Installation of Raspbian on Raspberry pi 3.

The raspbian is installed on 8 GB memory card. To install raspbian first step is to format the memory card in order to make it suitable for installation. To format memory card SD Memory card format tool 5.0 is used. Steps to format the memory card are as follows:

1. Download SD Memory card format tool from www.sdcard.org\downloads\.
2. Install the memory card formatting tool.
3. Format memory card as FAT.
4. Once formatting is completed, memory card is ready for use.

Once the memory card is formatted, it is ready for use. The next step is to download the operating system either raspbian or noobs from official website https://www.raspberrypi.org/downloads/. After downloading the system image of raspbian extract the zip file to some folder. Next step is to write this downloaded system image to memory card. For this purpose Etcher tool is used. Steps to write system image to memory card are as follows:

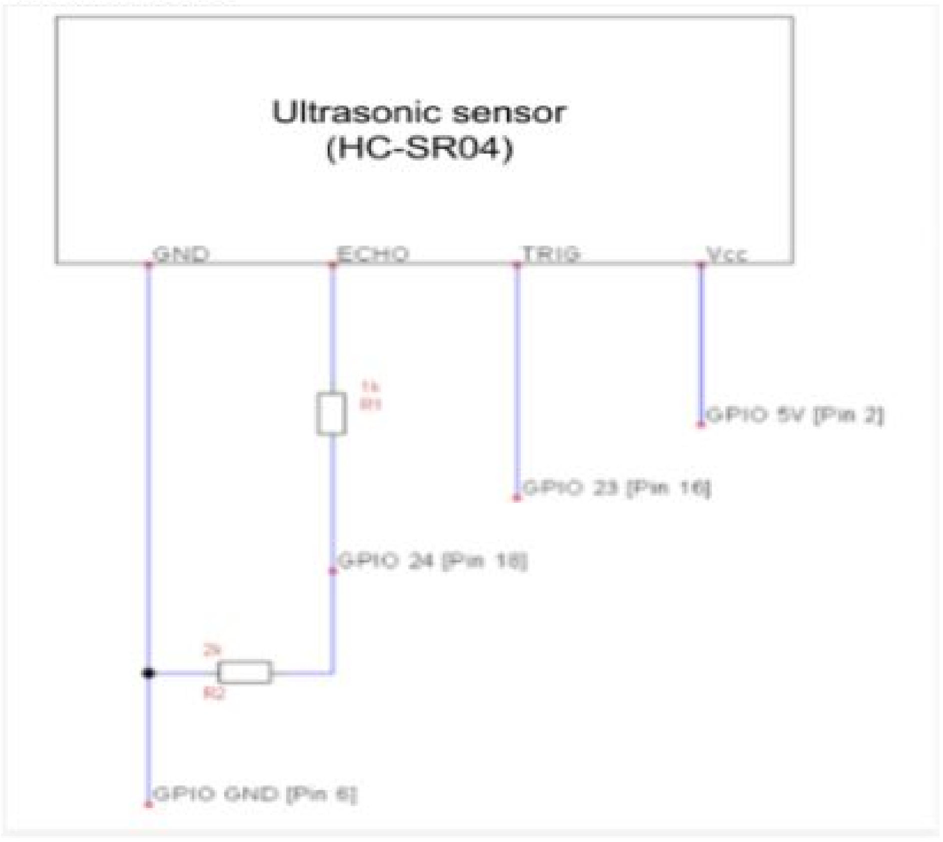
1. Download Etcher and install it from https://etcher.io/.
2. Connect an SD card reader with the SD card inside.
3. Open Etcher and select from your hard drive the Raspberry Pi .img or .zip file you wish to write to the SD card.
4. Select the SD card you wish to write your image to.
5. Review your selections and click 'Flash!' to begin writing data to the SD card.

Once the system image is written to memory card, insert that memory card into raspberry pi and power on the raspberry pi. On the first boot it will take you through all the normal installation steps. Once you are done with that raspberry pi is ready for use.

Connecting Ultrasonic sensor to raspberry pi 3.

HC-SR 04 ultrasonic distance sensor is used to measure water level in storage tank. This sensor operates on 5V. The maximum measuring distance is 200cm.The Ultrasonic transmitter transmits an ultrasonic wave, this wave travels in air and when it gets objected by water it gets reflected back toward the sensor this reflected wave is observed by the Ultrasonic receiver. Once the signal is received by receiver using time distance speed formula we can calculate the water level in the tank. 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 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, along with some clever math, to calculate the distance between the sensor and the reflecting object.

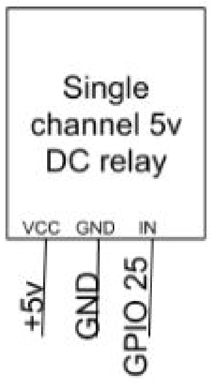
Connection with raspberry pi:



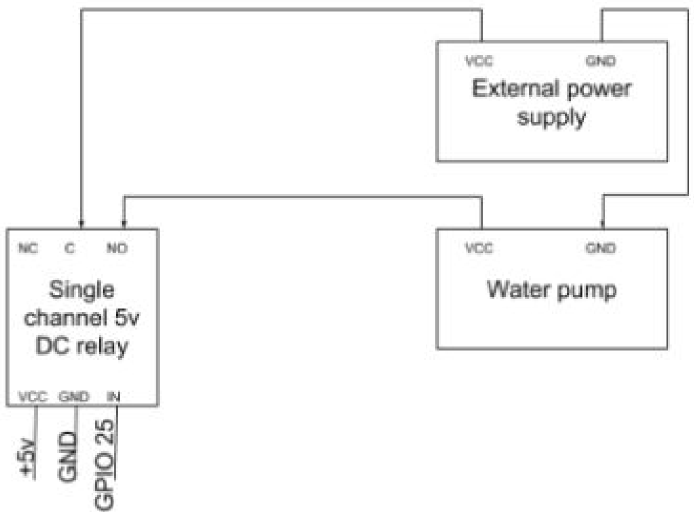
Connecting Relay to raspberry pi3.

