UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Internet of Things

Course Project
Monitoring of an agricultural field using LoRa

Professor: **Marco Di Felice**
Professor: **Luciano Bononi**

Students:
Gianmarco Canello
Marco Roncato

Academic year 2020/2021

1 Introduction

We chose this project since we where interested in investigating the capabilities and potential of the LoRa protocol. As we come from the automation engineering field, we were interested in telecommunication since it's a subject that is not so treated in our degree course and in our opinion it's a key knowledge that is employed in Industry 4.0.

Monitoring a field is a very important aspect in agriculture and knowing the parameters of the soil can help greatly in reducing water consumption and improving the quality of life of small farmers that may not have access to large scale monitoring like the larger companies in this field.

Our setup is a low-cost field monitoring group of sensors without any subscription fee that keeps data locally in the home network of the owner.
With an expense of under 300 euros the owner of the field can efficiently monitor various locations in the field with data coming from the moisture of the soil, the air temperature and humidity.

Our solution allows the owner to change on the fly the GPS coordinates of where the sensor stations are placed and, of course, the sample frequency of the data.
All the data is collected and can be visualized from anywhere (phone, tablet, computer) along with a prediction of future data. We also implemented a Telegram bot that can alert the owner when the soil is too dry or when the status of the field is ok; in this way the owner of the field does not need to perform a login to have access to basic information about the status of the soil.

2 Project's architecture

Our system is deployed by means of two ESP equipped by a Semtech LoRa™ SX1276 that transmits at 433 MHz data provided by a temperature sensor DHT22 and a capacitive soil sensor.

