with specific velocity depends on the medium of propagation. The sound waves which are having high frequency reflect from boundaries and produce distinctive echo patterns.

Features of an Ultrasonic Sensor

1. Supply voltage: 5V (DC).
2. Supply current: 15mA.
3. Modulation frequency: 40Hz.
4. Output: 0 – 5V (Output high when obstacle detected in range).
5. Beam Angle: Max 15 degrees.
6. Distance: 2 cm – 400 cm.
7. Accuracy: 0.3cm.
8. Communication: Positive TTL pulse.

Ultrasonic Sensor Working Principle

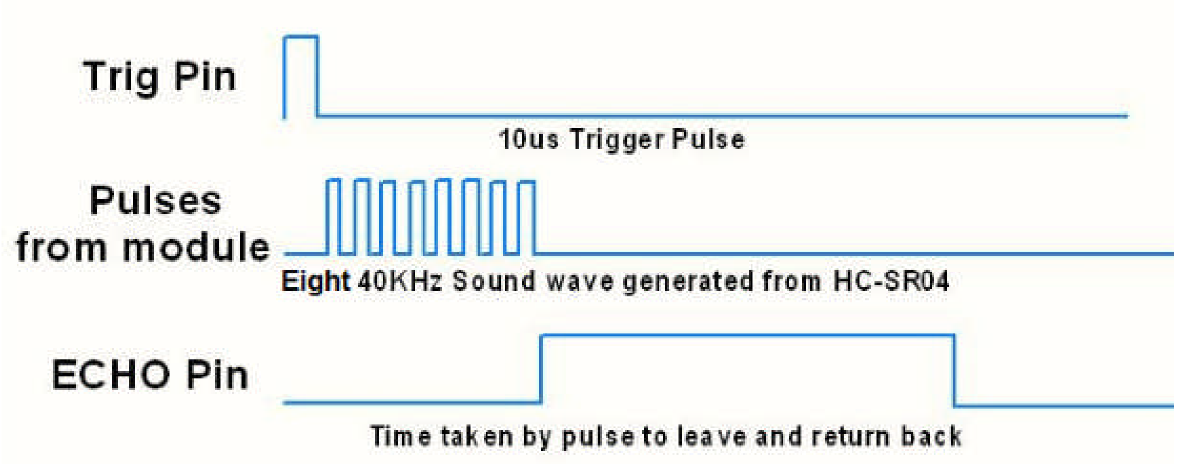
Ultrasonic sensors emit short, high-frequency sound pulses at regular intervals. These propagate in the air at the velocity of sound. If they strike an object, then they reflected back as an echo signal to the sensor, which itself computes the distance to the target based on the time-span between emitting the signal and receiving the echo.

Working Principle of Ultrasonic Sensor

Ultrasonic sensors are excellent at suppressing background interference. Virtually all materials which reflect sound can be detected, regardless of their colour. Even transparent materials or thin foils represent no problem for an ultrasonic sensor.

The ultrasonic sensors are suitable for target distances from 20 mm to 10 m and as they measure the time of flight they can ascertain a measurement with pinpoint accuracy. Some of our sensors can even resolve the signal to an accuracy of 0.025 mm. Ultrasonic sensors can see through dust-laden air and ink mists. Even thin deposits on the sensor membrane do not impair its function.

Timing Diagram of Ultrasonic Sensor



Timing Diagram of Sensor

1. First, need to transmit trigger pulse of at least 10 us to the HC-SR04 Trig Pin.
2. Then the HC-SR04 automatically sends Eight 40 kHz sound wave and wait for rising edge output at Echo pin.
3. When the rising edge capture occurs at Echo pin, start the Timer and wait for a falling edge on Echo pin.
4. As soon as the falling edge captures at the Echo pin, read the count of the Timer. This time count is the time required by the sensor to detect an object and return back from an object.

How to calculate Distance?

If you need to measure the specific distance from your sensor, this can be calculated based on this formula:
We know that, **Distance= Speed* Time.** The speed of sound waves is 343 m/s.
So,
Total Distance= (343 * Time of hight(Echo) pulse)/2
Total distance is divided by 2 because the signal travels from HC-SR04 to object and returns to the module HC-SR-04.

