

CERN

SUMMER STUDENT PROGRAMME

PROJECT REPORT

Monitoring System for ALICE Surface Areas

Author:

Oğuz Demirbaşı

Supervisors:

Dr. Ombretta Pinazza

Peter Matthew Bond

September 16, 2016

Contents

1	Introduction	2
2	Development Process	2
2.1	Collecting Sensor Data	2
2.2	Sending Data Using DIM	7
2.3	DIM Server	7
2.4	Developing the Monitoring Application	8

1 Introduction

I have been at CERN for 12 weeks within the scope of Summer Student Programme working on a monitoring system project for surface areas of the ALICE experiment during this period of time. The development and implementation of a monitoring system for environmental parameters in the accessible areas where a cheap hardware setup can be deployed were aim of this project.

The project was developed in 4 phases as shown in figure 1:

- Mount temperature, humidity and air quality sensors on a breadboard and read them from arduino board
- Setup a raspberry PI to read the data produced by Arduino from the serial port
- Program a server-client system using DIM (Distributed Information Management System) to serve the sensor data to the final application
- Prepare a SCADA application able to receive the data on DIM, archive and trend the volumes.

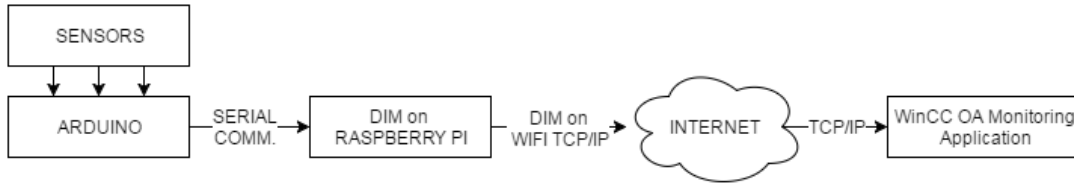


Figure 1: Workflow diagram of the project

2 Development Process

2.1 Collecting Sensor Data

Arduino board located in the center of first part of the project. Arduino UNO WiFi Edition was used and sensors were connected to that. Then a script which collects data from sensors and sends them via serial port was written and uploaded to the board. The code can be found in appendix section.

Totally five sensors were used in this project. Three of them are different temperature sensors (10K ohm thermistor, KY-013 and LM35 from Texas Instruments), one

humidity sensor (hih 4000 from Honeywell) and one air quality sensor (micro-e air quality click) shown in figure 2. 10K ohm resistor was used since thermistor can not be connected to Arduino directly, circuit schematic of thermistor is shown in figure 3.



Figure 2: Sensors which used in the project

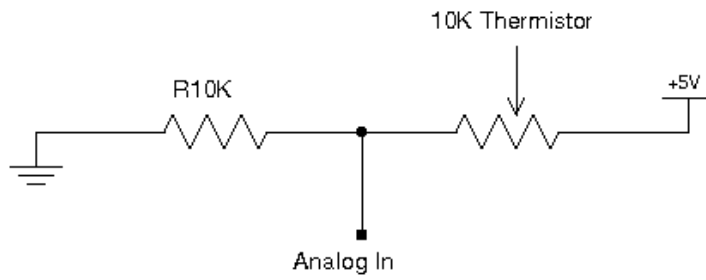


Figure 3: Circuit schematic of thermistor

Connection of KY-13, LM35 and hih-4000 as 3-legged sensors are easier to connect than a thermistor, one leg for ground, another leg for +5V and the other leg for analog input. Circuit design is shown in figure 4. Differently from the other sensors, air quality sensor is designed in micro-bus standard. So it is planned to be installed on a micro-bus click shield (shown in figure 5) for Arduino at the start but it did not work since the click shield and the other analog inputs can not be connected to Arduino at the same time. Then it soldered and connected to Arduino directly on the breadboard as shown in official documentation (Figure 6).

