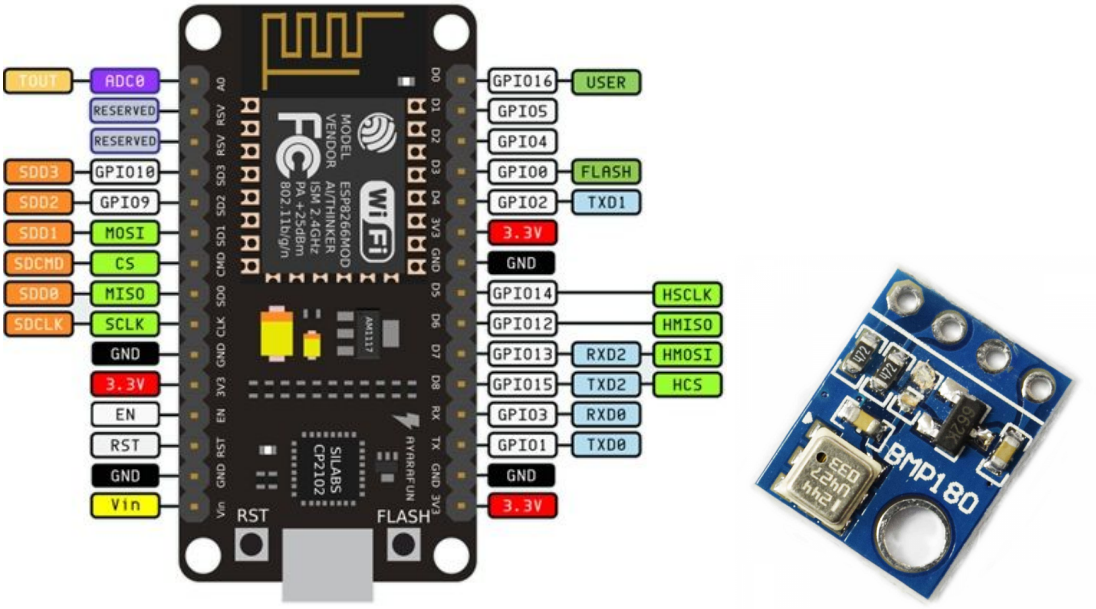
Wireless Leak Test Setup using NodeMCU

Suryanarayan Mondal

December 16, 2018

Abstract

The main goal is to build a wireless setup with which multiple RPC gaps can be tested for leak and button pop-up independently and in parallel.



(a) NodeMCU (b) BMP180 Breakout Board

Figure 1: Required Electronics

Installing Arduino IDE

Download and install **Arduino IDE** and follow the instruction on the website (`https://www.arduino.cc/en/main/software`) to install it on the computer at a preferred directory.

Open the Arduino IDE. The front face will look like figure 2.

Preparing Arduino IDE for NodeMCU

This process can be found online. It is better to check YouTube. The process mentioned here may become obsolete with newer version of software.

Open Arduino IDE. Then go to `File->Preferences` and navigate to **Additional Board Manager URLs** and paste `http://arduino.esp8266.com/stable/package_esp8266com_index.json` (shown in figure 3).

Then, open `Tools->Board->BoardsManager` and install esp8266 platform. The search field can be useful here (shown in figure 4).

Copy the Sketch

Find the **Sketchbook location** from the figure 3. Copy the contents inside the **Arduino_Bak** directory and marge them into the **Sketchbook Location** directory. Restart the Arduino IDE.