The single channel 5v relay is used to control the water pump. The operating voltage of the relay is 5v. This relay is connected to GPIO pin of raspberry pi. Whenever the moisture level is below threshold value water pump is switched on using the relay and also water pump is switched off once threshold level is reached.

Connecting relay to raspberry pi3:



Connecting Water pump to relay:



Python code to integrate all the sensors

final_script_fyp.py

```
import subprocess
import os
import socket
import csv
import time,sched
import RPi.GPIO as GPIO
import glob
import logging
import requests
import json
from errno import ENETUNREACH
headers={'Content-type': 'application/json'} #headers set to specify the content type of the data being sent to the URI
s = sched.scheduler(time.time, time.sleep) #used to start a new thread process
from threading import Timer
from suds.client import Client
URL_get_data = "http://192.168.43.104:5010/get_data" #URI to send JSON data to be inserted into get_data table
URL_update_mois_data="http://192.168.43.104:5010/update_mois_data" #URI to send JSON data to be updated into update_mois_data table
URL_delete_ip="http://192.168.43.104:5010/delete_ip" #URI to send JSON data to be deleated from the table

TRIG=23 # trigger pin of ultrasonic
ECHO=24 # echo pin of ultrasonic
mois = 0
RELAY_1 = 25
RELAY_2=16
GPIO.setmode(GPIO.BCM)
GPIO.setup(RELAY_1,GPIO.OUT)
GPIO.setup(RELAY_2,GPIO.OUT)
GPIO.setup(TRIG,GPIO.OUT)
GPIO.setup(ECHO,GPIO.IN)
moisture_threshold_low=550
moisture_threshold_high=80
water_tank_height=25
water_tank_status = ""
distance = 0
raspberry_id=1

def read_distance(): # measures using the ultrasonic sensor and returns the level of water in the tank
    try:

        water_level=0
        print "Water level sensor testing"
        GPIO.output(TRIG, False)
        print "Waiting For Sensor To Settle"
        time.sleep(2)
        GPIO.output(TRIG, True) # trigger starts sending pulse

        time.sleep(0.00001)

        GPIO.output(TRIG, False) # trigger stops sending pulse
        while GPIO.input(ECHO)==0: # loops till echo receives back the reflected pulse
            pulse_start = time.time()
        while GPIO.input(ECHO)==1:
            pulse_end = time.time()
        pulse_duration = pulse_end - pulse_start # calculates the duration of pulse traversed
        distance = pulse_duration * 17150
        distance = round(distance, 2)
        print distance
        final_distance = distance
        print int(100-((100*distance)/water_tank_height))
        return final_distance
    except:
        return 0


def run_status():  # checks for the functioning status of nodemcus
    try:
        data="rashmipawar921@gmail.com"
        json_data = json.dumps(data) # converts dictionary data into json
         r = requests.get(url = URL_delete_ip, json = json_data, headers = headers) # the data in json format is sent
to the specified URI using GET method
        print "running alive_status_nmcu.py"
        execfile("/home/pi/Desktop/alive_status_nmcu.py") # executes the specified file
    except:
        pass

def run_sensor():  # requests the nodemcus for sensor readings and sends it to server

    temp=0
    raspberry_id="1"
    water_tank_level=30
    water_pump_status="off"
    send_noti="1"
    water_level=20
    ip_list=[]
    f=open("/home/pi/Desktop/nmcu_ip.csv","r") # opens file in read mode
    file_contents=csv.reader(f) # reads each record from csv into a dictionary
    ip_list=list(file_contents) # converts dictionary into list


    moisture_readings={}
    for i in range(len(ip_list)): # iterates for each ip of the alive nodemcus

        try:
            deadline = time.time() + 5.0 # used to set a time limit of 5 seconds
            UDP_IP=ip_list[i][1]
            UDP_PORT=1885
                MESSAGE="aiscmm_smart_irrigation_169.254.152.165_sensor_data" # message sent to the nodemcu
requesting for sensor readings

            print "UDP target ip:", UDP_IP
            print "udp target port:", UDP_PORT
            print "msg:",MESSAGE

            sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM) # creates a socket
            try:
                    sock.sendto(MESSAGE,(UDP_IP, UDP_PORT)) # udp message is sent to the specific nodemcu using ip
and port
                print "Packet send"
            except IOError as e:
                if e.errno==ENETUNREACH:
                    pass
            sock.settimeout(deadline-time.time()) # closes connection after timeout
            chunks=[]
            bytes_recd=0
            chunks=sock.recv(4096) # receives data sent by nodemcu
            print chunks
            chunks_list=[]
            chunks_list=chunks.split(",") # list of sensor readings is created

            temp=chunks_list[0]
            mois = chunks_list[2]
            moisture_readings[i]=float(mois)
            raspberry_id="1"
            water_pump_status="off"
            send_noti="1"

            flag = 2
        except socket.timeout:
            print "time out"
            moisture_readings[i]=2000
            continue

    no_of_pipes=2 #static
    pipe_list1=[]
    pipe_list2=[]
    for i in moisture_readings:
        pipe_no=(i+1)%no_of_pipes;
        if pipe_no==1:
            pipe_list1.append(moisture_readings[i]) # appends into list readings of moisture near pipe 1
        else:
            pipe_list2.append(moisture_readings[i]) # appends into list readings of moisture near pipe 2
    mois_avg1=0;
    mois_avg2=0;
    faulty_count1=0
    faulty_count2=0
    mois_avg_list=[]
    for i in range(len(pipe_list1)):
        if pipe_list1[i]==2000:
            faulty_count1=faulty_count1+1
            continue
        else:
            mois_avg1=mois_avg1+pipe_list1[i]
             mois_avg1=mois_avg1/(len(pipe_list1)-faulty_count1) # calculates average of moisture readings near
pipe1
    mois_avg_list.append(mois_avg1)

    for i in range(len(pipe_list2)):
        if pipe_list2[i]==2000:
            faulty_count2=faulty_count2+1
            continue
        else:
            mois_avg2=mois_avg2+pipe_list2[i]
             mois_avg2=mois_avg2/(len(pipe_list2)-faulty_count2) # calculates average of moisture readings near
pipe1
    mois_avg_list.append(mois_avg2)

    print mois_avg_list

    water_tank_status = "off"
        try:

        for i in range(len(mois_avg_list)):
            try:
                   if(mois_avg_list[i]>moisture_threshold_low): # checks if moisture level measured is lesser than
required threshold
                distance = read_distance()
                water_tank_level=distance
                print(distance, water_tank_height)
                if(distance<water_tank_height): # checks if there is sufficient water in the tank
                    print("inside the pump")
                    if i==0:
                        GPIO.output(RELAY_1,True) # turns on the water pump 1

                        water_tank_status = "motor on"
                        print water_tank_status
                         data={"raspberry_id":raspberry_id,"pump_id":i+1,"mois":mois_avg_list[i]} # creates dictionary
of current values to be sent to server
                        print data

                        json_data = json.dumps(data) # converts dictionary to json
```