Interfacing Ultrasonic Sensor with GrovePI & RaspberryPI

Steps-

1. Connect Ultrasonic sensor to Digital port D7
2. Connect LCD to any of I2C ports
3. Run below code to fetch distance of obstacle from Sensor

Github Link
https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/GrovePI_Codes/16_Ultrasonic_GrovePI.py

---

Code-
```
from grovepi import *
from grove_rgb_lcd import *
import time

ultrasonic_ranger = 7

setRGB(0,100,0)

while True:
    distant = ultrasonicRead(ultrasonic_ranger)
    print(distant,'cm')
    setText_norefresh(str(distant)+'cm')
    time.sleep(0.5)
```

Interfacing Ultrasonic Sensor to DFRobot Hat & RaspberryPI

DFR Ultrasonic sensor has multiple properties. Due to its small size, this convenient sensor that allows plug and play has strong environmental applicability, high accuracy, wide measurement range, quite suitable for outdoor environments, especially those with rapid temperature changes. It is also an excellent choice for robots to avoid obstacles automatically, car reversing alarms, doorbells, warning alarms, subway safety line prompts, bank and cash machine one-meter line prompts, and so on.

Preparation

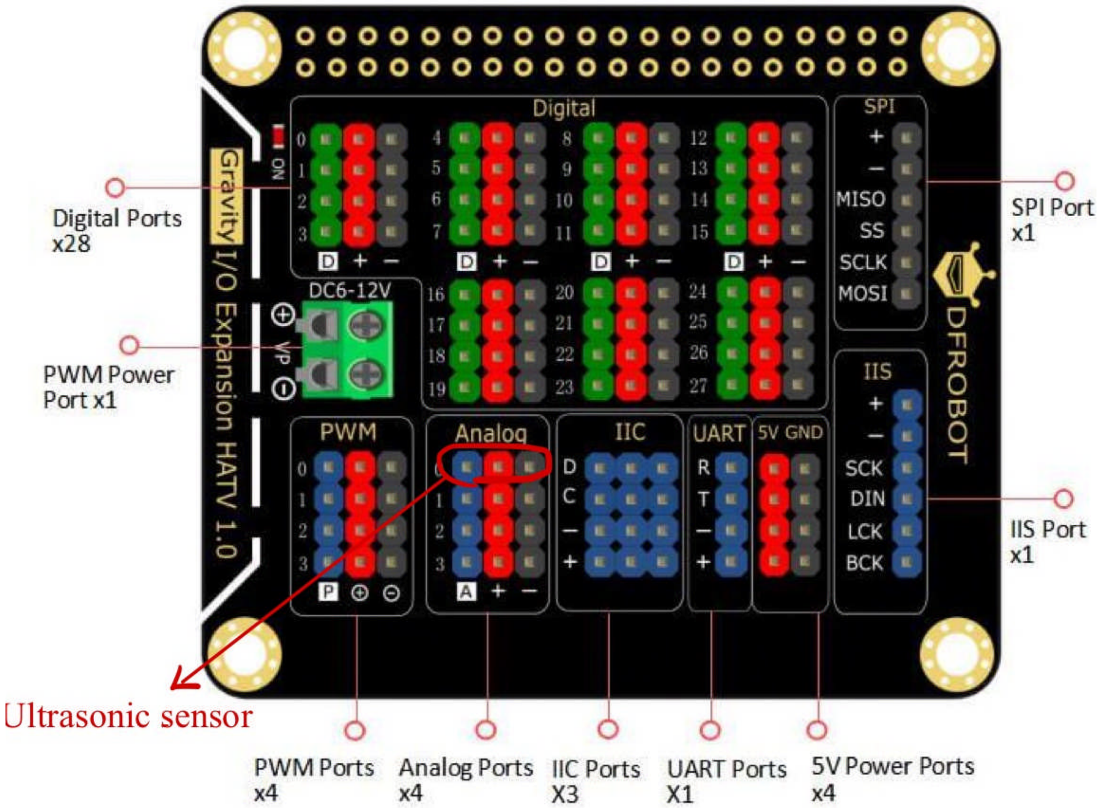
Hardware

1. Gravity: 37 Pcs Sensor Set
2. Raspberry Pi 4 Model B
3. IO Expansion HAT for Raspberry Pi 4B/3B+
4. 8GB + SanDisk Class10 SD/MicroSD Memory Card
5. 5V@3A USB Power Supply