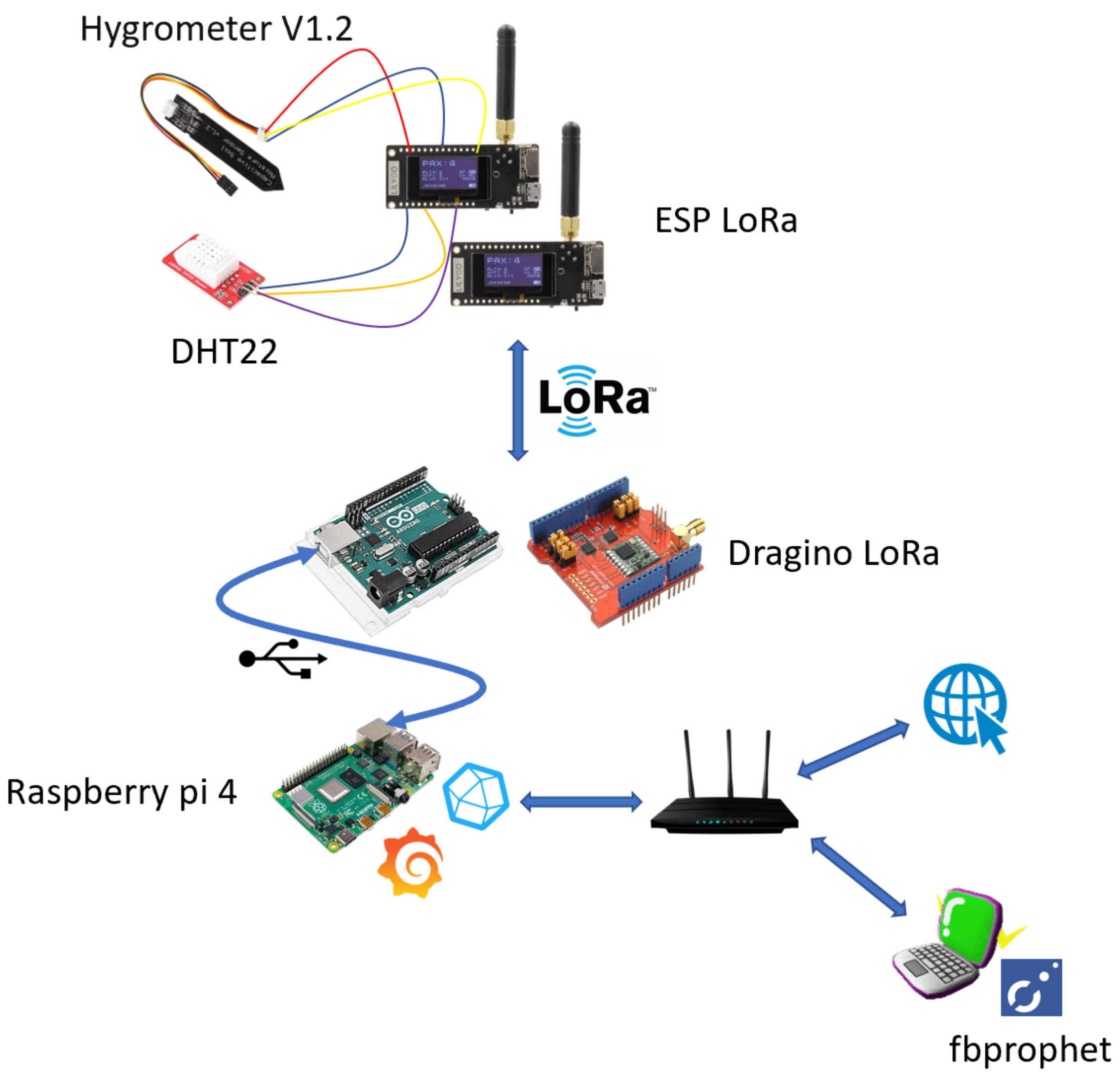


Figure 1: Representation of the data-flow in the project

This device sends and receives packets over LoRa communicating with a Dragino LoRa hat placed on top of an Arduino Uno, which is connected to a Raspberry Pi 4B via USB. The Raspberry acts as local server which collects data and can send packets to modify internal parameters of the ESP devices through the Arduino shield.

The Raspbery Pi hosts Influxdb database and Grafana,which are forwarded over the internet via port mapping, for the management and visualization of the data .

Finally a separate PC is employed to compute predictions over acquired data thanks to the adoption of fbprophet, an open source python library made by a Facebook team. Here we present a scheme of the project's architecture

3 Project's implementation

3.1 Hardware setup

When we initially started the project we wanted to receive data directly into the Raspberry. To do so we worked on a LoRa hat for Raspberry Pi.
This hat had the module SX1268 from Semtech and it would interact with the Raspberry through Python code. We started by running a basic receiver code but we soon noticed that this did not work because the test packet, that we were sending using the ESP device, was never captured.

After a lot troubleshooting, we concluded that the documentation given by the supplier of the hat was too scarce and it had some errors.

Also, we found out that the library used for the LoRa hat (Radiohead) is not compatible when working with devices from different vendors. This obstacle made us change the receiver device to the Dragino that is compatble with the LoRa library by sandeepmistry. This library holds compatibility between different vendors, so we could finally have a communication channel between the ESP devices and the Arduino Uno.

We read online that the Radiohead library was the most performing one, capable to obtain the maximum possible distance coverage for LoRa, but our application only needed basic coverage because we intend to monitor an open field without obstacles, also this library can be used only on devices from the same vendor. Nonetheless, we tested the actual coverage in our application and in urban environment with the lowest spreading factor value we were able to receive data from up to 500m of distance.

3.2 ESPs design

The ESP devices were delivered without any connected pins, so first of all, we soldered all the pins to the boards together with Angelo Trotta. Secondly, we noticed that there were OLED displays connected to every one of the ESP. We decided to include the use of the display in our project even if we are aware that the power consumption would have increased a lot. Moreover, we can turn off the displays by commenting just some lines of code, if the display is turned off, our project becomes much more power efficient! The display shows for demonstration purposes the ID of the sensor station and the packet that it's sending.

Connected to the pins of the ESP devices there are the moisture sensor and the DHT22 temperature and humidity sensor. We had some difficulties with the pin connection because if we chose any different data pin other than the one we used (pin 35 for the moisture sensor and pin 25 for the DHT22) then the ESP device would refuse to boot up.
The code running on the ESP devices is set to do a cyclic deep sleep pattern that is configurable in length from the Raspberry device using a Python script. The length of the sleep duration that corresponds to the sample frequency, is set by default to 30 seconds but in a real application we would need data only once every few minutes.