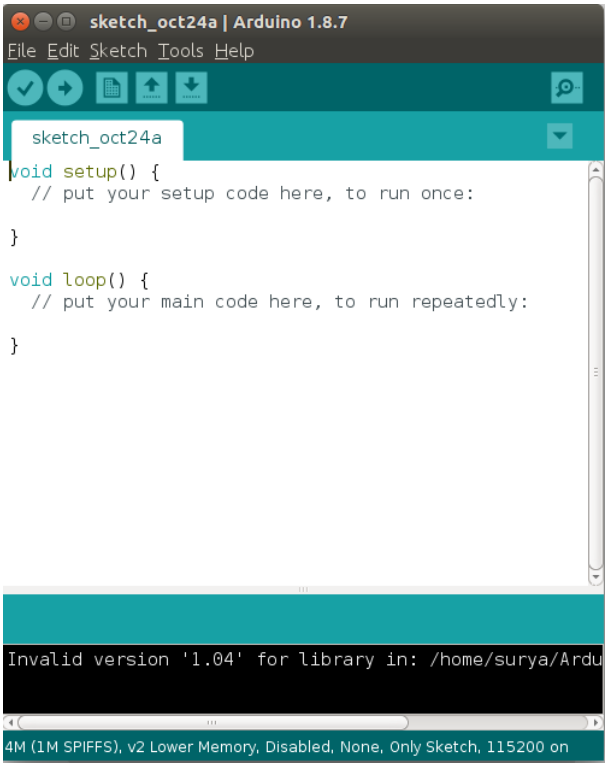


Figure 2: Arduino IDE

Connecting BMP180 to NodeMCUs

Four pins of BMP180 sensor has to be connected to NodeMCU. The connections are as follows.

BMP180		NodeMCU
VIN	———>	3.3V
GND	———>	GND
SCL	———>	D5
SDA	———>	D4

If the pins D4 or D5 is not available then it change it to any available digital pin on NodeMCU. But, the class file has to be modified accordingly. In this case, open `/{SketchbookLocation}/libraries/Adafruit-BMP085/Adafruit_BMP085.cpp` and modify the line mentioned bellow.

```
Wire.begin(D4,D5);
```

Uploading the Sketch to NodeMCUs

Open the Arduino IDE. Then, choose **NodeMCU 1.0** from `Tools->Board` after installation. Open the sketch from `File->Sketchbook->LeakTest_NodeMCU_BMP180`. Inside the sketch, these four fields needs to be configured.

```
int station = 2;
const int wifiNo = 3;
const String ssids[wifiNo] = {"iichep1_ng","iichep2_ng","iichep3_ng"};
const String passwords[wifiNo] = {"xx000112","yy001231","zsd343141"};
IPAddress ipAddress(192,168,0,177);
uint16_t udpPort = 5006;
int displayInterval = 3;
```

The field **station** has is the station number. Station number has be positive integer including zero. The station 0(zero) is to record the data of atmosphere. Other stations are for each RPC gap. No two NodeMCUs with similar station number must be operated at the same time. Please note down the maximum station number assigned, this has to be checked with the program which receives the data from these stations. **Note: This number can not be confused with the total count of stations.**