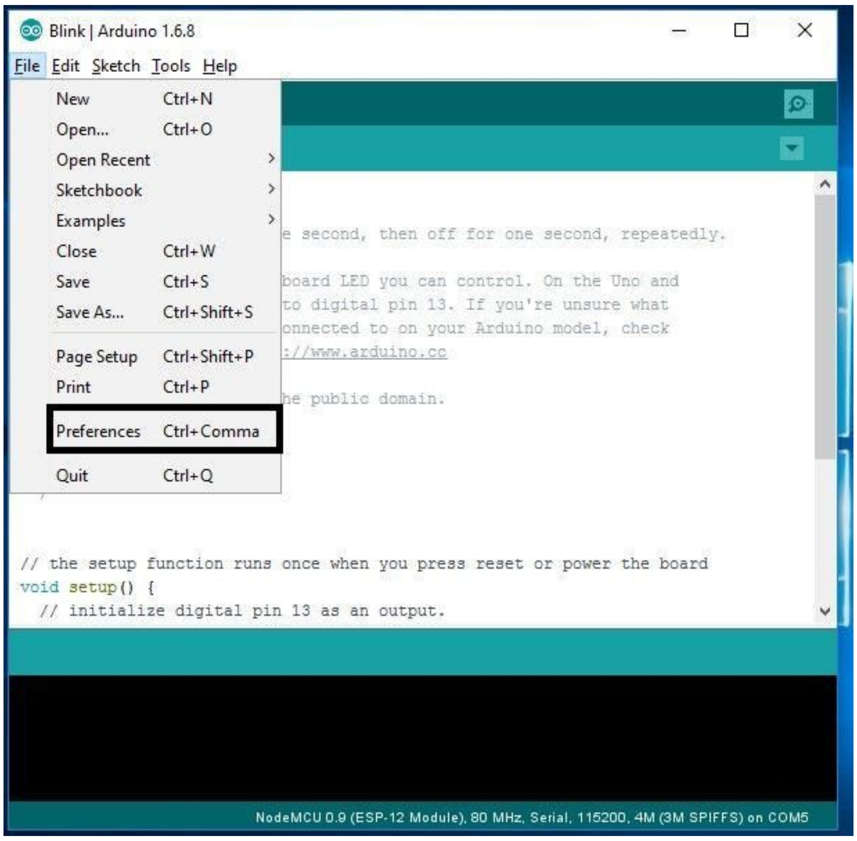
```
                                r = requests.get(url = URL_update_mois_data, json = json_data, headers = headers) # sends
json data to the URI
                            print "data is updated to db"

                            time.sleep(10)
                            GPIO.output(RELAY_1, False) # turns off the water pump 1
                            water_tank_status = "off"
                            print water_tank_status

                        else:
                            GPIO.output(RELAY_2,True) # turns on the water pump 2

                            water_tank_status = "motor on"
                            print water_tank_status
                             data={"raspberry_id":raspberry_id,"pump_id":i+1,"mois":mois_avg_list[i]} # creates dictionary
of current values to be sent to server

                            json_data = json.dumps(data) # converts dictionary to json
                                r = requests.get(url = URL_update_mois_data, json = json_data, headers = headers) # sends
json data to the URI
                            print "data is updated to db"

                            time.sleep(10)
                            GPIO.output(RELAY_2, False) # turns off the water pump 2
                            water_tank_status = "off"
                            print water_tank_status

                    else:

                        print water_tank_status

                else:

                    print water_tank_status
            except:
                continue


data={"temp":temp,"mois":mois_avg_list[i],"raspberry_id":raspberry_id,"water_tank_level":water_tank_level,
"water_tank_status":water_tank_status,"send_noti":send_noti} # creates dictionary of current values to be
sent to server

            json_data = json.dumps(data) # converts dictionary to json
            r = requests.get(url = URL_get_data, json = json_data, headers = headers) # sends json data to the URI
            print "data is updated to db"
        except:
            pass


while(1):
        s.enter(30, 1, run_sensor, ()) # schedules the process thread to run every 30 seconds
        s.enter(300, 2, run_status, ()) # schedules the process thread to run every 300 seconds
        s.run()
```