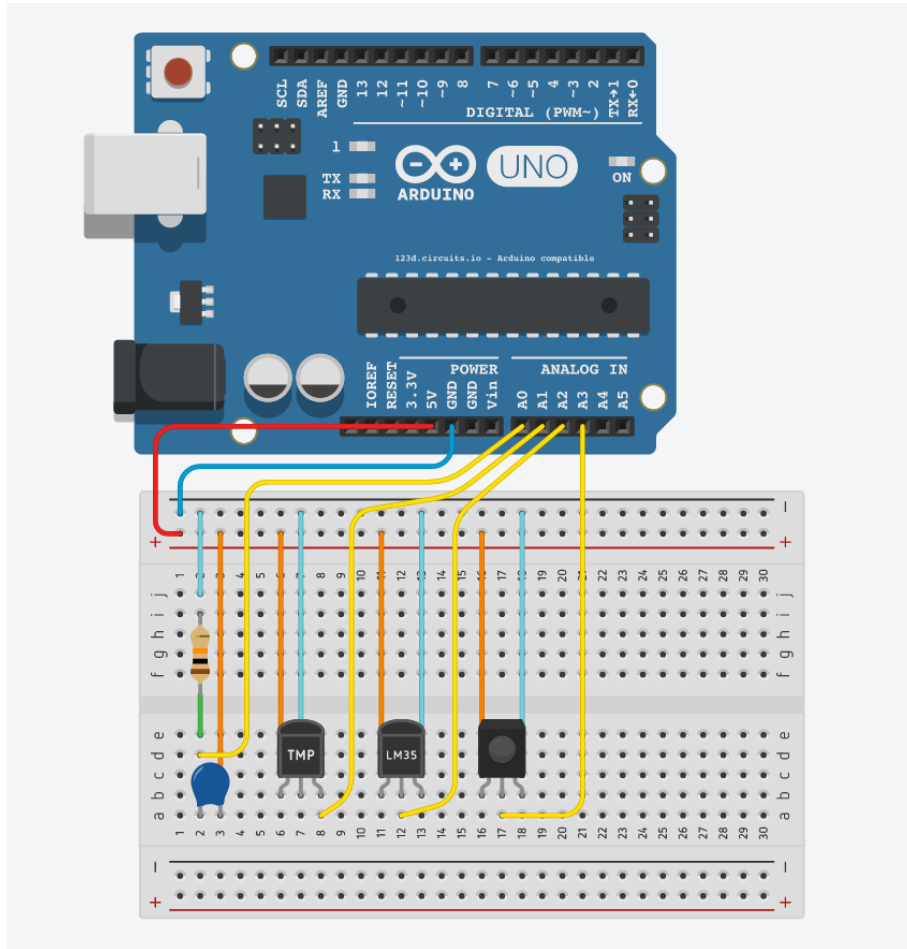


Figure 4: Circuit design of temperature and humidity sensors

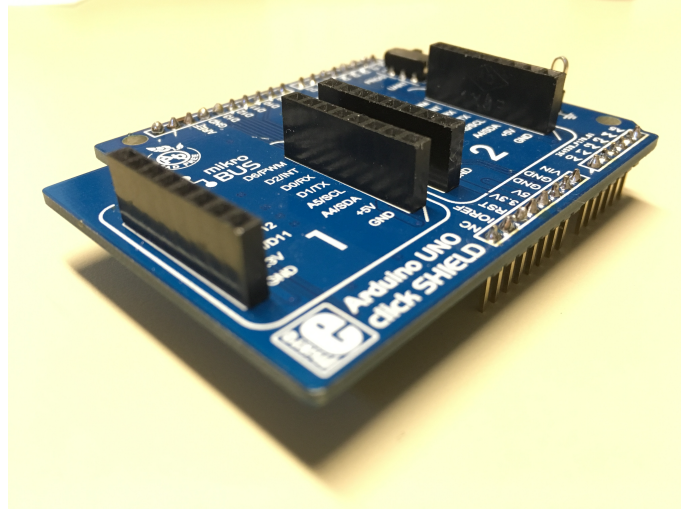


Figure 5: Click shield for Arduino

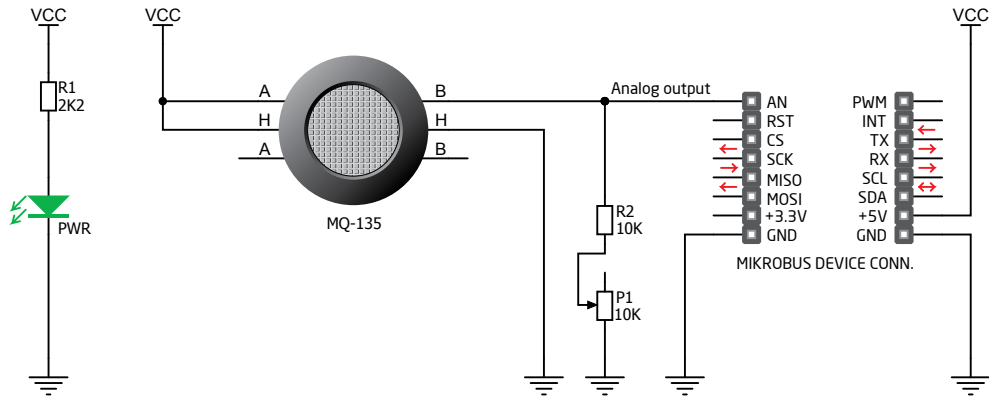


Figure 6: Air quality click circuit connection schema [3]

After all of the sensor connections, the Arduino code reading and calculating sensor data was completed. The last step in this part of project is showing all data in a LCD display in sequence. Standard 16×2 LCD display was used to do it. Legs were soldered to board of display, and the connection to the Arduino was completed. 10K potentiometer was used in this circuit. The final circuit is shown in figure 7.