Every time the device wakes up it takes the measurements using the `getReadings` function where there is also the computation of the SOH value, then it will use the function

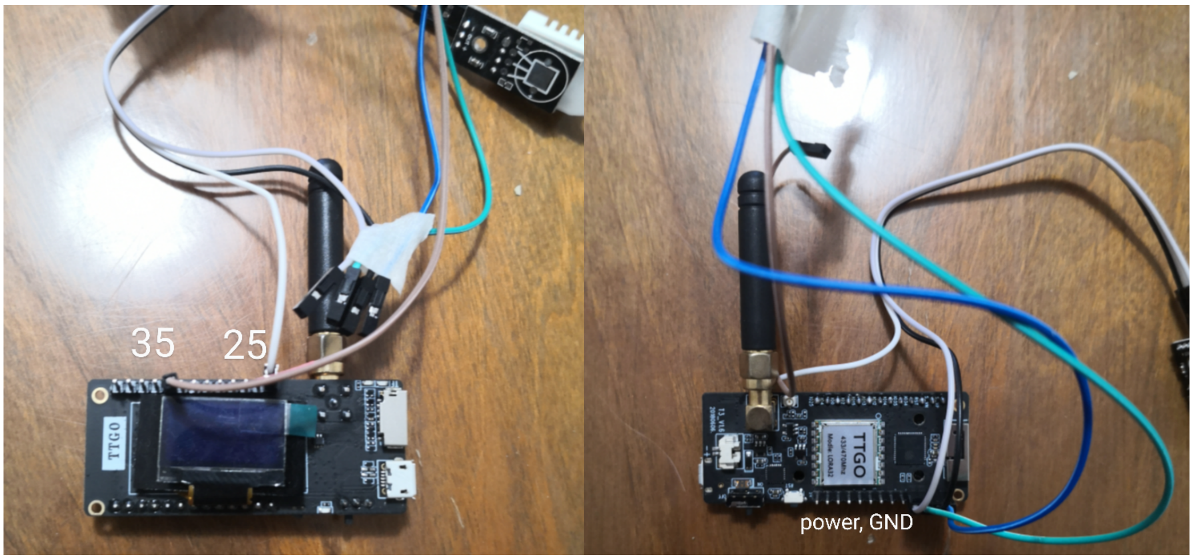


Figure 2: Representation of the pinout for the ESP devices

`sendReadings` to send the readings on LoRa by converting all the values to string and separating them using a comma to be able to convert them later in Node Red in JSON. Note that we used a syncword that was set up thanks to the function `LoRa.setSyncWord()` to make sure that the receiver only gets the packets form our two stations.

Afterwards, the ESP device will start listening for incoming packets that may change the configuration values of the GPS coordinates or the sampling frequency.

The listening period is set by default to be 12 seconds, but, to improve battery efficiency, we implemented a routine to go to sleep if the configurations aren't changed or if they do not attain to the said device. The function `getConfigData` is called when there is an available packet to be received. The packet is checked and if the value is the ID of the device, then the device instantly goes to sleep by returning the value 1, while if it's different than the ID and the first value is the same as the ID of the device, then the new GPS coordinates are changed and the new sample frequency is set.

LoRa radio settings were left as default because we found them to be best suited for our application: the default spreading factor is set to 7 and it's the most power efficient and reliable, the signal strength of the receiving Arduino Uno never went below -90dbm.

If for some reason the sensors need to be placed further away, the spreading factor can be increased to get better range, thanks to the library functions:
`Lora.setSpreadingFactor()`,`Lora.setSignalBandwidth()`,`Lora.setCodingRate()`.

Mind that by doing this the energy consumption of the transmitting devices can quadruple from SF7 to SF12 because the transmitter needs to be on for a longer period of time.

3.3 Dragino transceiver & Arduino

The Dragino is a LoRa hat, mounted on Arduino able to capture packets incoming from ESPs, afterward the received packet is forwarded to Arduino to be processed. Arduino is going to process only the packet received with the correct sync-word, that was decided a