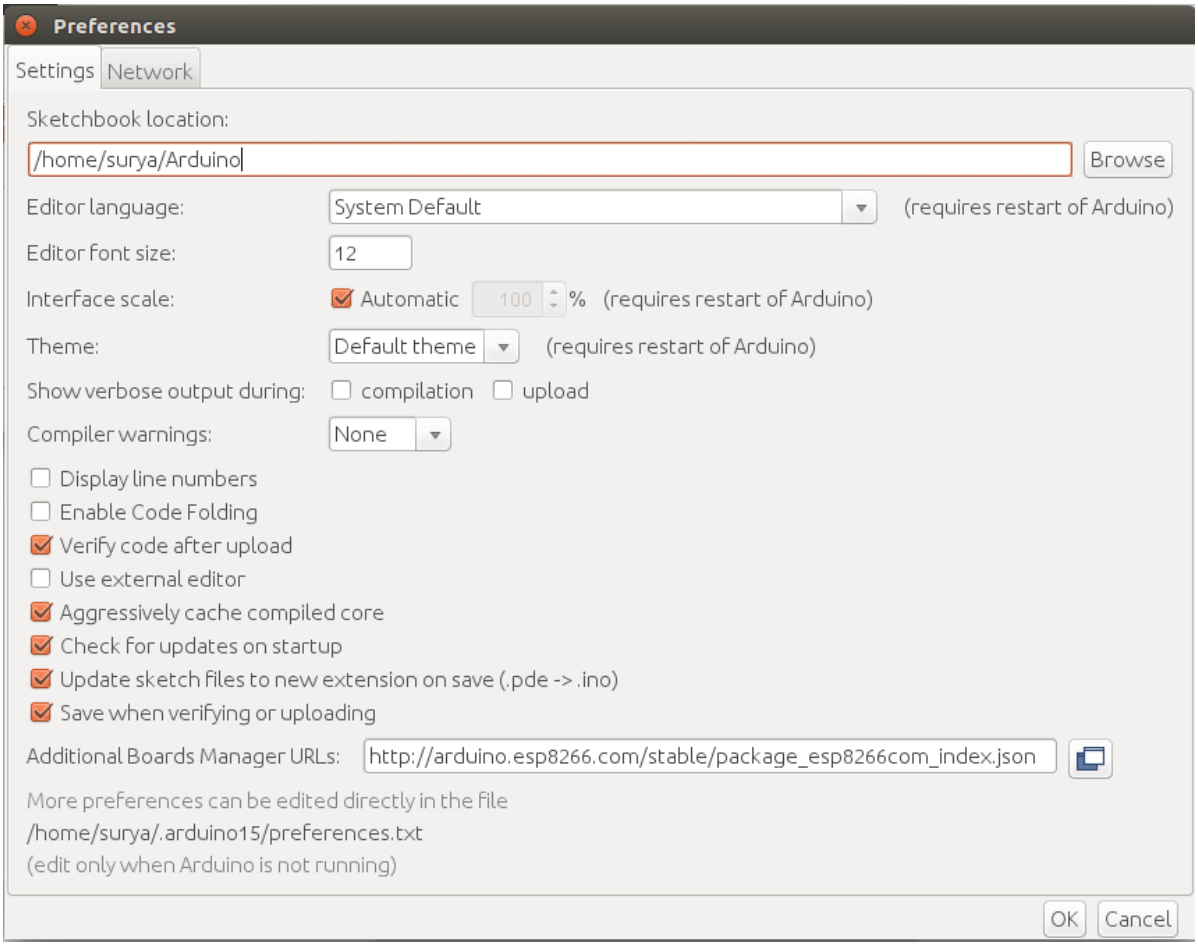


Figure 3: Preferences

Figure 4: Install Drivers

This code is written in such a way that the device will connect to the strongest WiFi. The device will have the ability to switch over to next stongest WiFi if disconneted. The set of SSIDs and Passwords available in the facility have to be specified inside the code in proper manner.

The field **wifiNo** specify the number of available WiFi.

The field **ssids** carry all the SSIDs of the WiFI Access Point of Router with which this module is going to be connected with.

The field **passwords** carry all the passwords associated with the SSIDs mentioned above with proper order.

The field **ipAddress** has to carry the exact IP Address of the system to where this module is sending the UDP data packets. **Here, the numbers are separated by commas.**

The field **udpPort** sets the port number for UDP. If the post number 5006 is not available on the system then change it to some suitable numbers consulting with the system engineer.

The field **displayInterval** sets the intervals of the data packets in seconds. This value has to

be an integer larger than 1(one). This field is also not required to be changed if not needed.

Then save the sketch.

Connect the NodeMCU to be configured to the system. Find the port number assigned to this module by typing `ls/dev/tty*` on any terminal (example: `/dev/ttyUSB0`, `/dev/ttyACM0`).

Open the Arduino IDE. Then, choose the proper port number from `Tools->Port`. Open the proper sketch to be uploaded. Then upload the sketch by `Sketch->Upload`.

After upload is completed, open Serial Monitor by `Tools->SerialMonitor` and make sure that the **baud** rate is at 9600. Then press reset on the NodeMCU module. MAC address will be displayed first. Sometimes, the MAC of the modules are needed to be registered with the Access Point or Router. If needed, then do the needful. After successful connection to the network, the IP Address assigned to the module will be displayed. If the connection to the BMP180 is proper, then the pressure and temperature will be displayed in proper interval.

Now, go to the system to where the NodeMCU is sending UDP packes and type **nc -ul 5006** on any terminal. The data packet containing station number, temperature and pressure must be displayed in proper interval. This module is now successfully configured and tested.[1]

At least two of these fully functional modules are needed for leak test. One module has to be configured as station 0(zero).

Procedure to the Leak Test

Copy the contents inside the **Code_Madurai_NodeMCU** directory and paste them into any desired directory on the system where the NodeMCUs are sending UDP packets. This directory will contain two daughter directories inside it - **bin** and **data**. All the softwares used to receive and log data from the NodeMCUs are inside **bin** directory. The **index file** and all the data files can be found in **data** directory.

