

Guide to practical classes in IoT design and integration

for students in Computer Science

Introduction

This experimental teaching module is first of the kind taught in our department. The study is based on the newest technologies that are not (yet) mastered in traditional teaching curricula. Many of the components and boards used for practical classes are not widely available, some of them are released on the international market only a few months or even weeks ago (October 2016).

We hope that you will appreciate the possibility to experiment with the newest IoT technologies. Most of the exercises are presented for the first time in this domain. We mean here especially the newest communication modules such as LoRa (RFM9X) and ESP-12 based boards for controllers and gateways.

At the software side we are probably first able to teach the embedded systems (MCUs) programming and development directly on SBCs such as Raspberry Pi3 or Odroid XU4. This is possible thanks to the Arduino IDE 1.6.6 experimental platform (released in August 2016) and the newest Linux packages for Radio Head libraries compatible with Arduino, ESP and Raspberry Pi architectures (released in September 2016).

No traditional PCs are needed and all SBC may be used with Lapdocs providing the screens, the keyboards and the power. However for the aim of convenience we will also use the traditional PC platform.

All components, the sensors-actuators, the boards, the shields, the communication modules and the additional elements are provided by **smartcomputerlab.org**

How do we work

We have to communicate a huge amount of information related to the components, the protocols, and the object oriented libraries for these components. That is why the exercises and the projects* are provided in an almost finished state. You have to **read** the documentation, to **analyze** the exercise/project content, to **experiment**, to **understand**, and to **modify** and/or **complete** the designs.

In some cases you have to work in "binôme", in some in two "binômes". Many of the exercises and projects are prepared as client-server model and need two communication sides. One side operated by one "binôme".

Important: All exercises are explained in the companion text book. This text also contains the Arduino (or Raspberry Pi) codes necessary to work on the exercises.

<http://www.smartcomputerlab.org/M1/IoT.book.prot.pdf>

The content and the rationale for the exercises/projects

The module targets the understanding and the mastering of the design/development of simple IoT systems based on the most recent , low cost, and open source **components**.

We think that the essential aspects for IoT are **connection** and **communication** technologies. That is why the following exercises we study progressively the connection and communication features, including buses, and communication modems.

These technologies allow us to link all kinds of sensors-actuators and processing boards into an IoT architectures and propose an almost unlimited possibilities for the design/development of IoT applications.

Note that the components and their functions as well as programming codes are explained in the first four chapters of the **accompanying book**.

The related text should be studied before the experimentation exercises.

The following is the list of the exercises and mini-projects.

1. **First example:** A simple Arduino Nano based architecture
2. **Second example:** A simple ESP8266 NodeMCU based architecture
3. **Third example:** A simple communication system for IoT with Arduino Nano and NRF24L01 modules
4. **Fourth example:** A simple communication system for IoT with Arduino Nano and LoRa SX1276
5. **Fifth example:** A simple communication system for IoT with Arduino Nano ESP8266 and LoRa SX1276 serial modems
6. **Sixth example :** Sending images/streams from ESP8266 to RPI3

0. Session 0 - Introduction and first steps (3 hours)

Before starting the work on the exercises and the project we have to prepare and understand our working environment that is based on Arduino IDE and Linux OS.

First we need to download the most recent version of Arduino IDE and install it on our system (Linux). Note that the recent version of Arduino IDE runs on both architectural platforms: Intel x86 and ARM. We need them both while we are going to use a classical PC and an RPI3 SCB. The required software can found on this page: <https://www.arduino.cc/en/Main/Software>.

After the downloading we need to decompress and install the Arduino IDE.

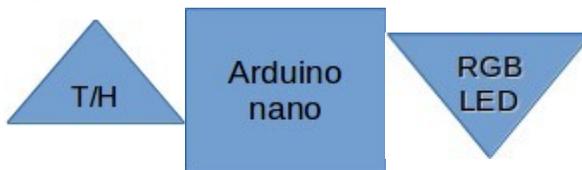
You will find an Arduino main directory and the libraries and sketches sub-directories.