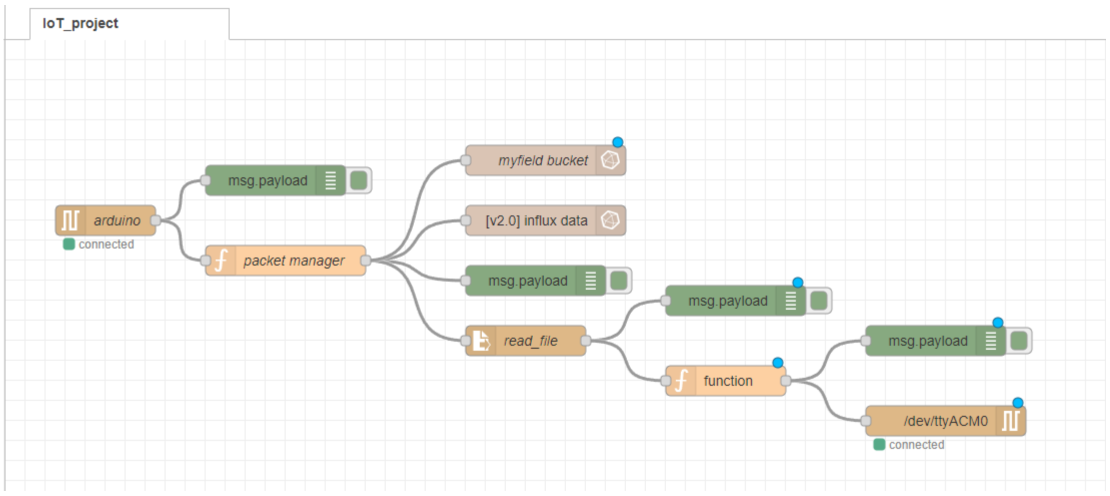


Figure 3: NodeRed scheme for the project

priori. The spreading factor was selected to be 7, the default, the same as the one selected for ESPs. The received packet is parsed in order to collect values that were opportunely divided by means of "," and finally those are printed on the serial USB port together with the signal strength that is added at the end of the received packet. In this way the Raspberry Pi will be able to read the oncoming packet by listening to the serial port.

Whenever a packet is received, the Dragino is switched to sender mode in order to send back a string of configuration parameters, received from serial USB port, to the ESP, in case no parameter were modified then the ID of the device that sent the last one, is sent back.

3.4 Raspberry & NodeRed

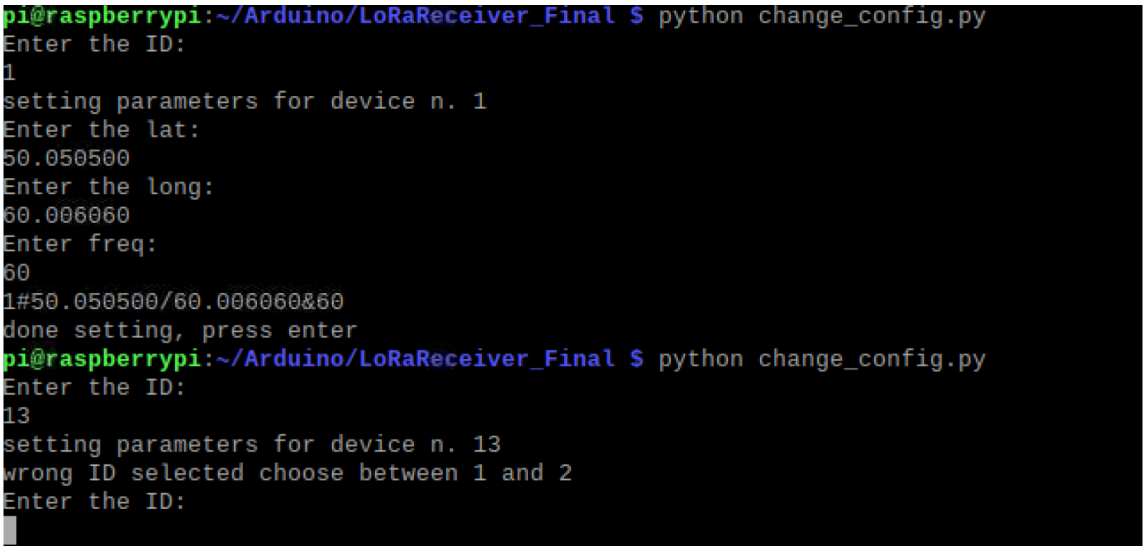
When Arduino releases the received data on the serial port, the Raspberry collects such data with Node-RED, a flow-based programming tool, in order to parse them and distinguish from which device they are coming from. At this point Node-RED organizes received measures in a JSON file to be uploaded to Influxdb later. Every JSON file is marked with a tag (i.e station1, station2) in order to let Influxdb be able to store data properly for the two stations. Soon after, a packet is uploaded to Influxdb, Node-RED checks the configuration file if any change of internal parameters(i.e frequency and GPS's coordinates) of ESPs is performed at the user side. The configuration file of the ESPs is stored inside the Raspberry and it's called *config.tx*, here are stored the parameters to be applied to the given ESP, it will be explained later.