alive_status_nmcu.py

```
import subprocess
import os
import socket
import csv
import time,sched
import RPi.GPIO as GPIO
import glob
import logging
import json
import requests
s = sched.scheduler(time.time, time.sleep)
from threading import Timer
from suds.client import Client

headers={'Content-type': 'application/json'} #headers set to specify the content type of the data being sent to
the URI
URL = "http://192.168.43.104:5010/insert_ip" #URI to send JSON data to be inserted into insert_ip table

def run_sensor_status(): # checks is the nodemcu is functioning else sends the ip of faulty to server
    faulty_nmcu=[]
    raspberry_id="1"

    send_noti="6"

    ip_list=[]
    f=open("/home/pi/Desktop/nmcu_ip.csv","r") # opens file in read mode
    file_contents=csv.reader(f) # reads each record from csv into a dictionary
    ip_list=list(file_contents) # converts dictionary into list

    for i in range(len(ip_list)):  # iterates for each ip of the nodemcus



        try:
            deadline = time.time() + 5.0 # used to set a time limit of 5 seconds
            UDP_IP=ip_list[i][1]

            UDP_PORT=1885
             MESSAGE="aiscmm_smart_irrigation_169.254.152.165_sensor_status" # message sent to the nodemcu
requesting for sensor status

            print "UDP target ip:", UDP_IP
            print "udp target port:", UDP_PORT
            print "msg:",MESSAGE
            sock=socket.socket(socket.AF_INET,socket.SOCK_DGRAM) # creates a socket
            sock.sendto(MESSAGE,(UDP_IP, UDP_PORT)) # udp message is sent to the specific nodemcu using ip and
port
            print "Packet send"
            sock.settimeout(deadline-time.time()) # closes connection after timeout
            chunks=[]
            bytes_recd=0
            chunks=sock.recv(4096) # receives data sent by nodemcu

            chunks_list=[]
            chunks_list=chunks.split(",")

            st=chunks_list[0]
            print st
            raspberry_id="1"
            send_noti="6"
            if st!="alive":
                faulty_nmcu.append(ip_list[i]) # unrechable ips are appended into the list


        except socket.timeout:
            print "time out"
            faulty_nmcu.append(ip_list[i])
            continue
    print faulty_nmcu
    return faulty_nmcu


faulty_nmcu=run_sensor_status()
for row in faulty_nmcu: # iterates through each faulty nodemcu ip

data={"nodemcu_id":row[0],"raspberry_id":1,"nodemcu_ip":row[1],"raspberry_ip":"169.254.152.165","email"
:"rashmipawar921@gmail.com"} # creates dictionary of current values to be sent to server

    json_data = json.dumps(data) # converts dictionary to json
    r = requests.get(url = URL, json = json_data, headers = headers) # sends json data to the URI
print "data is updated to db"
```

Steps to Setup Arduino IDE for NODEMCU ESP8266

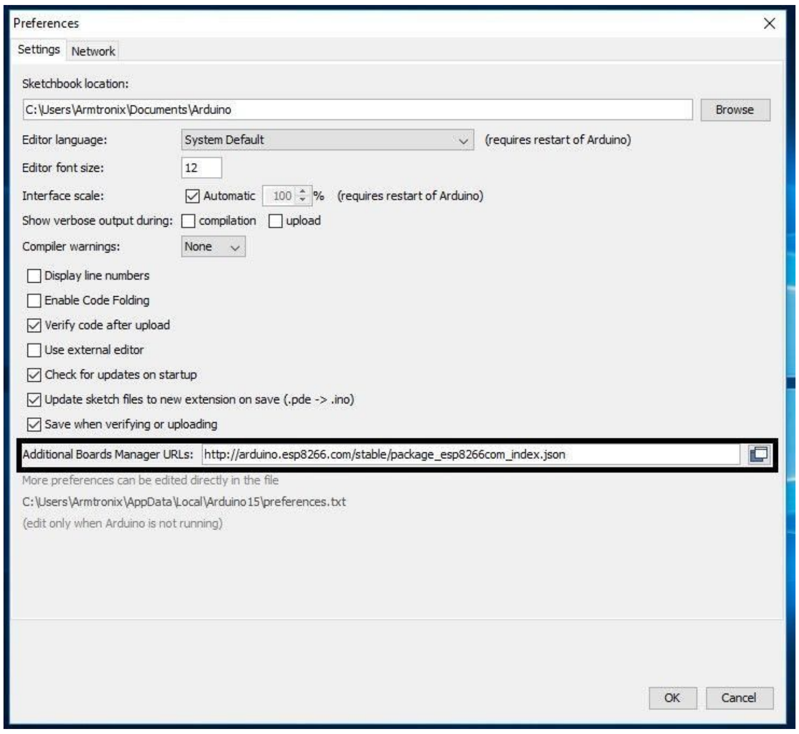
1. Install Arduino IDE software from the link http://www.arduino.cc/en/main/software.
2. Open File ->Preferences



```
e second, then off for one second, repeatedly.

board LED you can control. On the Uno and
to digital pin 13. If you're unsure what
onnected to on your Arduino model, check
://www.arduino.cc

he public domain.

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 13 as an output.
```

NodeMCU 0.9 (ESP-12 Module), 80 MHz, Serial, 115200, 4M (3M SPIFFS) on COM5

3.Adding ESP8266 Board Manager



Sketchbook location: C:\Users\Armtronix\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 12

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers

☐ Enable Code Folding

☑ Verify code after upload

☐ Use external editor

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☑ Save when verifying or uploading