The initial installation is sufficient to start the first designs with Arduino compatible boards: Uno, Nano, Mega. The use of more advanced boards requires the installation of these boards via the Preferences then Board Manager tools.

0.1 A simple front-end with DHT11/22 and Arduino Nano

In the first design with Arduino IDE we are going to use an Arduino Nano board with one sensor - DHT11/22 as the input and one RGB LED as the output.

Architecture:



Components:

DTH11 or DTH22



Arduino Nano



RGB LED

Connections

We are going to use three pins for DHT11/22: GND, 5V and D5 for data and four pins for RGB LED: GND, BLUE-pin 6, GREEN-pin 7 and RED – pin 8.

The following is the Arduino code (with 1 functional error !):

```
#include "DHT.h"
#define DHTPIN 5      // what digital pin we're connected to
#define DHTTYPE DHT11 // DHT 11
// #define DHTTYPE DHT22 // DHT 22
// Connect pin VCC of the sensor to +5V
// Connect pin 5 of the sensor to whatever your DHTPIN is
// Connect pin GND
DHT dht(DHTPIN, DHTTYPE);
void setup() {
  Serial.begin(9600);
  pinMode(6, OUTPUT); // BLUE
  pinMode(7, OUTPUT); // GREEN
  pinMode(8, OUTPUT); // RED
  Serial.println("DHTxx test!");
  dht.begin();
}
void loop() {
  digitalWrite(6, LOW); digitalWrite(7, LOW); digitalWrite(8, LOW);
  // Wait a few seconds between measurements.
  delay(2000);

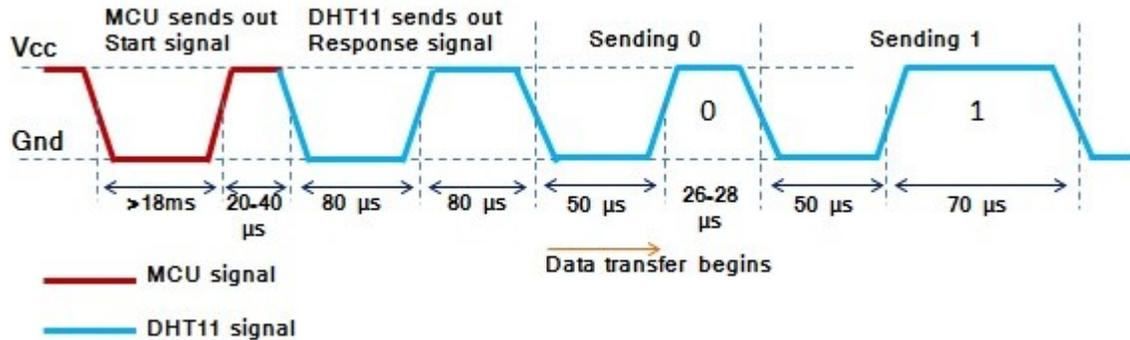
  // Reading temperature or humidity takes about 250 milliseconds!
  // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
  float h = dht.readHumidity();
  // Read temperature as Celsius (the default)
  float t = dht.readTemperature();
  // Read temperature as Fahrenheit (isFahrenheit = true)
  float f = dht.readTemperature(true);
  // Check if any reads failed and exit early (to try again).
  if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }
  Serial.print("Humidity: "); Serial.print(h); Serial.print("%\t");
  Serial.print("Temperature: "); Serial.print(t); Serial.println("*C");
  if(t<21.0) { digitalWrite(6, HIGH); } // set LED BLUE
  if(t>21.0 && t<23.0) { digitalWrite(7, HIGH); } // set LED GREEN
  if(t>23.0) { digitalWrite(8, HIGH); } // set LED RED
}
```

0.1.1 DHT11/22 communication protocol

The following timing diagram describes the data transfer protocol between a MCU and the DHT11 sensor. The MCU initiates data transmission by issuing a **Start** signal. The MCU pin must be configured as output for this purpose. The MCU first pulls the data line low for at least 18 ms and then pulls it high for next 20-40µs before it releases it.

