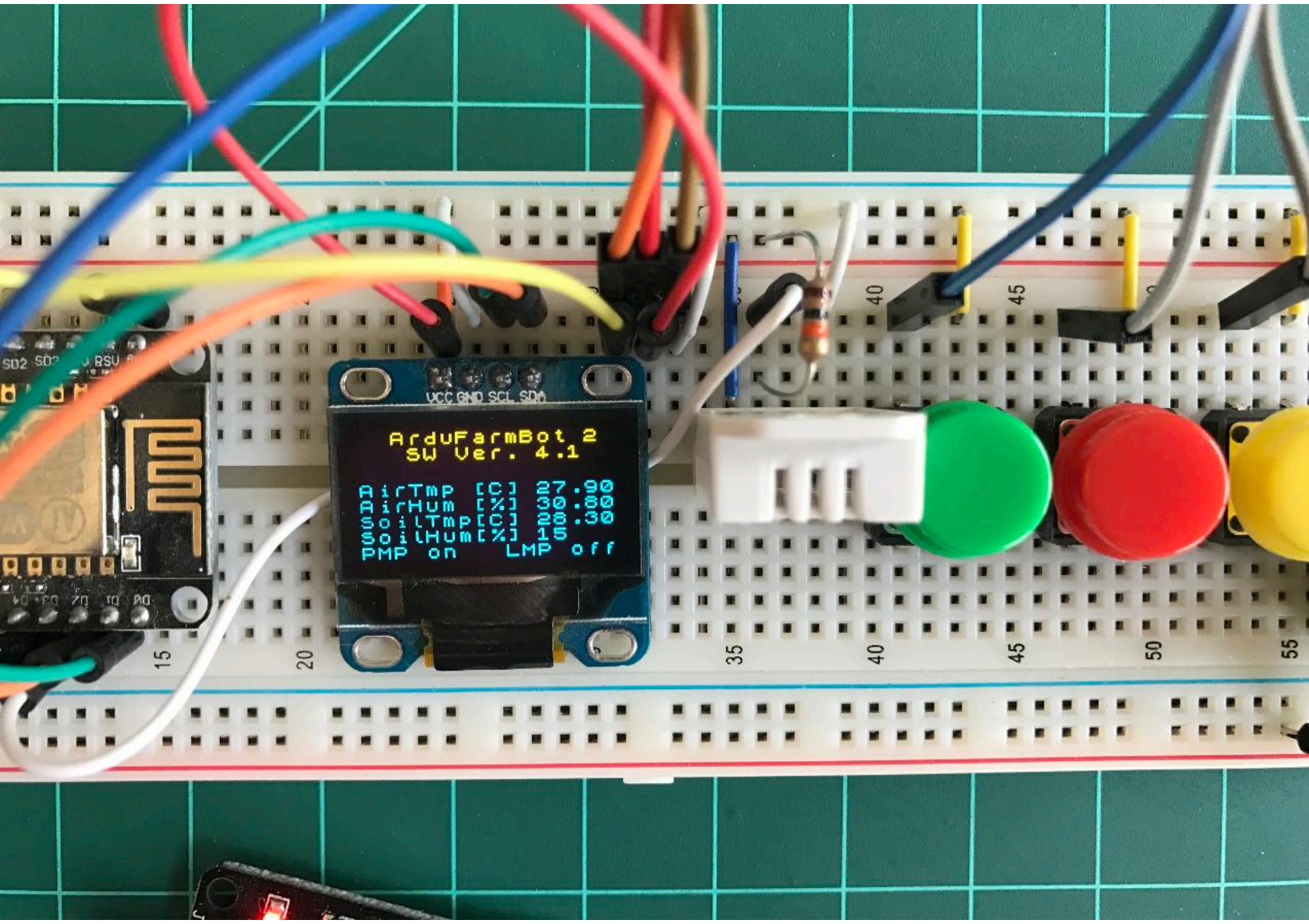
MJRoBot Tutorials

Volume 1



ArduFarmBot

Tomato garden automation with help of "Internet of Things" - IoT

MJRoBot.org

www.EngineeringBooksPdf.com

ArduFarmBot

Tomato garden automation with help of "Internet of Things" - IoT

Marcelo José Rovai

Mauricio Teixeira Pinto

www.EngineeringBooksPdf.com

Table of Contents

Preface – MJRoBot Tutorials 4

Preface - ArduFarmBot 5

The Book 6

Introduction 8

Part 1 - ArduFarmBot: local station 11
1.1 - Bill of Materials (“BoM”) for parts 1 and 2 12
1.2 - Installing, Programing and Testing Sensors 13
1.3 - Adding a LCD for local monitoring 20
1.4 - Actuators and buttons for local control 23
1.5 - Going deeper with a real Soil Moisture Sensor 28
1.6 - Completing the Local Control Station 32
1.7 - It’s show time! 37
1.8 - Changing to a “Small Form Factor” 42
1.9 - Laboratory functional tests 44
1.10 - “Test Drive”: Watering a Tomato plant with ArduFarmBot 46
1.11 - ArduFarmBot in action: Mauricio’s garden 47

Parte 2 - ArduFarmBot: Remote Station 49
2.1 - The IoT Approach 50
2.2 - Completing the Hardware 51
2.4 - “Data Storage Cloud”: The ThinkSpeak.com 56
2.5 - Commanding actuators from the web 60
2.6 - Implementing a dedicated Webpage 66
2.7 - Returning to the brain. A Sensor-Actuator Matrix approach 68
2.8 - Code optimization 72
2.9 - ArduFarmBot in action: Marcelo’s garden 76

Part 3 - The ArduFarmBot 2 78
3.1 - Bill of Materials (“BoM”) for part 3 80
3.2 - The NodeMCU 81
3.3 - Using Arduino IDE with NodeMCU com 83
3.4 - Installing the OLED display 87
3.5 - Capturing Air Temperature and Humidity 90
3.7 - Collecting Soil Temperature 94
3.8 - Completing the HW 97
3.9 - Local Control Station – Concluding the Code 103
3.10 - Making our Gardening System fully automatic 106
3.11 - Building an App BLYNK 110
3.12 - Changing code to introduce Blynk 113
3.14 - ArduFarmBot 2 real test 121

Conclusion 124

www.EngineeringBooksPdf.com

Preface – MJRoBot Tutorials

Our generation grew up from a science fiction world perspective, packed with cartoons and so many series that sowed in our unconscious this reality that we would live in the future.

What we see today is not very different from what we dreamed as children - things communicate with each other and interact with their owners which already has a name: "IoT - Internet of Things"!

Objectively it will allow (already allows) an interconnection of real world objects with the virtual world through sensors. Exchanging information about status, location, operation, problems, decisions, etc., where algorithms will be responsible for processing that information, responding to the object, and generating a large amount of data that will be stored on powerful servers in the cloud. And if you want, everyone can interact with your world.

There are many facilities that we find today with collaborative systems for sharing knowledge on the Internet. The great work is to systematize this knowledge in order to become practical for the average individual, in this case you. Making it useful to your day to day, giving you comfort, and improving your daily life.

Of course, you cannot predict the future but you can help to build it.

For those interested in participating or helping us build such reality is that we are launching these tutorials.

But beware of the "hackers" ... they may invade your refrigerator.

www.EngineeringBooksPdf.com

Preface - ArduFarmBot

Planting and harvesting are activities that have always been present in humanity. Man was an extractive specie who dominated the art of agriculture and ceased to be nomadic. He fixed roots.

After several millennia, planting and harvesting became a hobby for the common of mortals.

For many it can be an impossible task! How much water, how much heat should it have? How much time to harvest? Is it growing properly?

If you do not want to have these doubts you better go to the supermarket. But surely you will not have the same thrill of seeing your own creation turning alive and being able to tell your friends that the lunch salad was from your own vegetable garden.

Internet of Things comes to help you if you want help, of course.

It was by this thinking that we, ordinary mortals interested in electronics and a good “pasta alla bolognese”, decided to develop this project for tomato garden automatization.

We hope it will bring many fruits for you. Literally.

www.EngineeringBooksPdf.com

The Book

This book uses the electronic controller *ArduFarmBot* as a basis for learning how to work in both HW and SW, with: a) LCD and OLED type displays; b) LEDs and buttons; c) Activation of pumps and lamps via relays and d) Sensors such as: DHT22 (temperature and relative air humidity), DS18B20 (soil temperature), YL69 (soil moisture) and LDR (luminosity).

All key stages of the project are documented in detail through explanatory texts, block diagrams, high-resolution color photos, electrical diagrams using *Fritzing* application, complete codes stored in *GitHub* and *YouTube* videos.

Two versions of the electronic controller "ArduFarmBot" are developed in detail in this book. From capture of data coming from a garden, such as air and soil temperature, relative humidity, soil moisture and luminosity, the ArduFarmBot helps to control when a crop should receive heat and water. Control will happen automatically, locally and remote via internet

The book is divided into 3 parts.

In the first part, the *Arduino Nano* is the starting point for development of a *local* version of ArduFarmBot, that can be controlled both, manually and automatically.

In the second part, the book dives into automation design, introducing remote operation through the creation of a webpage. The *ESP8266-01* is used for Wi-Fi connection, sending data to a specialized web service in the field of IoT, the *ThingSpeak.com*.

In the third part, a second version of "ArduFarmBot" is developed, introducing the *NodeMCU ESP8266-12E*, a powerful and versatile IoT device, which replaces both the Arduino Nano and the ESP8266-01, used in the earlier parts of the book. In this last part of the book, a new service platform of the IoT universe, the *Blynk*, is also explored.

www.EngineeringBooksPdf.com

ArduFarmBot

www.EngineeringBooksPdf.com

Introduction

In the following pages, we will develop an automatic gardening system, the "ArduFarmBot" to decide timing and amount of heat and water a tomato plant should receive based on its data previously captured (temperature, relative air humidity, lightness and soil moisture).

In addition, the ArduFarmBot will allow operator's manual intervention either locally or remotely via Internet, to control a water pump and an electric lamp (the latter to be used in heat generation for plants).

In summary, the system should:

Receive as input:

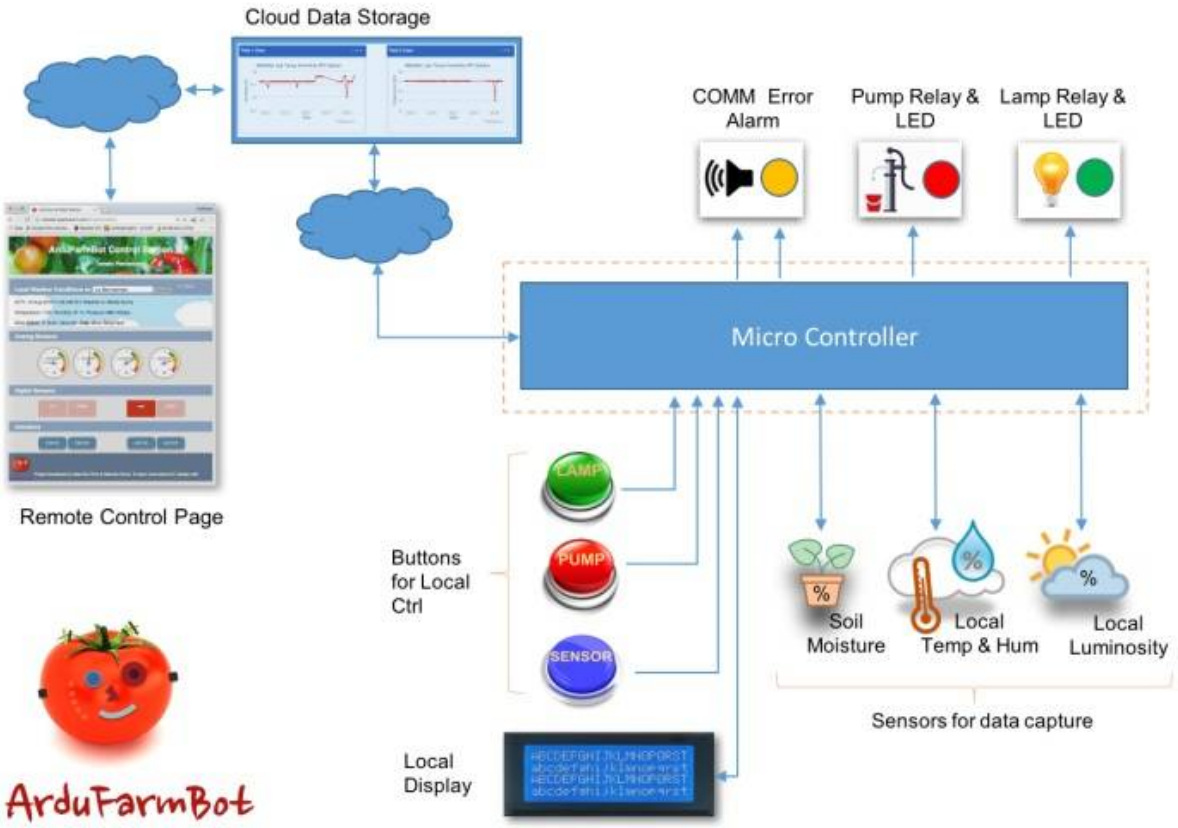
- Sensors (analog data):
  - Temperature
  - Humidity
  - Luminosity
  - Soil humidity
- Buttons:
  - Pump ON/OFF
  - Lamp ON/OFF

Provide as an output:

- Actuators:
  - Pump control Relay
  - Lamp control Relay
- Signalization (digital data):
  - Visual and sound for status/error indication
  - Pump status Visual
  - Lamp status Visual
- Data Display
  - All analog and digital data should be available for instant evaluation
- Data Storage
  - Historic data should be storage remotely and optionally also locally.

www.EngineeringBooksPdf.com

Underneath diagram shows the main components of the Project:



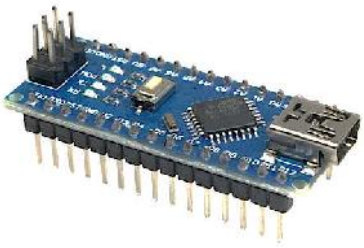
As well, video below describes the first laboratory prototype used for testing:

https://youtu.be/SwgKzfAvWlI

And this one shows how commands will work locally and remotely via Webpage:

https://youtu.be/fcRA6g8ZGS8

www.EngineeringBooksPdf.com



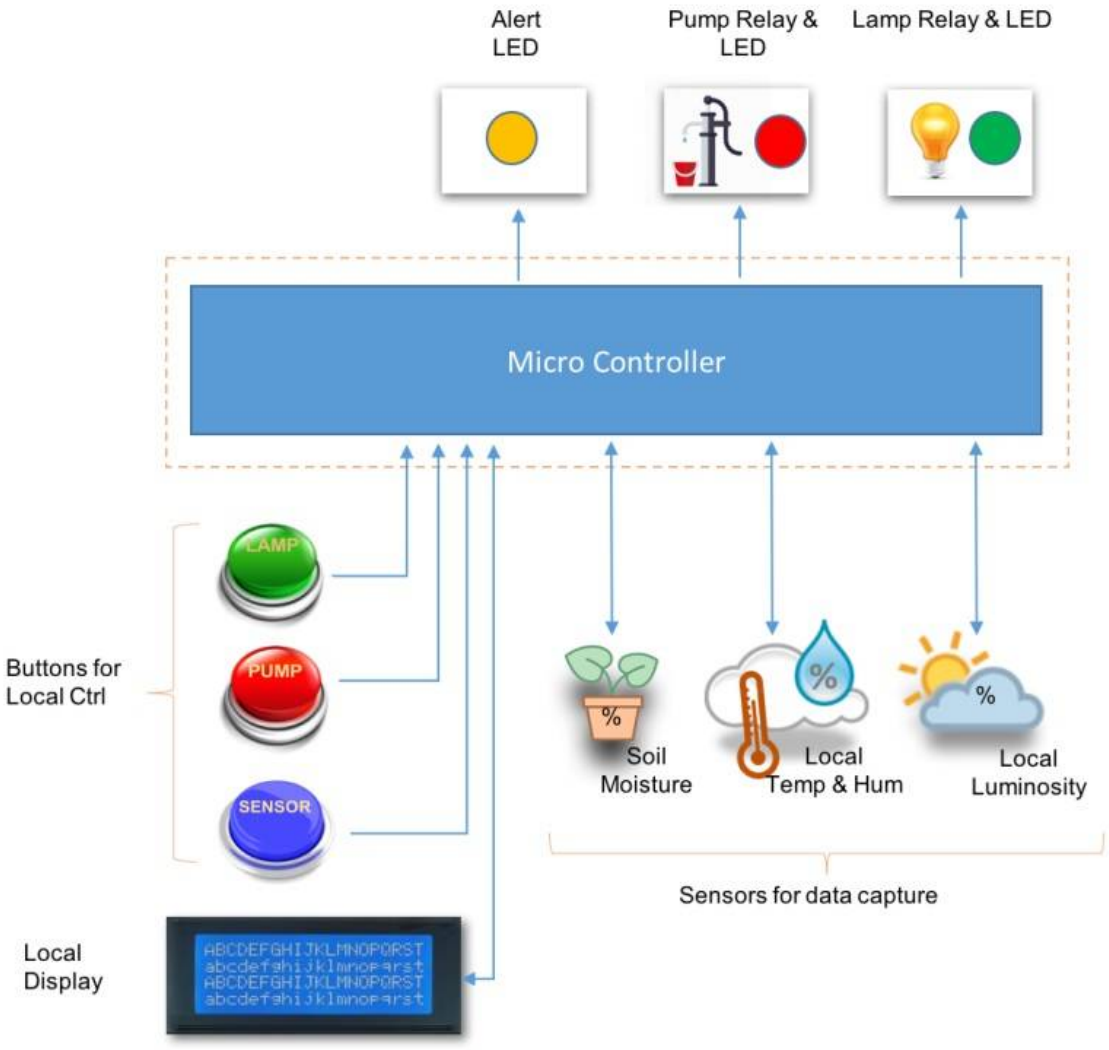
1

www.EngineeringBooksPdf.com

Part 1 - ArduFarmBot: local station

In this first part, we will explore local station developing HW and SW to work with sensors, actuators, learning how to display data, etc.

Below is a simplified block diagram of local station version 1:



www.EngineeringBooksPdf.com

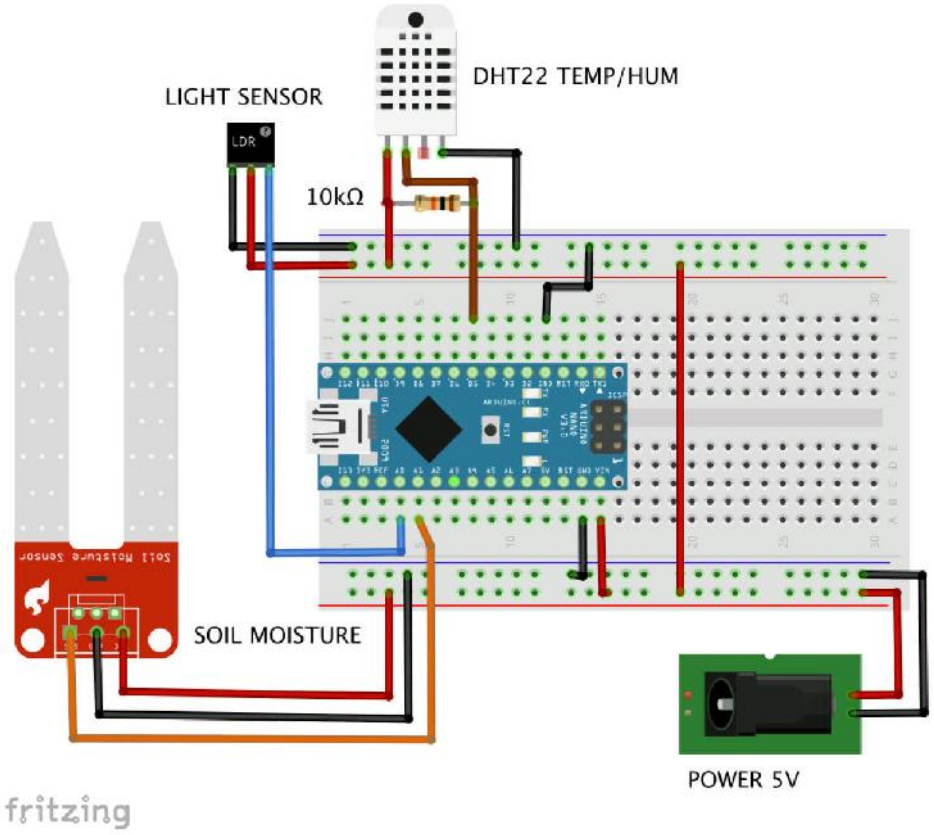
1.1 - Bill of Materials (“BoM”) for parts 1 and 2

The main ArduFarmBot components are (values in USD, for reference only):

- ✓ Arduino Nano – ($8.00)
- ✓ Temperature and Humidity Sensor DHT22 or DHT11 – ($4.00)
- ✓ Luminosity Sensor – AD-018 Photo resistor module or equivalent – ($1.00)
- ✓ 2X Soil Moisture Sensor – ($2.00)
- ✓ LCD I2C 20X4 ($14.00)
- ✓ LEDs (3X) ($1.00)
- ✓ Esp8266-01 – ($6.00)
- ✓ Active Buzzer – Ky-12 - ($0.60)
- ✓ 2 X 5v Relay Module - ($12.00)
- ✓ Jump wires - (S1.00)
- ✓ Resistor 10K ohms – ($0.03)
- ✓ Resistor 2.2K ohms – ($0.03)
- ✓ Resistor 1.0 K ohms – ($0.03)
- ✓ Resistor 220 ohms – ($0.03)
- ✓ Arduino Nano Shield (“Funduino”) – ($7.00)
- ✓ Membrane keyboard (4 Keys) – ($7.00)
- ✓ Plastic Box

www.EngineeringBooksPdf.com

1.2 - Installing, Programing and Testing Sensors



A. DHT22: Temperature and Humidity Sensor

The first sensor to be installed and tested is the DHT 22, a digital relative humidity and temperature sensor that by using a capacitive humidity sensor and a thermistor measuring surrounding air, spits out a digital signal on the data pin (no analog input pins needed).

According to its Datasheet:

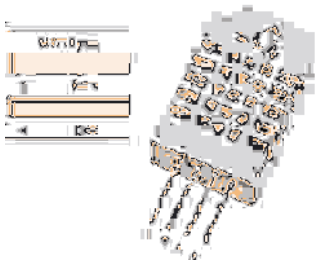
https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf,

The sensor should be powered between 3.3V and 5V (some datasheets may say 6V max) and will work from -40°C to +80°C (some datasheets may say +125°C) with an accuracy of +/- 0.5°C for temperature and +/- 2% for relative Humidity. Important to have in mind that its sensing period is in average 2 seconds (minimum time between readings).

For more details, please visit: https://learn.adafruit.com/dht

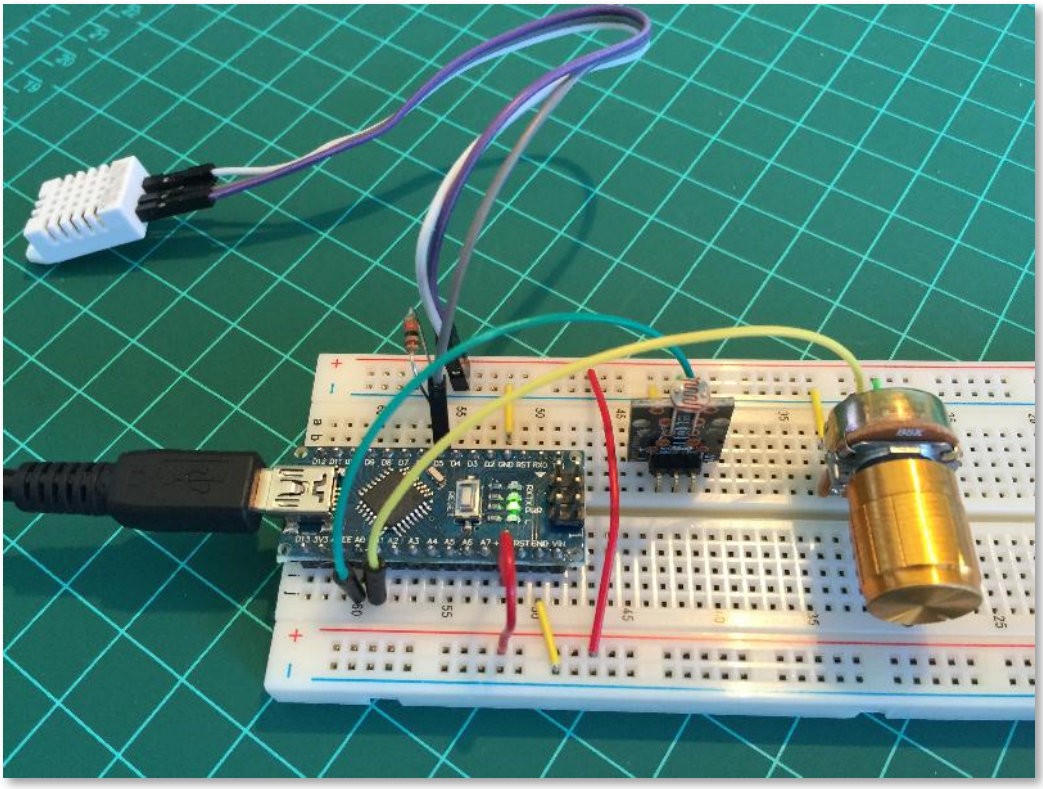
www.EngineeringBooksPdf.com

The DHT22 has 4 pins (facing the sensor, pin 1 is the most left) :



1. VCC (3 to 5V)
2. DATA (Data Output)
3. NC (No Connected)
4. GND (Ground)

Considering you will mostly use the sensor on distances less than 20m, a 10K ohms resistor should be connected between Data and VCC pins. Output pin should be connected to Arduino Pin5 as shown at previous diagram:



Once the sensor is installed at Arduino, download the DHT library from Adafruit GitHub repository:

https://github.com/adafruit/DHT-sensor-library

and install it in your Arduino's Library file.

After you reload your Arduino IDE, the "DHT sensor library" should be installed. Run the DHT Sensor code that follows, to verify that everything is up and running:

www.EngineeringBooksPdf.com

```
/*****************************************************************
 * DHT Sensor - Setup and Test
 * Based on the original code written by ladyada, public domain
 * MJRoBot 21Aug16
 *****************************************************************/
// Include DHT Library
#include <DHT.h>

// Sensor defiitions
#define DHTPIN 5        // DHT data pin connected to Arduino pin 5
#define DHTTYPE DHT22   // DHT 22 (if your sensor is the DHT 11,
only change this line by: #define DHTTYPE DHT11)

// Variables to be used by Sensor
float tempDHT; // on the final program we will use int insteady of
float. No need for high precision measurements
float humDHT;
float hic; // only used here for testing purposes

// Initialize the DHT sensor
DHT dht(DHTPIN, DHTTYPE);

void setup()
{
  Serial.begin(9600);
  Serial.println("DHT 22 Setup & Test");
  dht.begin();
}

void loop()
{
  // Wait a few seconds between measurements.
  delay(2000);

  //Read temperature and humidity values from DHT sensor:
  tempDHT = dht.readTemperature();
  humDHT = dht.readHumidity();

  // Check if any reads failed and exit early (to try again).
  if (isnan(humDHT) || isnan(tempDHT))
  {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Compute heat index in Celsius (isFahreheit = false)
  float hic = dht.computeHeatIndex(tempDHT, humDHT, false);

  // Show measurements at Serial monitor:
  Serial.print("   Temp DHT ==> ");
  Serial.print(tempDHT);
  Serial.print("oC  Hum DHT ==> ");
  Serial.print(humDHT);
  Serial.print("%  Heat index: ");
  Serial.print(hic);
  Serial.println(" oC ");
}
```


www.EngineeringBooksPdf.com

B. Luminosity Sensor

Hence the DHT is installed and tested, it's time for the luminosity sensor. For that, a simple LDR (Light Dependent Resistor) can be used. Basically, what we should to do is to have a Voltage Divider where one of the resistors is the LDR and the middle point of the divider should be used as an analog input for Arduino. This way as varying the light, the LDR resistance varies as well, the middle point voltage of the divider will also change proportionally.

For testing we will use a cheap LDR module (KY18) that has the voltage divider integrated. The module has 3 pins ("S" for data; "+" for VCC and "-" for GND). The pin "S" will be connected to Arduino Pin Analog 0 (A0). The "+" and "-" pins should be connected respectively to 5V and GND. When Power consumption is a concern, the "+" could be connected to one of the Arduino's digital output instead, that should be "HIGH" a few milliseconds before you read the voltage at pin A0, returning to "LOW" afterwards.

The function getLumen(LDR_PIN) reads a few times the sensor output (could be 3, 10 or more, depending on your case) calculating at the end the average of those readings. Also, once the output of Arduino's Analog Digital converter (ADC) is a number from 0 to 1023, we should "Map" those values to get the following results:

- "Dark Full" ➔ ADC output: 1023 ➔ 0%
- "Full Light" ➔ ADC output: 0 ➔ 100%

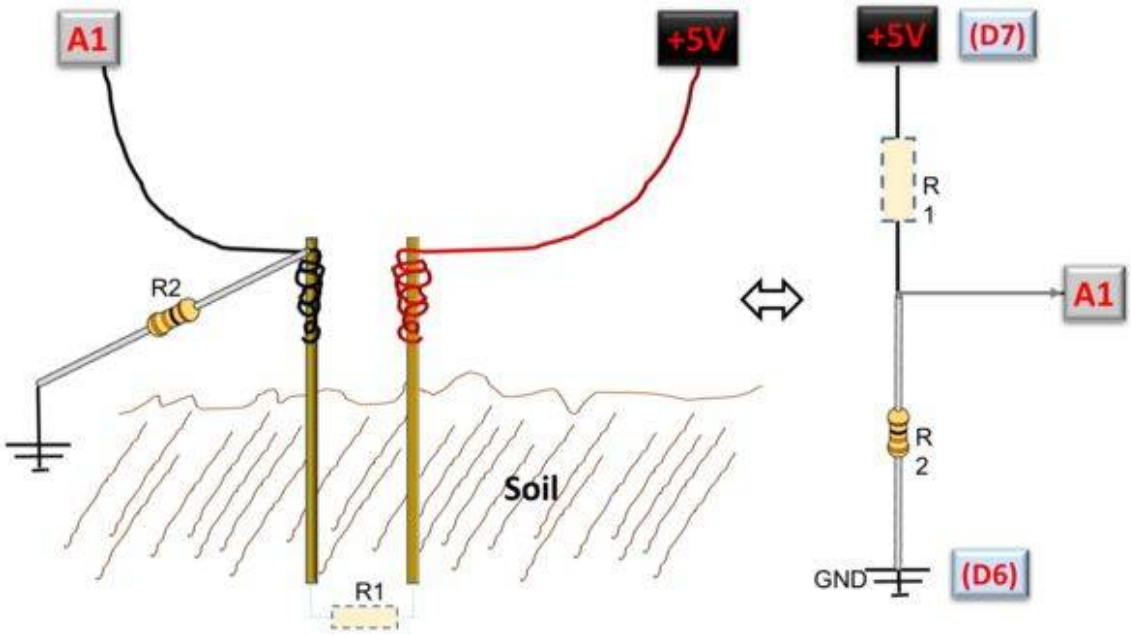
Below the coded function:

```
int getLumen(int anaPin)
{
  int anaValue = 0;
  for(int i = 0; i < 10; i++) // read sensor 10X ang get the
average
  {
    anaValue += analogRead(anaPin);
    delay(50);
  }

  anaValue = anaValue/10; //Light under 300; Dark over 800
  anaValue = map(anaValue, 1023, 0, 0, 100); //LDRDark:0  ==> light
100%

  return anaValue;
}
```


www.EngineeringBooksPdf.com

C. Soil Humidity Sensor

A sensor for testing soil moisture is very simple. It has the same principle as described previously for the Luminosity Sensor. A voltage divider to be used as input of one of Arduino's Analog Pin, but instead of a "Light Depending Resistor" we will have a "Soil Humidity Depending Resistor". The basic circuit is simple and can be seen below:



Unfortunately, reality is a little bit more complicated than this (but not that much). A simple sensor as described before would work just fine, but not for long. The problem is that having a constant current flowing through electrodes in one single direction will generate corrosion on them due to electrolysis effect. One way to solve it is to connect the electrodes not to VCC and Ground, but to an Arduino Digital ports. Doing that, first the sensor would be "energized" only when the reading should really happen and the current direction over the probes could be done on both directions, eliminating the electrolysis's effect.

For preliminary tests on developing the SW, a 10K Ohms potentiometer was used between +5V and GND to provide an output that simulates the Soil Moisture sensor output. For now, it is enough once we will discuss this sensor deeper on chapter 1.5.

Next, a simple test code based on the post "How to: Soil Moisture Measurement?":

http://forum.arduino.cc/index.php?topic=37975.0

www.EngineeringBooksPdf.com

```
/*****************************************************
Soil Moisture Sensor Test
*****************************************************/
#define SOIL_MOIST_PIN 1 // used for Soil Moisture Sensor Input
#define SMS_VCC 7
#define SMS_GND 6

int soilMoist; // analogical value obtained from sensor

void setup ()
{
  Serial.begin(9600);
  pinMode(SMS_VCC,OUTPUT);
  pinMode(SMS_GND,OUTPUT);

}

void loop (void)
{
  soilMoist = getSoilMoisture();
  Serial.print("Soil Moisture: ")
  Serial.print(soilMoist)
  Serial.println(" %")
}

/*****************************************************
* Capture soil Moisture data
*****************************************************/
int getSoilMoisture()
{
  int anaValue;

  digitalWrite(SMS_VCC,LOW);   // drive a current through the
divider in one direction
  digitalWrite(SMS_GND,HIGH);
  delay(1000);   // wait a moment for capacitance effects to settle
  anaValue=analogRead(SOIL_MOIST_PIN);

  digitalWrite(SMS_VCC,HIGH);   // reverse the current
  digitalWrite(SMS_GND,LOW);
  delay(1000);   // give as much time in 'reverse' as in 'forward'
  digitalWrite(SMS_VCC,LOW);   // stop the current

  anaValue = map(anaValue, 1023, 0, 0, 100);
  return anaValue;
}
```


www.EngineeringBooksPdf.com

D. Completing Sensor tests

Now that all sensors routines are ready and tested individually, let's create a specific function to read all sensors at once.

```
void readSensors(void)
{
  tempDHT = dht.readTemperature();   //Read temperature and
humidity values from DHT sensor:
  humDHT = dht.readHumidity();
  lumen = getLumen(LDR_PIN);
  soilMoist = getSoilMoist();
}
```

Perform some sensors tests while the program is running, like covering the LDR and see if the data goes from a high value to near "0" (see the Serial Monitor Print screen below). Do the same for Temperature and Air Humidity. Use potentiometer for Soil Moisture Sensor tests.