The configuration parameters are then forwarded to the Arduino by printing them to the serial port in order to be sent back to the ESP that is waiting a reply from it. If the parameter modified do not attain to the ESP, that started the communication, then the packet sent will be the ID of the said device, even if there is no change to the config file. In this way, the ESP device receiving its ID can easily ignore it and continue with its workflow.

In order to let the user interact with the raspberry, it was opportunely set up a port forwarding of its services on the Internet: the user can access the Influx database on port 8086, Grafana on port 3000, and Node Red on port 1880. To do this, we went into the router configuration and opened the ports.

The access to the database is safeguarded by a strong password. In this way, the user can visualize the status of the field and all the statistics from wherever there is an internet connection and a device able to visualize web pages.

In order to provide the new parameters to be sent to the devices it was deployed a simple python script "*setting_parameter.py*" in order to modify the *config.txt*,which contains the parameters of a device in a string formatted with special characters, that is going to be read by NodeRED in order to perform the callback action and print in the serial port of the Arduino.
Finally this string is sent by the Dragino device, and when it's received by the ESP it is decoded thanks to the presence of the special characters between values.



```
pi@raspberrypi:~/Arduino/LoRaReceiver_Final $ python change_config.py
Enter the ID:
1
setting parameters for device n. 1
Enter the lat:
50.050500
Enter the long:
60.006060
Enter freq:
60
1#50.050500/60.006060&60
done setting, press enter
pi@raspberrypi:~/Arduino/LoRaReceiver_Final $ python change_config.py
Enter the ID:
13
setting parameters for device n. 13
wrong ID selected choose between 1 and 2
Enter the ID:
```

Figure 4: Program execution to change the configuration settings

3.5 InfluxDB,Grafana and Alerts

Initially we installed the Influx version 1.8, but after some research, we found that the version 2.0 was better suited for our project because of the new introduced Flux language that can easily filter the data by using specialized queries, also because the fbprophet library requires Influx 2.0 to work.
To organize our workflow, we started by creating a test bucket called "test1" in which we tested sending data from Node Red to Influxdb, afterward a bucket "myfield" was created to gather real data from the field and for the live demo at the exam. This bucket is going to be used by Grafana to produce graphs and also it is going to be the bucket from which the prophet script is going to gather data.
Our bucket contains the following values:

- moisture
- temperature
- humidity
- SOH
- gps_lat
- gps_long
- RSSI

While the ID of the node is contained in a tag called "tag1". We decided to keep the parameters of the GPS coordinates as values because we implemented the possibility to remotely change the coordinates of where the sensor station is placed. In this way we can visualize in Grafana the change of these in a live tile

To query the database we use the Flux language that gives us the possibility to request a certain type of value thanks to the "filter" function, while we can get data from a certain window of time thanks to the function "range".
In this way, with the same queries that we use to look at data in Influxdb, we can do the same in Grafana to create the graphs we are looking for.
Thanks to the versatility of Grafana, we were able to create various alerts for our field:

- Alert about temperature
- Alert about soil moisture when the plants need water
- Alert about soil moisture when the plants are watered

Moreover it is always possible to insert new devices in the network without interruption of the service, provided that, any new device is correctly set with a new ID at the time of the boot in order to not overwrite other's data. The forecasting procedure will automatically detect new devices and then take their data in order to increase the quality of the forecast.
To create the Telegram bot we used the "Botfather" bot that is freely available. Once created, we inserted the necessary parameters of the bot like the chat ID and the bot token to be able to receive alerts from it.
Grafana, when an alert is triggered, sends the command to the bot that will post a message in our private group chat.