Next, the sensor responds to the MCU “**Start**” signal by pulling the line low for $80\mu\text{s}$ followed by a logic high signal that also lasts for $80\mu\text{s}$. Remember that the MCU pin must be configured to input after finishing the “**Start**” signal. Once detecting the response signal from the sensor, the MCU should be ready to receive data from the sensor.

The sensor then sends 40 bits (5 bytes) of data continuously in the data line. Note that while transmitting bytes, the sensor sends the most significant bit first.



DHT11 activating and reading protocol

0.1.2 To do:

1. Remove the functional error from the code.
2. Test the code with different sensors HT11 and DHT22 (what is the essential difference) :
3. What happens when temperature is exactly 23.0 degrees Celsius ?

0.2 A simple front-end with I2C sensors and ESP8266 MCU

In this design we are going to use a NodeMCU board with one sensor – HTU21D as the input and an OLED display as the output.

Architecture



Components



Connections

Both components are connected via I2C bus available directly on: D1 – SCL and D2 – SDA. We re using **3.3V** and **GND** connections.

The Arduino code to analyze:

```
#include <Wire.h>
#include "Adafruit_HTU21DF.h"
#include "OLED.h"
```

```
Adafruit_HTU21DF htu = Adafruit_HTU21DF();
OLED display(D2, D1); // SDA - D2, CL - D1 : wemos lora gateway
```

```
float t;
float h;
```

```

char mess[24];

void setup() {
  Serial.begin(9600);
  Serial.println("HTU21D-F test");

  if (!htu.begin()) {
    Serial.println("Couldn't find sensor!");
    while (1);
  }
  display.begin();
}

void loop() {
  t = htu.readTemperature();
  Serial.print("Temp: "); Serial.print(t);
  h = htu.readHumidity();
  Serial.print("\t\tHum: "); Serial.println(h);
  delay(2000);
  display.print("SMTR design 2");
  memset(mess,0x00,24);
  sprintf(mess,"Temp:%d",int(t));
  display.print(mess,2);
  memset(mess,0x00,24);
  sprintf(mess,"Humid:%d",int(h));
  display.print(mess,3);
  delay(1000);
}

```

0.2.1 To do

The HTU21D sensor provides precise values of temperature and humidity with two decimals registered on floating point format.

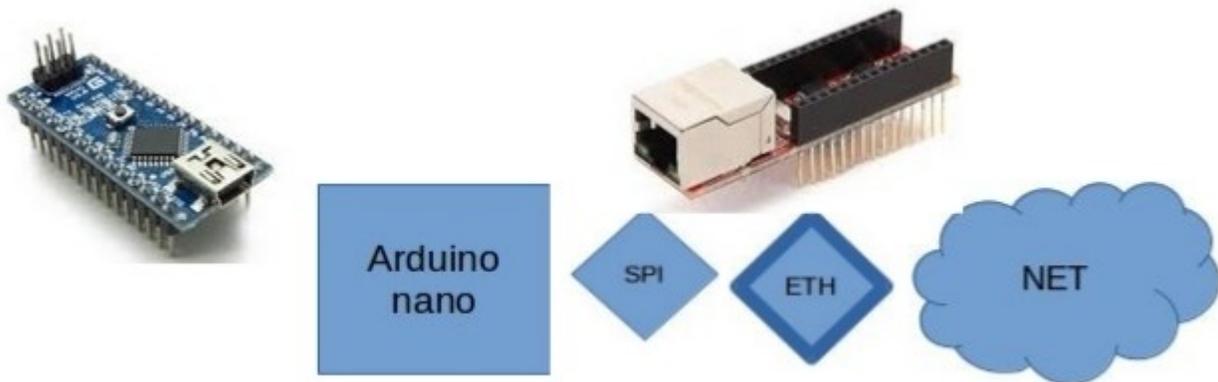
The code above with display function displays only integer values.