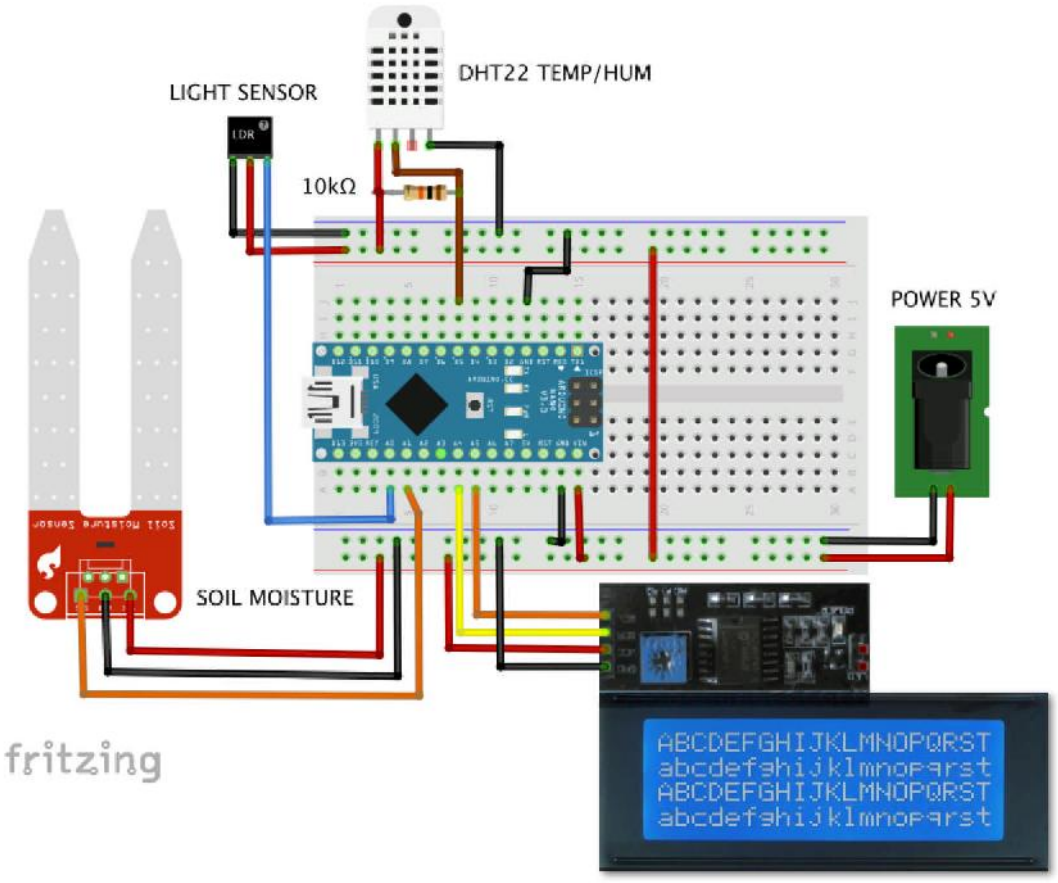
```
/dev/cu.wchusbserial1410                                        Send
ArduFarmBot SENSORS Setup & Test
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 78%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 23%  Soil Moisture ==> 45%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 25%  Soil Moisture ==> 41%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 9%  Soil Moisture ==> 30%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 7%  Soil Moisture ==> 27%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 8%  Soil Moisture ==> 26%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 68%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 77%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
    Temp DHT ==> 17oC  Hum DHT ==> 53%  Luminosity ==> 87%  Soil Moisture ==> 79%
Autoscroll                                   Both NL & CR      9600 baud
```

Complete code: *Sensors_Setup_and_Test.ino* can be downloaded from the ArduFarmBot file depository:

https://github.com/Mjrovai/ArduFarmBot/tree/master/Sensors_Setup_and_Test

www.EngineeringBooksPdf.com

1.3 - Adding a LCD for local monitoring



Not always we have a serial monitor available to analyze outputs of our sensors. Therefore, a LCD will be added to the project for local monitoring. The choice was for a high-quality 4 line 20-character LCD module that not only permits to set up the contrast through a potentiometer installed at its back, but also has a backlight and I2C communication interface.

For I2C Serial communication with the Arduino, the LCD interface provides 4 pins:

- GND, VCC, SDA and SCL

The SDA pin will be connected in our case to Arduino pin A4 and the SCL to pin A5, as shown at the above diagram.

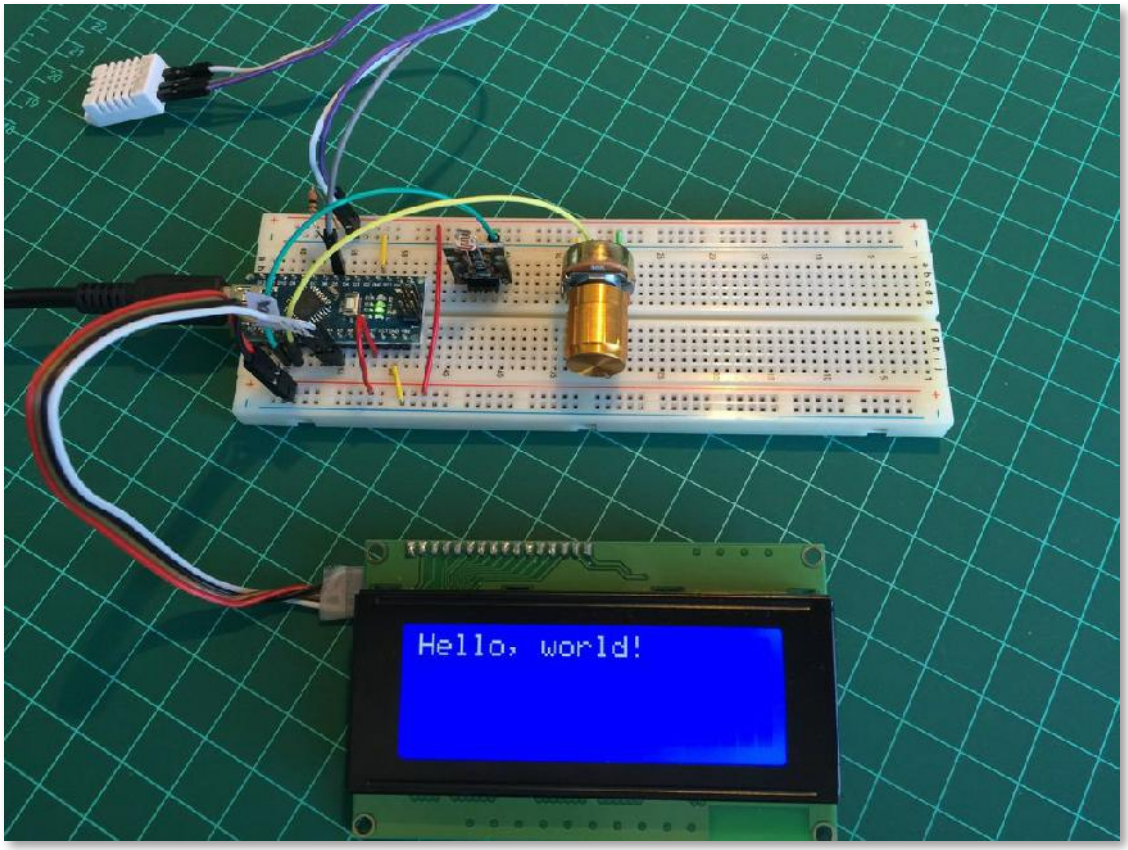
Once the 4 wires are connected, the next thing to do it is to download and install the I2C Library for your LCD Display:

https://github.com/fdebrabander/Arduino-LiquidCrystal-I2C-library

www.EngineeringBooksPdf.com

Open and upload to your Arduino the “Hello World” example that is included within the library, changing the default display set-up of: “16×2” by “20x4”. The address “0x27” worked fine in our case:

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Set the LCD address to 0x27 for a 20 chars and 4 line display
LiquidCrystal_I2C lcd(0x27, 20, 4);

void setup()
{
        // initialize the LCD
        lcd.begin();

        // Turn on the blacklight and print a message.
        lcd.backlight();
        lcd.print("Hello, world!");
}

void loop()
{
        // Do nothing here...
}
```



www.EngineeringBooksPdf.com

If you are not sure about yours LCD's I2C address, a simple I2C scan of your HW will show if there are I2C devices working properly and its address. The code can be found here:
http://playground.arduino.cc/Main/I2cScanner

In our case, we run the program and got at Serial Monitor:

```
Scanning...
I2C device found at address 0x27!
done
```

Let's incorporate the LCD on our last code, so we can see the sensors readings at LCD. The complete code can be download from ArduFarmBot GitHub:

https://github.com/Mjrovai/ArduFarmBot/tree/master/Sensors_Setup_and_Test_LCD/Sensors_Setup_and_Test_LCD.ino

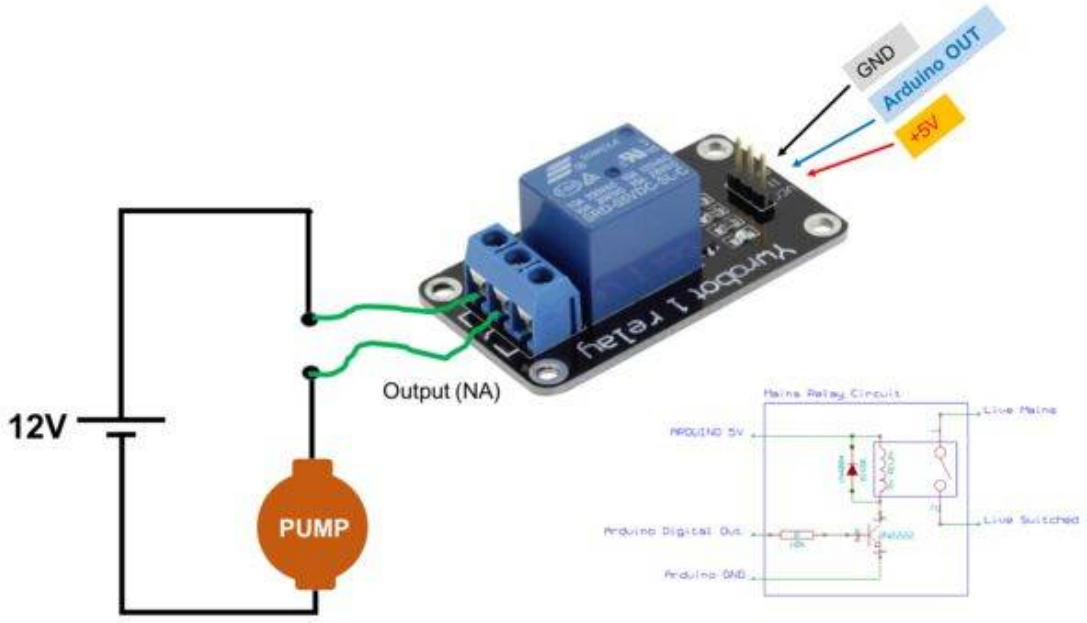
www.EngineeringBooksPdf.com

1.4 - Actuators and buttons for local control

A. Actuators

So far, we can read data from sensors and display it at Serial Monitor and LCD. It is time to do something with such data. Let's think about the actuators!

As discussed in the introduction, our final goal here is to take care of a tomato garden (orchard). Looking at the data provided by sensors, we will know the air temperature and humidity, luminosity and the most important how "dry" is the garden's soil. With that data at hand, our program should calculate if it is necessary to irrigate the garden, turning on the water pump or the electric lamp for warmth. For that, we will use small 5V Relay Module for Pump and Lamp activation. The Relay Module diagram circuit can be seen below.



Some modules have as inputs "G", "V", "S" or "S", "- ", "+" or "In", Vcc", "GND", etc.

Looking at the diagram, depending on your Relay Module, you must connect:

- Arduino 5V ➔ "V" or "+" or "Vcc"
- Arduino GND ➔ "G" or "-" or "GND"
- Arduino OUT ➔ "S" or "In" (in our case should be D10 for Pump and D8 for Lamp)

www.EngineeringBooksPdf.com

Usually you will see as output, 3 Pins: “NO”, “Ref”, NC”, that are: “Normal Open”, “Reference” and “Normal Closed”. We will use the pair: NO and Ref (center). At the above diagram, “NO” is the terminal to connect to “Live Mains” or the live positive of the Power Source (12VDC for Pump and 220VAC for Lamp). The “Ref” will be connected to Pump or Lamp as shown at above diagram.

To know more about relays, visit: “Controlling Power with Arduino “:
https://arduino-info.wikispaces.com/ArduinoPower

Simultaneously with the relays, 2 LEDs can be used to show if the relays are ON or OFF:

- LED Red: Pump
- LED Green: Lamp

For testing, it is great to have the LEDs on your Breadboard, but for a final project, you can take them out to save energy or maybe to use different digitals outputs for LEDs and Relays. They worked together, but will drive a reasonable amount of current from the Arduino (you will realize a drop of brightness at the LCD). Anyway, for the final assembly and testing we will discuss a lot of considerations regarding energy savings.

B. Local Buttons

Based on sensors readings, an operator could also decide manually to control the Pump and/or Lamp. For that, two push-buttons will be incorporate to the project. They will work on a “toggle” mode: If an actuator is “ON”, pressing the button will become “Turn-Off” and vice-versa. The button’s logic will be “normally closed”, which means that Arduino’s input will be constantly “HIGH”. Pressing the button “LOW” will be applied at the specific Arduino’s pin.

As we did with sensors, anytime that we will run the function loop(), a function readLocalCmd() will be executed. This function will read each button, updating the status of actuators variables (pumpSatus and lampStatus). Note that the function type debounce(pin) is called instead a direct digitalRead (pin). This is to prevent false readings from the pushbutton.

If you want to learn more about debouncing, please visit:
https://www.arduino.cc/en/Tutorial/Debounce

---

www.EngineeringBooksPdf.com

```
/*****************************************************************
* Read local commands (Pump and Lamp buttons are normally "HIGH"):
*****************************************************************/
void readLocalCmd()
{
  int digiValue = debounce(PUMP_ON);
  if (!digiValue)
  {
    pumpStatus = !pumpStatus;
    showDataLCD();
    aplyCmd();
  }

  digiValue = debounce(LAMP_ON);
  if (!digiValue)
  {
    lampStatus = !lampStatus;
    showDataLCD();
    aplyCmd();
  }
}
```

In the case where a button is pressed, another function will be called: aplyCmd(). And as per its name, will apply the corresponding command, turning the actuators ON or OFF:

```
/*****************************************************
* Receive Commands and act on actuators
*****************************************************/
void aplyCmd()
{
    if (pumpStatus == 1) digitalWrite(PUMP_PIN, HIGH);
    if (pumpStatus == 0) digitalWrite(PUMP_PIN, LOW);

    if (lampStatus == 1) digitalWrite(LAMP_PIN, HIGH);
    if (lampStatus == 0) digitalWrite(LAMP_PIN, LOW);
}
```


www.EngineeringBooksPdf.com

C. Considerations about "Timing" in the code

When we think about the 4 big "group of tasks" so far:

1. Read sensors
2. Read buttons (local Command)
3. Act on Pump/Lamp
4. Display data

It is easy to realize that the timing when these tasks should be executed are not necessarily the same. For example, to read the Temperature and Humidity data from DHT 22, we will need to wait at least 2 seconds between measurements, but samples in minutes will not make difference for our purposes here. For Soil Moisture sensor, with less measurements we do better (due probe corrosion generate by electrolysis) and finally, daylight will not vary instantly. But when we think about the actuators, as soon we press a button, we would like (and possibly need) a quick reaction.

So, the last instruction before the end of setup() will be the timer initialization using the millis(); function instead of spreading a lot of delays(); around the code:

```
startTiming = millis ();      // starting the "program clock"
```

During the loop(), the first instruction will be to increment the variable startTiming with a real timing counting.

```
elapsedTime = millis () – startTiming;
```

Subsequently, we will read the button status using the function readLocalCmd().

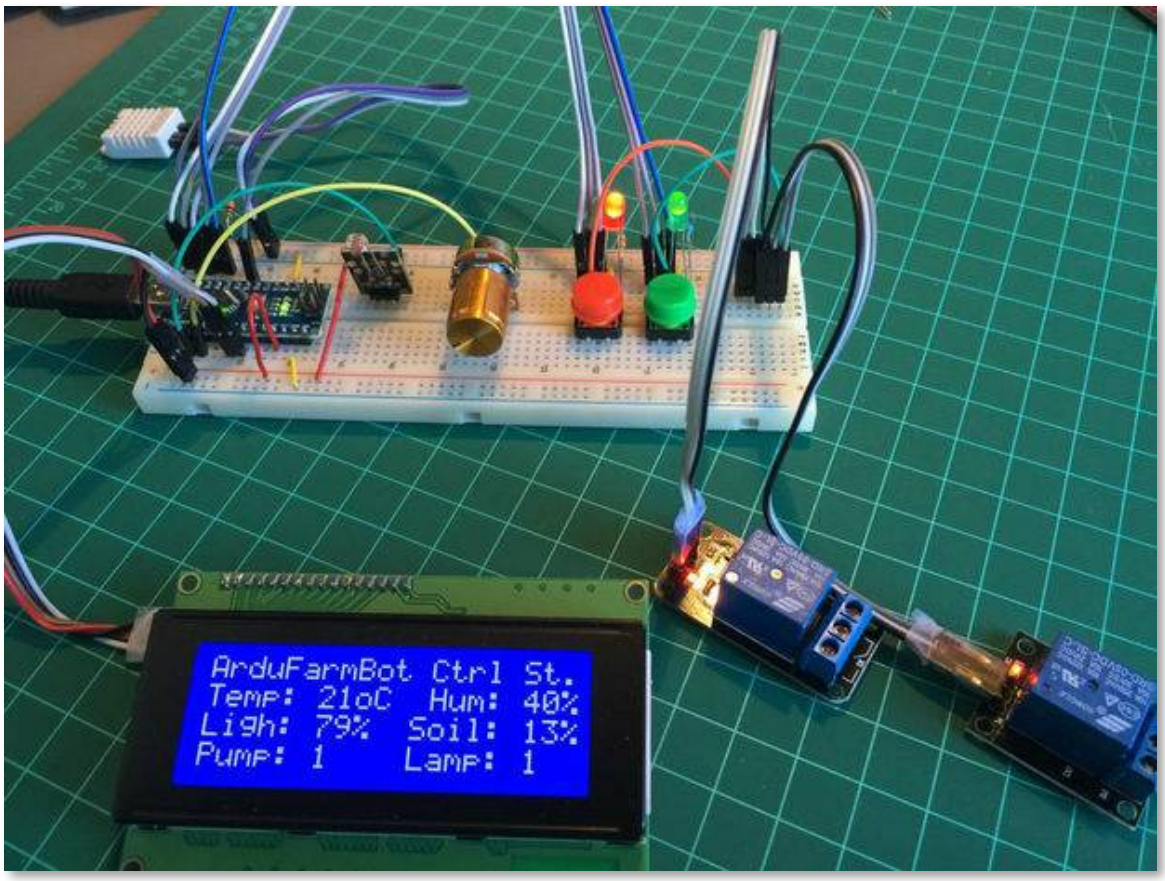
```
readLocalCmd ();      // Read local button status
```

This reading will happen any time that the program executes the loop().

www.EngineeringBooksPdf.com

Regarding the Sensors, we will do the readings, for example any 5 seconds and not at every loop:

```
IF (ELAPSEDTIME > (5000))
{
  readSensors();
  printData();
  startTiming = millis();
}
```



The complete code for Local Station test can be downloaded from ArduFarmBot GitHub:

https://github.com/Mjrovai/ArduFarmBot/tree/master/Ardufarmbot1_Local_Station

www.EngineeringBooksPdf.com

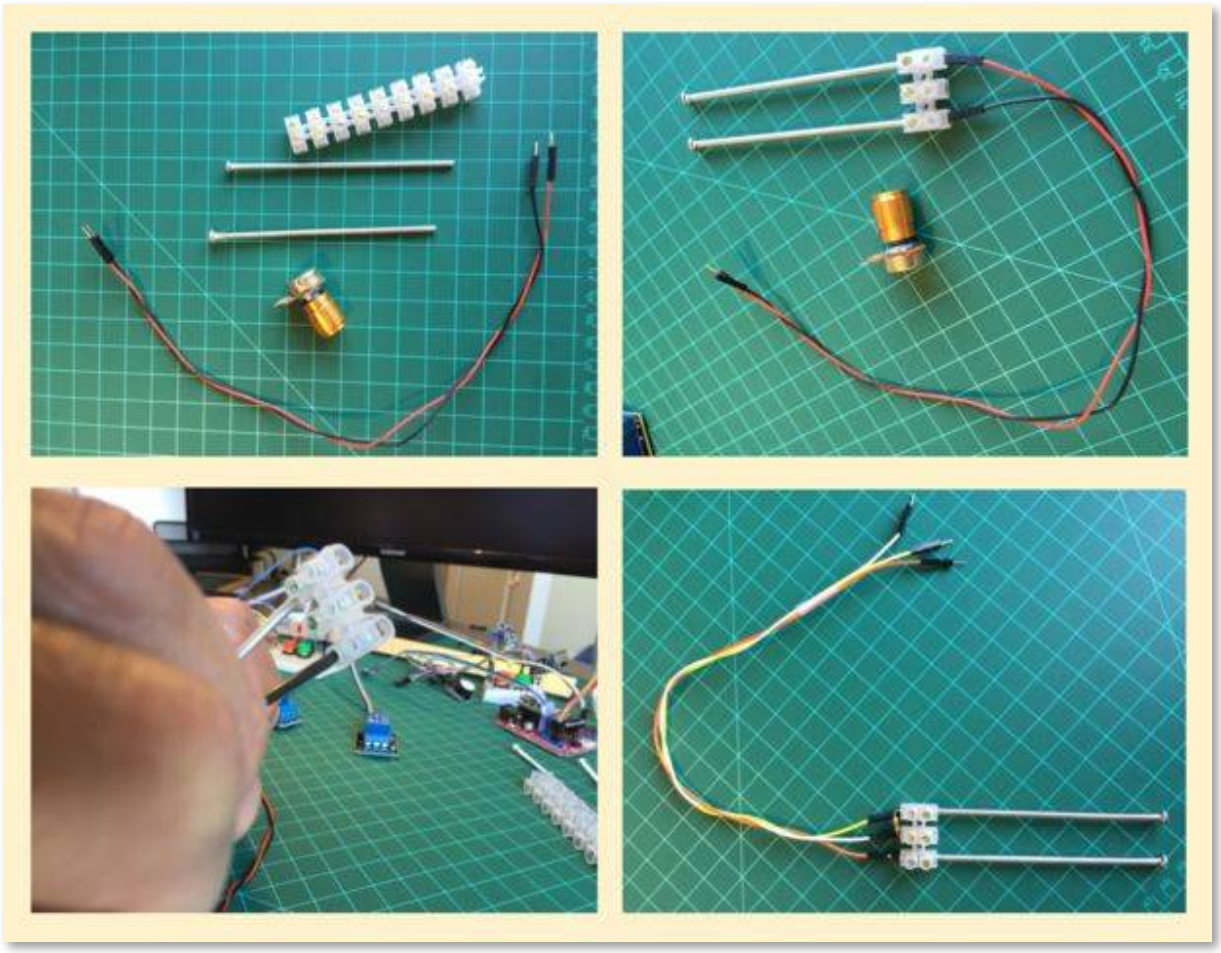
1.5 - Going deeper with a real Soil Moisture Sensor

You can skip this step in your project if you want; however, it will be interesting to go a little deeper with this simple but key sensor.

As briefly explained previously, a Soil Moisture Sensor is a simple “resistive voltage divider”.

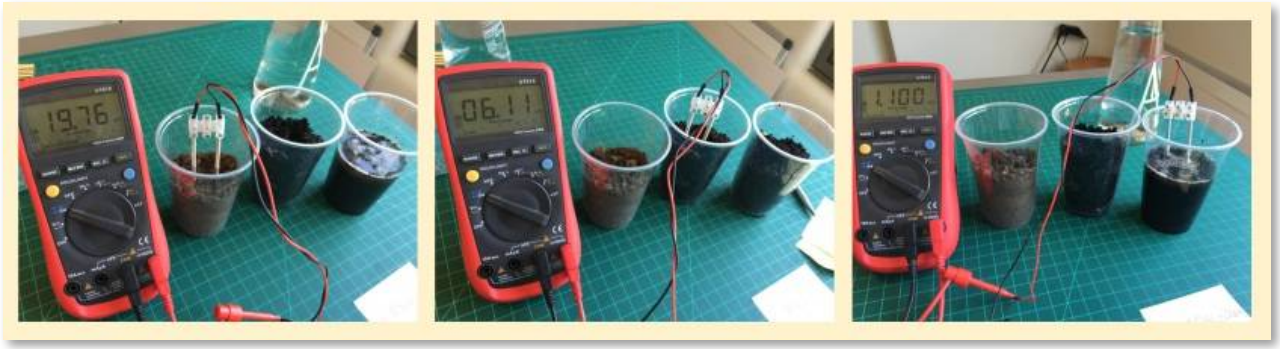
That said, we can construct a very simple sensor using two metal probes like galvanized nails, pins or bolts. Underneath you can see one created using only simple materials.

The first sensor was built only with the two bolts connected with two wires (black/red). On a second prototype, a third wire and a resistor were incorporated.



www.EngineeringBooksPdf.com

As described on 1.2 item C., the "R1" is the "soil resistance" (not the best scientific term, but it's OK). Taking 3 soil moisture samples for analysis ("Dry", "Humid" and "Wet"), we can measure the R1 value using a multimeter as following:



From the measurements, we got R1:

- Dry: R1 ➔ 20K ohm (aprox.)
- Humid: R1 ➔ 4K ohm to 6K ohm (aprox.)
- Wet: R1 ➔ 1K ohm (aprox.)

R2 it is the physical resistor that we will connect to complete the Voltage Divider (We will start with a 10K ohm potentiometer for set-up). Calculating Vin at Arduino A1, proportionally to VCC, we would get the equation:

*Vin = R2/(R1+R2)*VCC or Vin/VCC = 10K/(R1 + 10K)*100 [%]*

Using the real values measured with the Multimeter, we can anticipate that the results should be:

- Dry: 10K/30K*100 ➔ < 30%
- Humid: 10K/15K*100 ➔ ~ 67%
- Wet: 10K/11K*100 ➔ > 90%

www.EngineeringBooksPdf.com

By making the connections at Arduino and running the code developed so far, we get:

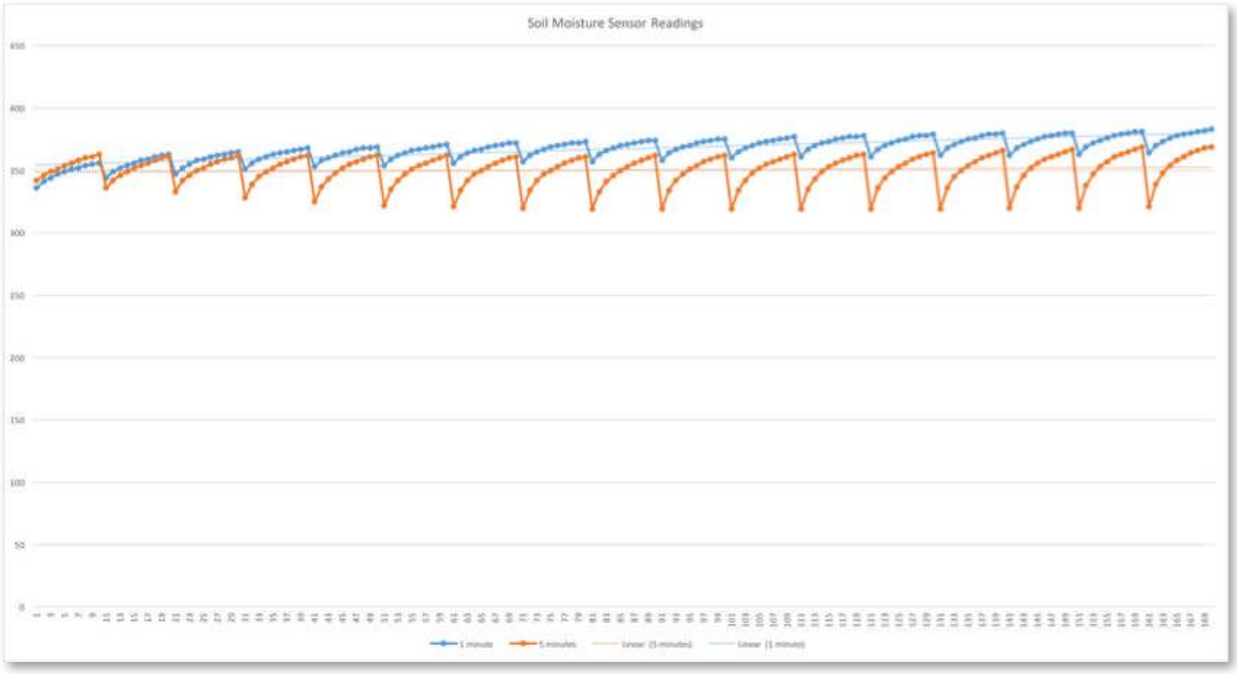
- Dry: 13%
- Humid: 62%
- Wet: 85%

Of course, because we moved the position of sensors, R1 changed BUT what really matters is the range of variation and not the absolute value. The sensor will be used for 3 states:

- Wet: Over 60% (no watering at all)
- Target: Between 40 and 60% (Where we want to work) and
- Dry: Below 30% (need turn on the pump to increase the humidity)

As you can see, using R2 as 10K worked fine, therefore we can take out the potentiometer and add a fixed resistor to our Soil Moisture Sensor (as shown in the 2nd prototype photo).

One thing that we realized by testing the sensors is that doing frequent measurements, will introduce an error on the readings, because the sensor also has a behavior as a "capacitor". Once we "energize" the sensor for data capture, we need also wait a reasonable time even after we cut off the Sensor Power supply, to "discharge the sensor". Reverting the current will help, but it is not enough.



www.EngineeringBooksPdf.com

The previous graphic shows 2 set of measurements:

1. Blue line: A cycle of 10 measurements with 1 seconds between samples and with 1 minute between cycles
2. Orange Line: A cycle of 10 measurements with 1 seconds between samples and with 5 minutes between cycles

With 1 second interval, each new sample will be increasing significantly. Waiting 1 minute after cut off power will decrease the "storage voltage effect", but will not eliminate it and the residual value will be added to next measurement. Increasing the interval of cycles to 5min for example will almost eliminate the error.

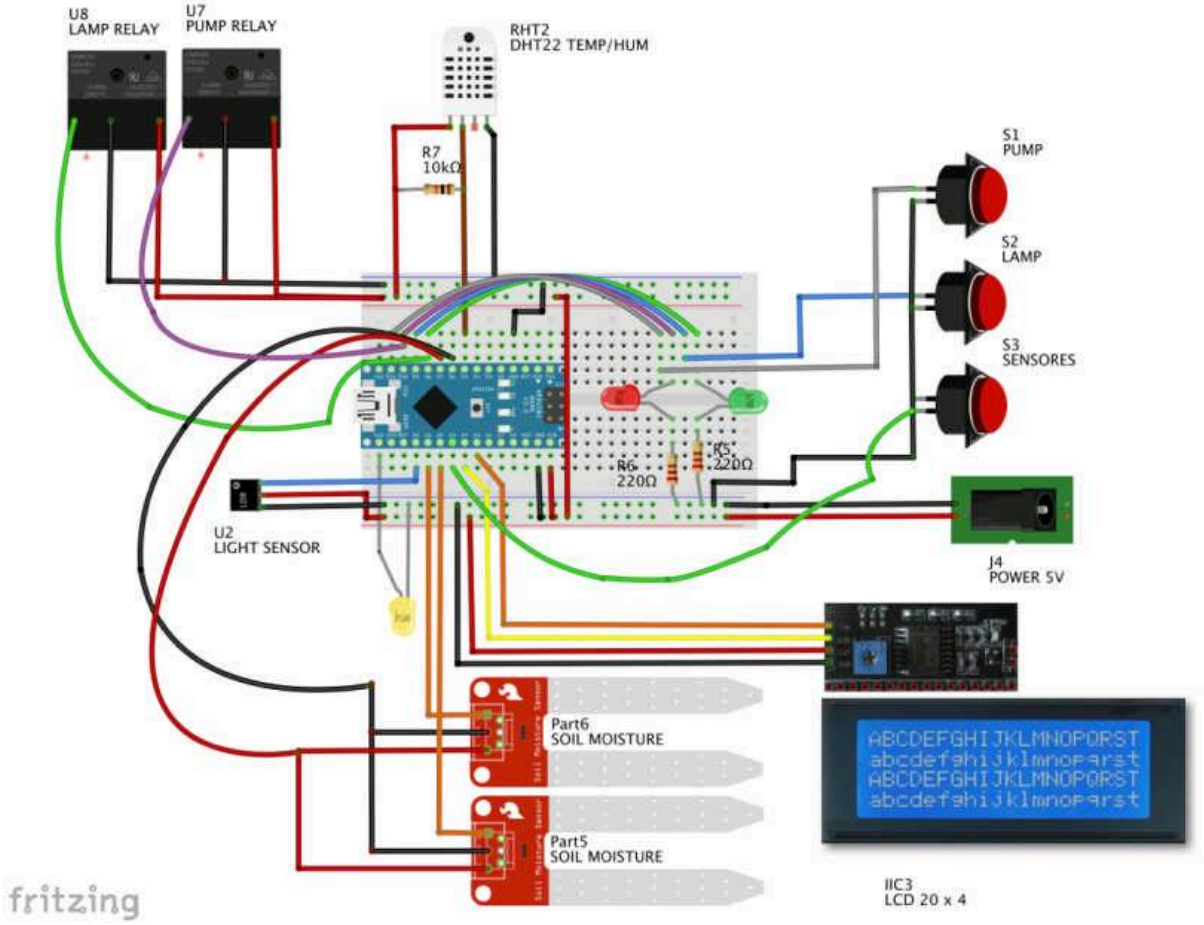
Based on the above results, the final code should not take samples with a frequency less than 10min, for example.

The video below shows the tests with the sensor:

https://youtu.be/bLhwOoBBWZ8

www.EngineeringBooksPdf.com

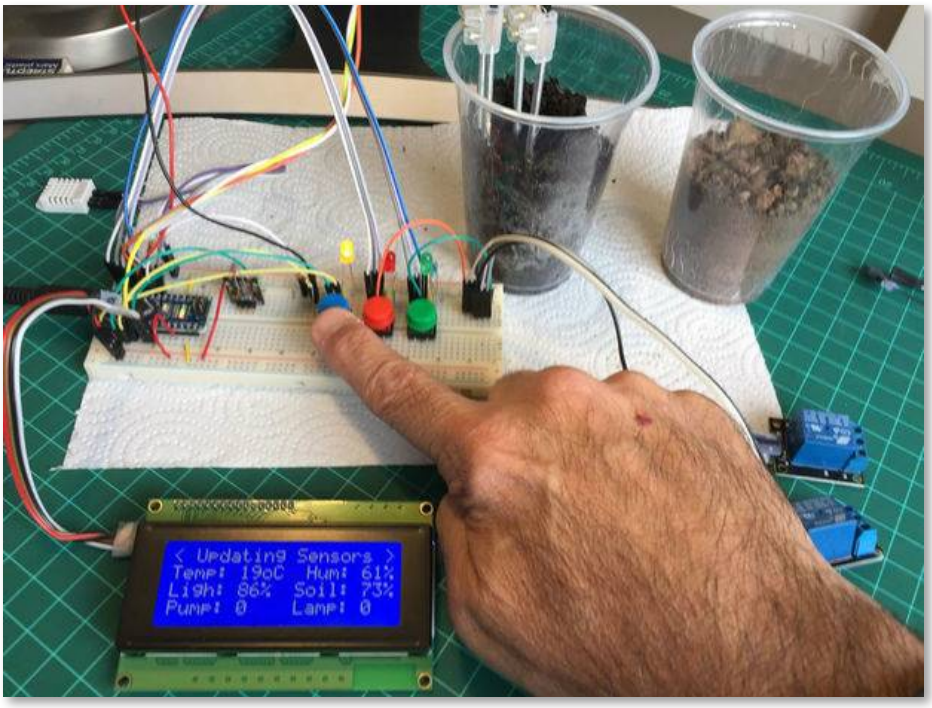
1.6 - Completing the Local Control Station



As we could see at 1.5, we will need to wait long cycles between Soil Moisture sensor measurements. It is Ok for our automatic needs, but for manual operation we will not want to "wait" 10, 15 or more seconds (or even minutes) for a sensor measurement. So, we will introduce a 3rd push-button to our project which will display the actual sensor data any time we want, independent of timing defined for automatic readings.

We will use the digital pin D17 (the same as A3) for the Push-Button and also introducing a "warning LED" (the yellow one at photo) connected to Pin 13. It will be "Light ON" when the sensors are been updated.

www.EngineeringBooksPdf.com



Folloing changed readLocalCmd() function:

```
/*****************************************************************
* Read local commands (Pump and Lamp buttons are normally "HIGH"):
*****************************************************************/
void readLocalCmd()
{
  int digiValue = debounce(PUMP_ON);
  if (!digiValue)
  {
    pumpStatus = !pumpStatus;
    showDataLCD();
    aplyCmd();
  }

  digiValue = debounce(LAMP_ON);
  if (!digiValue)
  {
    lampStatus = !lampStatus;
    showDataLCD();
    aplyCmd();
  }

  digiValue = debounce(SENSORS_READ);
  if (!digiValue)
  {
    digitalWrite(YELLOW_LED, HIGH);
    lcd.setCursor (0,0);
    lcd.print("< Updating Sensors >");
    readSensors();
    digitalWrite(YELLOW_LED, LOW);
  }
}
```


www.EngineeringBooksPdf.com

Another consideration is the introduction of a second Soil Moisture Sensor. On our final Project we will use up to 2 sensors in the garden area so we can get 2 soils moisture readings at different locations. We will use the average of those reading in the final code to decide when to Turn ON the pump, for example.

The Sensor "VCC and GND" will be the same (D7 and D6 respectively) and we will use the A2 for the second sensor. For simplicity, if only one sensor is used, the default is A1 and the code should ignore the A2 reading (a variable must be settle-up during Set-up).

The number of samples of each cycle will be defined by variable numSamplesSMS. In principle only one is enough here, taking into consideration that as much reading we do on a short time will introduce errors due to capacitance effect. If you start to see errors on the reading, maybe extra samples should be taking.

```
int soilMoist;
int soilMoistAlert = 0;
int DRY_SOIL = 30;
int WET_SOIL = 60;
int numSM = 1;  ➔  defines number of moisture sensors that are connected
int numSamplesSMS = 1; ➔ defines number of samples of each reading cycle
```

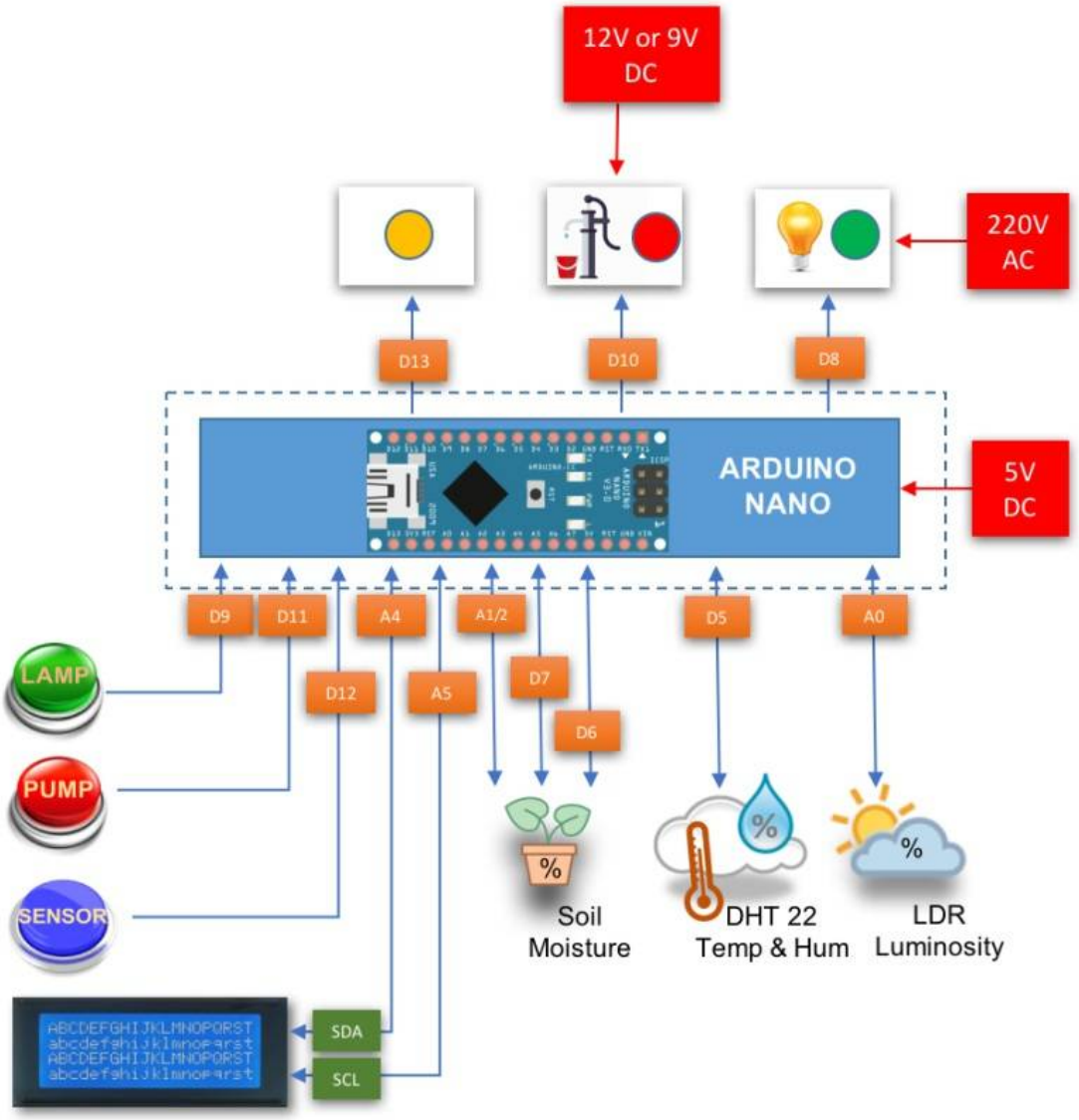

www.EngineeringBooksPdf.com

Below the new function for Soil Moisture Sensor readings:

```
/*************************************************
* Capture soil Moisture data
*************************************************/
int getSoilMoist()
{
  int i = 0;
  int anaValue1 = 0;
  int anaValue2 = 0;
  for(i = 0; i < numSamplesSMS; i++) // // "numSamplesSMS" defines
number of samples of each reading cycle
  {
    digitalWrite(SMS_VCC,LOW);   // drive a current through the
divider in one direction
    digitalWrite(SMS_GND,HIGH);
    delay(500);   // wait a moment for capacitance effects to
settle
    anaValue1 += analogRead(SOIL_MOIST_1_PIN);
    delay(500);   // wait a moment for ADC settle-up
    anaValue2 += analogRead(SOIL_MOIST_2_PIN);

    digitalWrite(SMS_VCC,HIGH);   // reverse the current
    digitalWrite(SMS_GND,LOW);
    delay(1000);   // give as much time in 'reverse' as in
'forward'
    digitalWrite(SMS_VCC,LOW);   // stop the current
    //delay (3000);
  }

  anaValue1 = anaValue1/(i);
  anaValue2 = anaValue2/(i);
  if (numSM == 2) anaValue1 = (anaValue1+anaValue2)/2; // "numSM"
variable, defines number of moisture sensors that are connected

  anaValue1 = map(anaValue1, 1015, 3, 0, 100); //1015:0 (en el air)
==> 003:100% (poniendo un "short circuit)
  Serial.println(anaValue1);
  return anaValue1;
}
```

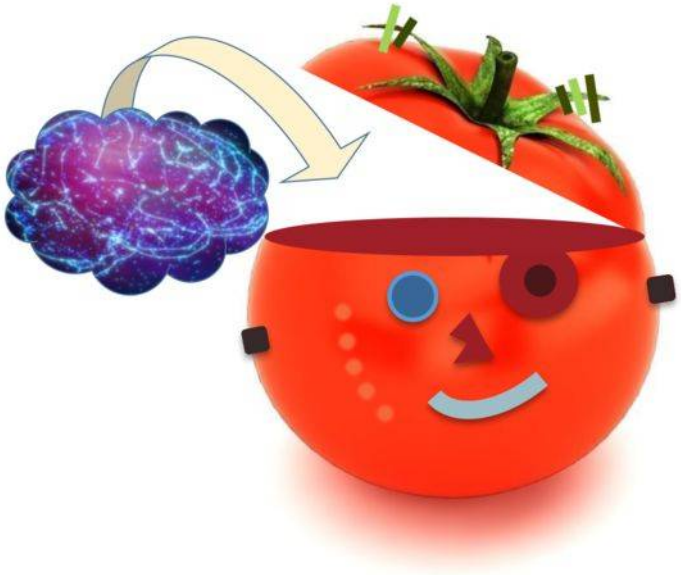

www.EngineeringBooksPdf.com

Underneath diagram shows the complete connections for the Local Control Station HW



www.EngineeringBooksPdf.com

1.7 - It's show time!



At this point, all HW is in place and almost all SW done. What is missing is the "logic" allowing our system to really perform the task of irrigating the garden automatically!

We need to include some "neurons" to our brain!

As discussed before, let's define the initial range where the Sensors will work. Those values should be changed using better practical values to be founded later on the real garden:

Soil Moisture:

- "WET": Over 60% (no watering at all)
- "Target Humid": Between 30% and 60% (Where we want to work) and
- "DRY": Below 30% (need to turn on the pump for increasing humidity)

Temperature:

- COLD: Below 15°C (Turn-On the Light/Heat)
- Optimum: between 20°C and 25°C
- HOT: Over 25°C (Do not Turn-On the Light/Heat)

Light:

- DARK (night): Below 40% (do not turn-on the Pump)
- LIGHT (day): Over 40%

You can optionally test "Hydroponic Plant Grow LED" Light systems. Those LED lamps can be used for both, to help faster growth due its special light frequency and provide heat in case of low temperature.

---

www.EngineeringBooksPdf.com

You must keep in mind that each type of seed has an optimum range of temperature where it will grow faster. For example, for tomatoes the minimum time for seeds to germinate will be 6 days on temperatures between 20ºC and 25ºC, going up for temperatures lower or higher than that:

Temperature: [ºC]	10	15	20	25	30	35
Time for Germination [days]	43	14	8	6	6	9

You can check more information about the relationship Temp/Germination days) on: http://tomclothier.hort.net/page11.html

Having those 4 reading (Temperature, Humidity, Soil Moisture and Light), we can have a matrix defining where we want that our tomatoes grow:

So, let's remember our sensor variables and define some new definitions:

To be used by DHT Sensor

- int tempDHT;
- int HOT_TEMP = 25;
- int COLD_TEMP = 15;

To be used by LDR Sensor

- int lumen;
- int DARK_LIGHT = 40;

To be used by SM Sensor

- int soilMoist;
- int DRY_SOIL = 40;
- int WET_SOIL = 60;

Based on the above definitions, let's think about some key assumptions:

1. If it's DRY and NOT DARK (day) ➔ PUMP = ON
2. If it's and DARK (night) ➔ PUMP = OFF
3. If it's COLD and not WET ➔ LAMP = ON
4. If it's COLD and WET ➔ LAMP = OFF (to protect the seed)

www.EngineeringBooksPdf.com

In this first part of the project we will keep it simple and will not explore all possible combinations and the role of air humidity on the equation. We will explore a more complex combination of sensors results on the 2nd part of this project, when we will apply the ArduFarmBot on a real garden.

The Code:

Let's create a new function that based on sensors reading, will deal automatically with actuators, turning on/off the Pump and Lamp: autoControlGarden(). This function as shown below, will be called on every cycle of sensors readings:

```
void loop()
{
  // Start timer for measurements
  elapsedTime = millis()-startTiming;

  readLocalCmd(); //Read local button status
  showDataLCD();

  if (elapsedTime > (sampleTimingSeconds*1000))
  {
    readSensors();
    autoControlPlantation();
    startTiming = millis();
  }
}
```

The function will have 2 main tasks:

- Pump Control
- Lamp Control

The Pump control segment will use a new variable: *soilMoistAlert*:

```
if (soilMoist < DRY_SOIL && lumen > DARK_LIGHT)
{
  if (soilMoistAlert == HIGH)
  {
    soilMoistAlert = LOW;
    turnPumpOn();
  }
  else soilMoistAlert = HIGH;
}
else soilMoistAlert = LOW;
```


www.EngineeringBooksPdf.com

This variable will be used to avoid “false true”. So, if we get a true in the test: soilMoist < DRY_SOIL and that it is not during the night (lumen > DARK_LIGHT), we will not immediately turn on the Pump, but instead we will wait the next cycle to verify if the “soil is really dry”. If the result is a “yes” (get a “true” for answer twice), the function turnPumpOn () will be called:

```
/*************************************************
* TurnPumOn
*************************************************/
void turnPumpOn()
{
  digitalWrite(PUMP_PIN, HIGH);
  pumpStatus = 1;
  showDataLCD();
  delay (timePumpOn*1000);
  digitalWrite(PUMP_PIN, LOW);
  pumpStatus = 0;
  showDataLCD();
}
```

The Pump should be On for a fixed amount of time, defined by the variable: timePumpOn in seconds.

Note that we also changed the function that display data on LCD, so the status of the Pump now will be:

- “0”: Pump OFF (pumpStatus = 0; and soilMoistAlert = 0;)
- “X”: Pump in alert (pumpStatus = 0: and soilMoistAlert = 1;)
- “1”: Pump ON (pumpStatus = 1; and soilMoistAlert = 0;)

```
lcd.print("Pump: ");
if (soilMoistAlert  == 1) lcd.print ("X");
else lcd.print(pumpStatus);
```

The same principle should be applied to Lamp control, but now a “Low Temperature” variable will be used to trigger the Lamp instead of “Dry soil” and if it is not “too wet”.

www.EngineeringBooksPdf.com

Below the complete function: autoControlGarden():

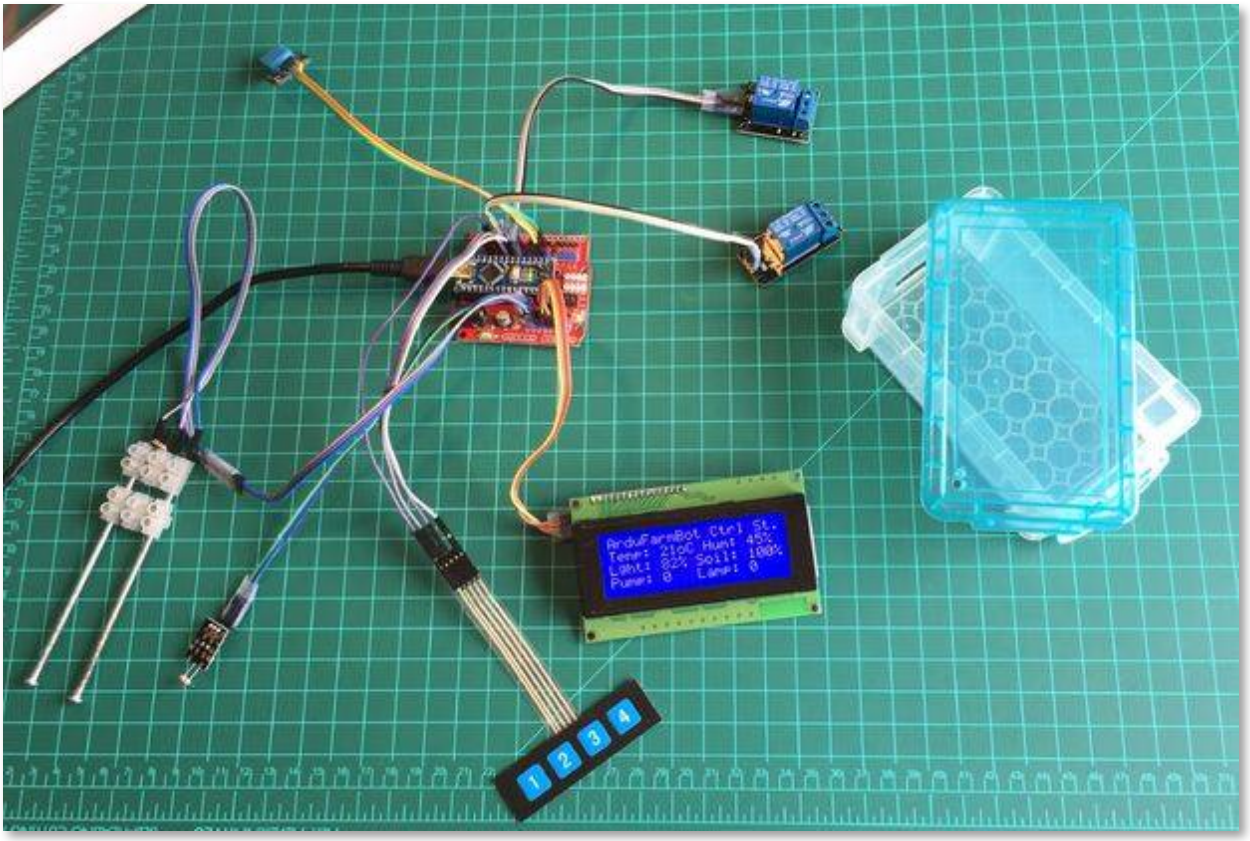
```
/*****************************************************
* Automatically Control the Plantation based on sensors reading
*****************************************************/
void autoControlPlantation()
{
//--------------------------------- PUMP ------//
  if (soilMoist < DRY_SOIL && lumen > DARK_LIGHT)
  {
    if (soilMoistAlert == HIGH)
    {
      soilMoistAlert = LOW;
      turnPumpOn();
    }
    else soilMoistAlert = HIGH;
  }
  else soilMoistAlert = LOW;