Additional Boards Manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json

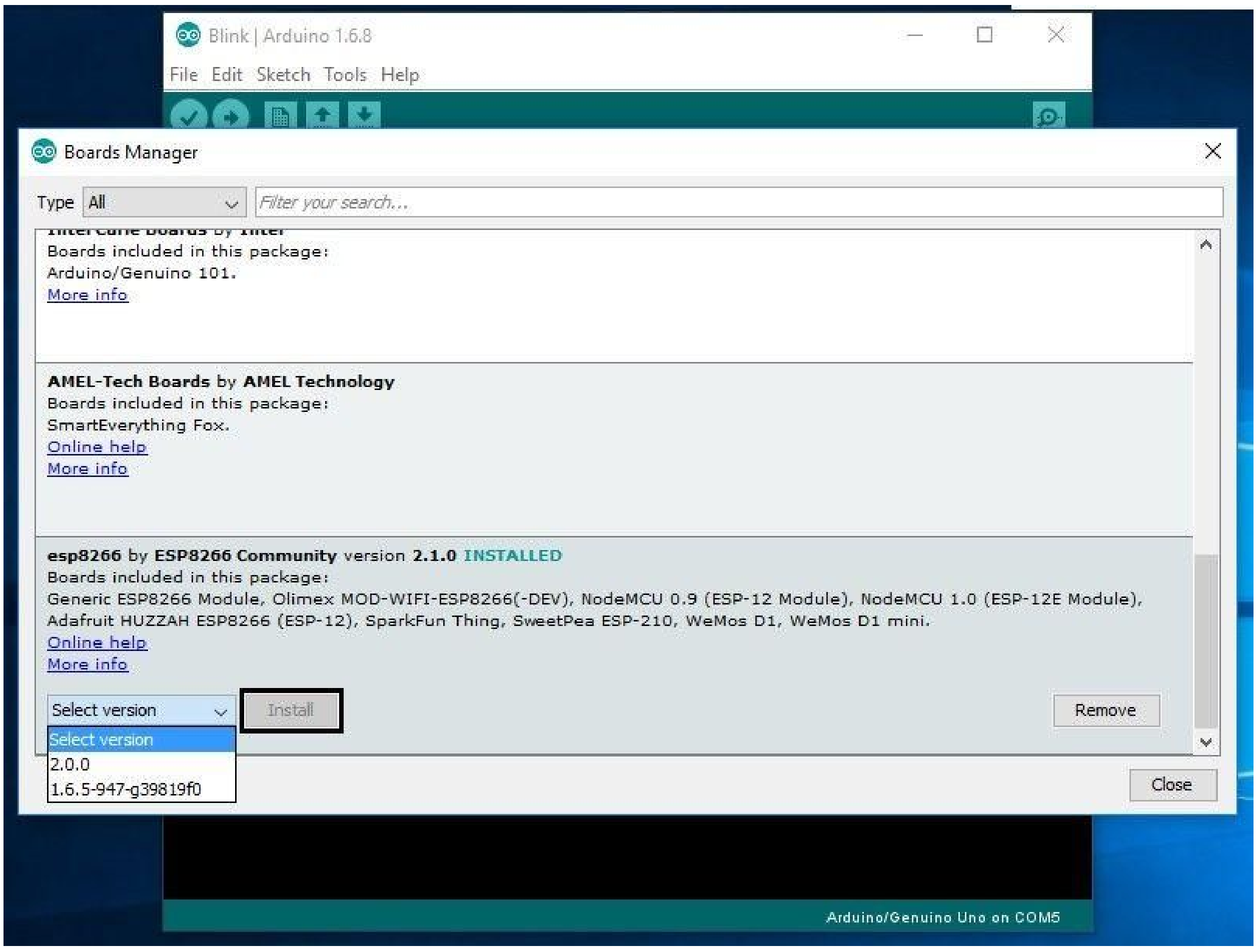
More preferences can be edited directly in the file

C:\Users\Armtronix\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

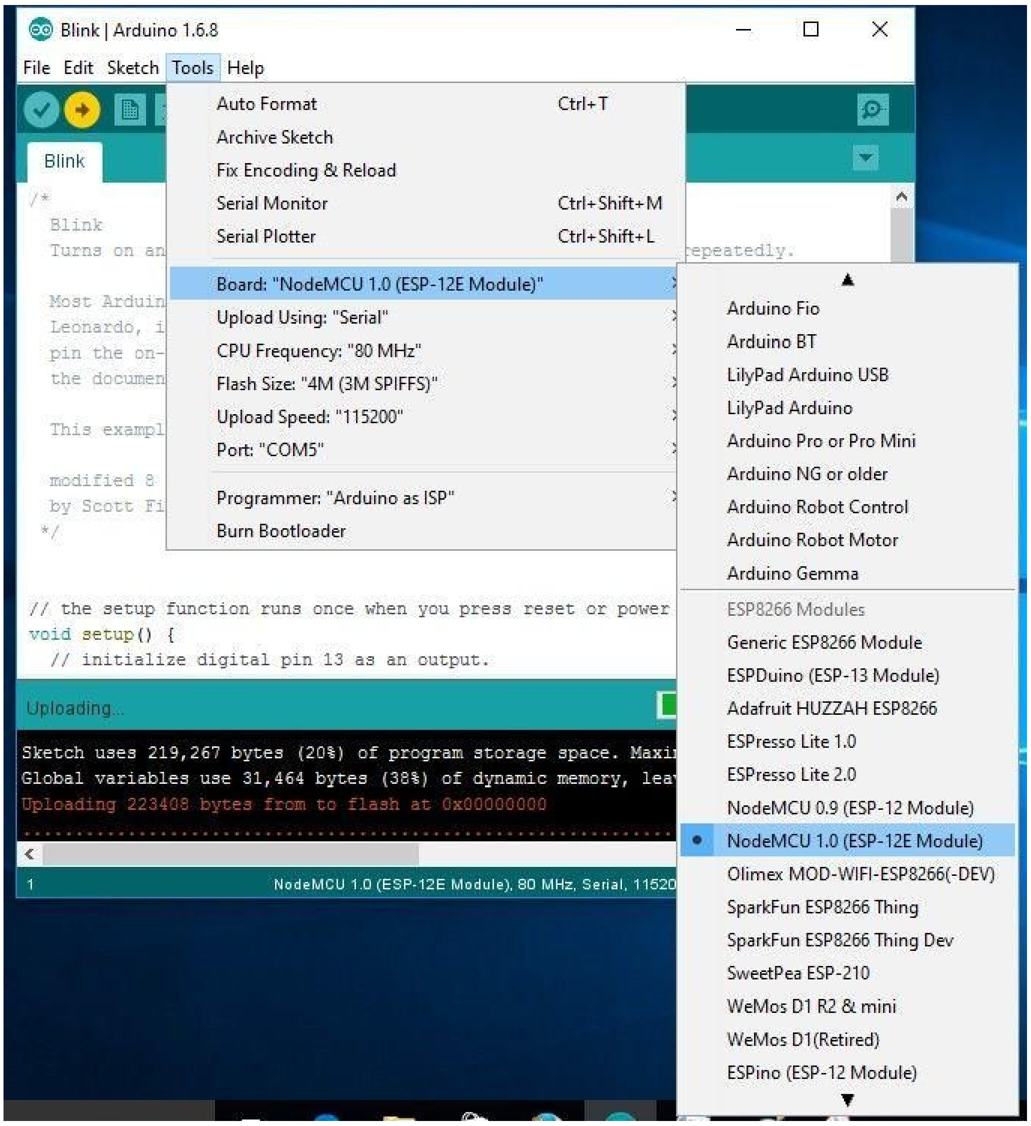
In the Additional Boards Manager enter below URL.