Modify this code in order to display the decimals.

1. Session 1 – back-end on Ethernet and WiFi

1.1 A simple Nano based architecture (1 to 2 hours)

This is a complete example of a client module with a “front-end” including a double sensor (temperature and humidity) and an Arduino Nano board used to read the sensor data directly via a simple digital line. The back-end of this sub-system integrates an Ethernet shield communicating with the main board (Nano) via an SPI bus. The Ethernet interface is further connected to the local network. At the server side of the network we use an RPI3 board running an IP/UDP server that allows us to receive (and send) the data from/to the Arduino board. The sensor is the module called DHT11 or DHT22; the software library contains **DHT.h** and **DHT.cpp** files.



Rationale

You experiment here with the simplest components providing a complete application. You use your previous knowledge concerning **Ethernet-Internet communication** (Computer Networks – ETN4)

This project takes the simplest processing and communication components: Arduino Nano and Ethernet. The Arduino board is used to capture the temperature/humidity parameters from one of the **simplest sensors – DHT11**.

To do:

We propose to complete the design by the introduction of an actuator. We can take an RGB LED, a buzzer or a relay. The relay may activate the heating device when the received temperature is too low. In our case the simplest choice would be an RGB LED with 3 states to signal the level of temperature: **blue** – to low, **green** – good, and **red** to high.

At the server side we send the corresponding message-signal to be decoded by the client and applied to the Arduino Nano pins linked to the RGB LED.

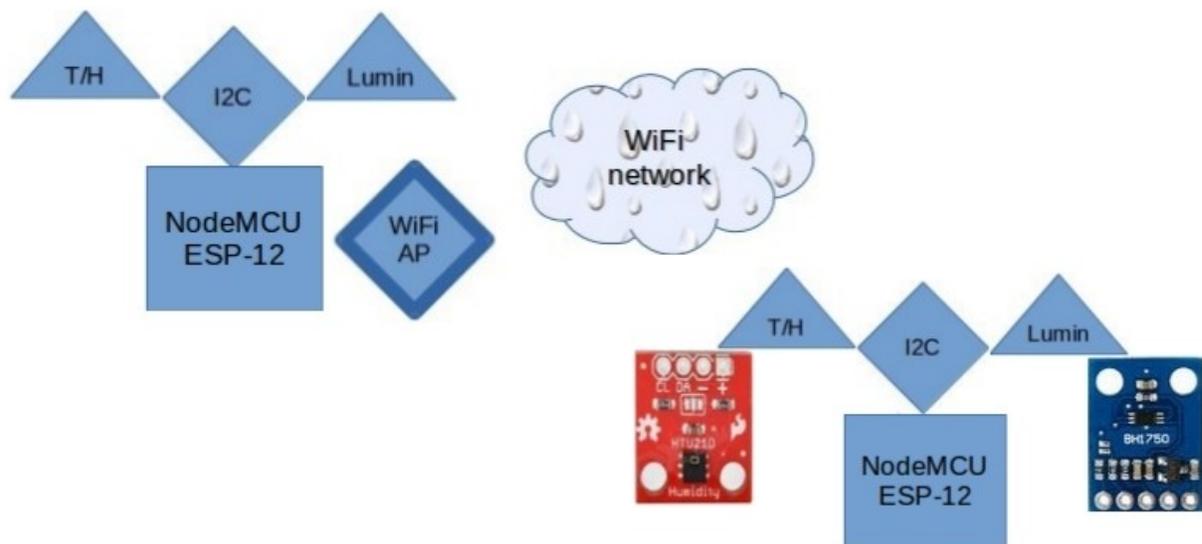
Attention

Duration of the exercise is 2 hours for assembling, programming, and experimentation (during the 3rd hour you have to give some “advice” to the second “binôme” working only 1 hour on this exercise).

1.2 A simple ESP8266 MCU based architecture (1 to 2 hours)

NodeMCU boards are based on ESP8266 System on Chip from Expressif. It combined features of WiFi access point and station + micro-controller. For more details look in the previous chapter.