//--------------------------------- HEAT ------//
  if (tempDHT < COLD_TEMP && soilMoist < WET_SOIL)
  {
    if (tempLowAlert == HIGH)
    {
      tempLowAlert = LOW;
      digitalWrite(LAMP_PIN, HIGH);
      lampStatus = 1;
    }
    else tempLowAlert = HIGH;
  }
  else
  {
    tempLowAlert = LOW;
    digitalWrite(LAMP_PIN, LOW);
    lampStatus = 0;
  }
}
```

At this point the ArduFarmBot is fully functional in terms of HW and SW and would work on our tomato garden. The complete code for the “Local Station” can be found at ArduFarmBot GitHub:

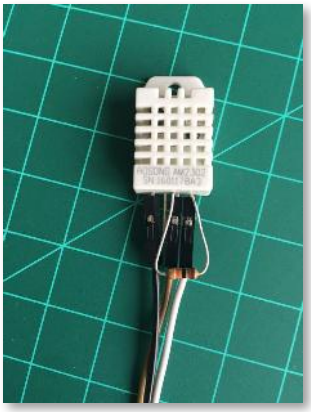
https://github.com/Mjrovai/ArduFarmBot/tree/master/ArduFarmBot1_Local_Control_Station_Final

www.EngineeringBooksPdf.com

1.8 - Changing to a “Small Form Factor”



Once we have our prototype fully functional, let’s reassemble it on a better way using the “Funduino Nano Shield” and a plastic box for helping on external tests. The great advantage of a Nano shield is that every component stays better assembled reducing bad contacts and noise. Also for external testing it is easier to have all main components on a small plastic box.

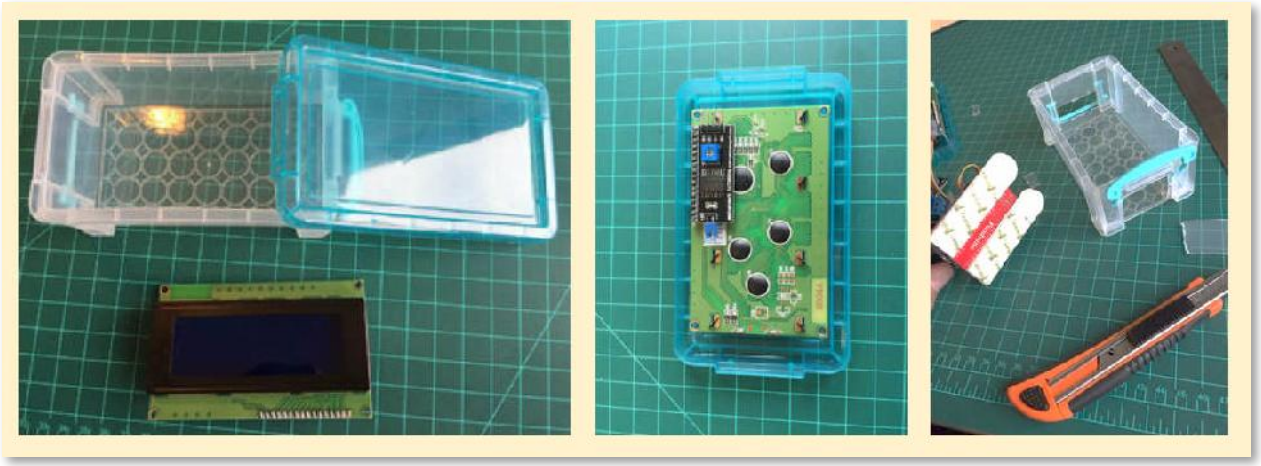


If you are using the DHT stand alone, you must add a 10K resistor between VCC and Signal (as shown in the photo). If you are using a sensor module, the resistor is already included. For this new test, we will use a DHT11 module (“blue color sensor”).

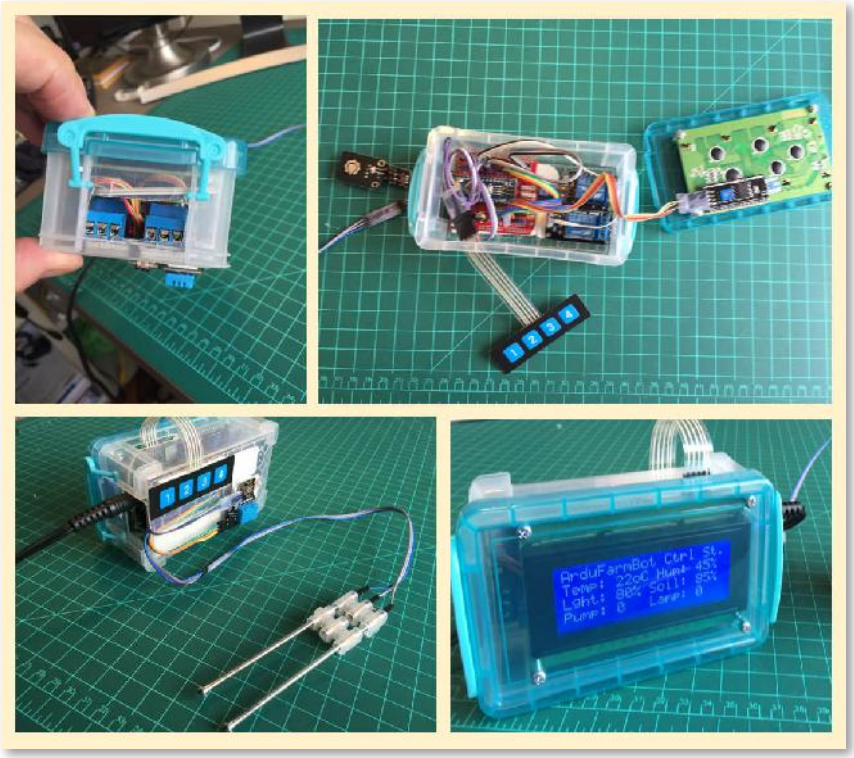
The result for our purpose is the same (only do not forget to change the line at the code to define the appropriate sensor that you will use: #define DHTTYPE DHT11).

www.EngineeringBooksPdf.com

The BOX



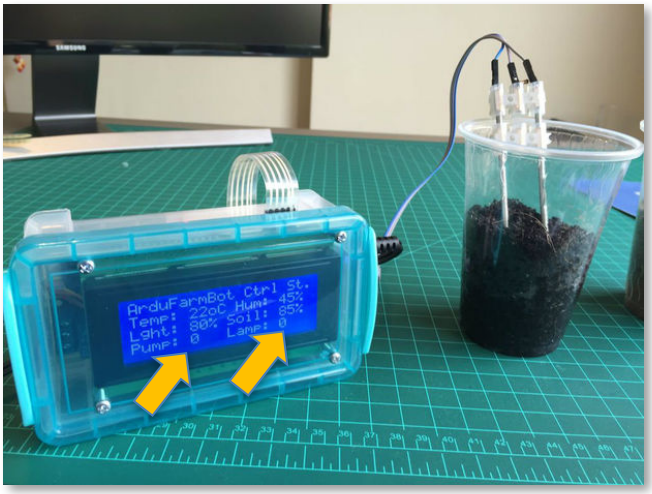
1. Make 4 holes at the plastic box for LCD installation.
2. Make lateral holes at the box so you can have the sensors out and have access inside for Nano (power up via external Power Supply or SW updates) and connection of actuators (Pump/Lamp) with Relays outputs.
3. Note that we used here the “1×4 Key Matrix Membrane Switch” as our control buttons.
4. You can decide the best way to fix the components at the box. We used ordinary 3M stuff for easy fix/remove (see the above photo).



www.EngineeringBooksPdf.com

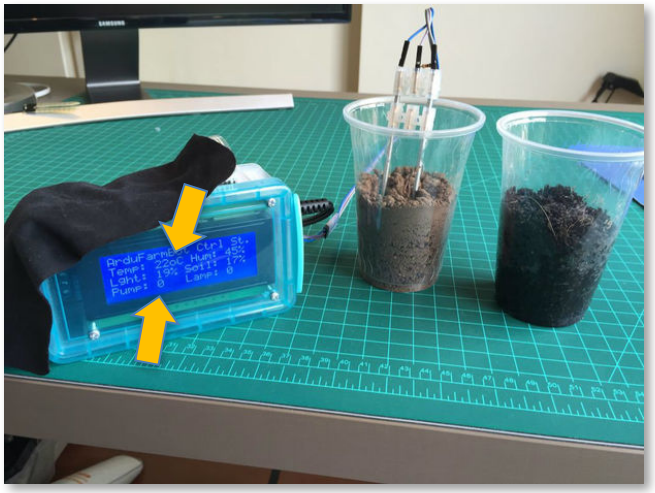
1.9 - Laboratory functional tests

After everything is placed and the SW uploaded, let's do some functional tests simulating various sensor's conditions to verify that all has been correctly assembled:



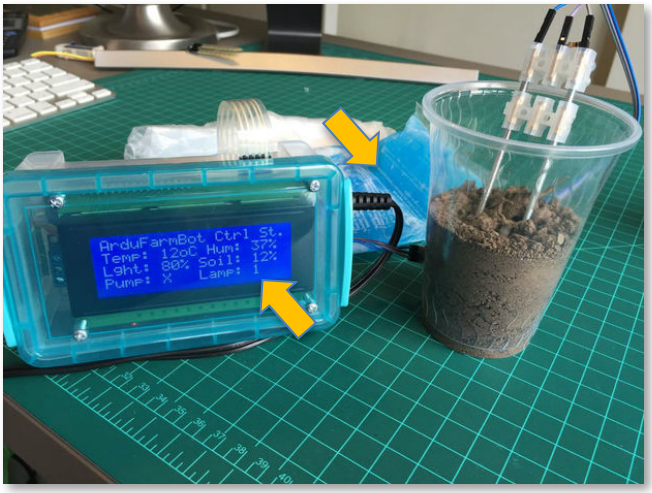
At normal light and temperature, introduce the Soil Moister Sensor Probe on a cup with wet soil sample. Observe the Photo 1 (Temp is 22ºC; Soil Hum. is 85% and Light is 80%). Nothing must happen. PUMP and LAMOP should be OFF ("0").

Keeping same light and temperature, let's move the Soil probe to the cup with dry soil sample. At photo 2, you may observe that the Pump was turned On (First went to "X" and after to "1" for a few seconds as explained on definitions).

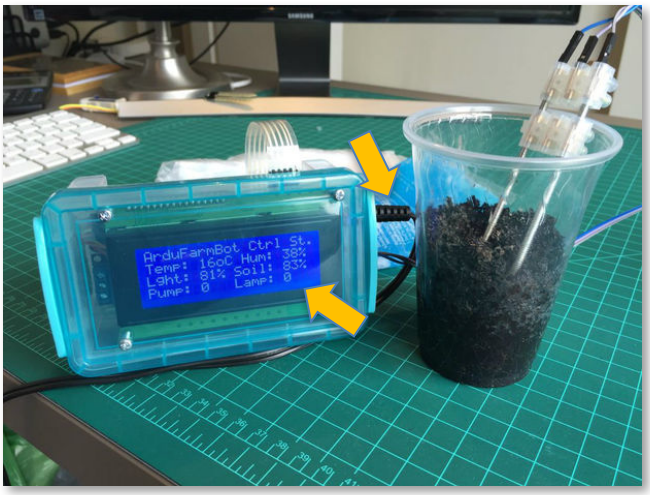


Now as shown at Photo 3, the LDR was covered and the Light % went down to 19%. In this case spite of the fact that the soil is dry, the Pump will not turn-on, because our controller understands that it is at night.

www.EngineeringBooksPdf.com



In the photo 4, we put ice at bottom of our box, close to the DHT Sensor. The temperature went down to 12ºC and the Lamp was Turned-On.



And finally, on photo 5, we keep the ice but change the probe again to the wet soil sample. In this case spite of the fact that it is cold, according with our matrix, the lamp remains off.

www.EngineeringBooksPdf.com

1.10 - “Test Drive”: Watering a Tomato plant with ArduFarmBot

For tests only, we used an electric pump that was available in the lab. This will not be the final one, but can illustrate how the project will work (Therefor, ignore drops that you will see at below video, it is only because water reservatory is to high in relation with water nose)

https://youtu.be/kIQePtzSjPo

www.EngineeringBooksPdf.com

1.11 - ArduFarmBot in action: Mauricio's garden

With everything we have learned so far, next step is to put the ArduFarmBot to control your real tomato garden. Based on this real experience we should calibrate and define better sensors and project parameters.

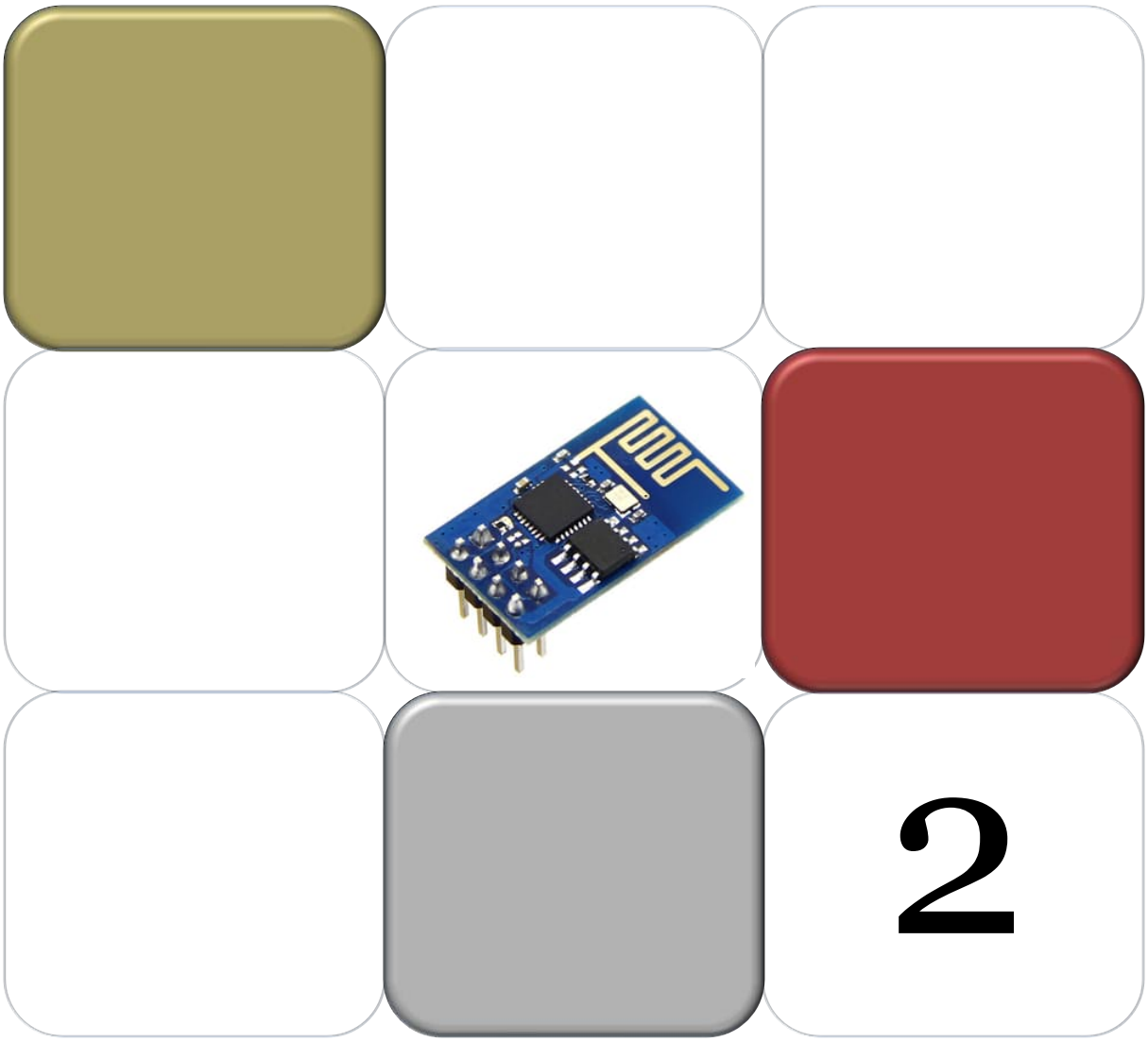
The photos show land preparation sequence and introduction of the seeds. The real results will be analyzed in more detail further in the next part of this project.

Movie below shows the ArduFarmBot in action:

https://youtu.be/HBhg5NtLcs0

Now, let's hope for a great salad!!!! Salut!

www.EngineeringBooksPdf.com



2

www.EngineeringBooksPdf.com

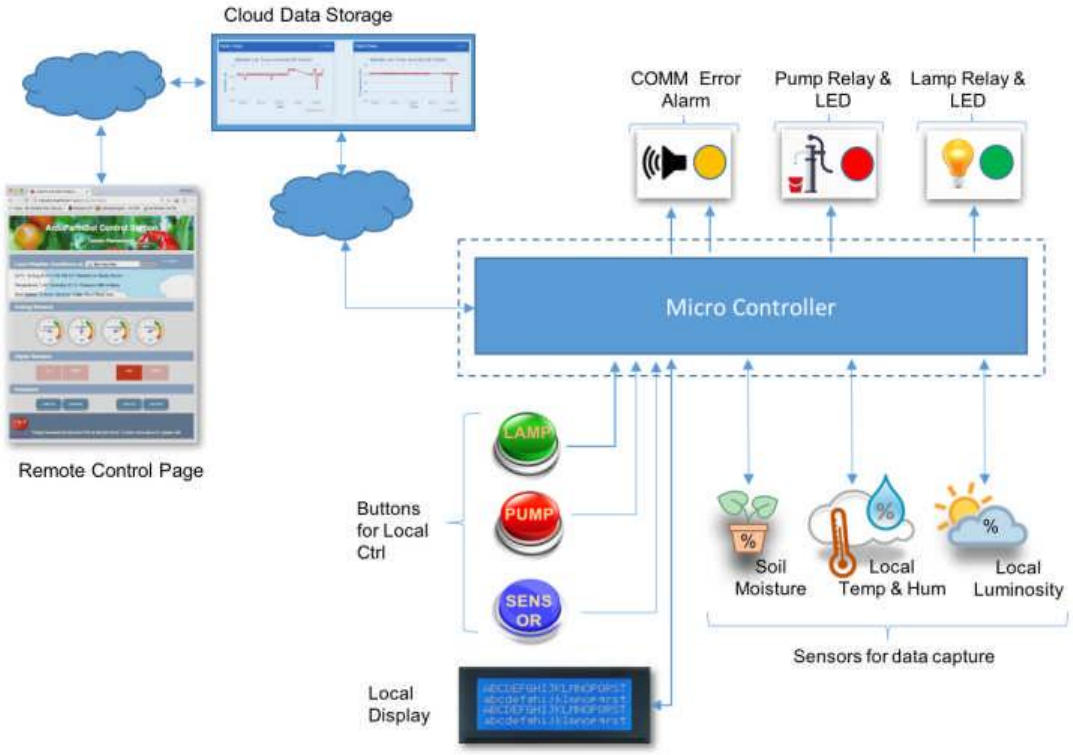
Parte 2 - ArduFarmBot: Remote Station

In the first part, we create a local control station, capturing information from a tomato garden as temperature, relative air humidity, luminosity and soil humidity. With these data, the ArduFarmBot could decide automatically the right amount (and when) the garden should receive heat and/or water. The local station developed on Part 1, also allows manual intervention of an operator to control at any time water pump and the electric lamp. On this Part 2, we will implement an IoT approach were this “manual intervention” can also be done remotely via Internet.

www.EngineeringBooksPdf.com

2.1 - The IoT Approach

Block diagram below shows following steps in this part of the project:



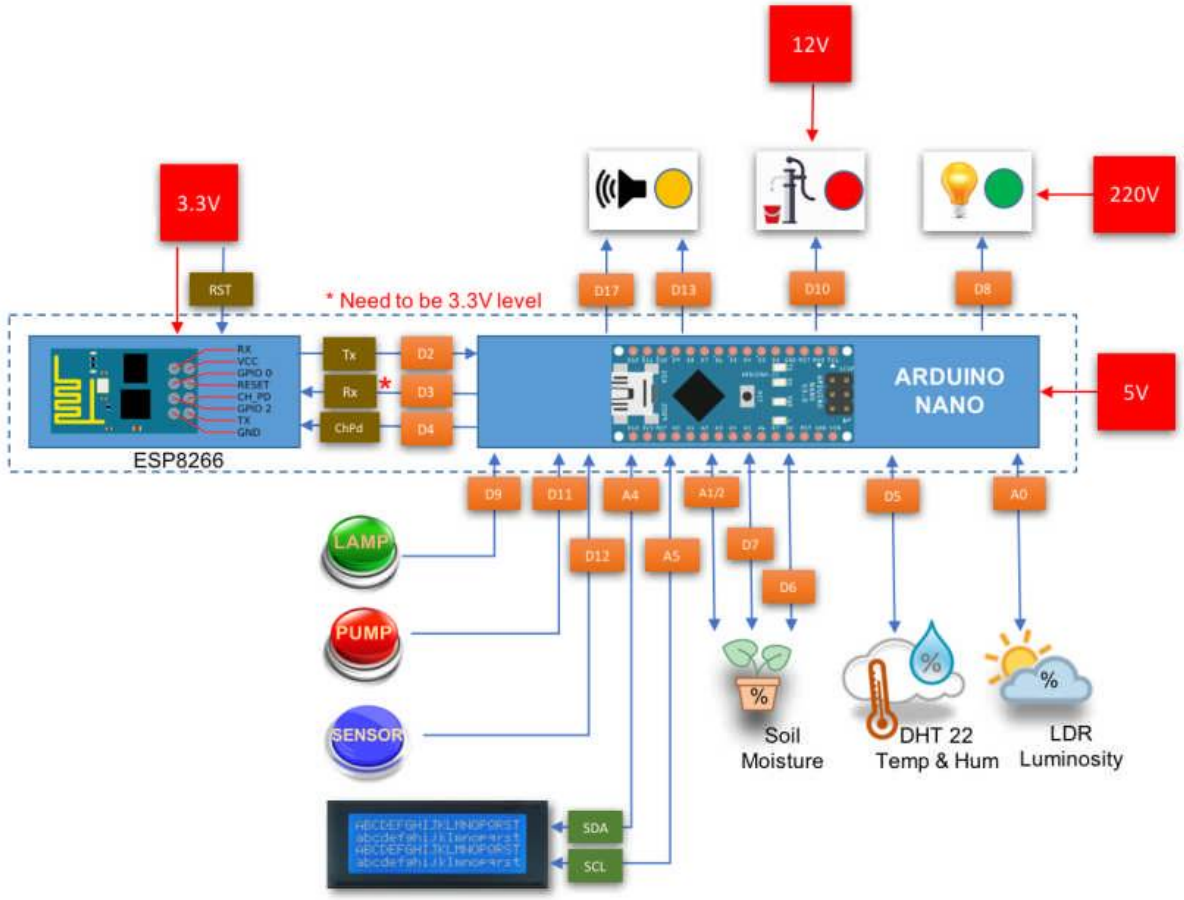
Note that the captured data will be sent to a "Cloud Storage service" (in our case Thinkspeak.com). Also, a dedicated website, the "Remote Control Page" will be monitoring and displaying those data in almost real time. This webpage will also permit the pump and lamp's remote activation.

www.EngineeringBooksPdf.com

2.2 - Completing the Hardware

To connect ArduFarmBot to the Internet, we will use the ESP8266-01, a simple, inexpensive and easy to program module for projects involving the Internet of Things (IoT). From the local station developed in Part 1, the only additional HW needed is the ESP8266 itself.

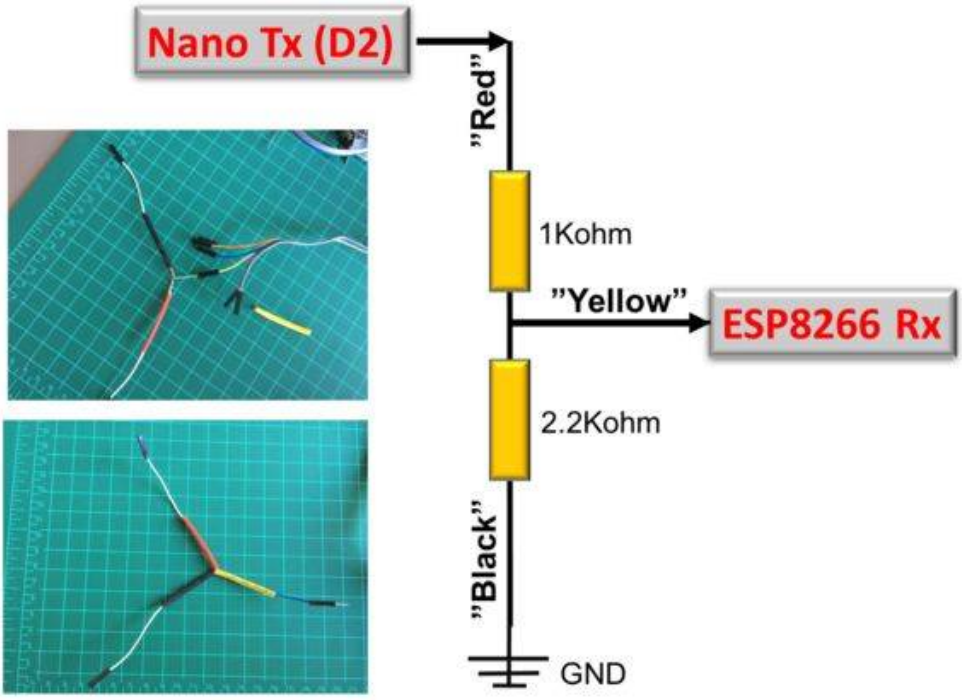
The block diagram below shows all connections to the Arduino pins and main components.



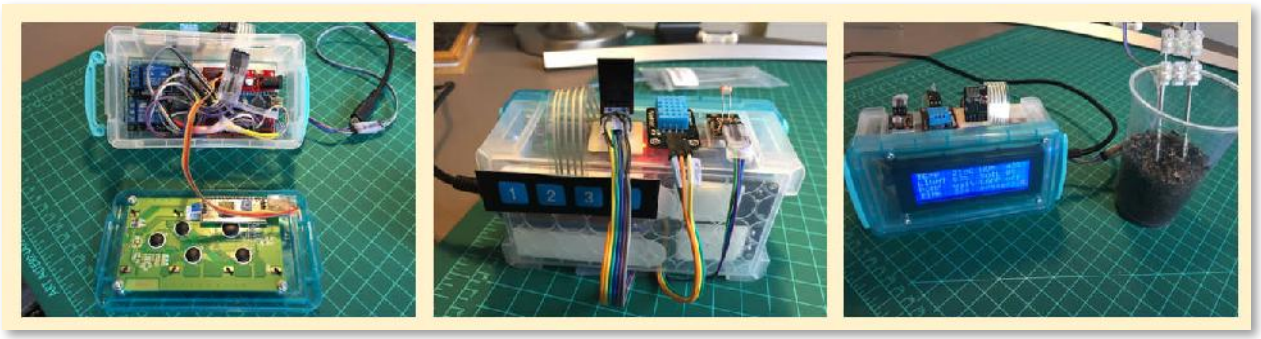
The only care that you must have is related with the voltage level. The ESP8266 works with 3.3V, so the Rx Pin which should not be connected directly to Nano Tx Pin (D3). A voltage level should be used.

www.EngineeringBooksPdf.com

In our case, we will build a voltage divider to be used as a voltage level converter.

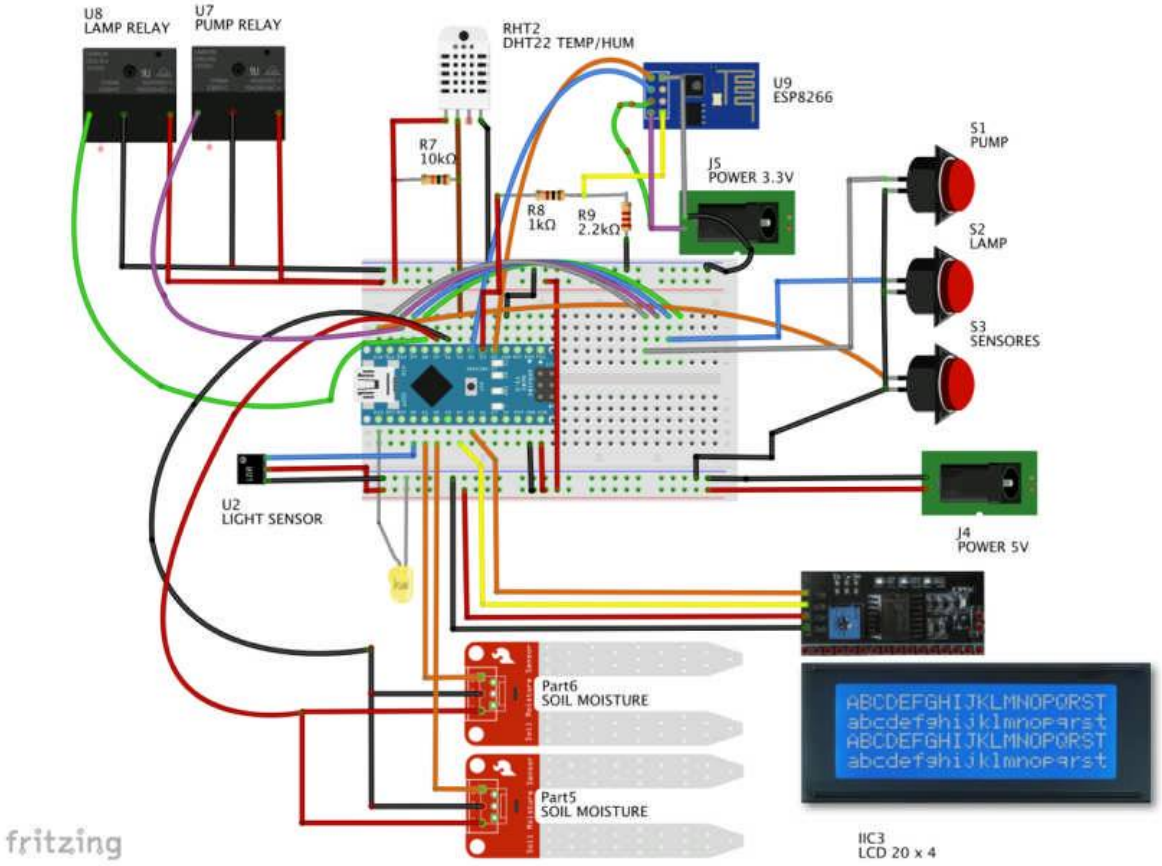


Note that we are using the ESP8266 connected to Nano Pin 2 (Tx) and Pin 3 (Rx), using library SoftSerial. If you want to "free" those digital pins, you can alternately use the Nano Serial pins 0 and 1. Only remember that you must disconnect them when uploading the code to Nano.



www.EngineeringBooksPdf.com

The electrical diagram here shows in more detail how to connect the ESP8266.



NOTE: If you want to connect the BUZZER, you should do it at pin D17 (same as pin A3). It is good to have a sound when you have a Comm. Error. You can use it during the test phase, leaving it out at final project (The code will work with/without it). It is up to you to have it or not.

You can use below code for testing and/or setup your ESP8266-01:

https://github.com/Mjrovai/ArduFarmBot/tree/master/ESP8266%20Test%20Setup

www.EngineeringBooksPdf.com

2.3 - Connecting the ESP8266 to the internet

Once the module is installed, we must first apply a “Reset” on its CH-PD Pin.

```
/*****************************************************
* Reset funtion to accept communication
*****************************************************/
void reset8266(void)
{
  pinMode(CH_PD, OUTPUT);
  digitalWrite(CH_PD, LOW);

  delay(300);
  digitalWrite(CH_PD, HIGH);
  Serial.print("8266 reset OK");
  lcd.clear();
  lcd.println("8266 reset OK       ");
}
```

After resetting, let’s connect it to your local network using your credentials in the code, change: USERNAME and PASSWORD, and to initiate the module as a “STA: Station Mode” (CWMODE = 1):

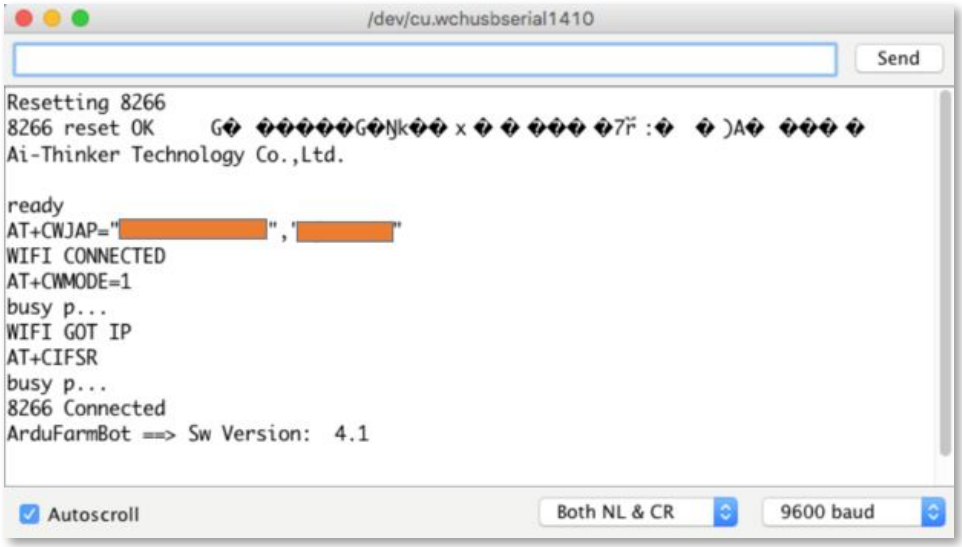
```
/*****************************************************
* Connect WiFi
*****************************************************/
void connectWiFi(void)
{
  sendData("AT+RST\r\n", 2000, DEBUG); // reset
  sendData("AT+CWJAP=\"USERNAME\",\"PASSWORD\"\r\n", 2000, DEBUG);
//Connect network
  delay(3000);
  sendData("AT+CWMODE=1\r\n", 1000, DEBUG);
  sendData("AT+CIFSR\r\n", 1000, DEBUG); // Show IP Adress
  lcd.clear();
  lcd.print("8266 Connected");
  Serial.println("8266 Connected");
}
```


www.EngineeringBooksPdf.com

To send data to ESP8266, the function sendData() was used:

```
/*******************************************************
* Send AT commands to module
*******************************************************/

String sendData(String command, const int timeout, boolean debug)
{
  String response = "";
  esp8266.print(command);
  long int time = millis();
  while ( (time + timeout) > millis())
  {
    while (esp8266.available())
    {
      // The esp has data so display its output to the serial
window
      char c = esp8266.read(); // read the next character.
      response += c;
    }
  }
  if (debug)
  {
    Serial.print(response);
  }
  return response;
}
```

Above functions will be called during the “Setup Phase” of our Code. If everything were done correctly, at Serial Monitor will be display similar messages as below:



www.EngineeringBooksPdf.com

2.4 - “Data Storage Cloud”: The ThinkSpeak.com

All captured data by the ArduFarmBot will be uploaded to the cloud, using the free service of “ThinkSpeak.com” (https://thingspeak.com/).

At Loop() function (after we capture data with *readSensors()*), we should call a specific function to upload the captured data: updateDataThingSpeak();

```
/************************************************
* Transmit data to thingspeak.com
************************************************/
void updateDataThingSpeak(void)
{
  startThingSpeakCmd ();

  cmd = msg ;
  cmd += "&field1=";     //field 1 for DHT temperature
  cmd += tempDHT;
  cmd += "&field2=";    //field 2 for DHT humidity
  cmd += humDHT;
  cmd += "&field3=";    //field 3 for LDR luminosity
  cmd += lumen;
  cmd += "&field4=";    //field 4 for Soil Moisture data
  cmd += soilMoist;
  cmd += "&field5=";    //field 5 for PUMP Status
  cmd += pumpStatus;
  cmd += "&field6=";    //field 6 for LAMP Status
  cmd += lampStatus;
  cmd += "\r\n";

  sendThingSpeakCmd();
}
```


www.EngineeringBooksPdf.com

In order to send those data, please start communication with ThingSpeak. We will do this by using function: startThingSpeakCmd ():

```
/*****************************************************
* Start communication with ThingSpeak.com
*****************************************************/
void startThingSpeakCmd(void)
{
  cmd = "AT+CIPSTART=\"TCP\",\"";
  cmd += IP;
  cmd += "\",80";
  esp8266.println(cmd);
  delay(2000);
  if(esp8266.find("Error"))
  {
    Serial.println("ESP8266 START ERROR");
    return;
  }
  Serial.println("Thinkspeak Comm Started");
  cmd ="";
}
```

Once the channel is opened with ThingSpeak and the “cmd” string is assembled with the data, it is time to upload all of it to corresponding channel at ThingSpeak, using function: sendThingSpeakCmd():

www.EngineeringBooksPdf.com

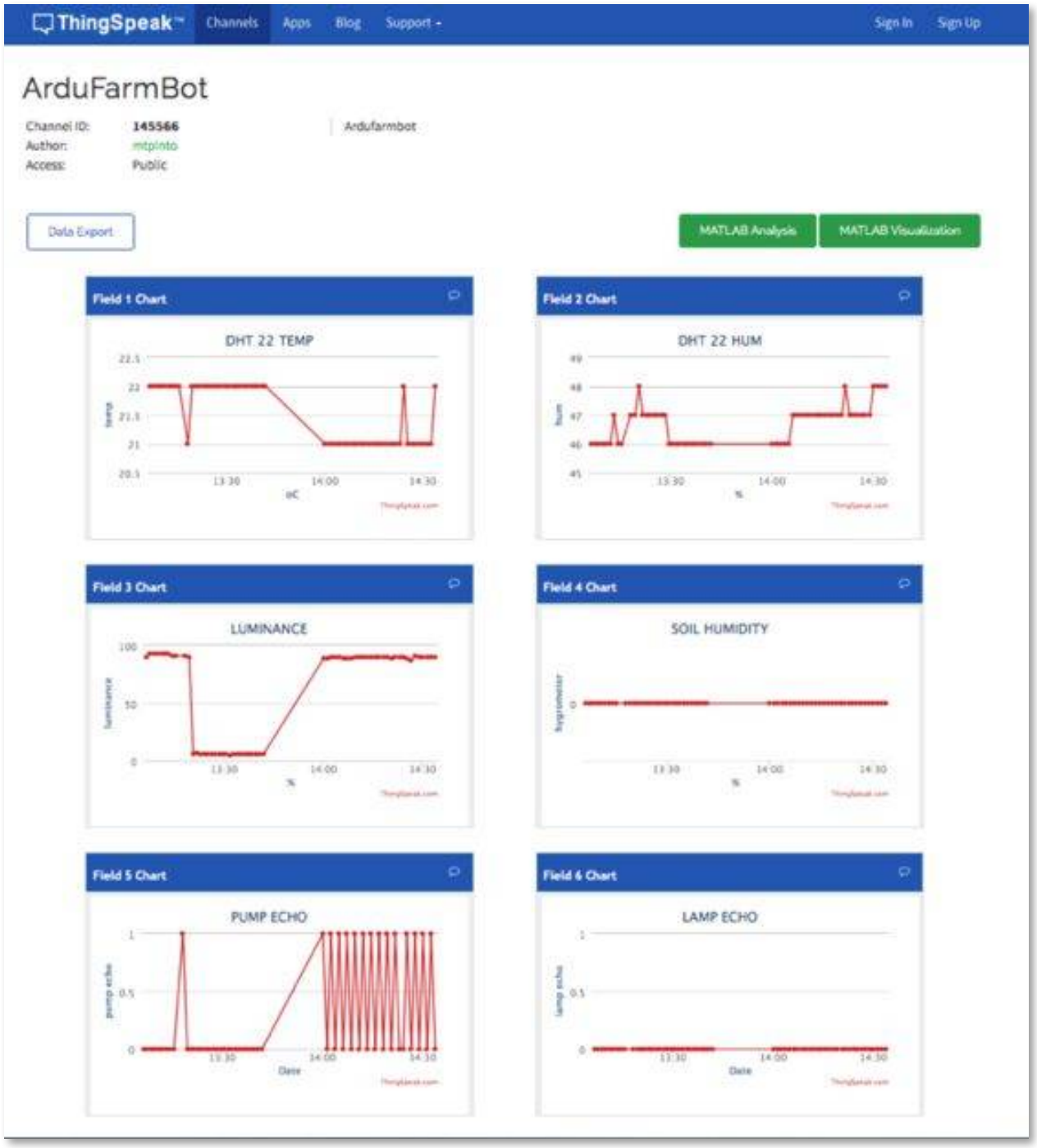
```
/****************************************************
* Update channel ThingSpeak.com
****************************************************/
String sendThingSpeakCmd(void)
{
  esp8266.print("AT+CIPSEND=");
  esp8266.println(cmd.length());
  if(esp8266.find(">")){
    esp8266.print(cmd);
    Serial.println("");
    Serial.println("");
    Serial.println(cmd);
    delay(500);

    String messageBody = "";
    while (esp8266.available())
    {
      String line = esp8266.readStringUntil('\n');
      if (line.length() == 1)
      { //actual content starts after empty line (that has length
1)
        messageBody = esp8266.readStringUntil('\n');
        Serial.print("Message received: ");
        Serial.println(messageBody);
      }
    }
    return messageBody;
  }
  else{
    esp8266.println("AT+CIPCLOSE");
    Serial.println("ESP8266 CIPSEND ERROR: RESENDING"); //Resend...
    error=1;
    return "error";
  }
}
```

Above functions were based on a great and detailed tutorial developed by Michalis Vasilakis. For more details, please see his tutorial:

https://www.instructables.com/id/Arduino-IOT-Temperature-and-Humidity-With-ESP8266-/

www.EngineeringBooksPdf.com

Photo below shows the ArduFarmBot channel at ThingSpeak.com:



www.EngineeringBooksPdf.com

2.5 - Commanding actuators from the web

At this point, we are uploading all collected data and storing them into the cloud. This is outstanding and useful for a remote monitoring, but what happens if based on those data we also want to turn-on the Pump or the Lamp, independent of local automatic program? To do that, we will also need to "Download" data from the Cloud and command the controller to act based on those commands.

We will create specifics fields on our ThinkSpeak channel to command the actuators:

Field 7:

- Data = 1 ➔ PUMP should be Turn ON
- Data = 0 ➔ PUMP should be Turn OFF

Field 8:

- Data = 1 ➔ LAMP should be Turn ON
- Data = 0 ➔ LAMP should be Turn OFF

OK, but how to set up those fields directly at ThingSpeak? We can do it, for example writing a "PlugIn" directly at ThinksPeak, or we can use an external website to do it (this will be our choice). Anyway, on both cases you should use a command like:

```
api.thingspeak.com/update?key=YOUR_WRITE_KEY&field7=1
```

With above command (and using your channel Write Key), you can write "1" at field 7, which means that the PUMP should be turned on. You can easily test it, writing above command-line at your browser, corresponding field at your channel will be changed. In return, the browser will show a white page with a single number on upper left corner, consistent with sequential data entry in your channel.

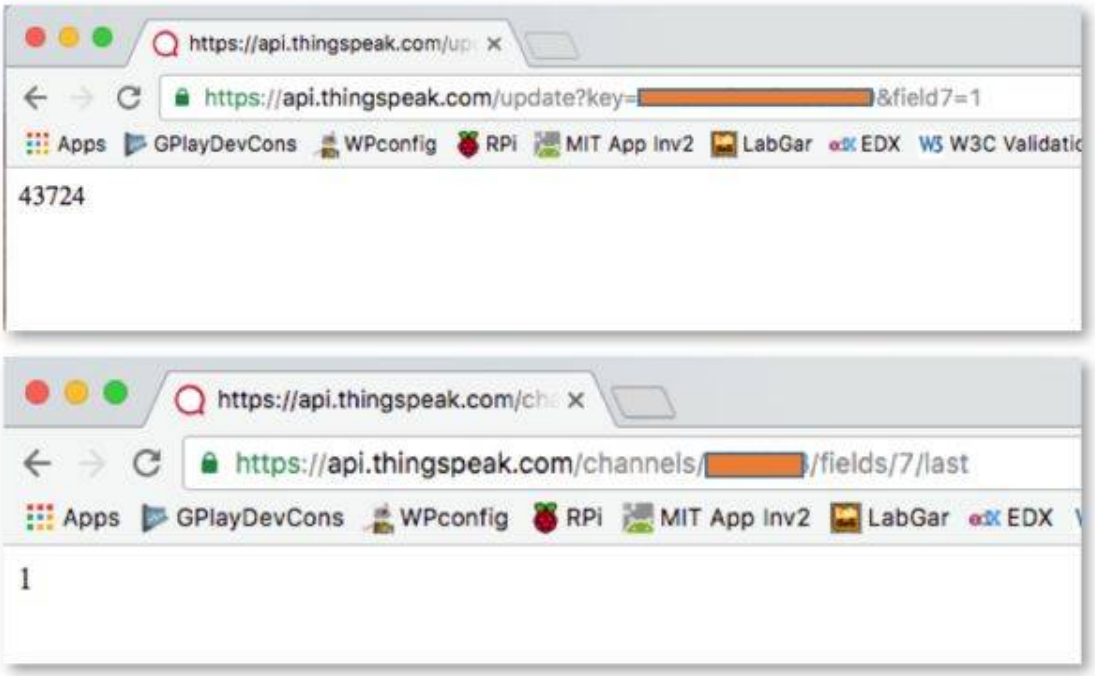
So far, 50% of the work is done! Now you must read this "command" (data on the field), down at local ArduFarmBot station.

Command to do this is shown below. It will get last data that was written at the specific field (that in our case will be a "command".

