Table of Contents

1. Laboratory on ESP 8266 2
1.1. Laboratory hardware in details 2
1.1.1. Actuators 2
1.1.2. Sensors 2
1.2. Connecting to the nodes 3
1.3. Programming 5
1.3.1. platformio.ini 5
1.3.2. Software libraries 6
1.4. Scenarios to implement during labs 6
1.4.1. Scenario 1: Basic operations on the 4x20 LCD screen 7
1.4.1.1. Prerequisites 7
1.4.1.2. Scenario 7
1.4.1.3. Result 7
1.4.1.4. Start 7
1.4.1.5. Steps 8
1.4.1.6. Result validation 9
1.4.2. Scenario 2: Presenting temperature and humidity on the LCD 9
1.4.2.1. Prerequisites 9
1.4.2.2. Scenario 10
1.4.2.3. Result 10
1.4.2.4. Start 10
1.4.2.5. Steps 10
1.4.2.6. Result validation 12

1. Laboratory on ESP 8266

While ESP 8266 (Espressiff) is, in general, a network processor with broad applications in IoT, here we focus on using it as an embedded-class microcontroller.

As you will have remote access to the pre-defined hardware, focus on the node's configuration as they vary between nodes.

All you need is a web browser and broadband Internet connection.

1.1. Laboratory hardware in details

You can book for use nodes 1 through 4 and 8 through 11 at the moment.

Every node contains sensors and actuators. Every node has 4×20 (4 lines, 20 characters) LCD display, you can observe via web camera, and also at least a DHT (temperature + humidity) sensor.



The 4×20 LCD display is connected via I2C bus to the ESP microcontroller and its address varies, depending on the node, see the documentation below.



The DHT sensor is connected to the ESP microcontroller using single GPIO and there are two types of them! Please see the documentation below.

1.1.1. Actuators

Here we focus on the 4×20 LCD screen solely. This type of LCD can display 4 rows, 20 characters each, and is controlled via the I2C interface.
The I2C bus is connected to D1/GPIO5 and D2/GPIO4 pins of the ESP microcontroller (D1 is SCL and D2 is SDA). All nodes share the same connection configuration.

There is a difference in the addressing, however:

- Nodes 1 through 4, 8 and 9 use 0x3F;
- Nodes 10 and 11 use 0x27;

You need to provide appropriate initialisation in the software library that controls LCD regarding the note you've booked.

1.1.2. Sensors

In the following scenarios, you will use a DHT, integrated temperature-humidity sensor.

Sensor types vary between nodes (Figure 1):

- Nodes 1 through 4 all use DHT 22 (white sensor);
- Nodes 8 through 11 all use DHT 11 (a blue one);

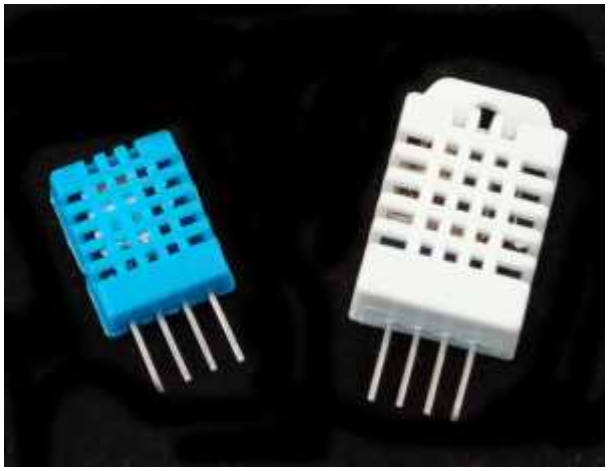


Fig. 1: DHT 11 (left) and DHT 22 (right) temperature and humidity sensors

You need to initiate related software library with appropriate sensor type, depending on the laboratory node you use.
In general, those sensors differ with accuracy; thus for raw experimentation, it is meaningless, whichever node you book to use.

Regarding DHTs, all nodes share the same configuration on the connection level: your DHT is always connected to the D4/GPIO2 pin of the ESP microcontroller.



Warning: sending continuous requests, without a delay, to the DHT sensors is a really bad idea as you will frequently get NaN (not a number) results. Consider introducing some at least 200ms delay between consecutive reads.