This example explores a front-end server architecture that involves two sensors one for the temperature, and the humidity and second for the luminosity. These sensors are connected in parallel to the MCU via an I2C bus.



Rationale

You experiment with relatively simple components including the connection via **I2C bus** and the communication with **WiFi modem** integrated in ESP-12 MCU. You should understand the I2C bus functions (addressing and data transfer) and the WiFi communication similar but not identical to Ethernet. Note that you must configure wireless access with an SSID and the password before using the WiFi interface.

To do:

1. We propose to modify the server code in order to send the temperature and humidity values with decimals.
2. Add an RGB LED and/or relay to the MCU board and send the control signal-message from the client side in order to switch on/off or light the RGB LED it accordingly to the received values of temperature and/or luminosity.

Attention

Duration of the exercise is 2 hours for assembling, programming, and experimentation (during the 3rd hour you have to give some “advice” to the second “binôme” working only 1 hour on this exercise.

Session 2 : Radio links with NRF24 and LoRa modems

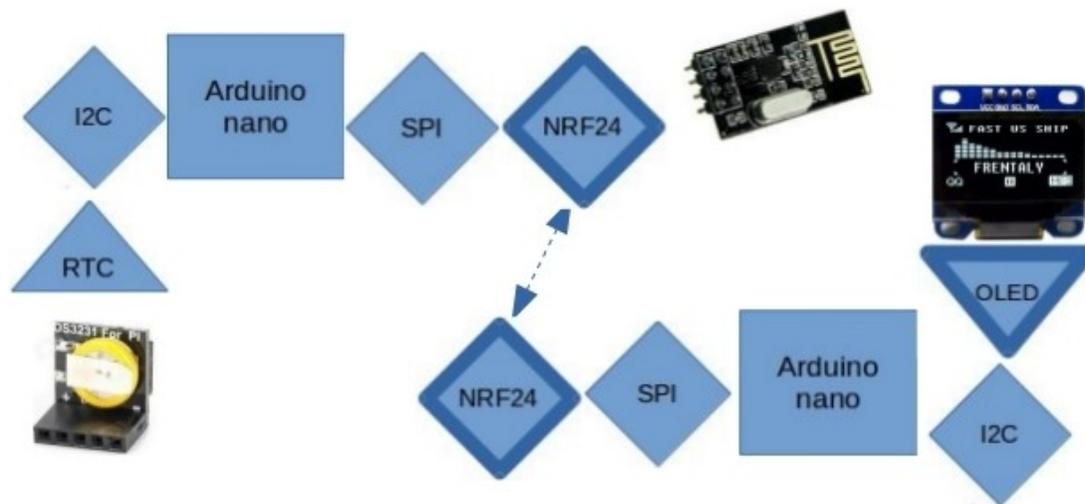
2.1 A simple communication system for IoT with Arduino Nano and NRF24L01 modules (1 to 2 hours)

In this example we are using NRF24L01 modems to transmit the messages/data from one Arduino Nano to another. The nRF24L01 is a highly integrated, ultra low power (ULP) 2Mbps RF transceiver IC for the 2.4GHz ISM band. The modulation mode is GFSK (Gaussian Phase Shift Keying).

With peak RX/TX currents lower than 14mA, a sub μ A power down mode, advanced power management, and a 1.9 to 3.6V supply range .

Depending on the model with or without LNA antenna the communication distance is at least between 20 and 200 meters. The selected data rate varies from 250 Kb/s to 2Mb/s.

The NRF24L01 module is connected via an SPI interface. To program the module we are using **RH_NRF24.h** library available for Arduino as well as for Raspberry Pi.



Rationale

Here you experiment with **simple radio components** – nRF24 using 2.4GHz ISM band with many 1Mb/s channels. NRF24 modules may operate with Arduino Nano/Mega or Raspberry Pi using the same Radio Head library. You should understand the I2C bus functions (addressing and data transfer) with RTC real time clock and **OLED display**.