http://arduino.esp8266.com/stable/package_esp8266com_index.json


Blink | Arduino 1.6.8
File Edit Sketch Tools Help
Boards Manager
Type All
Filter your search...
Intel Curie Boards by Intel
Boards included in this package:
Arduino/Genuino 101.
More info
AMEL-Tech Boards by AMEL Technology
Boards included in this package:
SmartEverything Fox.
Online help
More info
esp8266 by ESP8266 Community version 2.1.0 INSTALLED
Boards included in this package:
Generic ESP8266 Module, Olimex MOD-WIFI-ESP8266(-DEV), NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Adafruit HUZZAH ESP8266 (ESP-12), SparkFun Thing, SweetPea ESP-210, WeMos D1, WeMos D1 mini.
Online help
More info
Select version
Select version
2.0.0
1.6.5-947-g39819f0
Install
Remove
Close
Arduino/Genuino Uno on COM5

The Boards Manager window opens, scroll the window page to bottom till you see the module with the name ESP8266. Once we get it, select that module and select version and click on the Install button. When it is installed it shows Installed in the module as shown in the figure and then close the window.

4.Selecting ESP8266 Arduino Board



Board: "NodeMCU 1.0 (ESP-12E Module)"
Upload Using: "Serial"
CPU Frequency: "80 MHz"
Flash Size: "4M (3M SPIFFS)"
Upload Speed: "115200"
Port: "COM5"
Programmer: "Arduino as ISP"

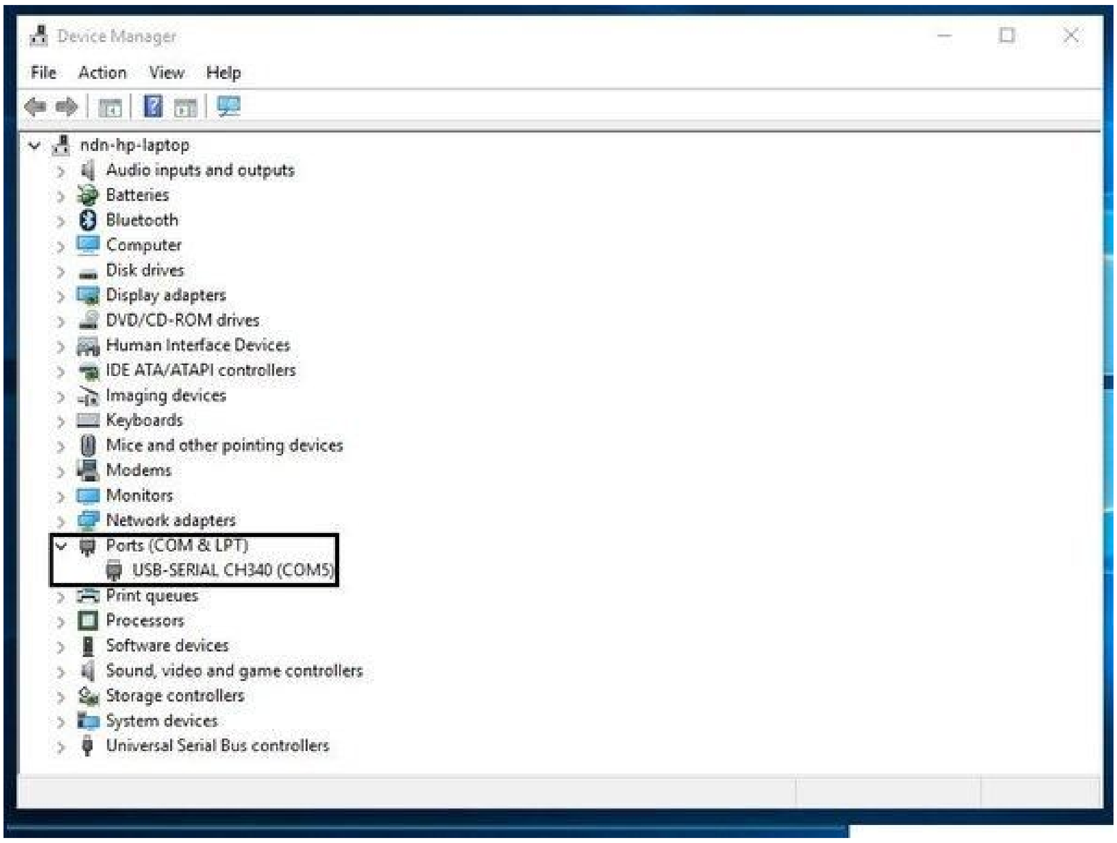
```
/*
  Blink
  Turns on an                                   repeatedly.

  Most Arduin
  Leonardo, i
  pin the on-
  the documen

  This exampl

  modified 8
  by Scott Fi
 */

// the setup function runs once when you press reset or power
void setup() {
  // initialize digital pin 13 as an output.
```

```
Sketch uses 219,267 bytes (20%) of program storage space. Maxi
Global variables use 31,464 bytes (38%) of dynamic memory, lea
Uploading 223408 bytes from to flash at 0x00000000
............................................................
```