1.2. Connecting to the nodes

You need to create an account (Figure 2), please see an instructional video on how to connect and create an account.

Please follow instructions and remember to fix your theme for Revolution! Your environment should look like in Figure 3.

Do not book a device for more than 30 minutes in a row (Figure 4): respect other students! You can continue working with devices out of regular working hours, see supervisor's guidance on lab availability.

The URL for accessing VREL lab is: http://distance.roboticlab.eu.

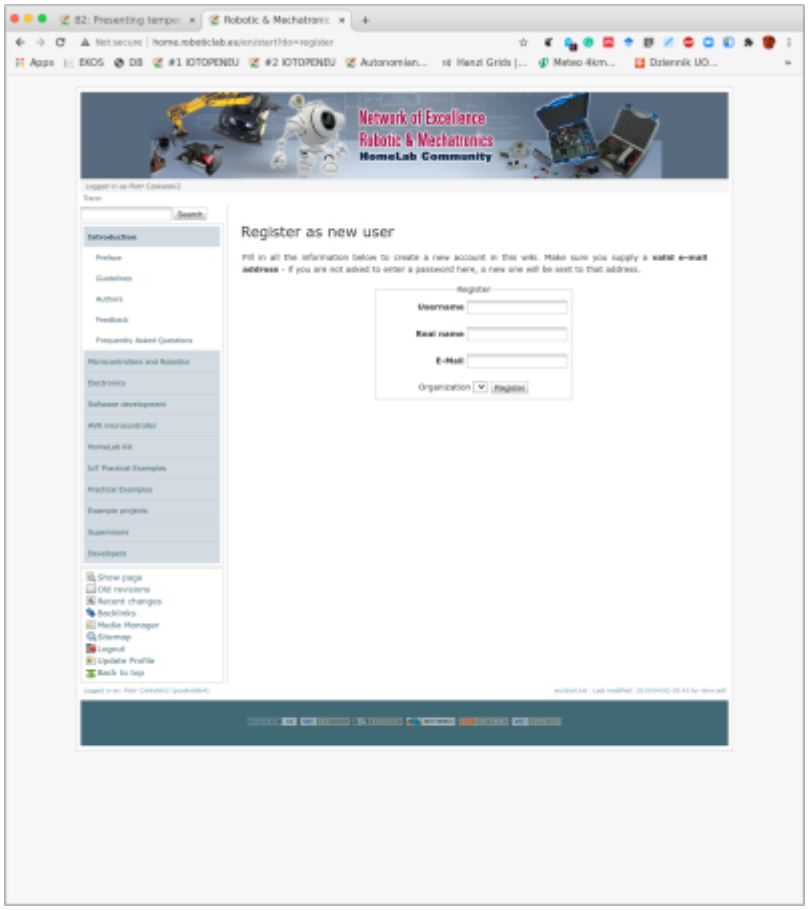


Fig. 2: VREL account registration page



Fig. 3: Laboratory node 8 ready for booking



Fig. 4: Booking of the VREL node

1.3. Programming

You program it the same way, as popular Arduino. The general program structure is composed of two functions:

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Setup is executed once after you upload your code (eventually when you press “Reset” in the interface), while `loop()` is executed indefinitely, once `setup()` has finished.

1.3.1. platformio.ini

One more step is necessary to finish the configuration of your environment: you need to tell the compiler, what is your target platform and which libraries will you use. To do so, there is a file named `platformio.ini` that you need to edit (see top tab bar with files, there is platformio.ini to the left).

Simply paste the following contents:

```
[env:d1_mini]
platform = espressif8266
board = d1_mini
framework = arduino
lib_deps = Wire, 1574, PubSubClient, DHT sensor library@1.3.5, CoAP simple
library, 31@1.0.3, 528@1.0.4, Adafruit Unified Sensor
```

Obviously, this contains all hardware and software libraries, even more than is necessary to use in case of the following two scenarios, but you will have it ready for future experimentation.



Remember to remove other content than presented above. Wrong development environment configuration can cause the inability to compile and run your code!