The use of OLED screen with **u8g** library allows to experiment with displaying functions.

Note that the nRF24 module may require an adapter providing more power from the 5V source.

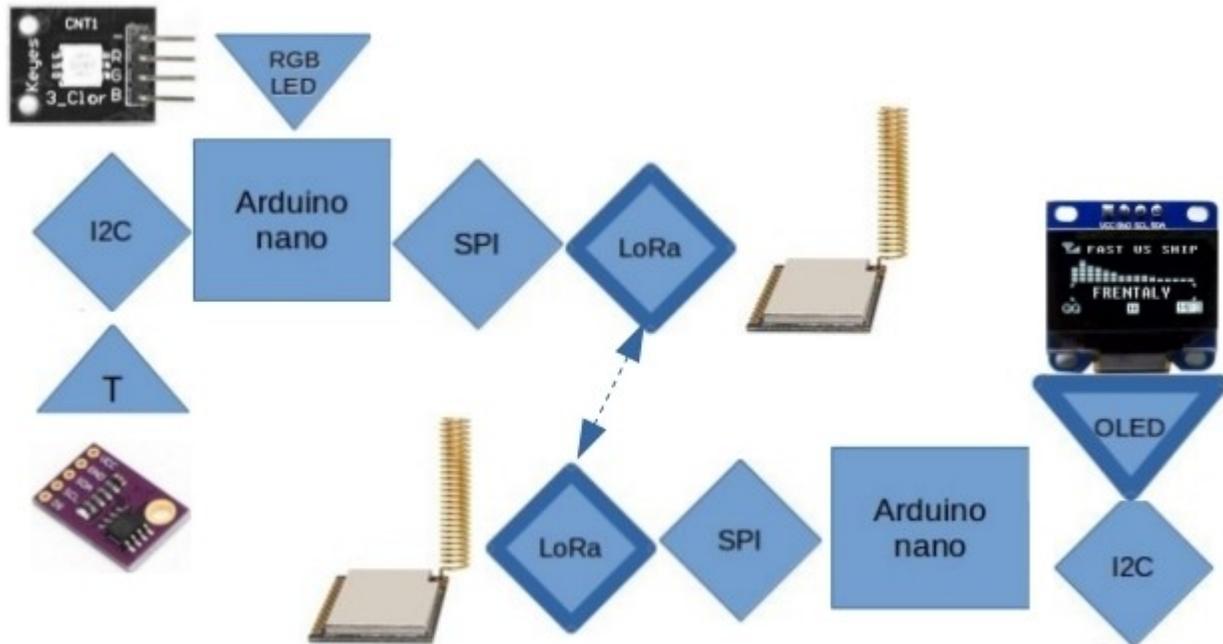
To do:

1. We propose to experiment with the radio frequency on both sides using **nrf24.setChannel(channel)** function
2. Add an RGB LED on the client side to indicate the state of the processing: waiting (red), receiving frame(blue), sending reply frame (green)

2.2 A simple communication system for IoT with Arduino Nano and LoRa SX1276 (1 to 2 hours)

The following LoRa client-server architecture is quite similar to the previous example with NRF24 modems. Here at the client side we have a front-end with a temperature sensor connected via I2C bus to the MCU board and an RGB LED to indicate the state of the “protocol”. At the back-end side we have the LoRa modem connected to the MCU via SPI bus.

At the server side the back-end is built with the same kind of LoRa modem . The front-end is made with an OLED display connected to the MCU via I2C bus.



Rationale

Here you experiment with **LoRa radio components** in LoRa mode. You should be able to understand the basic features of LoRa technology and its implementation in RFM95/96 circuits. The use of Radio Head library permits to experiment with different **LoRa modes** including SF-spreading factor, CR-code rate, and BW – frequency bandwidth.

To do:

1. Modify the frequency from 343 by 0.125MHz to 345 (8 channels/125KHz). See the note.
2. Modify the transmission power from 5 to 23 dB.