NodeMCU 1.0 (ESP-12E Module), 80 MHz, Serial, 11520

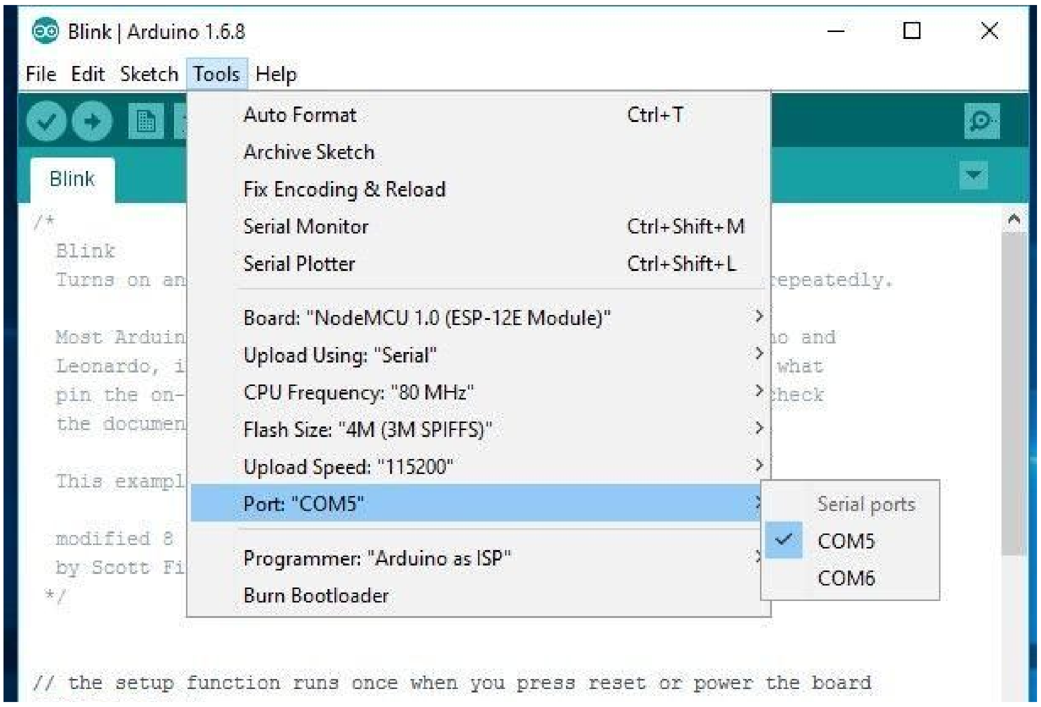
To run the esp8266 with Arduino we have to select the **Board: “Arduino/Genuino Uno”** and then change it to **NodeMCU 1.0 (ESP-12E Module)** or other esp8266 modules. This can be done by scrolling down, as shown in the figure.

5.Connecting ESP8266 to the PC



Let's connect the ESP8266 module to your computer through USB cable as shown in the figure. When module is connected to the USB, COM port is detected eg: here COM5 is shown in the figure.

6. Selecting COM Port



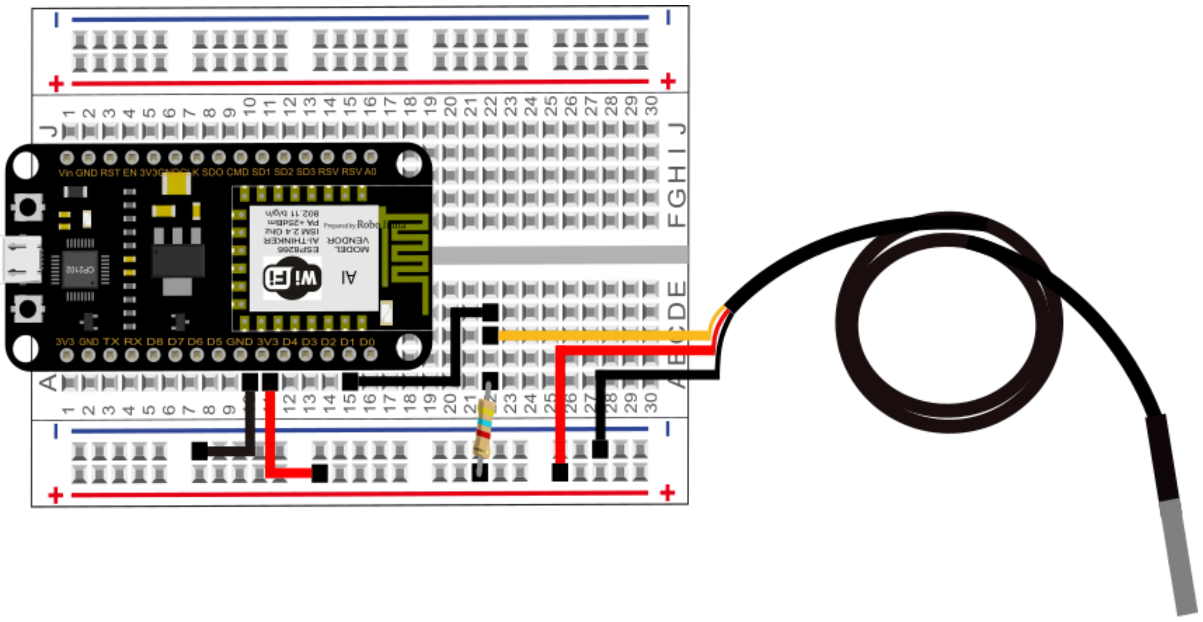
Board: "NodeMCU 1.0 (ESP-12E Module)"
Upload Using: "Serial"
CPU Frequency: "80 MHz"
Flash Size: "4M (3M SPIFFS)"
Upload Speed: "115200"
Port: "COM5"

```
// the setup function runs once when you press reset or power the board
```