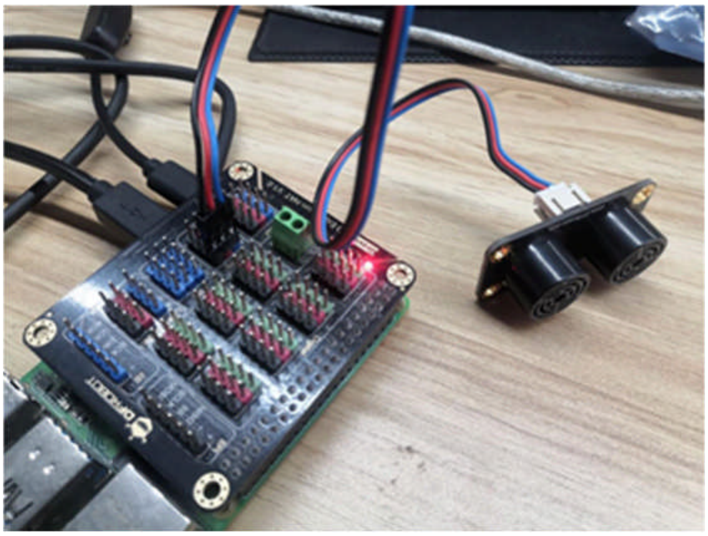
Connection

- Connect the Raspberry Pi correctly to devices such as the screen, power, keyboard and mouse.

- Connect this module to analog port A0 on Raspberry Pi expansion board.



- Find this position to connect the sensor correctly.
-



Github Link

https://github.com/Code-Unnati/Advance-Course/blob/master/Module-2/Unit-3/DFRobot_IoT_Codes/13_DFR_ultrasonic_distance_sensor.py

Code-

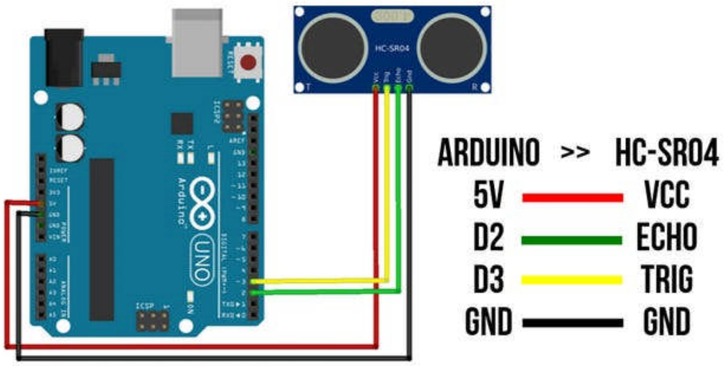
```
from dfadc import *

board_detect()

while board.begin() != board.STA_OK:
    print_board_status()
    print("board begin faild")
    time.sleep(2)
print("board begin success")

board.set_adc_enable()

while True:
    val = board.get_adc_value(board.A0)   # A0 channels read
    print(val)
    dist = val * 1276/(1023.0*4)/4*2.5
    dist=round(dist)
    print("Distance--",dist)
    time.sleep(2)
```

When we move the ultrasonic sensor, we can receive feedback to detect different distance values.

Interfacing Ultrasonic Sensor with Arduino

Below image shows the interfacing of the ultrasonic sensor with Arduino. The components required for the interfacing are,

- Arduino Uno
- Ultrasonic Sensor
- Connecting wires



Source: Arduino Interfacing of Ultrasonic Sensor

Kindly download the **Arduino IDE** for your system and upload the below code in your Arduino Uno,

Code-

```
#define echoPin 2 // attach pin D2 Arduino to pin Echo of HC-SR04
#define trigPin 3 //attach pin D3 Arduino to pin Trig of HC-SR04

// defines variables
long duration; // variable for the duration of sound wave travel
int distance; // variable for the distance measurement

void setup() {
  pinMode(trigPin, OUTPUT); // Sets the trigPin as an OUTPUT
  pinMode(echoPin, INPUT); // Sets the echoPin as an INPUT
  Serial.begin(9600); // // Serial Communication is starting with 9600 of baudrate speed
  Serial.println("Ultrasonic Sensor HC-SR04 Test"); // print some text in Serial Monitor
  Serial.println("with Arduino UNO R3");
}
void loop() {
  // Clears the trigPin condition
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  // Sets the trigPin HIGH (ACTIVE) for 10 microseconds
  digitalWrite(trigPin, HIGH);
```

```
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  // Reads the echoPin, returns the sound wave travel time in microseconds
  duration = pulseIn(echoPin, HIGH);
  // Calculating the distance
  distance = duration * 0.034 / 2; // Speed of sound wave divided by 2 (go and
back)
  // Displays the distance on the Serial Monitor
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
}
```

You can see the output on the serial monitor and it looks like,