In the **bin** directory, two programs written in python can be found. The program named **receiveUDP.py** has to be continuously running for the entire duration of test. In fact, there is no need to stop this program. Execute this program using **./receiveUDP.py** and leave the shell undisturbed. If data packets from **zeroth** module are received by this system successfully, then a file with suffix **_atm.dat** will be created inside **data** directory which will contain data in the following format,

`<UNIX_Epoch><Room_Temperature><Atmospheric_Pressure>`

If the line number of these **_atm.dat** files exceeds a preset number, then a new file will be created automatically. Maximum number of lines can be set by changing the filed textbf25000 within the following line in the program named **receiveUDP.py**,

if int(data_atm_cnt) == 25000:

These **_atm.dat** files are not required for the analysis, but stored for record keeping.

Now, multiple RPC gaps can be tested using this setup. The value of the variable in the following line in **receiveUDP.py** has to be more than the maximum station number assigned,

maxRPC = 100

Restart the program **receiveUDP.py** if it has been modified.

Each NodeMCU stations connected with each RPC gaps will send data to the system. Since, multiple stations are sending data to the program **receiveUDP.py**, an input specifying the name of the RPC connected to a specific station has to be given. This is done by the program named **RPC_Names.py** inside the **bin** directory. This program modifies a file named **RPC_Names** inside **data** folder, which contains the information about the RPCs being tested. If this file has been modified, the program **receiveUDP.py** reads it and correlate the stations with the name of the RPCs. Do not modify the file **RPC_Names** by hand, always used the program. Enter the necessary info using **RPC_Names.py** before starting the leak test for each RPC. While adding the name of a RPC, four info are needed - **Station Number**, **Name of the RPC**, **Run Number** and **Pressure offset on the station w.r.t. zeroth station if any**. One can also use this program to see the number of RPCs attached with the system.

Once the gap name is entered, one data file with gap name and run number will be created in **data** directory. If the zeroth and the other specific station connected with RPC gap is working properly, then the data will be written in the respective file in the following format.

`<UNIX_Epoch><Room_Temperature><Atmospheric_Pressure><RPC_Temperature><RPC_Pressure>`.

Now, the RPC gaps is needed to be pressurized up to a certain pressure difference. To see the pressure difference from the file continuously, open a terminal, navigate to the **bin** directory and execute the following command.

```
./RPC_Display.py <Station No> &
```

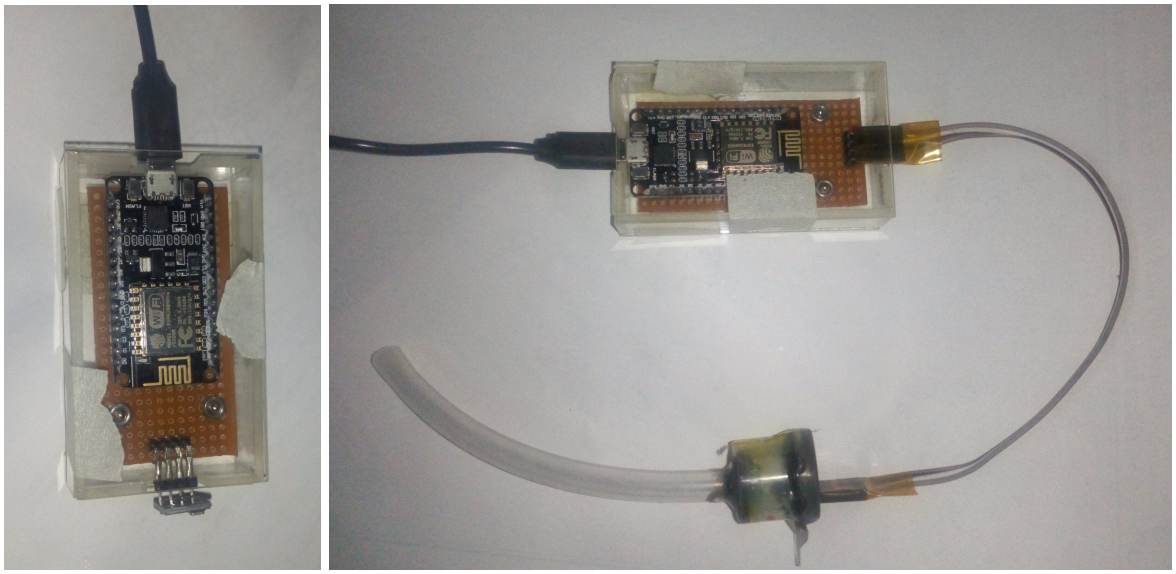
[1]The command **nc** can bind to only one device at the same port. Power down all other NodeMCU before running **nc** command.