7.Upload the program to NodeMCU and check the output on the Serial Monitor.

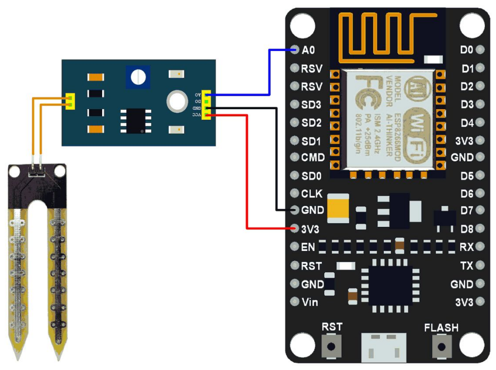
Connecting Temperature Sensor to the Node MCU

DS18B20 is used for measuring the soil temperature. It is one wire interface sensor, very common in PCB embedded electrical circuits. Its unique one wire protocol requires only one port pin for communication and needs no other external components to work. It is waterproof sensor hence it can be used in humid and wet environmental conditions. It can operate on 3.3v and 5.5v.
This sensor has 3 pins.
Vcc - 5V
Gnd - Gnd
Data



Connecting Moisture Sensor to the Node MCU

KG003 soil moisture sensor is used to measure the soil moisture. This is a simple water sensor that can be used to detect soil moisture. Module output is high level when the soil moisture deficit or output is low. The sensitivity can be adjusted by using the digital potentiometer. The output is available in both analog and digital format. In the project analog interfacing is used for accurate output. The soil moisture sensor consists of two probes. The two probes allow the current to pass through the soil and then it gets the resistance value to measure the moisture value. When there is more water, the soil will conduct more electricity which means that there will be less resistance. Therefore, the moisture level will be higher. Dry soil conducts electricity poorly, so when there will be less water, the soil will conduct less electricity which means that there will be more resistance. Therefore, the moisture level will be lower. The sensor has 4 pins. First two pins are Vcc and GND. The next two pins are output pins.A0 pin is giving analog output and D0 pin is giving digital output. We are using analog output of sensor to get readings. As raspberry pi does not have any analog pin, we have to use an ADC to convert analog output of sensor into digital. The ADC used here is ADS1115.



NODEMCU FINAL SCRIPT

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 2 // DS18B20 on NodeMCU pin D4
OneWire oneWire(ONE_WIRE_BUS);
#include<string.h>
#include<stdlib.h>

DallasTemperature DS18B20(&oneWire);
const char* nmcu_id="node_1";
const char* ssid = "123";//wifi name
const char* password = "lasya123";//password

WiFiUDP Udp;
unsigned int localUdpPort = 1885;  // local port to listen on
char incomingPacket[255];  // buffer for incoming packets
char  replyPacket[] = "";  // a reply string to send back
char status_reply[]="alive";

void setup()
{
  Serial.begin(115200);
  Serial.println();

  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println(" connected");

  Udp.begin(localUdpPort);
  Serial.printf("Now listening at IP %s, UDP port %d\n", WiFi.localIP().toString().c_str(), localUdpPort);
}


void loop()
```

```
{
 int packetSize = Udp.parsePacket();
 if (packetSize)
 {
   // receive incoming UDP packets
      Serial.printf("Received %d bytes from %s, port %d\n", packetSize, Udp.remoteIP().toString().c_str(),
Udp.remotePort());
   int len = Udp.read(incomingPacket, 255);
   if (len > 0)
   {
     incomingPacket[len] = 0;
   }
   Serial.printf("UDP packet contents: %s\n", incomingPacket);
   if (incomingPacket=="status")
   {
     Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
     Udp.write(status_reply);
     Udp.endPacket();
   }

   else{
    float temp_0;
 float temp_1;
 DS18B20.requestTemperatures();
 temp_0 = DS18B20.getTempCByIndex(0); // Sensor 0 will capture Temp in Celcius
 temp_1 = DS18B20.getTempFByIndex(0); // Sensor 0 will capture Temp in Fahrenheit

 Serial.print("Temp_0: ");
 Serial.print(temp_0);
 Serial.print(" oC . Temp_1: ");
 Serial.print(temp_1);
 Serial.println(" oF ");
 delay(1000);

 // read the input on analog pin 0:
 float moist = analogRead(A0);
 // print out the value you read:
 Serial.println(moist);
   delay(1000);
    sprintf(replyPacket,"%f,%f,%f",temp_0,temp_1,moist);


    Serial.println(replyPacket);

    // send back a reply, to the IP address and port we got the packet from
    Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());
    Udp.write(replyPacket);
    Udp.endPacket();
   }
 }
}
```

NOTE: Before running the above scripts create a csv file containing the ip addresses of all the nodemcus installed in the farm with the name: “nmcu_ip.csv”. Create another csv file with the name “ping_ip_list.csv”.