3.6 fbrophet

In order to perform a forecast we employed the library ”fbprophet” which implements the forecasting procedure to predict time series evolution. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.
It can be used either in R or Python, but in order to be installed it has rigid requirements regards system architecture and module versions: the Prophet library, in order to be employed, requires an AMD64 architecture processor and a specific version for its module, since it is not released yet for the latest version. The Prophet's core module is ”Pystan”, that is an optimization module that requires the 64-bit architecture in order to be used and a specific version of it(i.e. ver.$\leq$ 3.0). Clearly that implied Raspberry couldn't handle the forecasting task since it has an ARM64 architecture, then another platform has been used in order to make the prophet library work.

We tried to install fbprophet's dependencies in Debian 10, but unfortunately we couldn't manage to make it work since Pystan requires at least Debian 11 and in the said OS it works with a precise version of Pystan and other required modules. Finally we settled with Windows 10 by creating an Anaconda environment of Python 3.7 in order to properly install fbprophet, pytstan and other required modules.

The *prophet.py* was the first attempt of generating a forecast of the downloaded data from InfluxDB and it performs a forecast of the provided value (i.e. temp,moisture and humidity) inside the query. Afterwards the Pystan optimizer tries to create a model of the collected data by running an optimization problem. The forecast is gathered inside a query to be uploaded to Influx and later be visualised in Grafana.

Afterwards a new version of the script was deployed *prophetV2.py* in order to be able to run the the prediction periodically, and decide for which values generate the forecast.
To manage the routine and the interface a *summoner_of_prophet.py* was created to let the user the freedom of defining which measures to forecast, the prediction windows and the interval between two prediction, which will be forwarded to prophet in order to provide predictions.

One of the main difficulties we encountered in deploying the prophet was the setting of the correct timezone. Influxdb takes the timezone from the Raspberry Pi settings and set it in UTC, but when the prophet libraries read the dataframe the part of the data where the timezone is stored is not accounted for so this information was lost making the time prediction shifted in the past of about 2 hours with respect to the current time.
For this reason we needed to impose a time shift of two hours (i.e. GMT +1) in order to make the correct prediction aligning with the time frame of the data. Also we automated the process of changing from and to DST since the time offset between CET and UTC changes on the basis of daylight saving time, this was opportunely fixed by checking a flag from the function `time.localtime()`.

Another issue encountered was the overlapping of new prediction with the past ones in the forecast data frame, then we implemented a way to flush the prediction data frame every time a new one is ready to be uploaded, to do so, we used an API query in order to tell Influxdb to delete all previous forecasts. The flush query was done at the beginning of the prophet function by means of the library *requests* that implements HTTP messages. Therefore a post message is forwarded to Influxdb with the command to flush all previous forecast records.

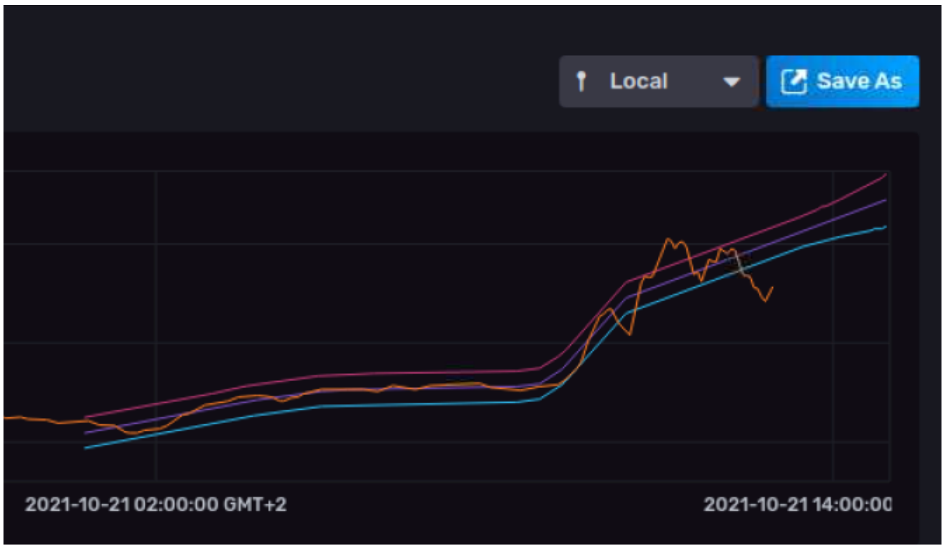


Figure 5: An example of the application of fbprophet

Finally, to make the prophet routine runnable in any x64 architecture without installing all the needed libraries we used *pyinstaller* library in order to compile the project and then export it in `.exe` extension which generates an executable program.