Note, sometime when you copy-paste content from this PDF file, there may be some extra characters included (i.e. text formatting), that are not visible on the screen, but they do exist and they may cause errors during compilation. To avoid, use copying through a proxy, dummy text editor, i.e. Notepad or Notepad++. First copy from PDF to Notepad then from Notepad to the web browser.

Remember to save file!

1.3.2. Software libraries

In general, you need to use two software libraries: for DHT and LCD.
The LCD library is LiquidCrystal_I2C one while using DHT requires two libraries from Adafruit: Adafruit_Sensor and DHT.

And obviously, if you use some preconfigured values (i.e. as when you refer to the D1 and D0 when initialising I2C communication library), you need to include the `Arduino` library that contains declarations of the constants and some useful functions, i.e. `delay(xxx)`.

```
#include <Arduino.h>
//Hardware specific libraries
#include <LiquidCrystal_I2C.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

1.4. Scenarios to implement during labs

There are two scenarios to be implemented within the scope of this laboratory work:

- Basic operations on the LCD;

- Presenting temperature and humidity on the screen;

1.4.1. Scenario 1: Basic operations on the 4x20 LCD screen

Whatever you do, you expect to have some output of the system. Sometimes there is a blinking LED, sometimes information about connected/disconnected network and some other time simply trace algorithm progress. In laboratories where you have physically connected MCU to your programming device (i.e. computer), you usually would choose `Serial` port to report about what is going on. However here all you have is access via the video stream. Perhaps those are reasons you will use this LCD display in your every lab work.



Note, the video stream from the camera is limited in bandwidth and presents some 5-10fps maximum (usually around 1 fps) so you shouldn't change display content nor LED state faster than once every 2 seconds, to let you notice any change!

1.4.1.1. Prerequisites

There are no other prerequisites than LCD I2C library. Mind, LCD is controlled via the I2C bus. LCD Display is 4×20 characters. LCD is controlled via I2C extender: PCF8574.



There are two types of I2C extenders differing their I2C address:

- Nodes 1 through 4, 8 and 9 use 0x3F address
- Nodes 10 and 11 use 0x27 address

1.4.1.2. Scenario

Initialize LCD screen, clear it, then write some fancy text on it, i.e. “Hello SMIW!” in the first line. Then your first name in the second line and the name of the city you're in, in the third. Note, out of the box, the LCD screen does not contain polish special characters like *śćżółńź*, so do not include them unless you re-define character definitions (advanced). In the fourth line, print right-aligned number of `loop` iterations (delay it for 1 second between updates - to observe video stream lag and delays). Note, delays are provided in ms, not in s.

1.4.1.3. Result

You should see the texts and ticking counter in the video stream.

1.4.1.4. Start

There are no special steps to be performed.

1.4.1.5. Steps

1.4.1.5.1. Step 1

Include LCD driver library (and obviously you need an Arduino library as well):

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

1.4.1.5.2. Step 2

Instantiate software controler component for the LCD display:

```
LiquidCrystal_I2C lcd(0x3F,20,4);  // set the LCD address to 0x3F for nodes
1 through 5, 8 and 9
//LiquidCrystal_I2C lcd(0x27,20,4); // for nodes 10 and 11 only!
                                    // for a 20 chars and 4 line display
```

1.4.1.5.3. Step 3

Declare some variables: counter `i`, its length `n` and `buffer` for the into to string conversion:

```
int i = 0;
char buffer [50];
int n;
```

We will use them in the further part of the code.

1.4.1.5.4. Step 3

Initialize display - we suggest to do it in `setup()` function:

```
...
  lcd.init(D2,D1);                  // initialize the lcd I2C
  lcd.backlight();                  // switch on the backlight
...
```

1.4.1.5.5. Step 4

Clear the contents, set cursor and draw static text - still in `setup()` function:

```
...
  lcd.home();
  lcd.print("Hello World!");
```

```
...
```

The `setCursor(x,y);` function will help you to position in the place you want the text to be rendered:

```
...
    lcd.setCursor(0, 1);
...
```

Indexing is zero-based and upper left corner is (0,0).

1.4.1.5.6. Step 5

Implement `loop()` to draw number of loop executions:

```
...
  i++;
  n=sprintf(buffer,"%d",i);
  lcd.setCursor(20-n,3);
  lcd.print(i);
  delay(1000);
...
```



`sprintf` uses number of wildcards that are rendered with data. Refer to the c/c++ documentation on `sprintf`. Here `%d` means: having integer number render it to string and as we do not specify number of digits, it is up to the engine to convert it properly. `delay(time)` is measured in milliseconds.

1.4.1.6. Result validation

Observe text, its position and counter ticking in lower, right corner.

1.4.2. Scenario 2: Presenting temperature and humidity on the LCD

In this scenario, you will present temperature and humidity as read by the attached DHT22/DHT11 sensor, on the LCD screen.

1.4.2.1. Prerequisites

You need to know, how to print and position text on the LCD display. Use LCD I2C library to control it (and obviously you need an Arduino library as well):

```
#include <Arduino.h>
```

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

The temperature and humidity sensor is all-in-one, DHT22 sensor (white) or DHT11 (blue), connected to the pin D4. Observe your camera to check which sensor is equipped with your lab and consult node documentation.
To read data you may use a dedicated library (libraries):

```
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

1.4.2.2. Scenario

Once you initialise sensor and LCD display, read sensor data (temperature and humidity, mind they're float values, not integers) and then display them in the loop. Give each loop execution some 5-10s delay.



Do not try to read temperature and humidity too frequent. Once every 5s is quite enough both for information and safe enough to not let your readings race with the communication protocol. If you read too frequent, your sensor may not be able to deliver data on time and you will obtain a 'nan' (not a number) instead of temperature and humidity.

The first line of the LCD should display: “*Temperature is:*“
The second line should provide temperature within the *”NN.N C*” form, in Celcius.
The third line should display: “*Humidity is:*“
Finally, the fourth should present relative humidity: *”NN.N%Rh*“

You will use `sprintf` function to format string and convert from float to string.

1.4.2.3. Result

You should be able to see temperature and humidity readings on the LCD display using the video stream.

1.4.2.4. Start

There are no special steps to be performed.

1.4.2.5. Steps

1.4.2.5.1. Step 1

Include LCD and DHT sensor driver libraries (and obviously you need an Arduino library as well):

```
#include <Arduino.h>
#include <LiquidCrystal_I2C.h>
#include <Adafruit_Sensor.h>
#include <DHT.h>
```

1.4.2.5.2. Step 2

Instantiate software controler components for the LCD display and DHT sensor:

```
LiquidCrystal_I2C lcd(0x3F,20,4);  // set the LCD address to 0x3F for nodes
1 through 5, 8 and 9
//LiquidCrystal_I2C lcd(0x27,20,4); // for nodes 10 and 11 only!
                                    // for a 20 chars and 4 line display
DHT dht(DHTpin,DHT22,50); //for nodes 1 through 4
//DHT dht(DHTpin,DHT11,50); //for nodes 8 through 11
```

1.4.2.5.3. Step 3

Declare a buffer for `sprintf` function:

```
char buffer [10];
```

1.4.2.5.4. Step 4

Initialize LCD and DHT sensor:

```
...
  lcd.init(D2,D1);     // initialize the lcd
  lcd.backlight();
  lcd.home();
...
  dht.begin();
  delay(1000);         // give it a second to let the sensor initialize
...
```

1.4.2.5.5. Step 5

Read in the loop:

```
...
  float hum = dht.readHumidity();
  float temp = dht.readTemperature();
...
```

To convert float into the string with formatting use `sprintf` function:

```
...
  sprintf(buffer,"%2.1f C",temp);
...
  sprintf(buffer,"%2.1f%%Rh",hum);
...
  delay(5000);
...
```



`sprintf` uses number of wildcards that are rendered with data. Refer to the c/c++ documentation on `sprintf`. Here

```
%2.1f
```

means: having float number render it to string using two digits before decimal point and one after. Temperature in our lab is always above 10C.

```
%%
```

is an escape character to print a

```
%
```

(percent) symbol. `delay(time)` is measured in milliseconds.

1.4.2.6. Result validation

Observe temperature and humidity readings on the LCD. The temperature in our lab usually ranges between 20-30C (not, we observed over 30C during really hot summer!) while humidity is usually between 35-60%Rh.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:macro:embedeed:lab:esp?rev=1606306672

Last update: **2020/11/25 13:17**