This will open a window which shows the pressure difference for that station. Screenshot of the window is in Figure 5.

Then, remove the name of the RPC using the same program after finishing the test using the program **RPC_Names.py**. Otherwise, the program will keep writing into the data file unnecessarily.

Figure 5: Preview of Pressure Difference

Check some photos of instruments and screenshots.



(a) Station 0^{th} (b) Station n^{th} with Sensor Housing

Figure 6: NodeMCU and BMP180 Housing

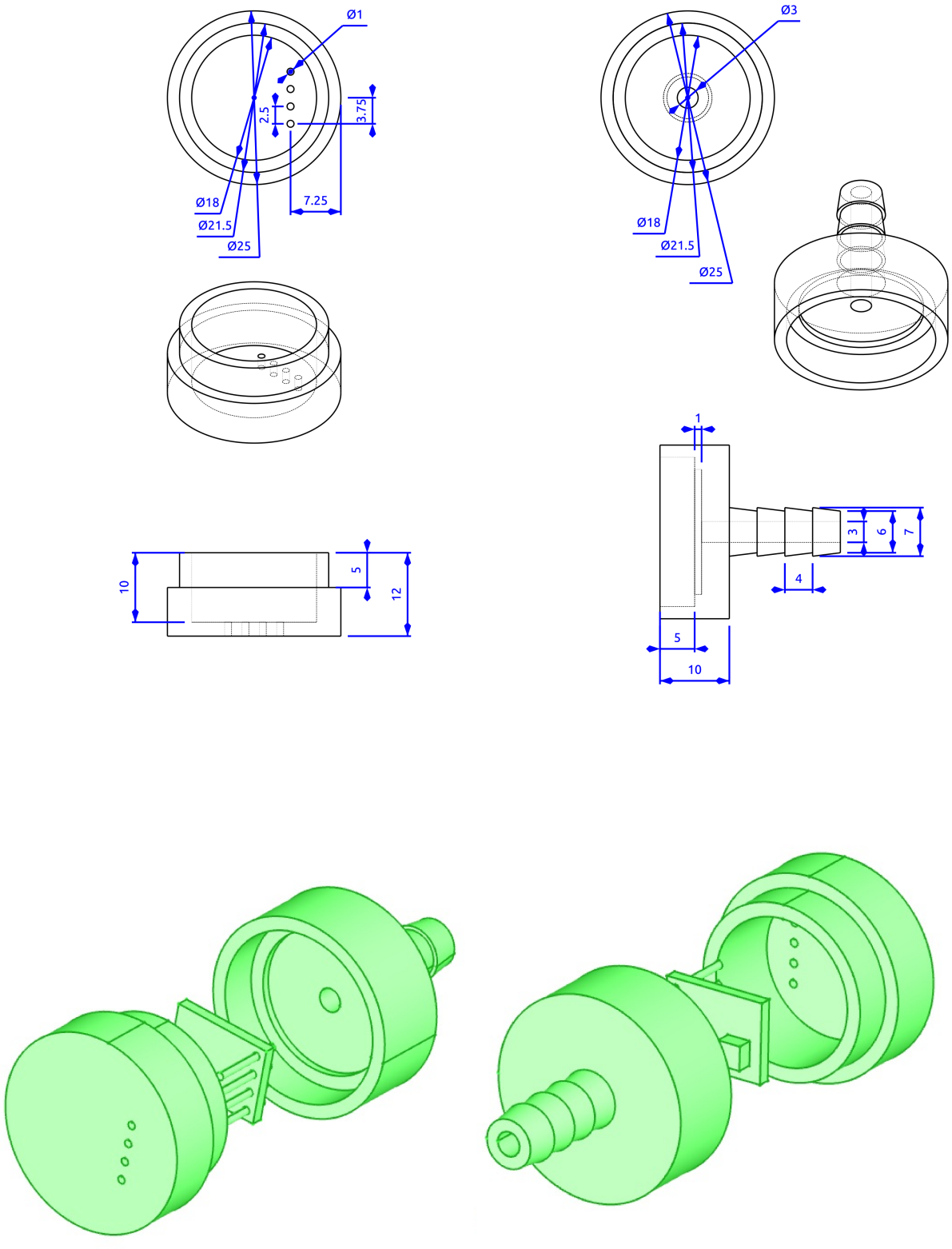


Figure 7: Sketch of BMP180 Housing