4 Results

In the end we developed a field monitoring suite:

- **Flexible**: We can expand the network with new sensors without modifying the code and be able to adapt to any kind of field, since the monitoring is done without any previous assumption on the field location. The location and the sample frequency of the sensors can be set remotely by launching a simple program.
- **Robust**: LoRa protocol offers a robust and reliable transmission over a large distance, our application with the current settings offers a coverage of up to 500m in urban environment (as we personally experimented), and by setting a syncword we are able to filter out messages from other devices.
- **Useful**: the Telegram notifications can warn us when the plants are in need of water or when it rains in the field. The Grafana dashboard lets us see all the data from any device that can run a browser. A simple program can set the prediction for any field of our data and do a live prediction with a refresh period settable on the fly.

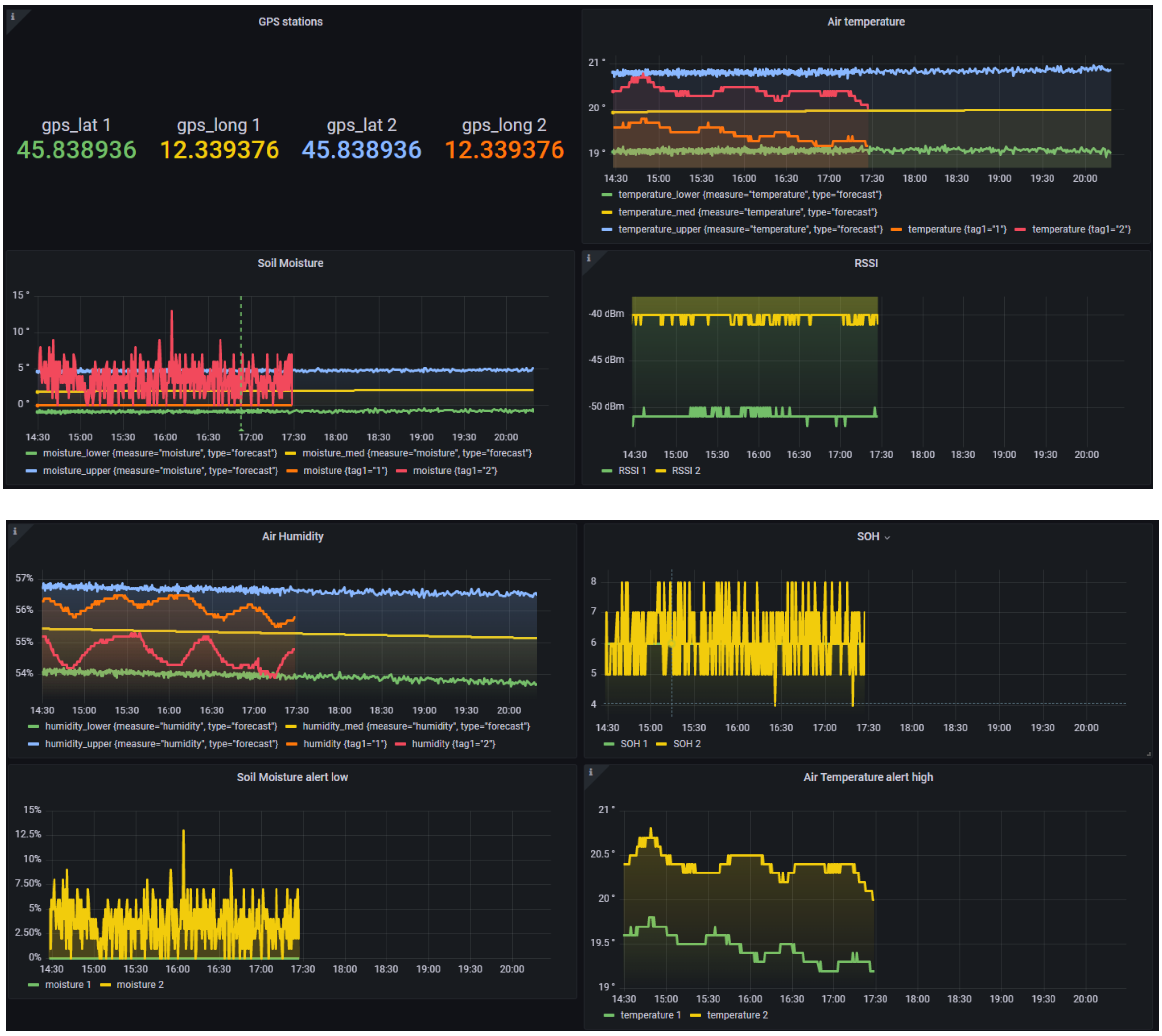
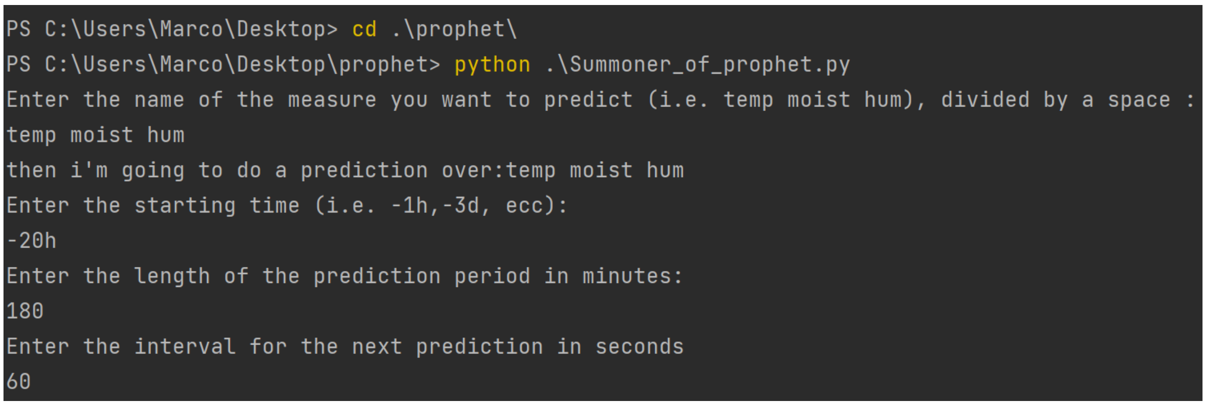

