# Weather monitoring wireless sensor network

Ivans Dorohovs (id14016)

Toms Ozols (to11018)

Oļegs Pešudovs (op14002)

May Htoo Maung (m-14003)

# General description

This “Weather monitoring wireless sensor network” project is a continued work of Reinholds Zviedris project, which he describes in his promotional work called “OBJECT MONITORING USING LOW POWER CONSUMPTION EMBEDDED DEVICES AND HETEROGENEOUS WIRELESS SENSOR NETWORKS”. In this promotional work he developed sensor network architecture and put together both sensor node and base station hardware parts.

Our goal was to develop software for three different hardware parts: texas instrument (for sensor node), arduino and raspberry pi (both for base station).

# Sensor Node

**Description:**

Sensor node will have all required sensors (wind speed and direction, air humidity and temperature, rain amount) attached to it and will gather and store data from them, occasionally sending it to the station node;

**Diagram of operation:**



**A description of sensors used:**

**TEMP, HUMIDITY, WIND DIRECTION, WIND SPEED, RAIN**

**FOR TEMPERATURE AND HUMIDITY:**

data pin: 11

library: SHT1x.h

global variables: float tempC; float tempF; float HUMIDITY;

Functions: void readSensor();

**FOR WIND DIRECTION:**

data pin: 14

Final data: float Direction;

global variable: float currentWindDirection;

float ADJUST3OR5; float PowerVoltage; in voltageToDegrees();

#define VARYVALUE 0.05 in fuzzyCompare();

functions: boolean fuzzyCompare(); float voltageToDegrees();

**FOR WIND SPEED:**

data pin: 15

Final data: float WindSpeed;

global variables: long currentWindCount;

unsigned long lastWindTime;

float currentWindSpeed;

unsigned long currentTime in serviceInterruptAnem();

functions: void serviceInterruptAnem();

**FOR RAIN:**

data pin: 13;

Final data: float totalRain();

global variables: long currentRainCount;

float sampleTime;

float startSampleTime;

unsigned long currentRainMin;

unsigned long lastRainTime;

float rain\_amount in flash();

unsigned long currentTime in serviceInterruptRain();

functions: void serviceInterruptRain();

**RADIO COMMUNICATION:**

data pin: 5, 2;

library: SoftwareSerial.h

Data: String RadioData;

Global variables: SoftwareSerial RT; // instance

# Station Node

**Description:**

Station node receives information from the sensor node, uploads it to the web and stores it locally. It operates on a raspberry pi microcontroller, but since it has no option to turn itself on and off (duty cycle), it also requires help from arduino, to do it instead.

**Arduino flow:**



**Pi flow:**



**Local storage:**

For storing the received measurements locally, an Sqlite3 database solution was used.

Sqlite3 doesn’t support date time types, so instead a text type was used for storing the measurement timestamp.

Table model:

|  |  |  |
| --- | --- | --- |
| **Property name** | **Type** | **Description** |
| timestamp | TEXT | The date and time of the measurement |
| temperature | REAL | Temperature reading |
| humidity | REAL | Humidity reading |
| windDirection | INT | An integer representation of the wind direction reading |
| windSpeed | REAL | Wind speed reading |
| rain | REAL | Rain |

**Plotly REST call examples:**

* Endpoints:

For communication with plot.ly, its grid API was used.

<https://api.plot.ly/v2/grids> - as the base URL.

* Authentification:

Uses HTTP basic authentication (might be deprecated and migrated to OAuth 2.0 in the future)

Requires a tuple of the plot.ly user name and api key.

* Plotly request headers:  
  Plotly calls require only one client defined request header:  
  'Plotly-Client-Platform' – which is the currently used client language/environment.  
  In our case it is: ‘python’
* Plotly grid creation request:  
  Post request to the base URL, with the example request body.  
  From the response, We take the Id of the newly created grid and store it in configuration.  
  Request body:

"data": {

"cols":{

"Timestamp":{ "data": [], "order": 0 },

"Temperature":{ "data": [], "order": 1 },

"Humidity":{ "data": [], "order": 2 },

"Wind direction":{ "data": [], "order": 3 },

"Wind speed":{ "data": [], "order": 4 },

"Rain":{ "data": [], "order": 5 }

}

}

* Plotly add rows to grid request body:  
  Post request to the base url/ + the grid identifier (eg: “BstGrupa5:6”)/ + “row” – using the following request body.  
  If the request is unsuccessful we send a new request to create a new grid.  
  Plotly only accepts numeric values, so We’re converting the measurement date and time to a unix timestamp.

Each value inside of each array, represents a value for a column by its order.

Request body:

"rows": {

[1484675464, 20.1, 100.1, 1, 28.1, 33.3],

[1484675499, 21.8, 101.2, 1, 25.3, 33.3],

[1484675563, 18.5, 110.1, 3, 25.8, 33.4]

}

**Python libraries used:**

* RPi.GPIO:
  + Used to access the raspberry pi gpio ports.
* Serial:
  + Used to access the UART serial port.
* Json:
  + A library to simplify work with json data objects
* Sqlite3:
  + A light-weight local server-less database script.
  + Used to store received measurements on the Pi.
* Requests:
  + An Http client library.
  + Used to perform Http calls to Plot.ly.

# References

1. <http://reinholds.zviedris.lv/wiki/_media/rzviedris-thesis-summary-11-08-2015.pdf> - “Object monitoring using low power consumption embedded devices and heterogeneous wireless sensor networks” by Reinholds Zviedris.