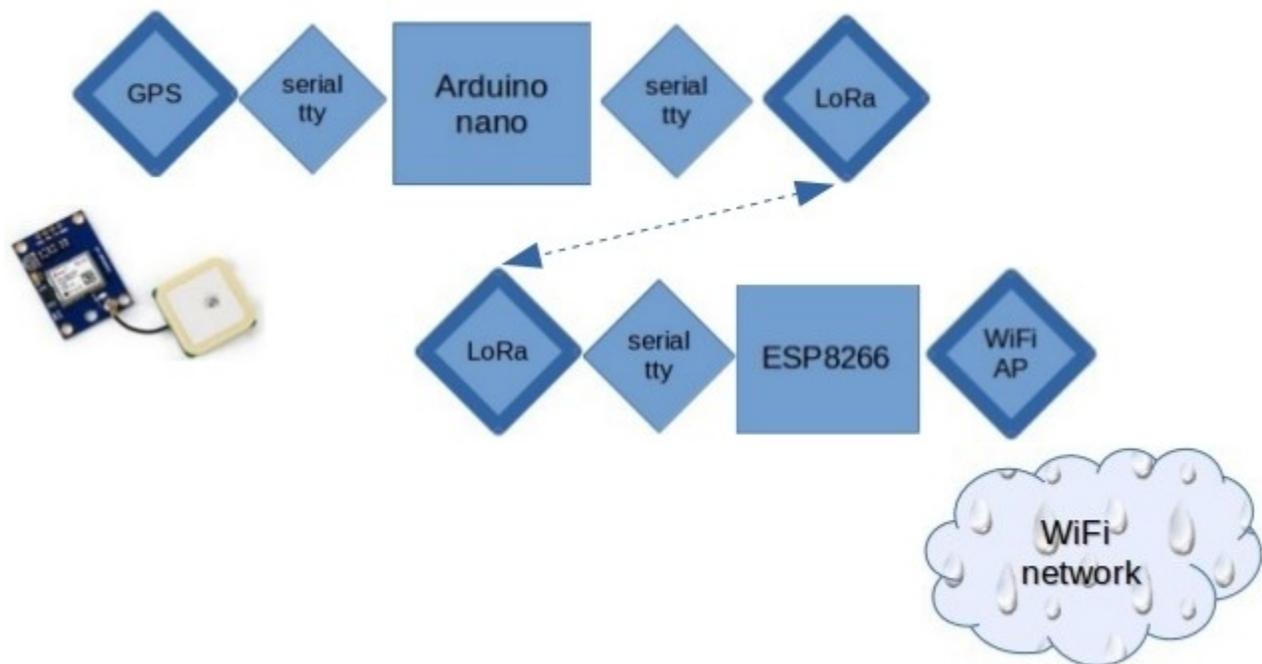
Add the transfer of temperature and luminosity and modify the display on server side.

Note: In the file **RH_RF95 . cpp** (Arduino libraries) choose different **LoRa modes** concerning the bandwidth (from 125KHz to 500KHz), the code ratio (4/5 to 4/8 and the spreading factor (from 512 to 2048 or 4096 chirps/bit). The same changes must be done at both sides, the client side and the server side.

Session 3: GPS and Video Camera on LoRa and WiFi

3.1 A simple communication system for IoT with Arduino Nano ESP8266 and LoRa SX1276 serial modems (1 to 2 hours)

In this example we are using simple serial communication via fixed and wireless links. First we connect via a serial interface the GPS receiver to the Arduino Nano board. This terminal board communicates with the base board implemented by an ESP8266 MCU via a LoRa serial link. Finally, the ESP8266 MCU provides the **softAP** allowing us to access the received data via TCP protocol.