GPS stations
gps_lat 1 45.838936
gps_long 1 12.339376
gps_lat 2 45.838936
gps_long 2 12.339376
Air temperature
temperature_lower {measure="temperature", type="forecast"}
temperature_med {measure="temperature", type="forecast"}
temperature_upper {measure="temperature", type="forecast"}
temperature {tag1="1"}
temperature {tag1="2"}
Soil Moisture
moisture_lower {measure="moisture", type="forecast"}
moisture_med {measure="moisture", type="forecast"}
moisture_upper {measure="moisture", type="forecast"}
moisture {tag1="1"}
moisture {tag1="2"}
RSSI
RSSI 1
RSSI 2
Air Humidity
humidity_lower {measure="humidity", type="forecast"}
humidity_med {measure="humidity", type="forecast"}
humidity_upper {measure="humidity", type="forecast"}
humidity {tag1="1"}
humidity {tag1="2"}
SOH
SOH 1
SOH 2
Soil Moisture alert low
moisture 1
moisture 2
Air Temperature alert high
temperature 1
temperature 2

Figure 6: Grafana dashboard with predictions for the next 3 hours



```
PS C:\Users\Marco\Desktop> cd .\prophet\
PS C:\Users\Marco\Desktop\prophet> python .\Summoner_of_prophet.py
Enter the name of the measure you want to predict (i.e. temp moist hum), divided by a space :
temp moist hum
then i'm going to do a prediction over:temp moist hum
Enter the starting time (i.e. -1h,-3d, ecc):
-20h
Enter the length of the prediction period in minutes:
180
Enter the interval for the next prediction in seconds
60
```

Figure 7: An example of execution of the summoner

5 Acknowledgements

We'd like to thank Angelo Trotta and Marco Di Felice for the time they granted us and for the priceless help in fixing the fbprophet code.