```
api.thingspeak.com/channels/CHANNEL_ID/fields/7/last
```


www.EngineeringBooksPdf.com

Same way as we did before, you can test the command-line, using your web browser. In this case, the browser will show you the data on that specific field. See the Photo below:



Back to “earth”, let’s write a function that will read this “last field”:

```
/*************************************************
* Read data from field7 of thingspeak.com
*************************************************/
int readLastDataField7()
{
  startThingSpeakCmd ();

  // "GET /channels/CHANNEL_ID/fields/7/last";
  cmd = msgReadLastDataField7;
  cmd += "\r\n";

  String messageDown = sendThingSpeakCmd();
  Serial.print("Command received: ");
  Serial.println(messageDown[7]);

  int command = messageDown[7]-48;
  return command;
}
```


www.EngineeringBooksPdf.com

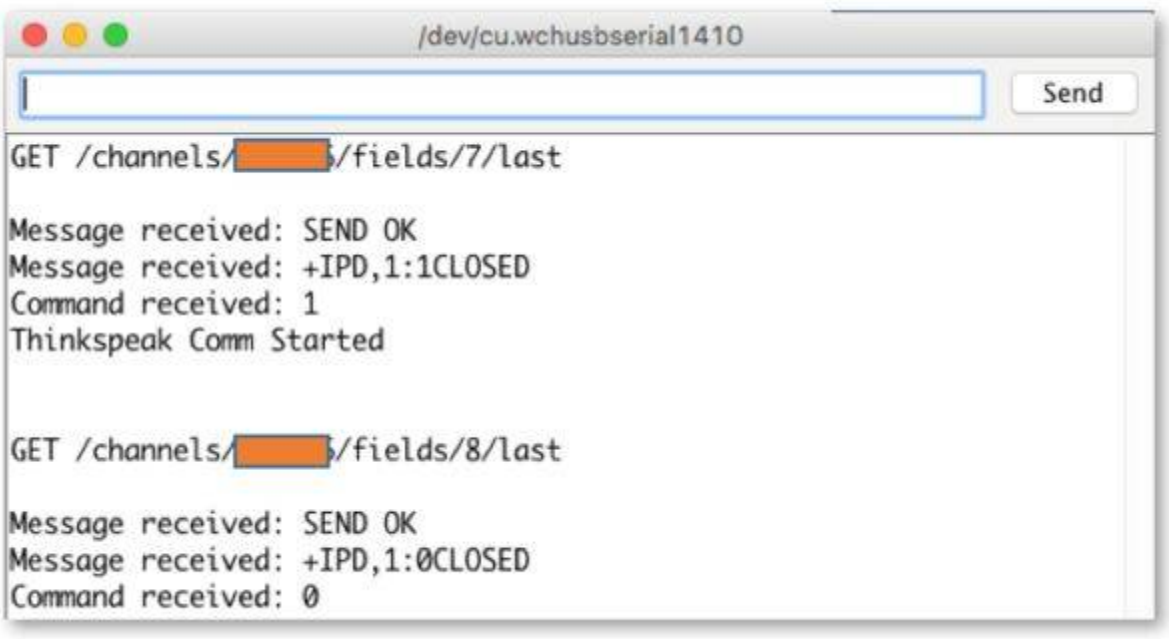
Previous function will return data on field 7 (“1” or “0”). A similar function should be written for Field 8.

Once we have the content of both fields, we should use them on a function that will command actuators similarly as we did with “manual command function”:

```
/*****************************************************
* Receive Commands from thingSpeak.com
*****************************************************/
void receiveCommands()
{
  field7Data = readLastDataField7();
  if (field7Data == 1)
  {
    digitalWrite(PUMP_PIN, HIGH);
    pumpStatus = 1;
    showDataLCD();
  }
  if (field7Data == 0)
  {
    digitalWrite(PUMP_PIN, LOW);
    pumpStatus = 0;
    showDataLCD();
  }

  delay (500);

  field8Data = readLastDataField8();
  if (field8Data == 1)
  {
    digitalWrite(LAMP_PIN, HIGH);
    lampStatus = 1;
    showDataLCD();
  }
  if (field8Data == 0)
  {
    digitalWrite(LAMP_PIN, LOW);
    lampStatus = 0;
    showDataLCD();
  }
  delay (500);
}
```


www.EngineeringBooksPdf.com

Therefore, from now on you can use the command-line in your browser to Turn On/Off Pump and Lamp remotely. Photo below shows how receiving command will appear at your Serial monitor:



Another important consideration is “coordination” between local and remote command. We must change the readLocalCmd() function to also update the Thinkspeak Field 7 and 8 respectively with Pump and lamp status (on their corresponding “IF statement”. See complete code at end of Part 2):

```
field7Data = pumpStatus;

field8Data = lampStatus;
```

Now field7Data and field8Data are in sync with the webpage commands and with local command actions when you press a button. So, let’s update aplyCmd() function, which is responsible to turn on/off the actuators:

www.EngineeringBooksPdf.com

```
/*************************************************
* Receive Commands and act on actuators
*************************************************/
void aplyCmd()
{
  if (field7Data == 1) digitalWrite(PUMP_PIN, HIGH);
  if (field7Data == 0) digitalWrite(PUMP_PIN, LOW);

  if (field8Data == 1) digitalWrite(LAMP_PIN, HIGH);
  if (field8Data == 0) digitalWrite(LAMP_PIN, LOW);
}
```

After beginning your tests, you will realize that any manual command at local or via web will be overcome by automatic actions defined through function autoControlGarden(); At this point you should consider who will be the "boss", having the last word! In our case, here we will define the following:

- Every loop cycle, at least most times, we will search to see if a button is pressed.
- Around every minute, we should do a "pooling" at ThinkSpeak and see if we have received an order from there.
- Around every 10 minutes, we will read the sensors, update the data on ThinkSpeak and more important take the automatic actions. Those actions will be taken independent of what were selected manually becoming the one that will be kept.

You can change it as you wish, which is a benefit about using a programmable processor for controlling things!

---

www.EngineeringBooksPdf.com

So, 2 timers will be used now, one for pooling remote commands and another one for reading the sensors (same one that we have used before):

```
long sampleTimingSeconds = 75; // ==> ******** Define Sample time
in seconds to read sensores *********
int reverseElapsedTimeSeconds = 0;
long startTiming = 0;
long elapsedTime = 0;
long poolingRemoteCmdSeconds = 20; // ==> ******** Define Pooling
time in seconds for new ThingSpeak commands *********
long startRemoteCmdTiming = 0;
long elapsedRemoteCmdTime = 0;
```

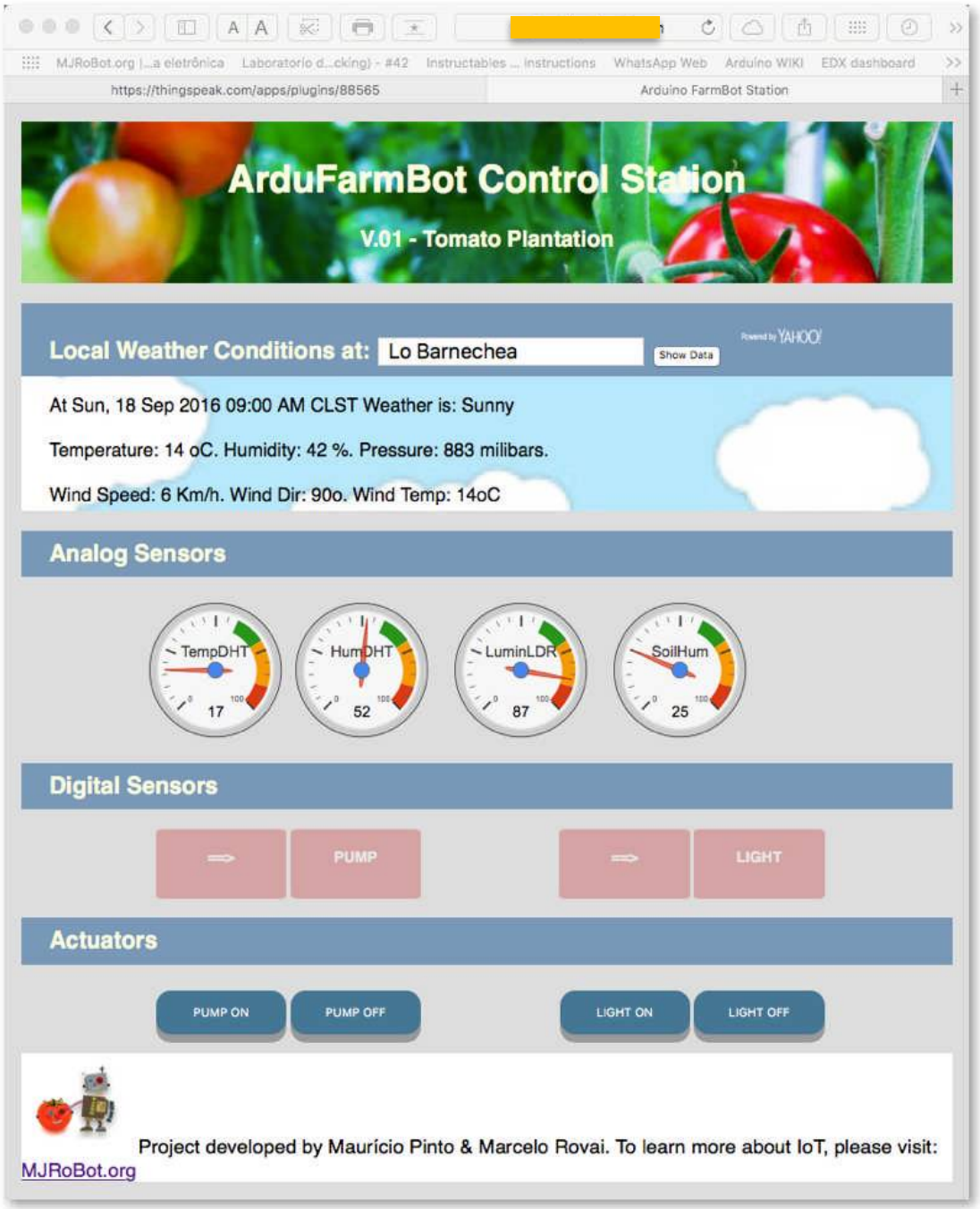
So, the loop() function should now be rewritten as below:

```
void loop()
{
  elapsedRemoteCmdTime = millis()-startRemoteCmdTiming;   // Start
timer for pooling remote commands
  elapsedTime = millis()-startTiming;   // Start timer for
measurements
  reverseElapsedTimeSeconds = round (sampleTimingSeconds -
elapsedTime/1000);

  readLocalCmd(); //Read local button status
  showDataLCD();

  if (elapsedRemoteCmdTime > (poolingRemoteCmdSeconds*1000))
  {
    receiveCommands();
    updateDataThingSpeak();
    startRemoteCmdTiming = millis();
  }

  if (elapsedTime > (sampleTimingSeconds*1000))
  {
    readSensors();
    autoControlPlantation();
    updateDataThingSpeak();
    startTiming = millis();
  }
}
```


www.EngineeringBooksPdf.com

2.6 - Implementing a dedicated Webpage

At this point our ArduFarmBot is operational and can be controlled from the web. You can monitor the data on *Thinkspeak* site (https://thingspeak.com/) and send commands using a browser, but of course this "web solution" can not be considered an "elegant" one! The best way to implement a full IoT solution is to develop a complete webpage that will display all data, also having buttons to activate the actuators.



www.EngineeringBooksPdf.com

We choose the Byethost (https://byet.host/), a free webhost, very easy and simple for handling your pages.

> *We will not detail how to develop such page, as this is not the center purpose of this book, but we will leave on the ArduFarmBot GitHub, complete HTML5, CSS3 and JavaScript source codes*

It is important to reinforce that the page does not work directly with the ArduFarmBot Local Control Station. What it is really doing is interacting with *ThinkSpeak* channel as below:

1. Reading sensor data on fields 1, 2, 3, 4
2. Reading actuator status on fields 5 and 6
3. Writing data on fields 7 and 8
4. Reading Local weather data from Yahoo services

Item 4 above is not crucial for the project, but always is good to have additional data available in case you want take some remote actions regarding your tomato garden.

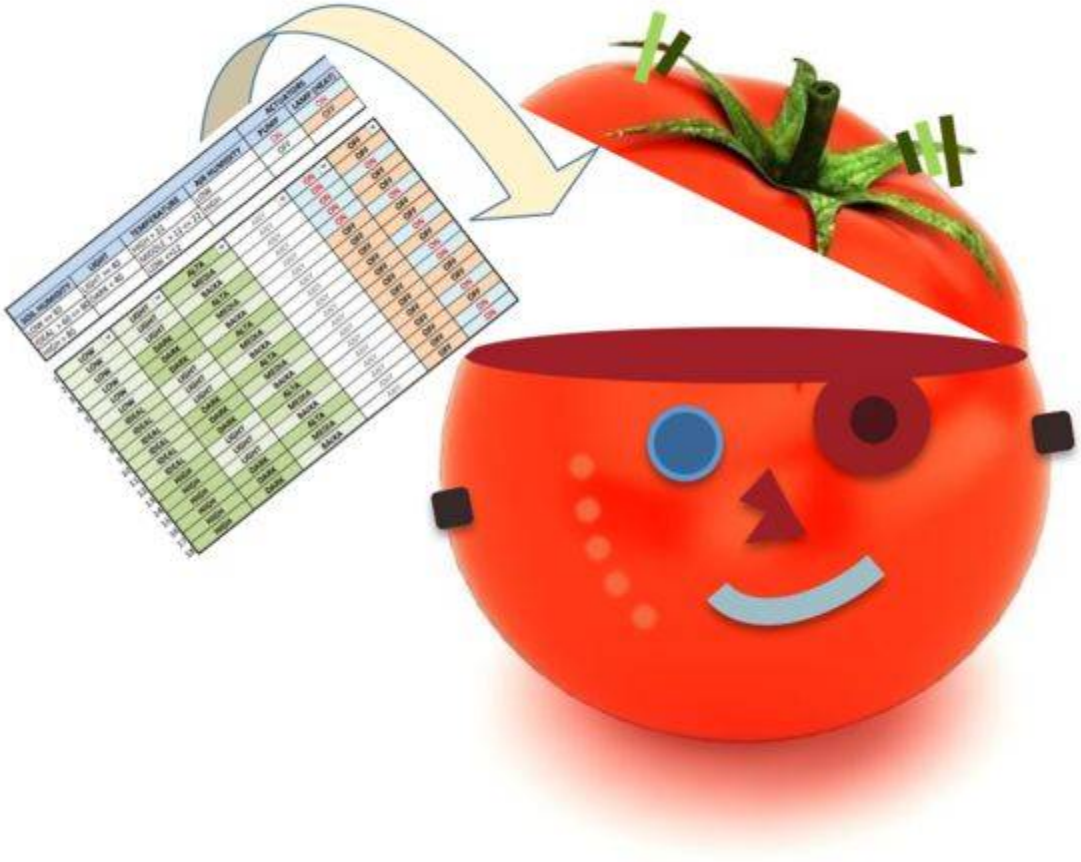
> *Other important consideration is that you can store those data on another ThingSpeak Channel and download it to your Arduino, showing weather data on local LCD display. This can be another cool idea for a project! We leave it here as a suggestio).*

The complete HTML, CSS and JavaScript source codes can be downloaded from here:

https://github.com/Mjrovai/ArduFarmBot/tree/master/WebPage

www.EngineeringBooksPdf.com

2.7 - Returning to the brain. A Sensor-Actuator Matrix approach



At the beginning of this project, we defined some preliminary considerations on how the actuators should act depending on sensors' reading. We did only simple choice approach, but what will happen if we have a more complex situation? Several different conditions? What we will develop is a "Sensor – Actuator matrix approach".

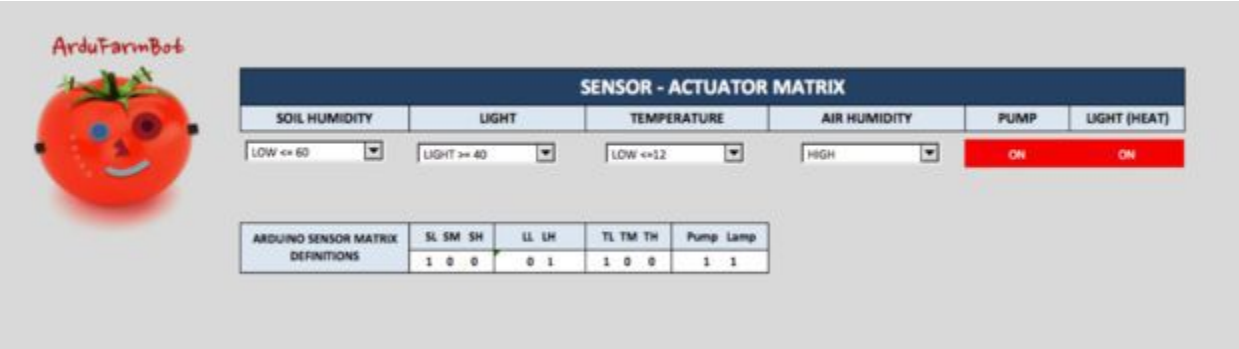
For each sensor, a condition was defined as well as how the output should be for every actuator, all reflected in a matrix. Result reflected on the Excel spreadsheet included below.

www.EngineeringBooksPdf.com

				ACTUATORS	
SOIL HUMIDITY	LIGHT	TEMPERATURE	AIR HUMIDITY	PUMP	LAMP (HEAT)
LOW <= 60	LIGHT >= 40	HIGH > 22	LOW	ON	ON
IDEAL > 60 <= 80	DARK < 40	MIDDLE > 12 <= 22	HIGH	OFF	OFF
HIGH > 80		LOW <=12			

1	LOW	LIGHT	ALTA	ANY	ON	OFF
2	LOW	LIGHT	MEDIA	ANY	ON	OFF
3	LOW	LIGHT	BAIXA	ANY	ON	ON
4	LOW	DARK	ALTA	ANY	ON	OFF
5	LOW	DARK	MEDIA	ANY	ON	OFF
6	LOW	DARK	BAIXA	ANY	OFF	ON
7	IDEAL	LIGHT	ALTA	ANY	OFF	OFF
8	IDEAL	LIGHT	MEDIA	ANY	OFF	OFF
9	IDEAL	LIGHT	BAIXA	ANY	OFF	ON
10	IDEAL	DARK	ALTA	ANY	OFF	OFF
11	IDEAL	DARK	MEDIA	ANY	OFF	ON
12	IDEAL	DARK	BAIXA	ANY	OFF	ON
13	HIGH	LIGHT	ALTA	ANY	OFF	OFF
14	HIGH	LIGHT	MEDIA	ANY	OFF	OFF
15	HIGH	LIGHT	BAIXA	ANY	OFF	ON
16	HIGH	DARK	ALTA	ANY	OFF	OFF
17	HIGH	DARK	MEDIA	ANY	OFF	ON
18	HIGH	DARK	BAIXA	ANY	OFF	ON

There are two spreadsheets in the file, a table with a filter and a version were you can select multiple sensor conditions and see how the actuators will work depending on the selection.



www.EngineeringBooksPdf.com

Once the Matrix is defined, we must translate it to our code. An array of 18 lines and 10 columns was created to “copy” conditions Sensor-Actuator Matrix:

```
//                      +---SOIL---+-LIGHT-+---TEMP---+---ACTUAT----+
 //                       SL  SM  SH  LL  LH  TL  TM  TH   Pump   Lamp
boolean SDF [18] [10] = {{ 1,  0,  0,  0,  1,  0,  0,  1,    1,    0 },
                         { 1,  0,  0,  0,  1,  0,  1,  0,    1,    0 },
                         { 1,  0,  0,  0,  1,  1,  0,  0,    1,    1 },
                         { 1,  0,  0,  1,  0,  0,  0,  1,    1,    0 },
                         { 1,  0,  0,  1,  0,  0,  1,  0,    1,    0 },
                         { 1,  0,  0,  1,  0,  1,  0,  0,    0,    1 },
                         { 0,  1,  0,  0,  1,  0,  0,  1,    0,    0 },
                         { 0,  1,  0,  0,  1,  0,  1,  0,    0,    0 },
                         { 0,  1,  0,  0,  1,  1,  0,  0,    0,    1 },
                         { 0,  1,  0,  1,  0,  0,  0,  1,    0,    0 },
                         { 0,  1,  0,  1,  0,  0,  1,  0,    0,    1 },
                         { 0,  1,  0,  1,  0,  1,  0,  0,    0,    1 },
                         { 0,  0,  1,  0,  1,  0,  0,  1,    0,    0 },
                         { 0,  0,  1,  0,  1,  0,  1,  0,    0,    0 },
                         { 0,  0,  1,  0,  1,  1,  0,  0,    0,    1 },
                         { 0,  0,  1,  1,  0,  0,  0,  1,    0,    0 },
                         { 0,  0,  1,  1,  0,  0,  1,  0,    0,    1 },
                         { 0,  0,  1,  1,  0,  1,  0,  0,    0,    1 },
                        };
```

To work with the Matrix, we created a function defSensorStatus (). This function tests for each line if the condition of the 8 first columns are TRUE. If Yes, the condition of the last 2 columns are executed.

For example:

if (1 and 0 and 0 and 0 and 1 and 0 and 0 and 1) { pumpStatus = 1; lampStatus = 0}

else if (1 and 0 and 0 and 0 and 1 and 0 and 1 and 0) { pumpStatus = 1; lampStatus = 0}

And so on ...

www.EngineeringBooksPdf.com

Inside previous function another array has been created with status of each sensor reading:

```
boolean snsSts[8]={0, 0, 0, 0, 0, 0, 0, 0}; // SL, SM, SH, LL, LH, TL, TM, TH
```

This variable array also will be used for LOG register.

The Excel spreadsheet can be download from the ArduFarmBot GitHub depository:

https://github.com/Mjrovai/ArduFarmBot/blob/master/Matrix_Sensor_Actuator.xlsx

www.EngineeringBooksPdf.com

2.8 - Code optimization

During the process of developing ArduFarmBot we realize that some changes on the original specification were needed:

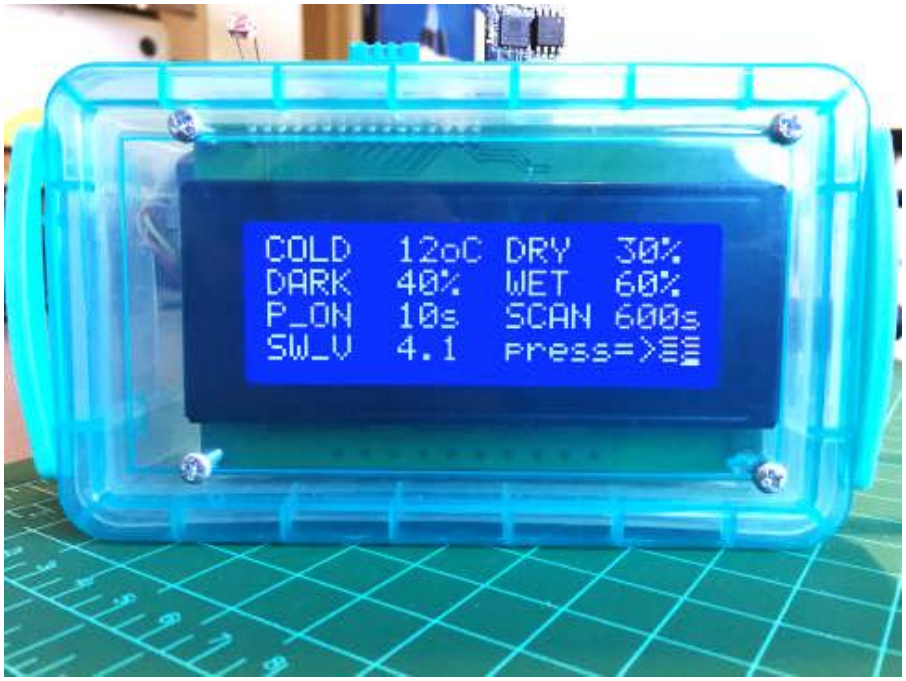
A. LCD Display:

The LCD display should be OFF by default and any time that a sensor reading is needed we can manually turn it "ON". This condition was implemented on the code and the button "Sensors Read" should be used as in "toggle" mode to Turn ON/OFF the LCD at any time. Turning ON or OFF the display will also update sensors readings for displaying only (readings will not be used by ArduFarmBot on its regular functions).

B. Initial Setup:

When ArduFarmBot is Turned ON (or reset), the LCD will display "Initial Set-up". To start running the program, the button "Sensors" must be pressed (or a 60s should pass without any action). Initial information shown is:

- COLD Temperature (i.e. 12°C)
- DRY Soil Humidity (i.e. 30%)
- WET Soil Humidity (i.e. 60%)
- DARK light (i.e. 40%)
- P_ON Pump time be ON (i.e. 10s)
- SCAN Time to read sensors (i.e. 600s)
- SW_Vertion (i.e. 4.1)



www.EngineeringBooksPdf.com

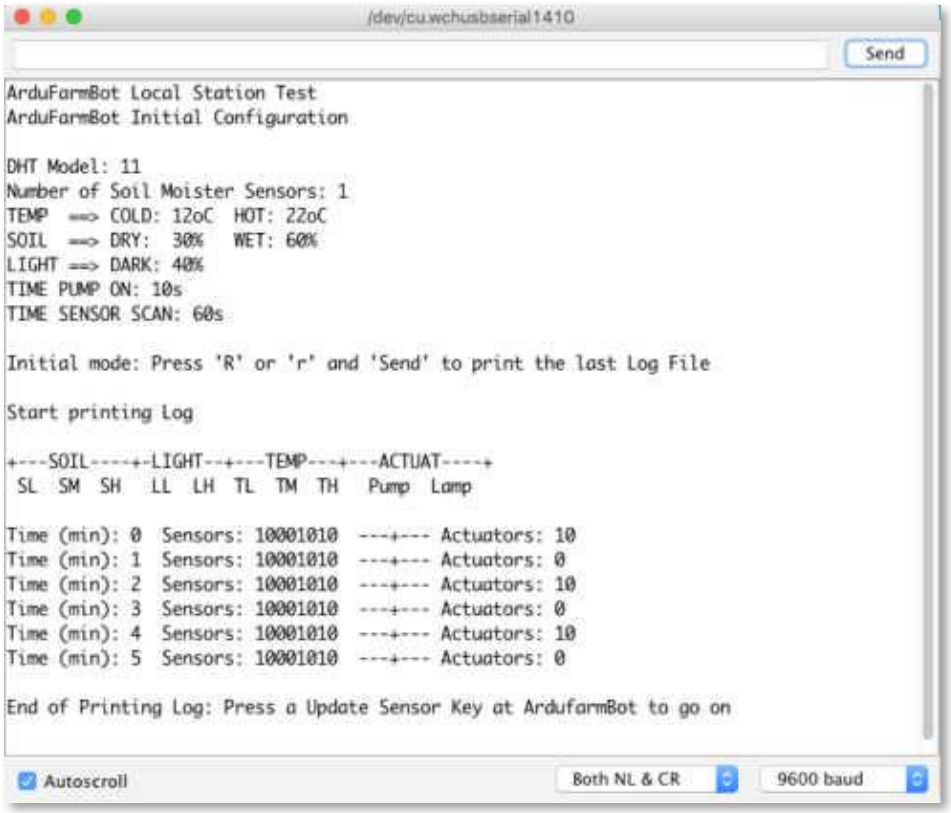
C. Log Record:

For audit purposes, we have created a LOG with the readings and actuations of our ArduFarmBot. At every reading cycle, the function: storeDataLogEEPROM() is executed:

```
/*****************************************************
* Storage of Log data at Arduino EEPROM
*****************************************************/
void storeDataLogEEPROM(void)
{
  for (int i = 0; i<8; i++)
  {
    logData = logData + (snsSts[i])<< 1;
  }
  EEPROM.write (memoAddr, logData);
  memoAddr++;
  logData = 0;
  logData = logData + pumpStatus;
  logData = logData << 1;
  logData = logData + lampStatus;
  EEPROM.write (memoAddr, logData);
  EEPROM.write (0, memoAddr+1);
  logData = 0;
  if ((memoAddr+1) == 1023) memoAddr=1;
  else memoAddr++;
}
```

As commented in the last step, what will be stored at the Arduino EEPROM is the content, bit a bit of the array snsSts[] plus Pump and Lamp status. You can see below LOG on Serial Monitor:

www.EngineeringBooksPdf.com



All ArduFarmBot codes were divided in different files for easier understanding. Note that 2 new files were added on this second part:

- **communication.ino** (ThingSpeak and ESP8266 specific funtions)
- **stationCredentials.h** (ThinkSpeak Channel ID and specific Keys for writing on the channel)

Finally, once the code ended with a reasonable size, we decided to store constant data in flash (program) memory instead of SRAM. For that, we use the PROGMEM keyword that is a variable modifier.

For example, instead of using:

```
#define DHTPIN 5,
```

We will use:

```
const PROGMEM byte DHTPIN = 5;
```


www.EngineeringBooksPdf.com

The keyword PROGMEN tells the compiler "to put the information into a flash memory", instead of into SRAM, where it would normally go. You must also include the library avr/pgmspace.h at main file of your code.

Another good procedure to reduce SRAM use is to comment (or delete) all Serial.Print() lines that you have used for debugging during development.

You can find the complete ArduFarmBot Arduino code at the project GitHub:

https://github.com/Mjrovai/ArduFarmBot/tree/master/ArduFarmBot_Control_Station_V4_1_BLOG

Do not forget to change the dummy data on credentials.h with your Channel Id and Write Key. Also on communication.ino, use your real Username and password to connect the ESP 8266 at Internet.

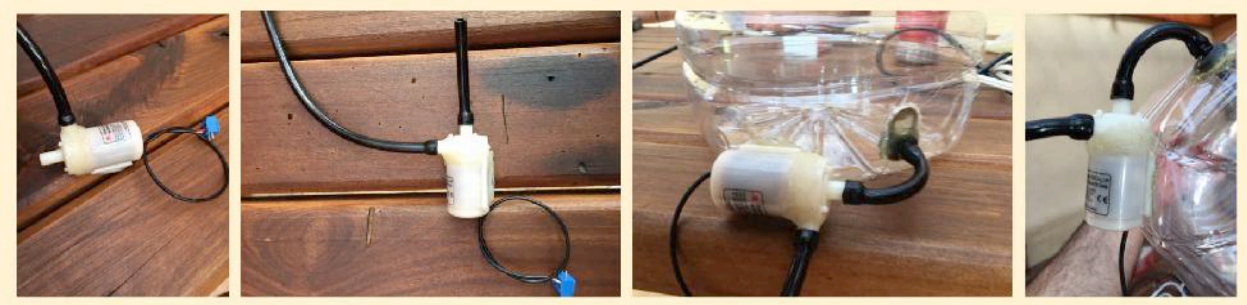
---

www.EngineeringBooksPdf.com

2.9 - ArduFarmBot in action: Marcelo's garden

Preparing the water reservatory:

Connecting the water pump:



Seed germination and initial grow (around 45 days):

Best plant selection and transgarden:

www.EngineeringBooksPdf.com

3

www.EngineeringBooksPdf.com

Part 3 - The ArduFarmBot 2

In the first two parts of this tutorial, we developed a fully automated Arduino-based system working both, manually and via Internet, with help of online services as ThingSpeak.com. In this 3rd part, starting from the original specifications, we will develop the "ArduFarmBot 2", that will be based on the different IoT platforms: the *NodeMCU ESP8266* and *BLYNK*.

A. Specifications:

Based on data collected from any garden such as temperature and humidity (for both, air and soil), the ArduFarmBot 2 will decide the right amount (and when) the orchard should receive heat and water. The system should also allow manual intervention of an operator to control a water pump and electric lamp to generate heat for planting. This manual intervention should be possible to be performed both locally and remotely via the Internet. Summarizing, the system should receive:

Input

- *Sensors:*
  - Air Temperature
  - Air Relative Humidity
  - Soil Temperature (new)
  - Soil Moisture (humidity)
- *Buttons:*
  - Pump ON / OFF
  - Lamp ON / OFF

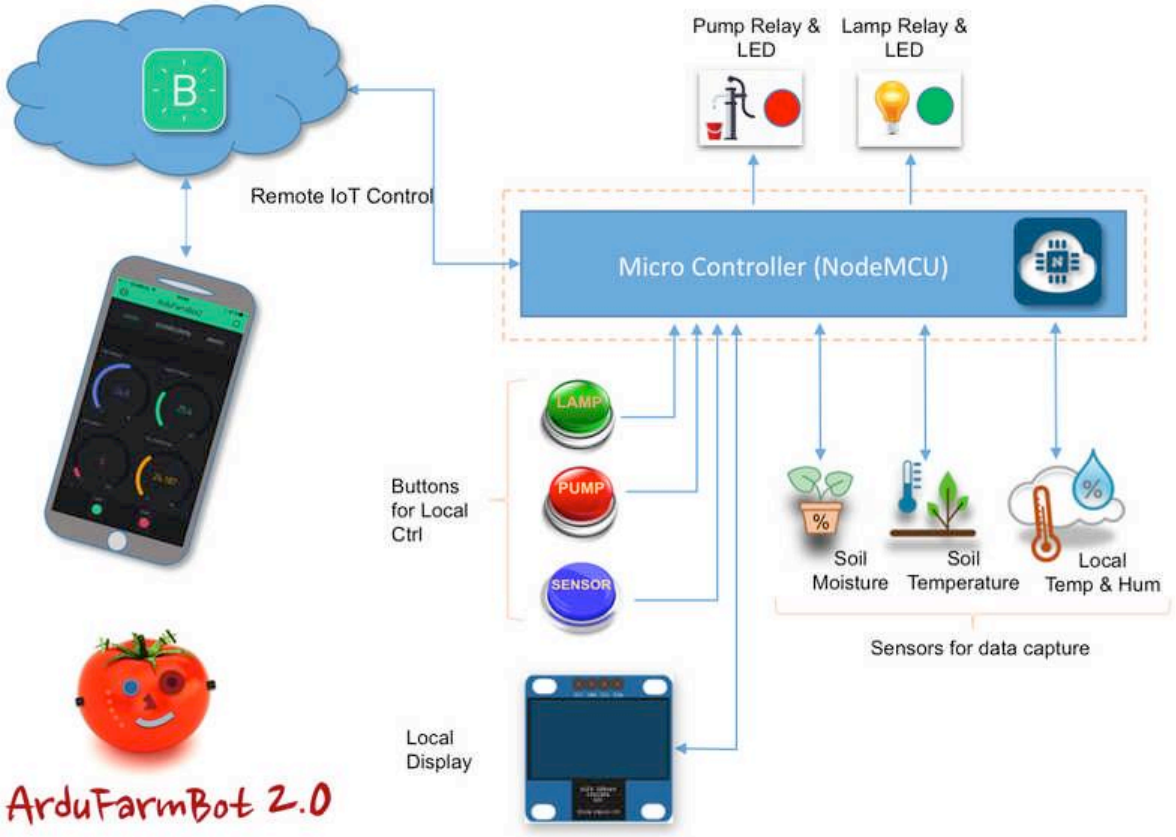
Output:

- *Actuators*:
  - Relay for Pump control
  - Relay for Lamp control
- *Automatic messages must be sent on main events as:*
  - Pump ON
  - Lamp On
  - System Off-line
- *Data Display*
  - All analog and digital data should be available for instant evaluation
- *Data Storage*
  - Historic data should be storage remotely

www.EngineeringBooksPdf.com

B. Block Diagram:

The block diagram shows main components of the project.



www.EngineeringBooksPdf.com

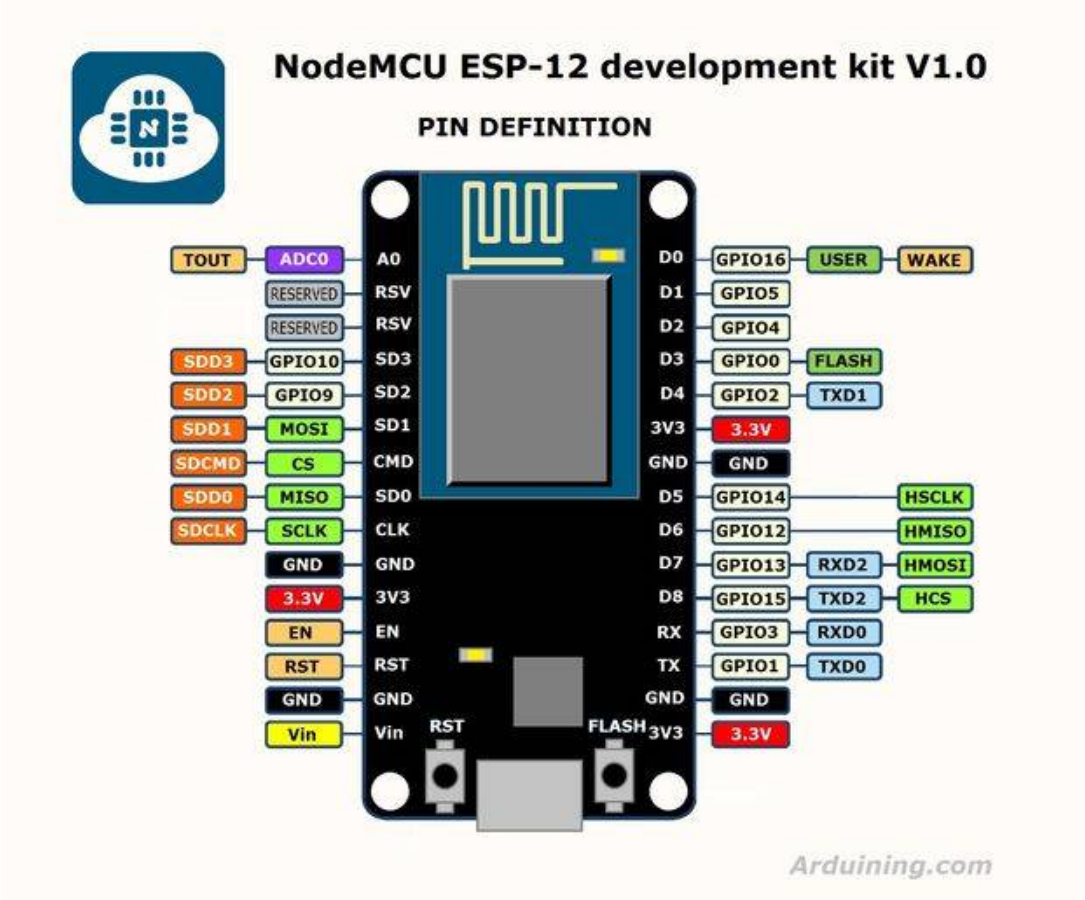
3.1 - Bill of Materials ("BoM") for part 3

The main ArduFarmBot components are (values in USD, for reference only):

- ✓ NodeMCU ESP8266 12-E – ($9.00)
- ✓ Temperature and Humidity Sensor DHT22 – ($4.00)
- ✓ Soil Moisture Sensor – ($7.00) (Optional, can be DIY)
- ✓ Waterproof Digital Temperature Temp Sensor Thermal Probe DS18B20($3.00)
- ✓ 0.96″ I2C IIC SPI Serial 128X64 Yellow&Blue OLED LCD ($9.00)
- ✓ Buttons (3X) – ($1.00)
- ✓ LEDs (2X) ($0.20)
- ✓ DC 12V 2CH 2 Channel Isolated Optocoupler High/Low Level Trigger Relay Module ($9.00)
- ✓ Jump wires (S1.00)
- ✓ Breadboard ($3.00)
- ✓ 4.7K ohms resistor – ($0.03)
- ✓ 10K ohms resistor – ($0.03)
- ✓ 220 ohms resistor – ($0.03)
- ✓ 5V/ 2A External Power Supply ($8.00)
- ✓ Mini DC Water Pump ($9.00)

www.EngineeringBooksPdf.com

3.2 - The NodeMCU



NodeMCU ESP-12E is the integrated version of the popular ESP8266, a Serial to Wi-Fi "System On a Chip" (SoC) which came to the scene for the first time back in 2013, and released the following year. ESP8266 was developed by Shangai-based company Espressif Systems, an IC manufacturer focused on the development of RF chips, Wi-Fi particularly.

There are several modules in the market that use ESP8266 chip, they are named ESP-NN, where NN is a number 01, 02, ... 12, sometimes followed by a letter. Those modules typically carry the ESP8266 SoC, flash memory, a crystal, and in most cases, an onboard antenna. At below link, you can find the full list of ESP8266 based devices found in the market.:

http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family,

Main modules are without doubt: the ESP-01 and the ESP-12E. On the ArduFarmBot 2 project, we will use the *ESP-12E Development Board (NodeMCU DevKit 1.0)*. Its Pin-out is shown above.

www.EngineeringBooksPdf.com

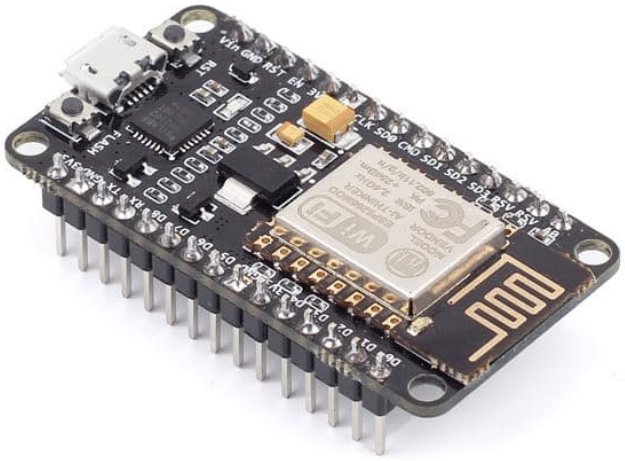
For further usage of the ESP-12E module, power regulation and USB connectivity have been added to the Node MCU, that includes:

- UART USB Adapter: Silicon Labs CP2102
- NCP1117 3.3VDC Voltage Regulator
- Micro-USB connector
- Additional GND, Vin, 3.3VDC pins for easy access during development.

In short, the NodeMCU ESP-12E is a device ready to use, simply install the USB drivers to your computer and start writing programs that connect to your Wi-Fi network!

Technical Specifications

- Support STA/AP/STA+AP 3 working modes;
- Built-in TCP/IP protocol stack, support multiple-channel TCP Client connection (max 5);
- 0～D8, SD1～SD3: used for GPIO, PWM (D1-D8), IIC, ect; driven ability can be arrived at 15mA;
- AD0: one-way 10 bits ADC;
- Power input: 4.5V~9V (10VMAX), support USB powered and USB debug;
- Working current: ≈70mA (200mA MAX, continue), standby＜200uA;
- Transmission data rate: 110-460800bps;
- Support UART/GPIO data communication interface;
- Support update firmware remotely (OTA);
- Support Smart Link;
- Working temperature: -40°C～＋125°C ;
- Driven mode: double large-power H bridge driven
- Weight: 7g.



www.EngineeringBooksPdf.com

3.3 - Using Arduino IDE with NodeMCU com

We will program and use the NodeMCU almost as a regular Arduino, using its IDE. It is important to remember that any new "custom firmware" will replace anything previously stored in the chip's flash memory, including the original firmware loaded at factory where the AT commands were common used. Although we can use the manufacturer's SDK to develop our custom firmware, it is much easier to use the good and old Arduino IDE.

Let's start:

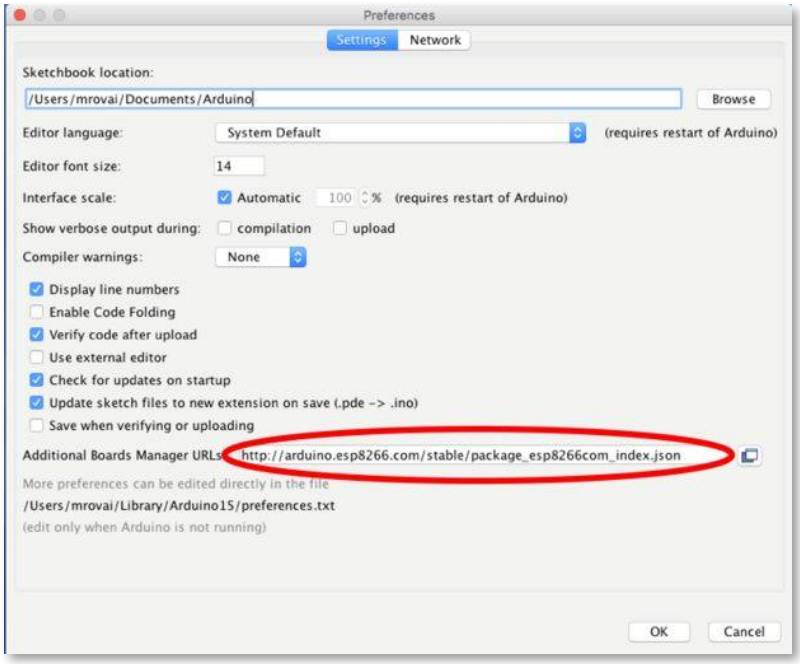
In the Arduino IDE:

⇒ Open PREFERENCES window and

⇒ Enter below URL into the Additional Boards Manager URLs field, and

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

⇒ Select OK.



www.EngineeringBooksPdf.com

⇒ Select MENU option Tools → Board → Boards Manager...and scroll down to locate the option esp8266 by ESP8266 Community which should be the last item on the list, and

⇒ click INSTALL

Installing the USB Drivers

The USB to Serial UART module included on the board is the Silicon Labs' CP2012, for which we usually need to install the readily available Virtual COM Port (VCP) drivers. In the case of MAC, the device file created to communicate with the CP2102 has the name /dev/cu.SLAB_USBtoUART**.**

You can find the appropriate drive for your computer at following link: https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

---

After restarting the Arduino IDE we can now select the board we are using from the MENU:

⇒ Tools → Board → NodeMCU 1.0 (ESP-12E Module).

www.EngineeringBooksPdf.com

Then, we specify correct CPU Frequency

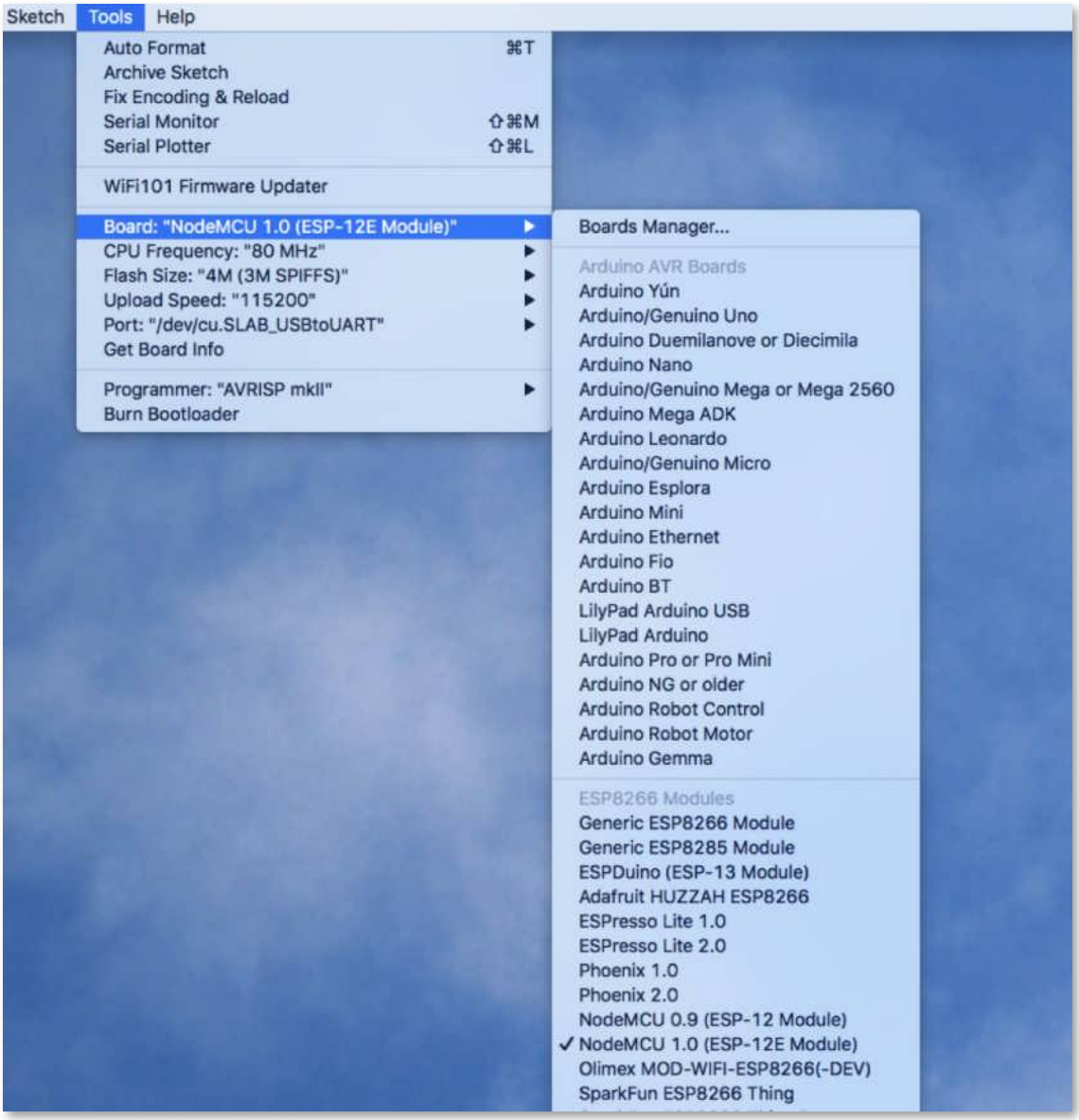
⇒ Tools → CPU Frequency: → 80MHz

And Upload Speed:

⇒ Tools → Upload Speed: → 115200).

Finally, the last step is to select the correct option for the Port:

⇒ Tools → Port → /dev/cu.SLAB_USBtoUART

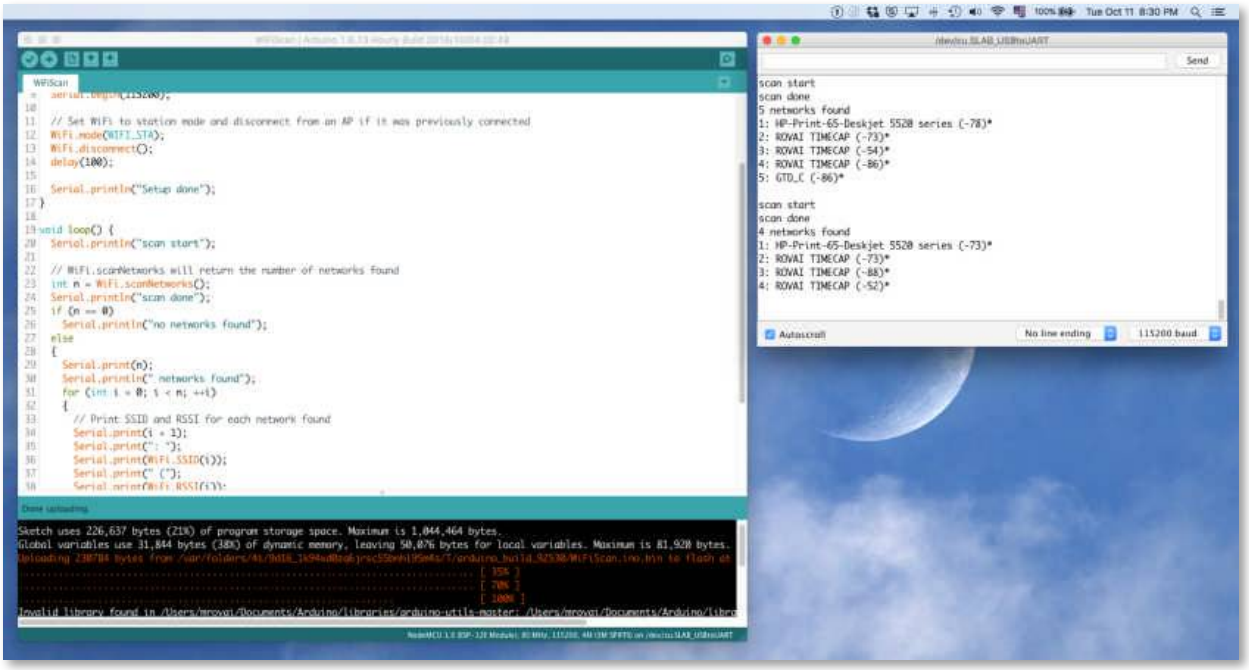


www.EngineeringBooksPdf.com

At this point we are ready to write our own firmware and upload it, but let's first try one of the examples:

⇒ File → Examples → ESP8266WiFi → WiFiScan

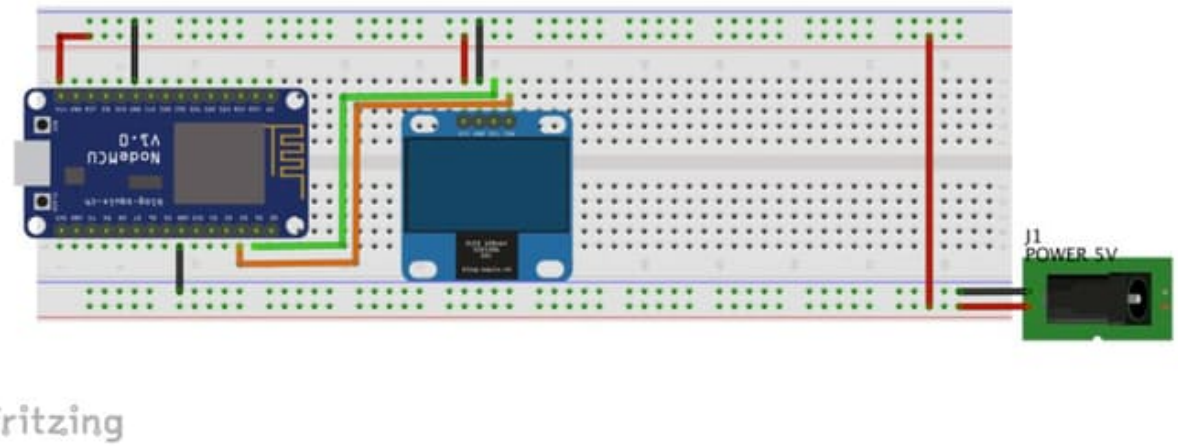
After uploading it, we can open the Serial Monitor window and observe the results.



Note that we need to match the baud rate, so check that 115200 is selected from the drop-down menu!

www.EngineeringBooksPdf.com

3.4 - Installing the OLED display



A great companion to our ESP-12E is the small OLED type display: SSD 1306. It will be very useful in our project, locally showing captured data, messages, etc. The model used is a 128 x 64-pixel display that communicates via I2C, the SSD 1306, whose main characteristics are:

- 128 pixels at horizontal and 64 pixels at vertical. So, if you use 8×8 characters, we will get a "16X8" Display (8 lines of 16 characters each).
- The SSD1306 can be powered with 5V (external) or 3.3V directly from the NodeMCU module. The first option was the chosen one for the project (5V).
- It is a I2C display, so we will connect it to the NodeMCU I2C pins, using:
    - SCL ➔ D1 (equivalent Arduino pin A5)
    - SDA ➔ D2 (equivalent Arduino pin A4)

After connecting the display, let's download and install its library on our Arduino IDE. We will use the ACROBOT library version:

https://github.com/acrobotic/Ai_Ardulib_SSD1306

www.EngineeringBooksPdf.com

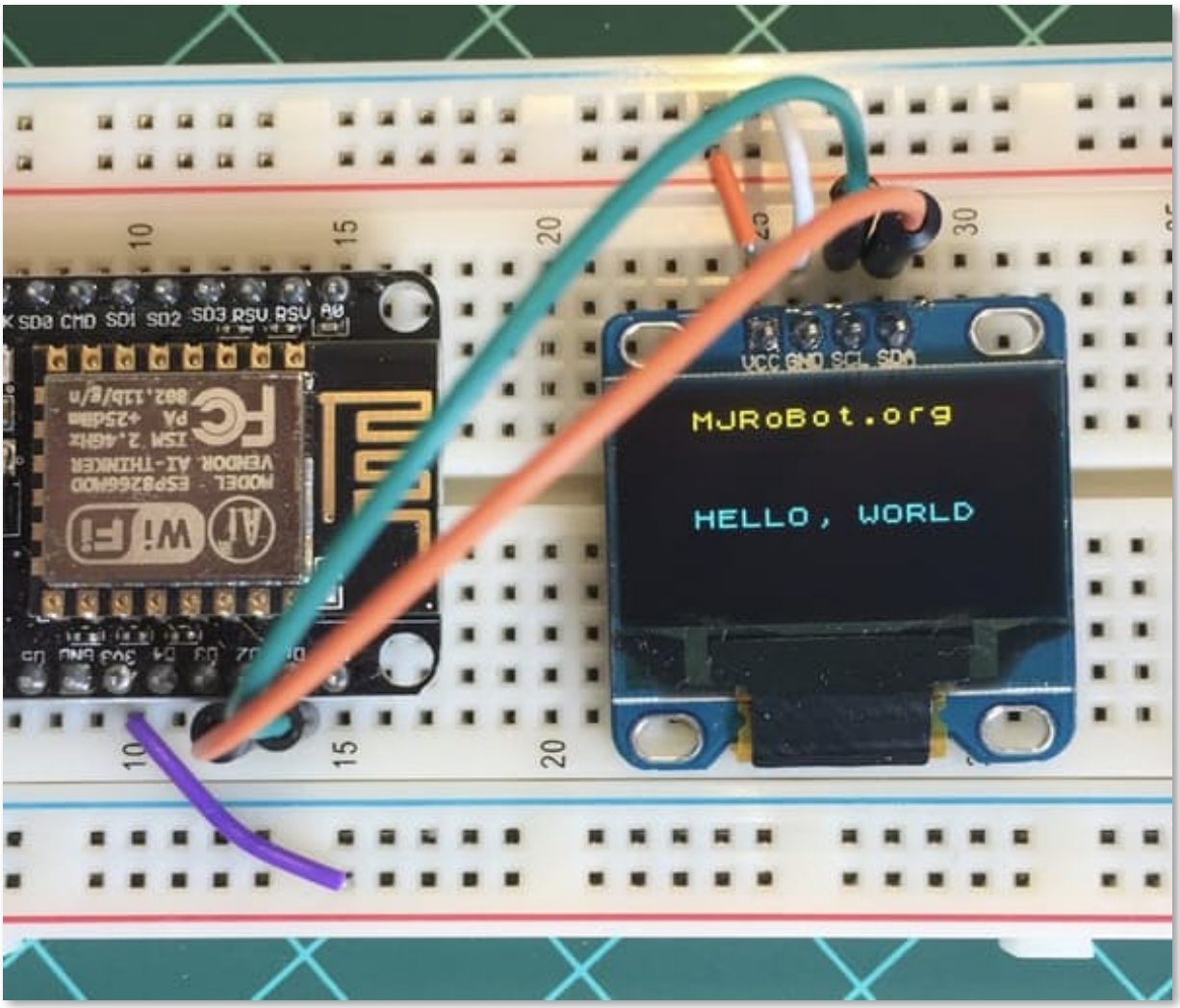
Once you have re-started the IDE, the library should be already installed.

Let's now, upload below sketch to test our OLED display:

```
/**************************************************
***************
*  NodeMCU and OLED display "Hello World" test
*
*  MJRoBot 15march17
**************************************************
***************/

#include <Wire.h>
#include <ACROBOTIC_SSD1306.h>

void setup()
{
  Wire.begin();
  oled.init();                      // Initialize SSD1306
OLED display
  oled.clearDisplay();              // Clear screen
  oled.setTextXY(0,0);              // Set cursor
position, start of line 0
  oled.putString("  MJRoBot.org");
  oled.setTextXY(4,0);              // Set cursor
position, start of line 4
  oled.putString("  HELLO, WORLD");
}

void loop()
{
}
```

Note that when you do not define a different size of text character, the default is 8X8. To define a different one, use as: oled.setFont(font5x7);

www.EngineeringBooksPdf.com

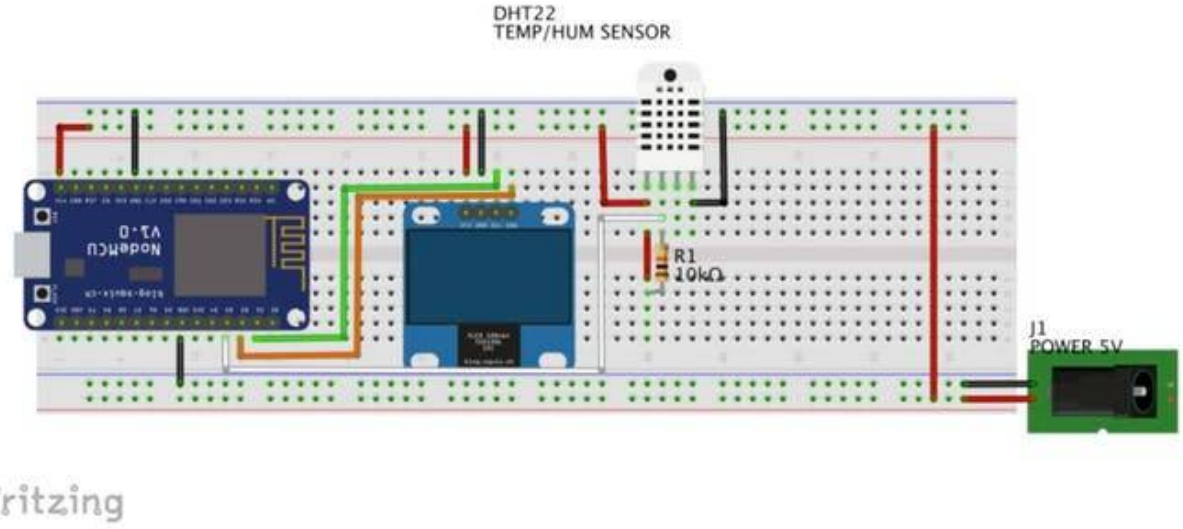


The “Hello World” test code can be downloaded from ArduFarmBot2 GitHub:

https://github.com/Mjrovai/ArduFarmBot-2/tree/master/OledTest

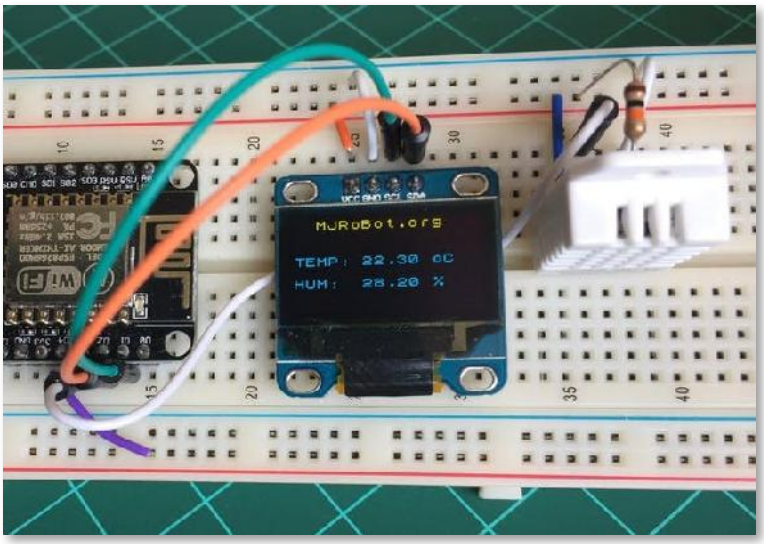
www.EngineeringBooksPdf.com

3.5 - Capturing Air Temperature and Humidity



As described in detail on iten 1.5, one of most used sensors for capturing weather data is the DHT22 (or it's brother DHT11), a digital relative humidity and temperature sensor. So, let's use it in the project.

As usually you will use the sensor on distances less than 20m, a 10K resistor should be connected between Data and VCC pins. Output pin will be connected to NodeMCU pin D3 (see the diagram above).



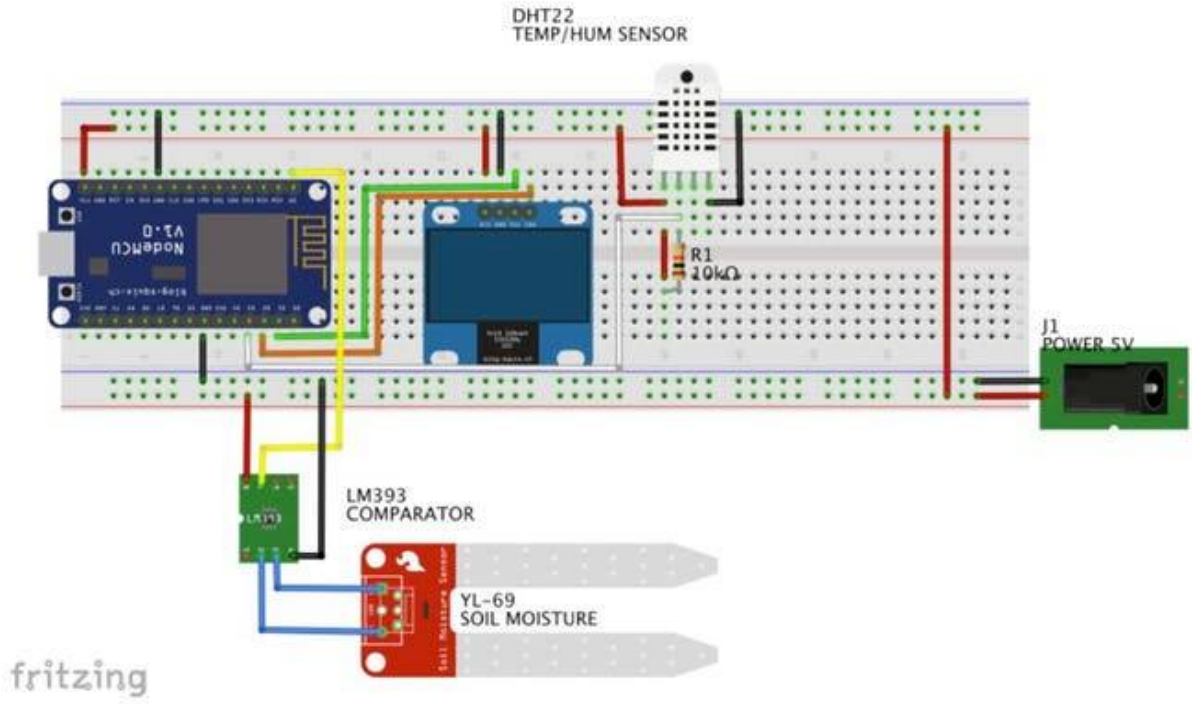
Once the sensor is installed at our module, verify if the DHT library is installed into your Arduino's Library file as explained on iten 1.5.

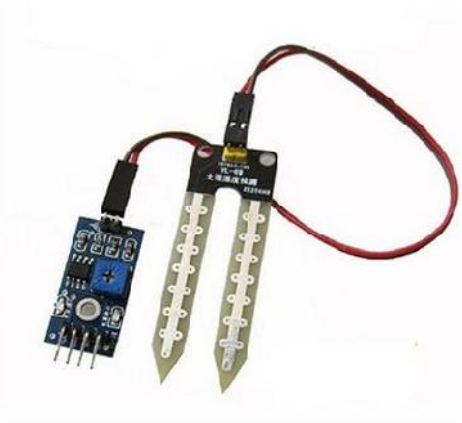
Download and run the DHT Sensor code test from ArduFarmBot 2 GitHub to verify that everything is running OK:

https://github.com/Mjrovai/ArduFarmBot-2/tree/master/DHT22_Test_with_Oled

www.EngineeringBooksPdf.com

3.6 - Capturing Soil Moisture Humidity





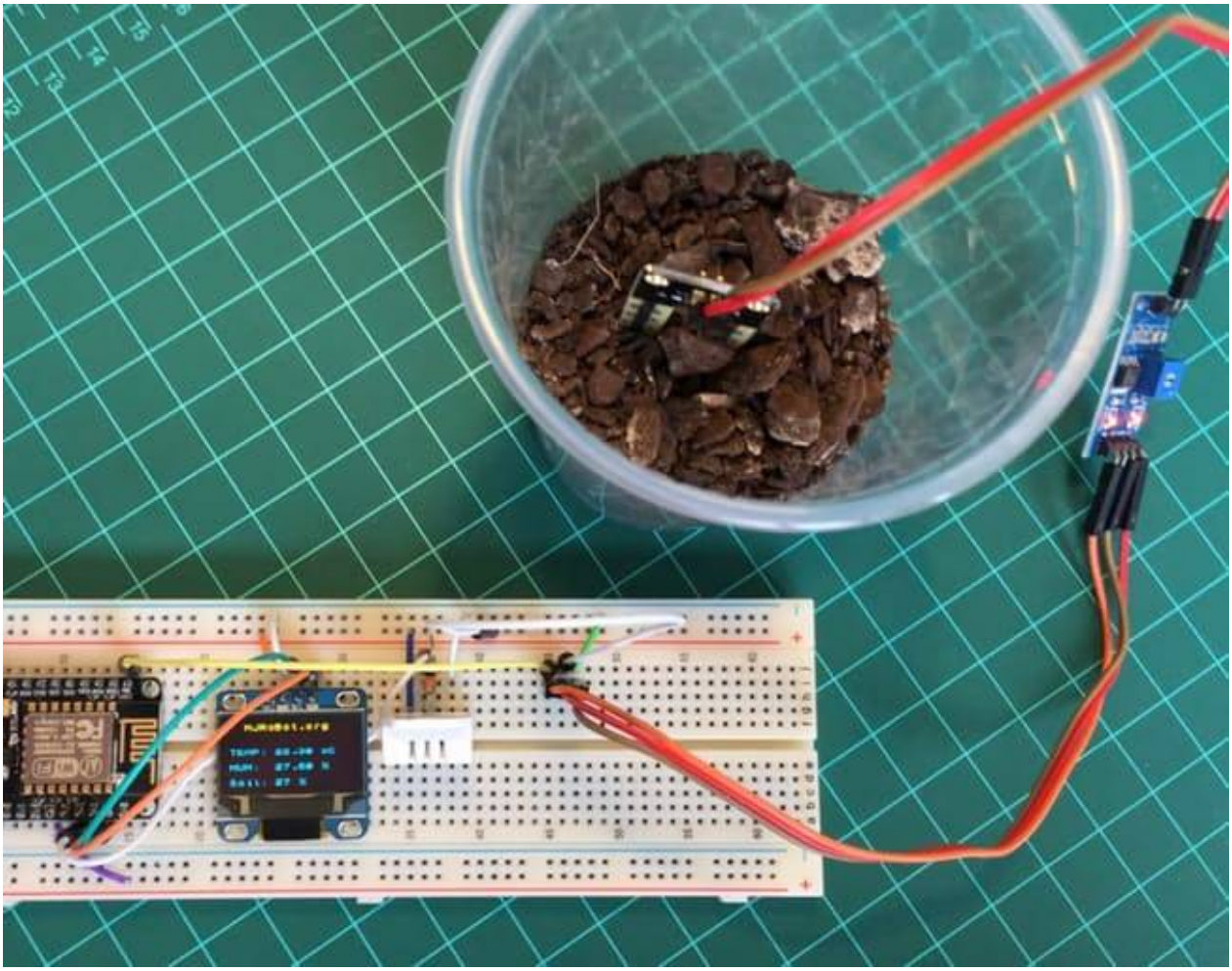
On the ArduFarmBot project Part 1, we have studied how to work with a Soil Moisture Hygrometer to measure soil humidity. There, we have explored a passive DIY type of sensor and here as an alternatively, we will use an electronic, very common in the market: the YL-69 sensor and LM393 Comparator module soil medium Hygrometer.

The LM393 module has 2 outputs, one digital (D0) that can be set-up using the potentiometer that exist on it and an analog one (A0). This module can be sourced with 3.3V, what is very convenient when working with an NodeMCU.

What we will do is to install the LM393 4 pins to NodeMCU as below:

- LM393 A0 ➔ NodeMCU A0
- LM393 VCC ➔ NodeMCU GPIO D3 or VCC (5V)
- LM393 GND ➔ NodeMCU GND
- LM393 D0 ➔ Not Connected

www.EngineeringBooksPdf.com



It is important to note that the correct procedure is to connect the Sensor VCC to one of NodeMCU Digital Pin defined as output. Doing that, the LM393 is "powered" only when it needs to perform a reading. This is important not only to save energy but also to prevent probe corrosion. Using a passive DYI home sensor as the one in the Part 1, the sensor would work without problem, but having to power one of the comparator module, the NodeMCU could have problems connecting the "soilMoisterVcc". If it is your case, alternatively, the LM393 can be connected direct to the external VCC (5V) as shown in the above electrical diagram. The code does not need to be changed, it works on both configurations.

www.EngineeringBooksPdf.com

A simple routine can be written to read the analog port:

```
/*****************************************************
 * Get Soil Moister Sensor data
 *****************************************************/
void getSoilMoisterData(void)
{
  soilMoister = 0;
  digitalWrite (soilMoisterVcc, HIGH);
  delay (500);
  int N = 3;
  for(int i = 0; i < N; i++) // read sensor "N" times
and get the average
  {
    soilMoister += analogRead(soilMoisterPin);
    delay(150);
  }
  digitalWrite (soilMoisterVcc, LOW);
  soilMoister = soilMoister/N;
  Serial.println(soilMoister);
  soilMoister = map(soilMoister, 600, 0, 0, 100);
}
```

Few comments about above routine:

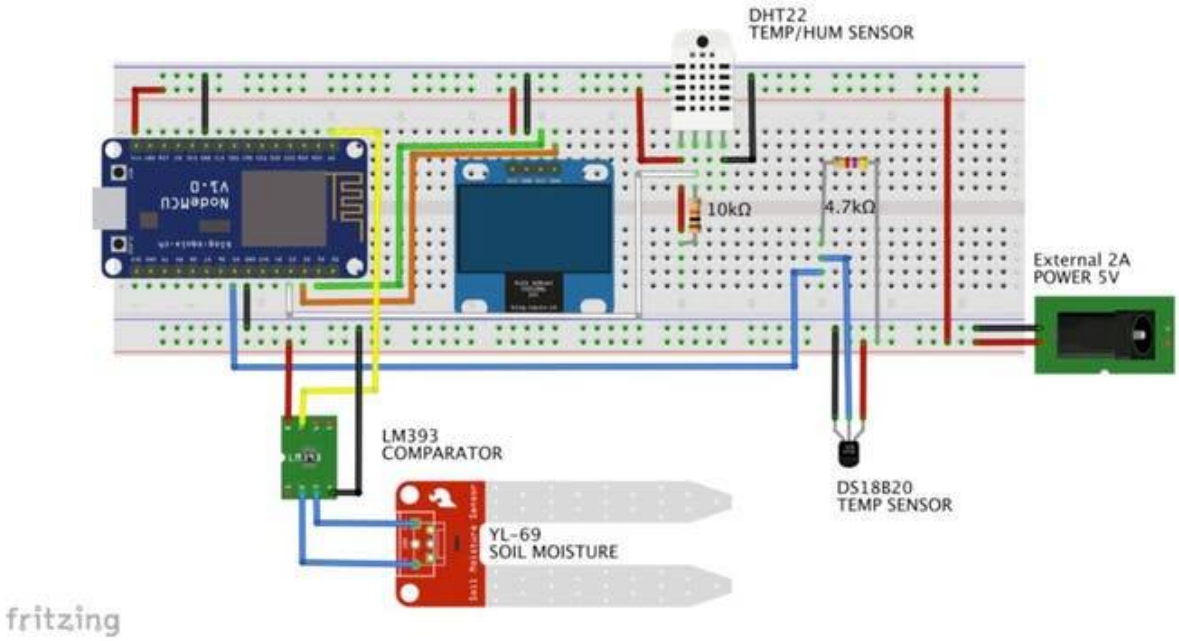
- We use MAP to setup the range in percentage.
    - Making a "short circuit" at the sensor probes (equivalent to "100% of humidity") we got a value of around 0 at ADC output and
    - Leaving it "in the air" the value displayed at Serial Monitor would be around 600 (5V Powered source).
- The sensor data is captured 3 times and an average is taken.

Below you can download a partial test code for this stage of the project:

https://github.com/Mjrovai/ArduFarmBot-2/tree/master/ArduFarmBot2_PartialTest

www.EngineeringBooksPdf.com

3.7 - Collecting Soil Temperature



On the experiments done in the field with ArduFarmBot, we verified an important relationship between humidity and soil temperature. So, we will leave here optionally, an extra sensor for soil temperature measurement, in case you want to study deeper into the subject. But it is important to note that for our project here, we will only use the air temperature, provided by DHT22.

We will use on this part of the project, a waterproof version of the DS18B20 sensor. It is very useful for remote temperature on wet conditions, as a humid soil (our case here). The sensor is isolated and can take measurements until 125ºC (Adafruit does not recommend using it over 100ºC due its cable PVC jacket).

The DS18B20 is a digital sensor which makes it good to use even over long distances! These 1-wire digital temperature sensors are relatively precise (±0.5°C over much of the range) and can give up to 12 bits of precision from the onboard digital-to-analog converter. They work great with the NodeMCU using a single digital pin, and you can even connect multiple sensors to the same pin, once each one has a unique 64-bit ID burned in factory to differentiate them.

www.EngineeringBooksPdf.com

The sensor works from 3.0 to 5.0V and has 3 wires:

- Black: GND
- Red: VCC
- Yellow: 1-Wire Data

Here, you can find the DS18B20 Datasheet:

https://cdn-shop.adafruit.com/datasheets/DS18B20.pdf

To use the DS18B20 properly, two libraries will be necessary:

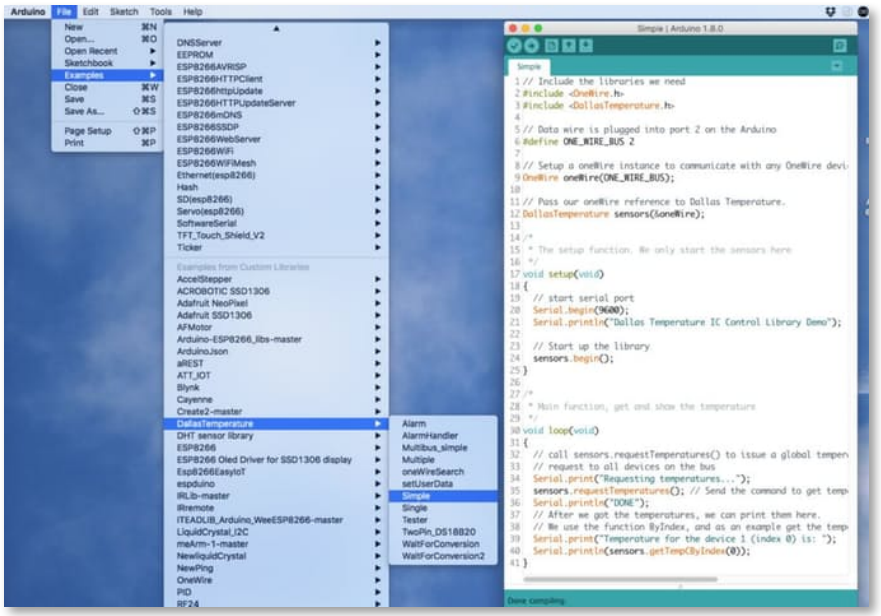
OneWire

- https://github.com/adafruit/ESP8266-Arduino/tree/esp8266/libraries/OneWire

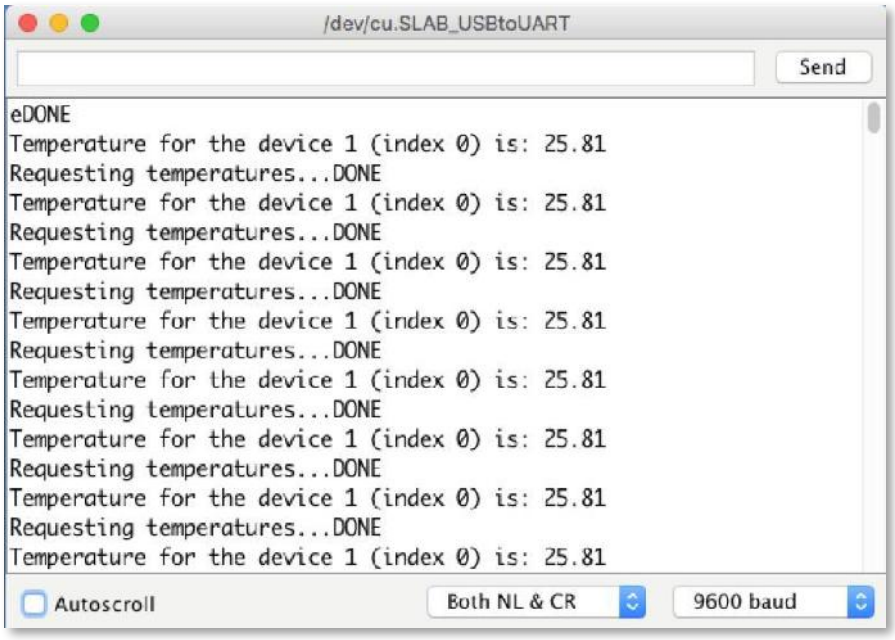
Dallas Temperature

- https://github.com/milesburton/Arduino-Temperature-Control-Library

Install both libraries in your Arduino IDE Library depository. After the IDE restarts, test the sensor using the code “Simple.ino” included on the Library Examples, as shown at the photo. Upload the code in your NodeMCU and monitor the temperature using Serial Monitor.



www.EngineeringBooksPdf.com



Picture above shows expected result. Secure sensor in your hand, temperature should shift closely to 32 / 34°C.

The OneWire library MUST be the special one, modified to be used with ESP8266, otherwise you will get an error during compilation

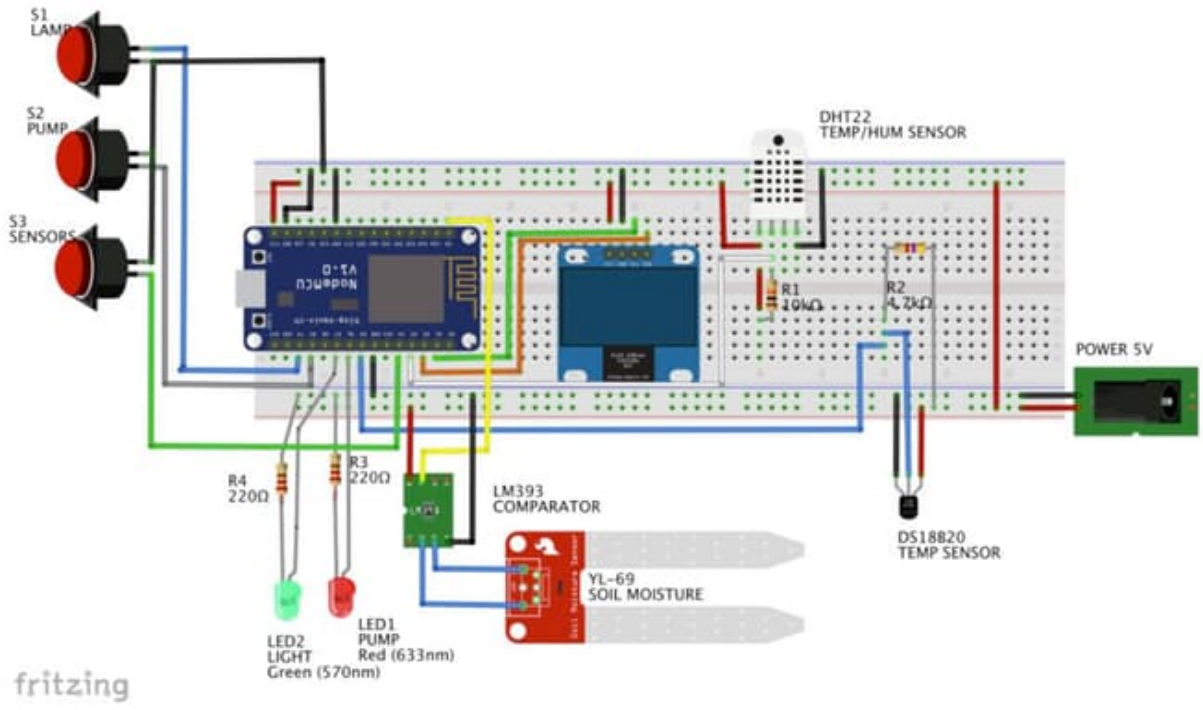
---

You will find the last version of OneWire library at Adafrut GitHub (link at previus page) or from the below ArduFarmBot2 GitHub depository:

https://github.com/Mjrovai/ArduFarmBot-2/tree/master/OneWire

www.EngineeringBooksPdf.com

3.8 - Completing the HW



Following above diagram, complete required system HW installing the buttons and LEDs.

> *The NodeMCU can become instable when its 3.3V output pins are used to powering other external devices depending on how much power is consumed. That's fine when we have only one or two external devices, but in the case of our project, several components are being used. So, for safety, connect all sensors (DHT22, DS18B20 and LM393 / YL69) and the OLED Display through the external 5V, leaving the NodeMCU only to provide control signals.*

A. LEDs

Note that LEDs connected on NodeMCU, are for testing only. They will “simulate” Pump (Red LED) and Lamp (Green LED). For final circuit, Relays will be connected to those outputs as described on the next Step.

www.EngineeringBooksPdf.com

B. Buttons

Based on sensors' readings, an operator could also decide to manually control Pump and/or Lamp. For that, three push-buttons will be incorporate to the project:

- RED: Pump Manual Ctrl
- GREEN: Lamp manual Ctrl
- YELLOW: Sensor Read (*)

(*) To update sensors, "light on" the OLED and present data (explained at next step)

Buttons will work on a "toggle" mode: If an actuator is "ON", pressing the button will "Turn-OFF" it and vice versa. The button's logic will be "normally closed", which means that NodeMCU Input will be constantly "HIGH". Pressing the button, a "LOW" will be applied at the specific pin (please see the block diagram at the end of this item).

To read the local command, a function readLocalCmd () should be executed. This function will read each button, updating the status of actuators variables (pumpStatus and lampStatus).

```
/*******************************************************
*********
* Read local commands (Pump and Lamp buttons are
normally "HIGH"):
*******************************************************
********/
void readLocalCmd()
{
  boolean digiValue = debounce(PUMP_ON_BUTTON);
  if (!digiValue)
  {
    pumpStatus = !pumpStatus;
    aplyCmd();
  }

  digiValue = debounce(LAMP_ON_BUTTON);
  if (!digiValue)
  {
    lampStatus = !lampStatus;
    aplyCmd();
  }
}
```

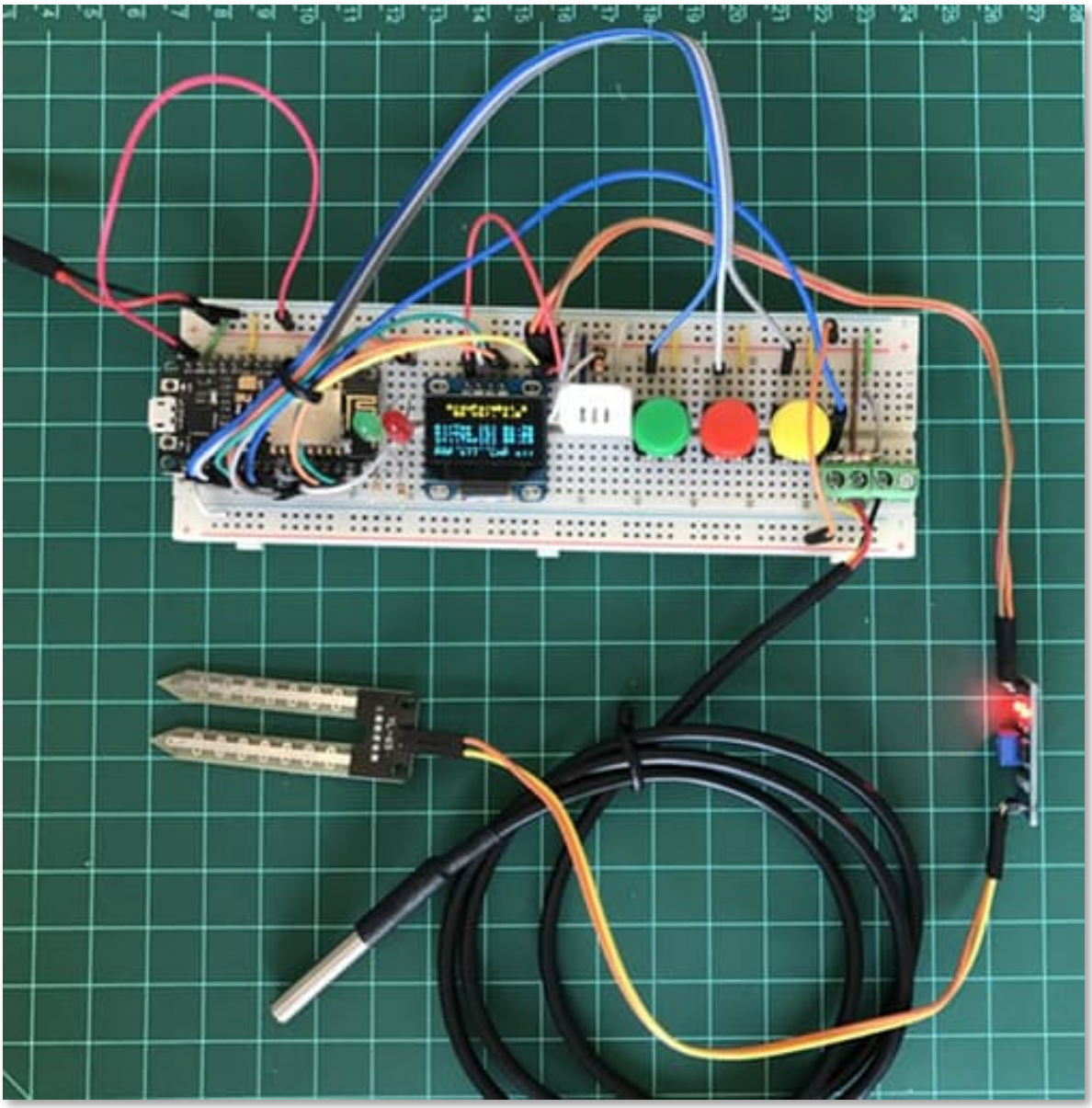
Note that the function type debounce(pin) is called instead a direct digitalRead (pin). As explained on Part 1, this is to prevent false readings from the pushbutton.

www.EngineeringBooksPdf.com

In case a button is pressed, another function will be called: aplyCmd(). And as per its name, it will apply corresponding command, turning the actuators ON or OFF:

```
/*****************************************************
* Receive Commands and act on actuators
*****************************************************/
void aplyCmd()
{
  if (pumpStatus == 1)
  {
    digitalWrite(PUMP_PIN, HIGH);
    displayData();
  }
  else
      {
        digitalWrite(PUMP_PIN, LOW);
        displayData();
      }

  if (lampStatus == 1)
  {
    digitalWrite(LAMP_PIN, HIGH);
    displayData();
  }
  else
      {
        digitalWrite(LAMP_PIN, LOW);
        displayData();
      }
}
```


www.EngineeringBooksPdf.com



C. Code considerations:

When we think about the 4 big “group of tasks” so far:

- Read sensors
- Read buttons (local Command)
- Act on Pump/Lamp
- Display all Data

www.EngineeringBooksPdf.com

We will realize that the timing when such tasks must be executed are not necessarily the same. For example, to read the Temperature and Humidity data from DHT 22, we will need to wait at least 2 seconds between measurements, even minutes are OK. For Soil Moisture sensor, the less measurements we do, the better (due probe corrosion generate by electrolise). But when we think about the actuators, as soon as we press a button, we would like (and possibly need) a quick reaction.

So, we must use here a "timer" to correctly control timing of those tasks. We could do this using the millis(), as we did on the Part 1 and 2, but let's use the opportunity to introduce another great tool here: SimpleTimer.h

---

To install the Library, follow the instructions on this link:

http://playground.arduino.cc/Code/SimpleTimer

The library must be included on the main body of your code, following by a timer definition:

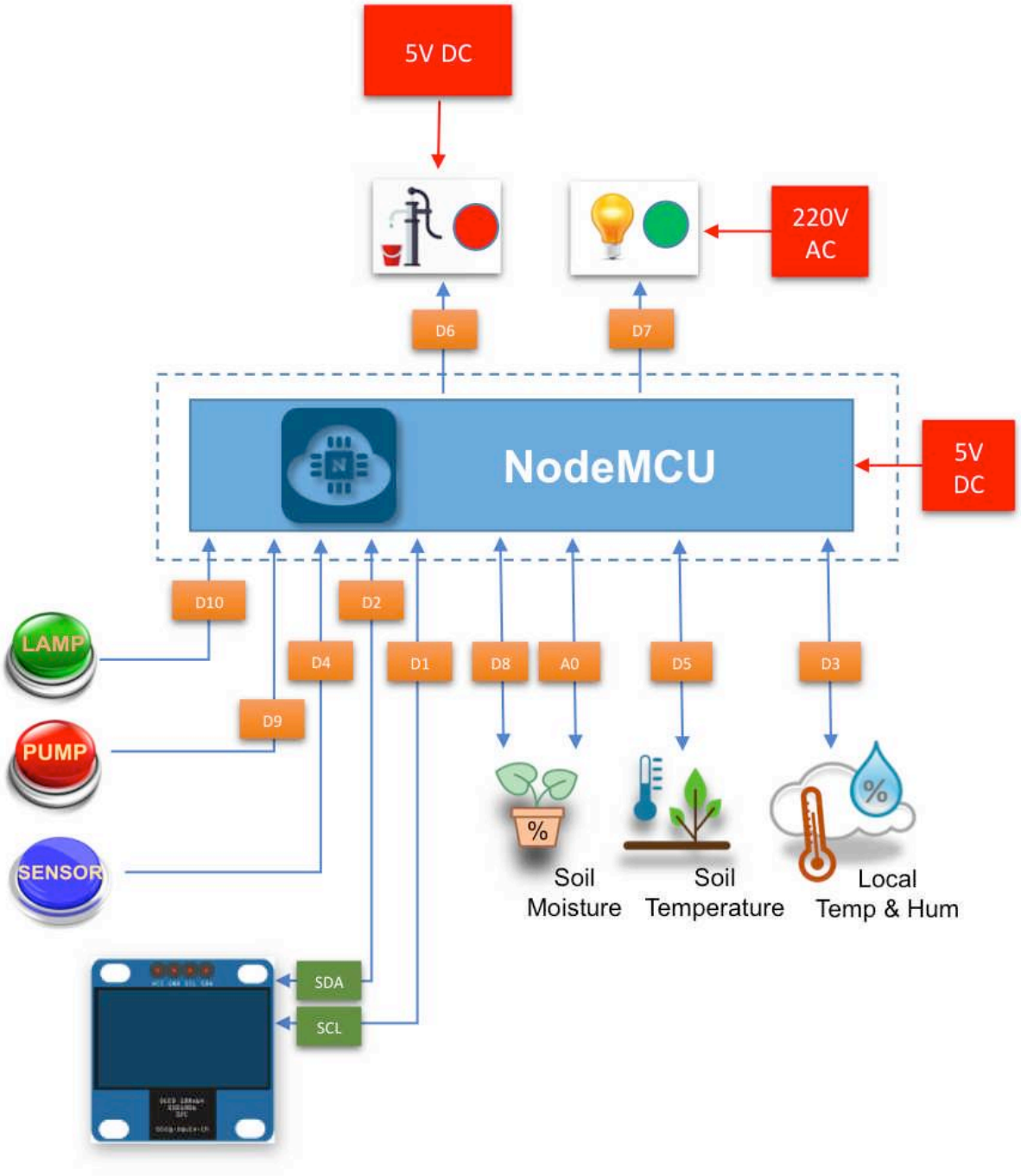
```
SimpleTimer timer;
```

Next, you should define the timers:

```
timer.setInterval(1000L, readLocalCmd);  // Read buttons at every 1s
timer.setInterval(2000L, getSoilTempData); // Read Soil Temp at every 2s
timer.setInterval(2000L, getDhtData);     // Read DHT Sensor at every 2s
timer.setInterval(10000, getSoilMoisterData); // Read Soil Hum at every 10s
timer.setInterval(10000, displayData); // Display Data at OLED at every 10s
```

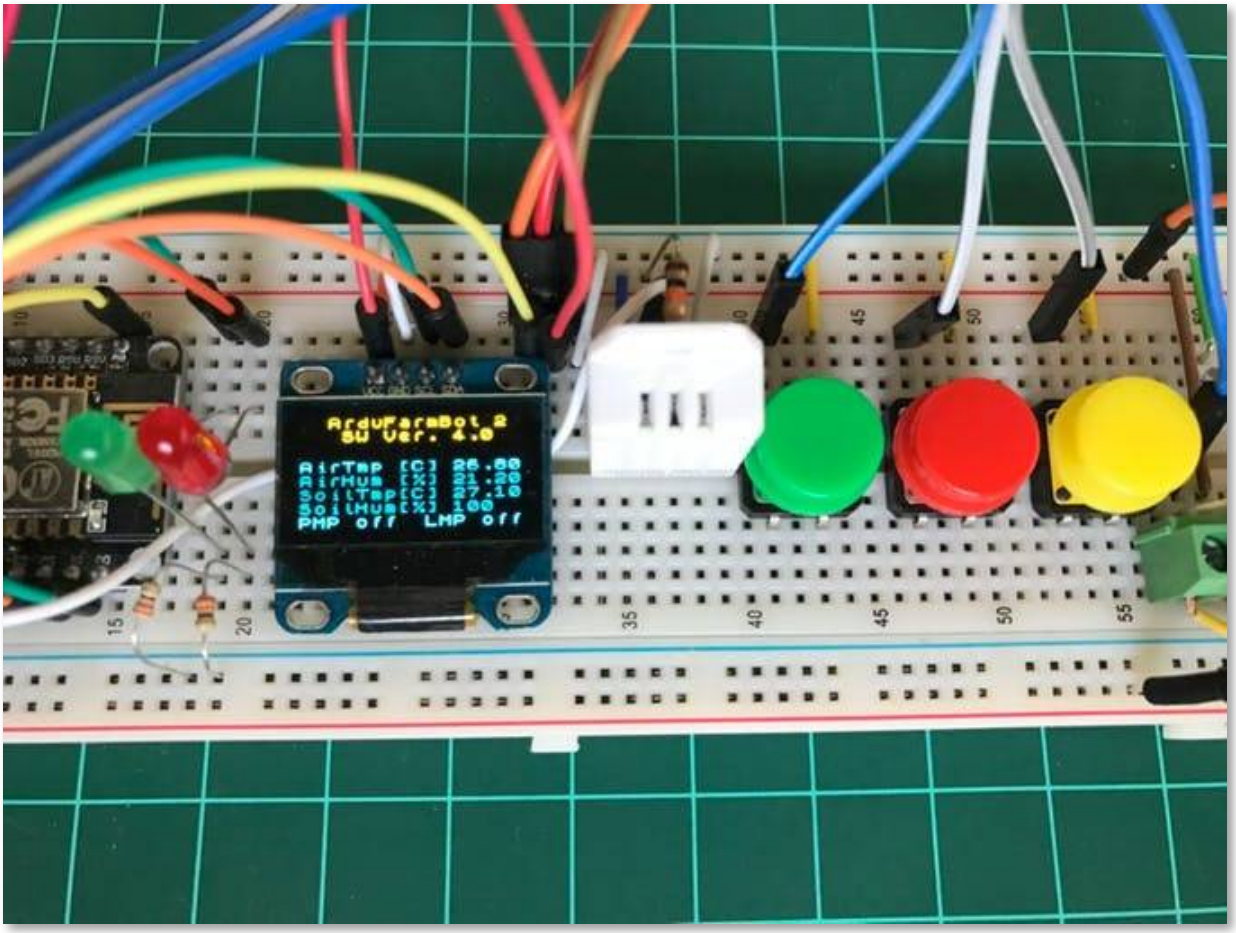

www.EngineeringBooksPdf.com

Below blocks diagram, shows NodeMCU pins and its connections with external devices:



www.EngineeringBooksPdf.com

3.9 - Local Control Station – Concluding the Code



Sensor Read Button

As we could see at previous step, we will need to wait long cycles between Soil Moisture sensor measurements. It is Ok, for our automatic needs, but for manual operation we will not want to “wait” 10, 15 or more seconds (or even minutes in real case). Also in real world, it makes no sense to keep the OLED display “ON” all the time (its default should be “dark” or “OFF”).

Therefore, we will introduce a 3rd push button to our project that will display the actual sensor data at any time, independent of automatic readings predefined timing. We will use this same button to display data on the OLED when sensors are updated.

www.EngineeringBooksPdf.com

Below changed readLocaCmd() function:

```
/*********************************************************
******************
* Read local commands (Pump, Lamp and Sensor buttons are
normally "HIGH"):
*********************************************************
*****************/
void readLocalCmd()
{
  boolean digiValue = debounce(PUMP_ON_BUTTON);
  if (!digiValue)
  {
    pumpStatus = !pumpStatus;
    aplyCmd();
  }

  digiValue = debounce(LAMP_ON_BUTTON);
  if (!digiValue)
  {
    lampStatus = !lampStatus;
    aplyCmd();
  }

  digiValue = debounce(SENSORS_READ_BUTTON);
  if (!digiValue)
  {
    turnOffOLED = !turnOffOLED;
    if (!turnOffOLED)
    {
      oled.setTextXY(0,0); oled.putString("UPDATING
SENSORS");
      getDhtData();
      getSoilMoisterData();
      getSoilTempData();
      oledStart();
      displayData();
    }else oled.clearDisplay(); //turn off OLED
  }
}
```


www.EngineeringBooksPdf.com

At this point, all HW are completed (using LEDs as actuators) and we can only have SW parts to be put together.

You can download complete code for testing your “local Station” on its Manual mode only, from the ArduFarmBot 2 file depository:

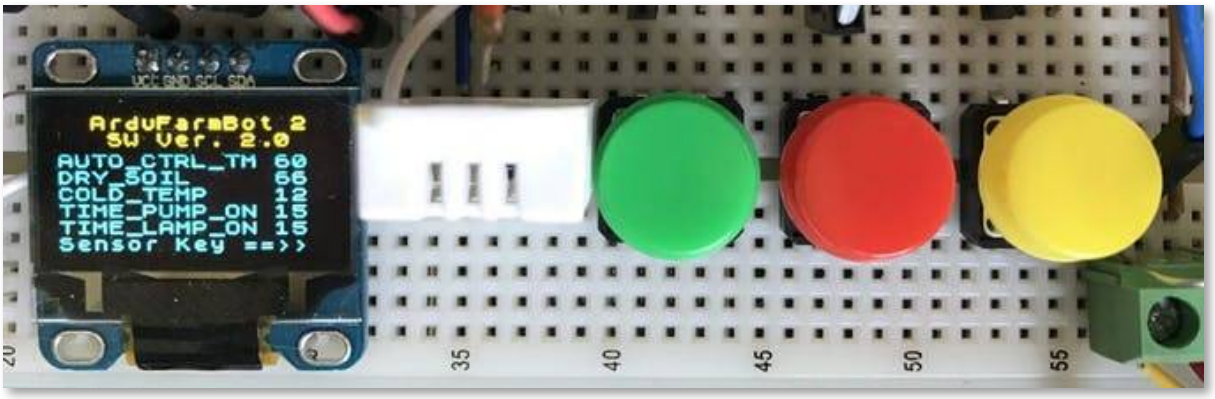
https://github.com/Mjrovai/ArduFarmBot-2/tree/master/ArduFarmBot2_Local_Manual_Ctrl_V1

Following video shows the ArduFarmBot 2, operating on a Local and Manual mode:

https://youtu.be/xJ9sSJw54PM

www.EngineeringBooksPdf.com

3.10 - Making our Gardening System fully automatic



At this point all HW are in place and as we saw previously, the station can be controlled by a local operator via buttons. What is missing is the "logic" allowing our system to really perform the task of irrigating the garden automatically! We need to include some "brain" to our ArduFarmBot project.

To begin, let's define the initial interval where sensors must work. Those values should be changed later using practical values obtained from a real garden:

Soil Moisture:

- "WET": Over 88% (no watering at all)
- "Target Humid": Between 66% and 88% (Where we want to work at) and
- "DRY": Below 66% (need to turn on the pump in order to increase humidity)

Air Temperature:

- COLD: Below 12oC (Turn-On the Light/Heat)
- Optimum: between 12oC and 22oC
- HOT: Over 22oC (Do not Turn-On the Light/Heat)

Having these 4 readings (Air Temperature, Air Humidity, Soil Moisture and Soil Temperature), same we did previously on iten 2.7, we can build a complex matrix defining how we want that our automatic Gardening System works.

www.EngineeringBooksPdf.com

So, let's define parameters to be used on our code:

```
/* Automatic Control Parameters Definition */
#define DRY_SOIL      66
#define WET_SOIL      85
#define COLD_TEMP     12
#define HOT_TEMP      22
#define TIME_PUMP_ON  15
#define TIME_LAMP_ON  15
```

TIME_PUMP_ON and TIME_LAMP_ON are the time in seconds that both pump and lamp must be ON during automatic operation.

Based on above parameters, let's think about some very simple assumptions to be implemented on the code:

- If it's DRY ➔ PUMP = ON
- If it's COLD ➔ LAMP = ON

In this part of the project, we will keep it simple and will not explore all possible combinations and the role of Air humidity or soil temperature on the equation. But feel free to update and implement the matrix developed on item 2,7. We will leave it as a homework.

---

www.EngineeringBooksPdf.com

The Code:

Let's create a new function that based on sensors reading, will deal automatically with actuators, turning on/off Pump and Lamp: autoControlGarden().

This function will be called on every Cycle of Sensors readings:

```
/*********************************************************
*******
* Automatically Control the Plantation based on sensors
reading
*********************************************************
******/
void autoControlPlantation(void)
{
  if (soilMoister < DRY_SOIL)
  {
    turnPumpOn();
  }

  if (airTemp < COLD_TEMP)
  {
    turnLampOn();
  }
}
```

The function will have 2 main tasks:

- Pump Control
- Lamp Control

```
*****************************************************
* Turn Pump On for a certain amount of time
*****************************************************/
void turnPumpOn()
{
  pumpStatus = 1;
  aplyCmd();
  delay (TIME_PUMP_ON*1000);
  pumpStatus = 0;
  aplyCmd();
}
```


www.EngineeringBooksPdf.com

```
/*************************************************
* Turn Lamp On for a certain amount of time
*************************************************/
void turnLampOn()
{
  lampStatus = 1;
  aplyCmd();
  delay (TIME_LAMP_ON*1000);
  lampStatus = 0;
  aplyCmd();
}
```

Finally, let's use the Sensor Read button ("yellow one") to not only pause the program for a certain time during the start-up but also to display the most important initial parameters, as shown at previous photo.

At this point the ArduFarmBot is fully functional in terms of HW and SW.

The ArduFarmBot2 code in its version of "Local and Automatic control" can be downloaded from the GitHub file depository:

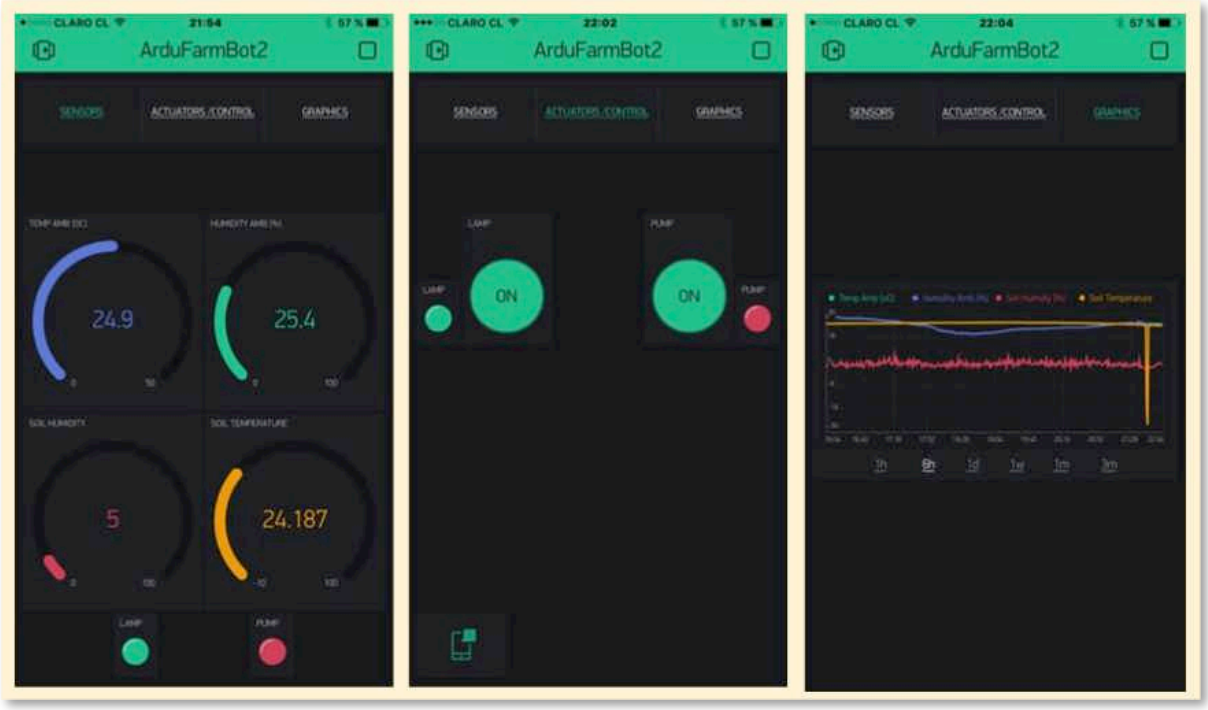
https://github.com/Mjrovai/ArduFarmBot-2/tree/master/ArduFarmBot2_Local_Automatic_Ctrl_V2

Following video shows the ArduFarmBot 2, operation on Local and Automatic modes:

https://youtu.be/sE_Lwgdjwbw

www.EngineeringBooksPdf.com

3.11 - Building an App BLYNK



It is really very easy to build IoT projects using BLYNK. The first thing you need is to have the BLINK App installed on your phone and its Library on the Arduino IDE.

If you do not have them yet, please follow these steps:

1. Download BLYNK app for Apple Iphone or Google Android
2. Install BLYNK Library for Arduino:
   https://github.com/blynkkk/blynk-library/releases/tag/v0.3.10

Note that you will download a zip file (There are 5 files there that you must manually install in your Arduino Library).

Once the Arduino IDE is reloaded, you should be OK to start using BLINK on your IoT project.

For more information about BLYNK, visit: http://www.blynk.cc/

---

www.EngineeringBooksPdf.com

Now, let's go to our app at the SmartPhone and follow the steps:

⇒ Open Blynk app.

⇒ Tap on "Create New Project" screen

⇒ Give a name for your project (For example "ArduFarmBot 2")

⇒ Select an appropriated Hardware Model: "NodeMCU"

⇒ Take note from Authorization Token (you can e-mail it to yourself to ease copy&past on your code)

```
char auth[] = "YourAuthToken";
```

⇒ Press "OK". A Blank screen with dots will appear.

⇒ Tap the Screen to open the "Widget Box"

Let's take a moment and think about our ArduFarmBot 2 Blynk App and define which Widgets will be installed. Revisiting general specification at introduction, we can summarize that our app will be needed for:

- Read all Sensors and verify actuators status
- Take remote actions, "turning on/off" Pump and Lamp
- Sending messages when System is "off-line" and/or an actuator is ON
- Record general sensors data

To organize things, let's split above "tasks" in 3 tabs:

- SENSORS
- ACTUATORS / CONTROL
- GRAPHICS

"Tabs" will be the first Widget to be installed. Enter on it and define previous "Tab names"

www.EngineeringBooksPdf.com

Next, go to each Tab and install Widgets as described below:

SENSORS

- Gauge: “Temp Air [°C]” Blue; input: V10 0 to 50; frequency: 5 sec
- Gauge: “Humidity Air [%]” Green; input: V11 0 to 100; frequency: 5 sec
- Gauge: “Soil Humidity [%]” Red; input: V12 0 to 100; frequency: 5 sec
- Gauge: “Soil Temperature[°C]” Yellow; input: V13 -10 to 50; frequency: 5 sec
- LED: “PUMP” Red; V0
- LED: “LAMP” Green; V1

ACTUATORS / CONTROL

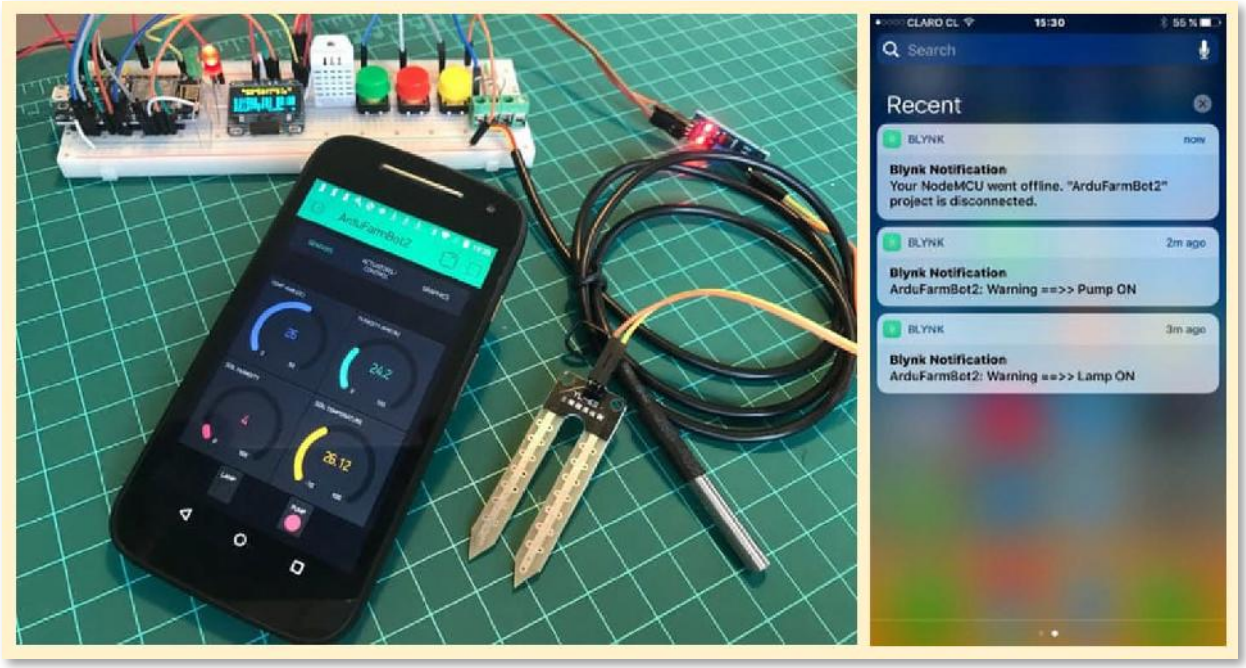
- Button: “PUMP” Red; output: V3 0 to 1; mode: Switch; label: on: ACT, off: OK
- Button: “LAMP” Green; output: V4 0 to 1; mode: Switch; label: on: ACT, off: OK
- LED: “PUMP” Red; V0
- LED: “LAMP” Green; V6
- Notifications: Notify When HW goes offline: ON

GRAPHICS

- Data to Show:
    - V10 “Temp Air”
    - V11 “Humidity Air”
    - V12 “Soil Humidity “
    - V13 “Soil Temp”

www.EngineeringBooksPdf.com

3.12 - Changing code to introduce Blynk



To run a Blynk app together with your code, you will need:

- Including BlynkSimpleEsp8266 library at beginning of your code
- During setup(), initiate Blynk credentials:
  - Blynk.begin(auth, ssid, pass);
- Define a timing to send local data to Blynk server:
  - timer.setInterval(5000L, sendUptime);
- Call the function Blynk.run(); at loop()
- Create the function sendUtime(); where you will introduce sensor data to be sent to Blynk Server:
  - Blynk.virtualWrite(VirtualPin, sensor data);

www.EngineeringBooksPdf.com

The program must include now the file StationCredentials.h:

```
char auth[] = "YOUR PROJECT TOKEN"; // Blynk project: "ArduFarmBot2"
char ssid[] = "YOUR LOCAL WIFI NAME";
char pass[] = "YOUR WIFI PASSWORD";
```

Other considerations:

To use the "Virtual LED" at Blynk app, you must define them at beginning of your code as below:

```
WidgetLED PUMPs(V0);  // Echo signal to Sensors Tab at Blynk App
WidgetLED PUMPa(V5); // Echo signal to Actuators Tab at Blynk App
WidgetLED LAMPs(V1);  // Echo signal to Sensors Tab at Blynk App
WidgetLED LAMPa(V6); // Echo signal to Actuators Tab at Blynk App
```

To "turn on" or "turn off" the PUMPs LED that relates to virtual PIN V0, for example, call functions respectively:

- PUMPs.on() or
- PUMPs.off()

We will include commands at applyCmd() function, so LEDs on Blynk app will mimic real LEDs of our project.

For notifications, we should also include the command:

```
Blynk.notify ("Mensagem a ser enviada");
```

on the same applyCmd() function, one for the Pump and another for the Lamp.

www.EngineeringBooksPdf.com

Below, the new function:

```
/*****************************************************
* Receive Commands and act on actuators
*****************************************************/
void aplyCmd()
{
  if (pumpStatus == 1)
  {
    Blynk.notify("ArduFarmBot2: Warning ==>> Pump ON");
    digitalWrite(PUMP_PIN, HIGH);
    if (!turnOffOLED) displayData();
    PUMPs.on();
    PUMPa.on();
  }
  else
      {
        digitalWrite(PUMP_PIN, LOW);
        if (!turnOffOLED) displayData();
        PUMPs.off();
        PUMPa.off();
      }

  if (lampStatus == 1)
  {
    Blynk.notify("ArduFarmBot2: Warning ==>> Lamp ON");
    digitalWrite(LAMP_PIN, HIGH);
    if (!turnOffOLED) displayData();
    LAMPs.on();
    LAMPa.on();
  }
  else
      {
        digitalWrite(LAMP_PIN, LOW);
        if (!turnOffOLED) displayData();
        LAMPs.off();
        LAMPa.off();
      }
}
```


www.EngineeringBooksPdf.com

To receive a command from a Blynk button, a function BLYNK_WRITE() must be defined outside a function, loop() or setup(). For that, a code was created, one for each Blynk Button (PUMP and LAMP):

```
/*****************************************************
*********
* Read remote commands
*****************************************************
********/
BLYNK_WRITE(3) // Pump remote control
{
  int i=param.asInt();
  if (i==1)
  {
    pumpStatus = !pumpStatus;
    aplyCmd();
  }
}

BLYNK_WRITE(4) // Lamp remote control
{
  int i=param.asInt();
  if (i==1)
  {
    lampStatus = !lampStatus;
    aplyCmd();
  }
}
```

Below, video shows the automatic operation of ArduFarmBot 2, now including Blynk:

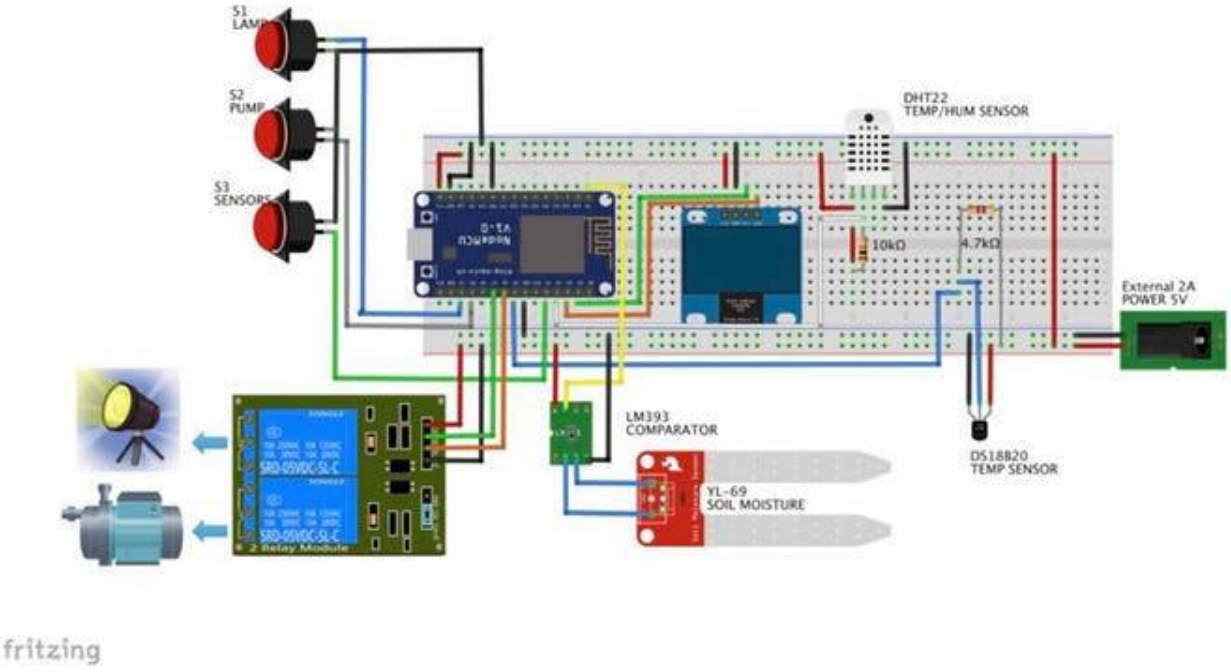
https://youtu.be/Y9xoh-iykzg

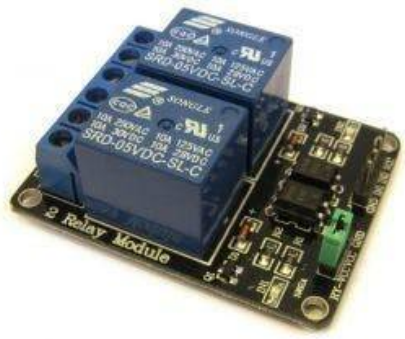
The ArduFarmBot2 code in its version of “Remote and Automatic control”, using Blynk can be download from the project GitHub file repository:

https://github.com/Mjrovai/ArduFarmBot-2/tree/master/ArduFarmBot2_Ext_Auto_Ctrl__V3_0

www.EngineeringBooksPdf.com

3.13 – Relays as actuators





As discussed at "introduction", our final goal here is to take care of a garden. With data provided by sensors, we will know the air and soil temperature, air relative humidity and the most important how "dry" is the soil. With those data on hand, our program should calculate if it would be necessary to irrigate the garden, turning on a water pump or to turn on an electric lamp to provide the appropriate heat to the crop.

For that, we will use a small dual 5V Relay Module for Pump and Lamp activation.

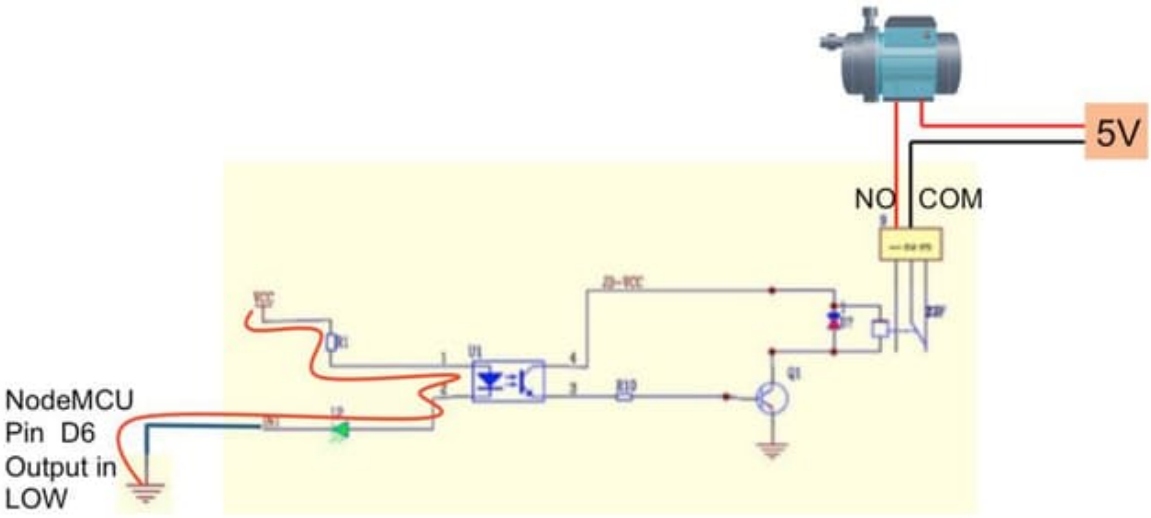
Usually you will see as output, 3 Pins for each relay: "NO" ("*Normal Open*"), "Ref" or "COM" ("*Reference" or "Common*") and "NC" ("*Normal Closed*").

We will use the NO and COM for each Relay.

Depending of your Relay Module, labels could be different

---

www.EngineeringBooksPdf.com



Looking at the diagram, you should connect:

- Power Supply 5V ➔ (4) “Vcc”
- NodeMCU D6 ➔ (3) “IN1” (Pump)
- NodeMCU D7 ➔(2) “IN2” (Lamp)
- NodeMCU GND ➔ (1) “GND”

On the above example, the “COM” is the terminal to connect to external Positive Pin of the 5V Power Supply (in the case of the Pump) or the 220VAC for the Lamp. The “NO” will be connected to Pump (or Lamp).

In the case of the Relay chosen and confirming on the above diagram, normally the IN1 and IN2 must be at HIGH and its activation will happen with a LOW level (less than 2V). With a LOW level from NodeMCU, the current will flow from VCC to NodeMCU Pin D6, activating the optocoupler input. The Relay output will close and the NO will close, turning ON the Pump on the example.

Regarding the V3.0 version of the code developed in the previous step, we must "reverse" the logic of the actuators (or digital pins as output). The NodeMCU pins D6 and D7 should normally be HIGH. So, the setup () function should be changed:

```
digitalWrite(PUMP_PIN, HIGH); // Relay module (inverted logic: normally HIGH)
digitalWrite(LAMP_PIN, HIGH); // Relay module (inverted logic: normally HIGH)
```


www.EngineeringBooksPdf.com

And also invert the conditions at *applyCmd()* function:

```
*************************************************
* Receive Commands and act on actuators
*************************************************/
void aplyCmd()
{
  if (pumpStatus == 1)
  {
    Blynk.notify("ArduFarmBot2: Warning ==>> Pump ON");
    digitalWrite(PUMP_PIN, LOW); // Relay module (inverted logic: activate with LOW)
    if (!turnOffOLED) displayData();
    PUMPs.on();
    PUMPa.on();
  }
  else
      {
        digitalWrite(PUMP_PIN, HIGH); // Relay module (inverted logic: normally
HIGH)
        if (!turnOffOLED) displayData();
        PUMPs.off();
        PUMPa.off();
      }

  if (lampStatus == 1)
  {
    Blynk.notify("ArduFarmBot2: Warning ==>> Lamp ON");
    digitalWrite(LAMP_PIN, LOW); // Relay module (inverted logic: activate with LOW)
    if (!turnOffOLED) displayData();
    LAMPs.on();
    LAMPa.on();
  }
  else
      {
        digitalWrite(LAMP_PIN, HIGH); // Relay module (inverted logic: normally
HIGH)
        if (!turnOffOLED) displayData();
        LAMPs.off();
        LAMPa.off();
      }
}
```

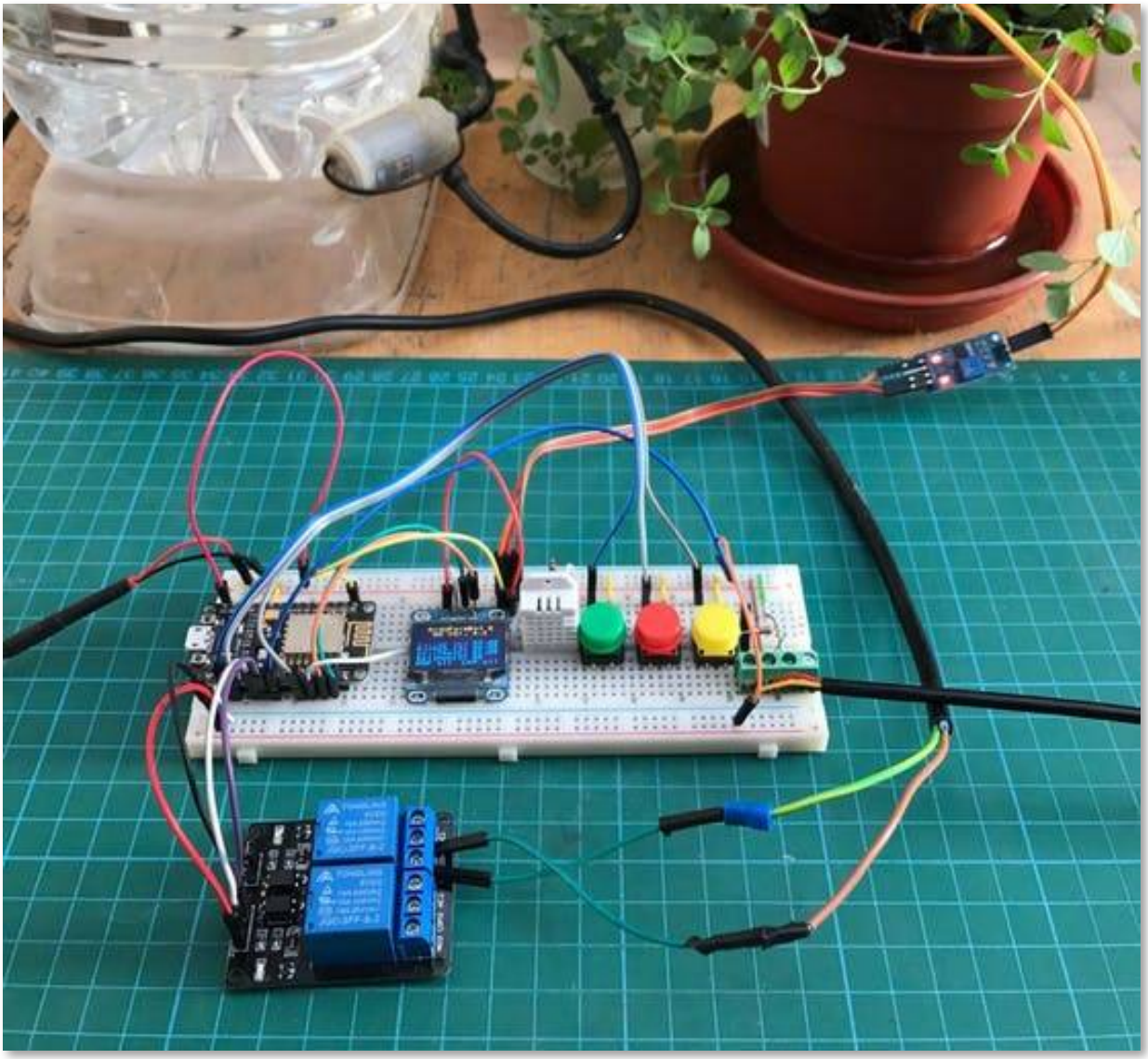

www.EngineeringBooksPdf.com

The ArduFarmBot2 code in its version of “Remote and Automatic control”, using Blynk and real Relay (activation LOW) can be download from project file repository:

https://github.com/Mjrovai/ArduFarmBot-2/tree/master/ArduFarmBot2_Ext_Auto_Ctrl__V4_0

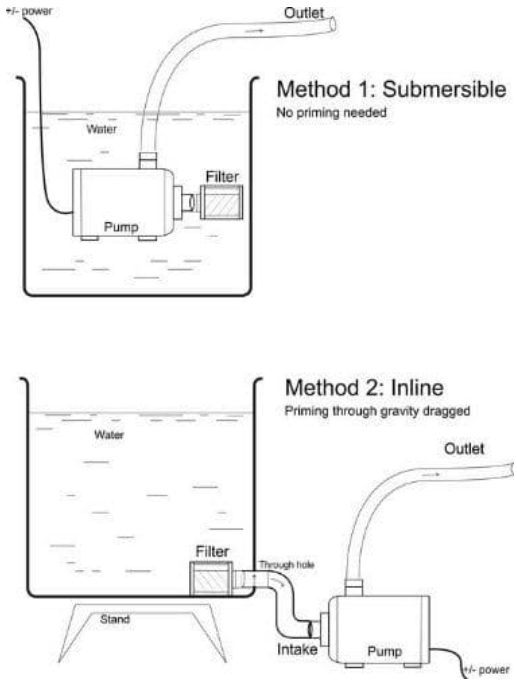
www.EngineeringBooksPdf.com

3.14 - ArduFarmBot 2 real test



The pump chosen is a mini 5V DC water pump (it works "drowned").

www.EngineeringBooksPdf.com



You can install it submerse in water or “inline”. We have used the second one.

Afterwards, connect one of the Pump’s wire to Relay IN1 and the other one to the External 5V Power Supply Pin (+). Take the Power Supply Pin (-) and connect it to the Relay COM1.

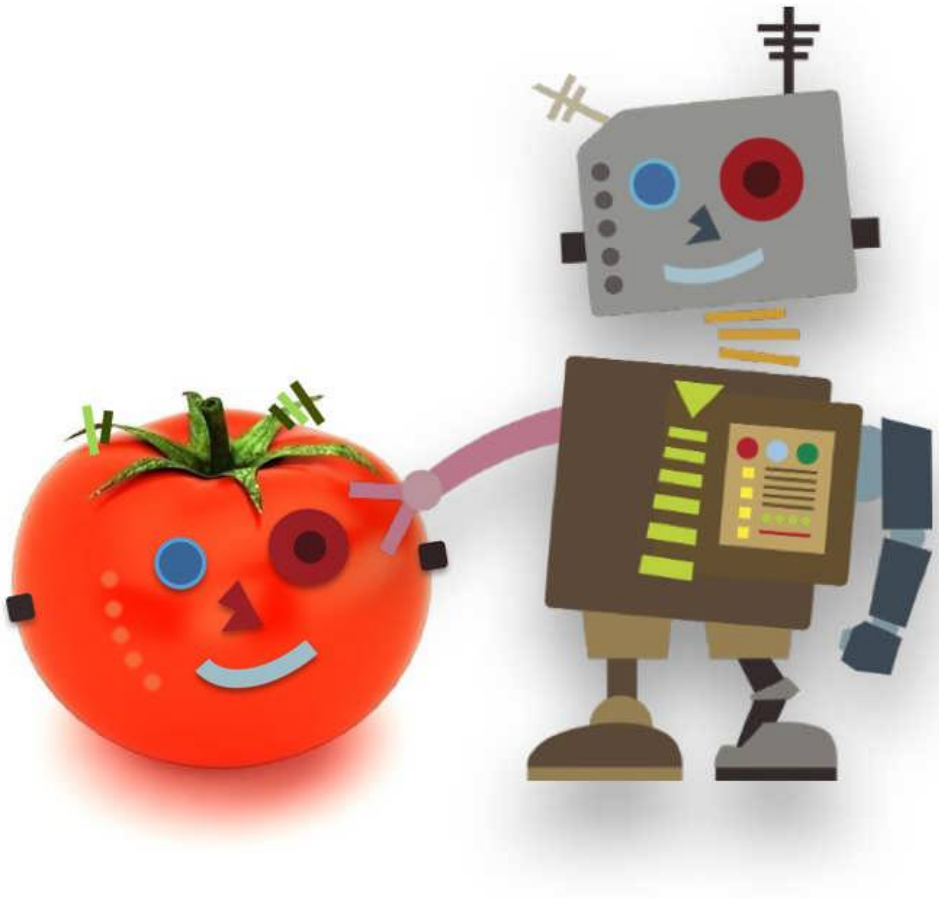
Movie below shows a manual local and remote operation of the ArduFarmBot 2:

https://youtu.be/cgTGbSn-g-0

And, finally the below video shows its automatic operation:

https://youtu.be/ZRoxKFEuGuQ

www.EngineeringBooksPdf.com



www.EngineeringBooksPdf.com

Conclusion

The last part of our project is still to be written and will certainly be a good Italian pasta, but with organic tomato sauce!

To get the updated files, always check the ArduFarmBot file repositories:

Part 1 e 2: https://github.com/Mjrovai/ArduFarmBot

Part 3: https://github.com/Mjrovai/ArduFarmBot-2

By the way, in the photo below you can see the first signs of life in Mauricio's garden and his tomatoes a few months later:

We hope that this project can help others find their way in the exciting world of electronics and IoT!

Regards from "the South of the World"!

Marcelo e Mauricio

Santiago de Chile, February 2017

www.EngineeringBooksPdf.com

Please, visit the Blog https://MJRoBot.org

And give a “Like” at Facebook page: https://www.facebook.com/mjrobot.org/

www.EngineeringBooksPdf.com