Rationale

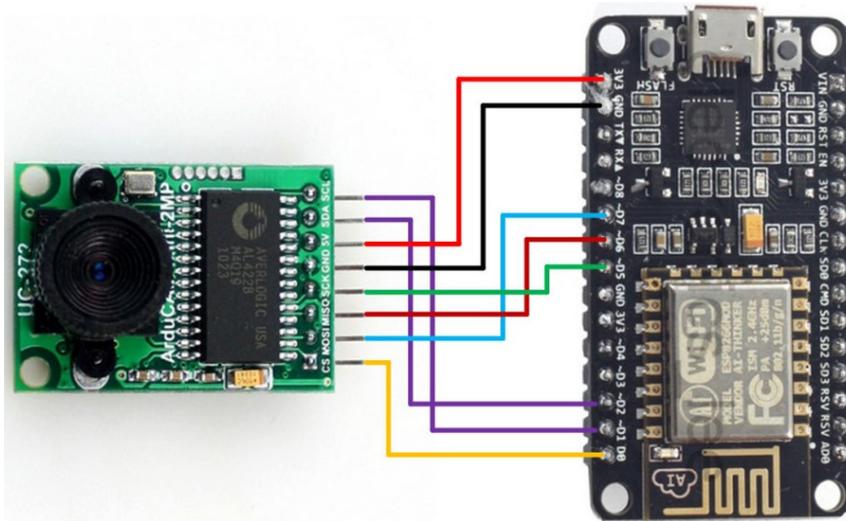
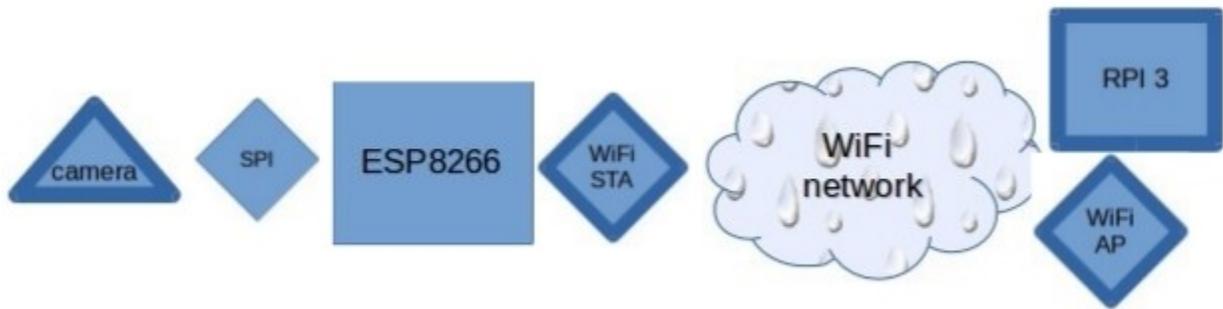
Here you experiment first with GPS module that allows you to understand different characteristics of GPS receiver. Then you use **LoRa radio components** in **serial mode** to compare it with **LoRa mode**. The exercise also illustrates the use of softAP on an ESP node

To do:

1. Replace the LoRa modem with serial UART connection by a one with the SPI bus and driven via Radio Head RF_95 library functions. Use Wemos D1 module instead of NodeMCU.
2. Add an OLED display to LoRa-Wifi module to show the received GPS coordinates on the local gateway.

3.2 Sending images/streams from ESP to RPI3 (1 to 2 hours)

In this example we show how to use the ArduCAM 2MP camera with connected to ESP board (Wemos D1) via the SPI and I2C buses.. With this application example we can take photo and send it to the Web. We also can take the photos continuously as video streaming and send it to the Web.



Rationale

In this exercise you experiment with a multimedia device – ArduCam integrating an FPGA circuit providing a real time JPEG compression of high definition images (up to 2Mpix). You see how to use both I2C and SPI buses to control the device and to receive high speed media stream. Note that the compressed images can not be stored directly in MCU device and must be sent immediately to the external user – WEB client on an SBC.

To do:

1. Modify the above example using Wemos D1 board instead of NodeMCU.
2. Add WiFi manager to select the access point and memorize its parameters.
3. Use several cameras (modules) to test the application