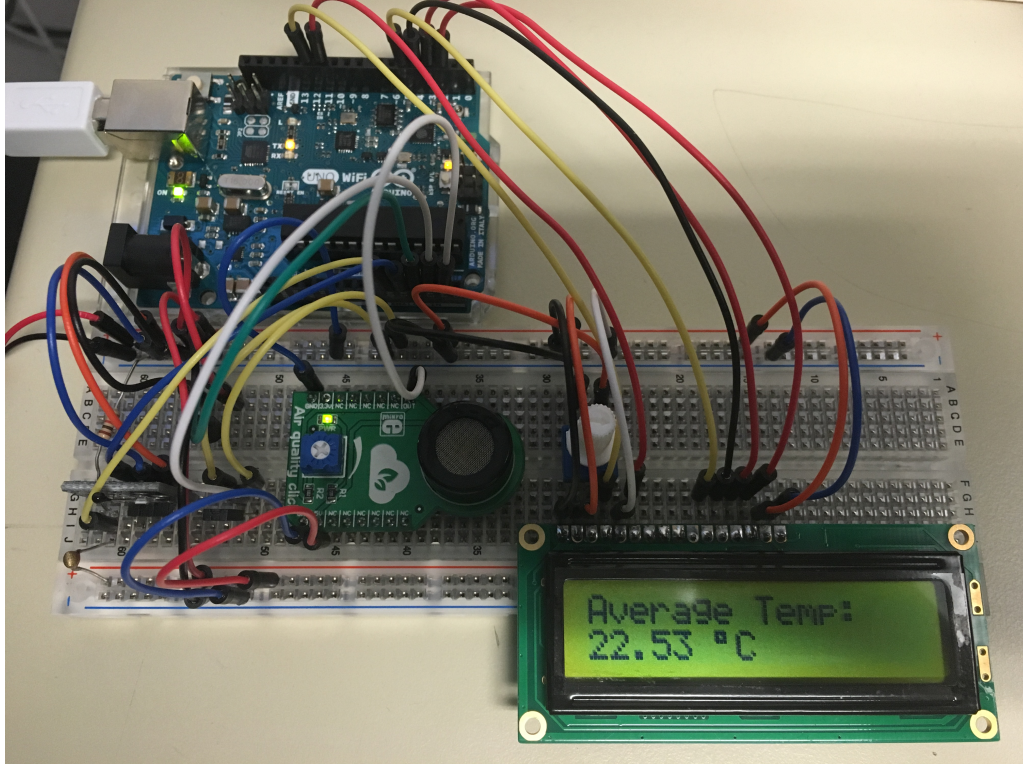


Figure 7: Final circuit

2.2 Sending Data Using DIM

A Raspberry Pi 2 computer was used for collecting sensor data coming from the serial port and sending them to a DIM server. Firstly GCC and G++ libraries were installed on Raspberry Pi and then PyDim library was installed. A problem about library access was facing when the first software run. A library linker (Path of the library files) was added to “/etc/ld.so.conf” in order to avoid that problem. Then the Python code was written. Developing a python code was not too complicated since some functions are already included for capturing serial data.

It was decided to develop a C++ code which does the same as the Python code since C++ is a more robust language. It was a bit more complicated than the Python code. Some special libraries for serial communications were installed first. Then a code that captures serial data was written. After that “dis.hxx” library that comes with DIM library was included and the rest of code which sends data to a DIM server was written. You can find all the codes in appendix section. The important point is that a variable can be sent directly in Python but a struct must be used in C++. All details can be found in the codes in appendix.

2.3 DIM Server

In this project, Distributed Information Management System was used for client server communication. In order to use this protocol both the client must have DIM Library and the server applications have to be run on the server computer. Client part was mentioned in 2.2. Server applications come with the PyDim package. A Dim Node Server (DNS) id the package allows clients to find services published trough the DIM protocol, shown in figure 8. “webdid.exe” activates a web page for monitoring data from the server. (“/bin/webdid.exe” didn’t work and “/Web-DID/webdid.exe” has been used.)

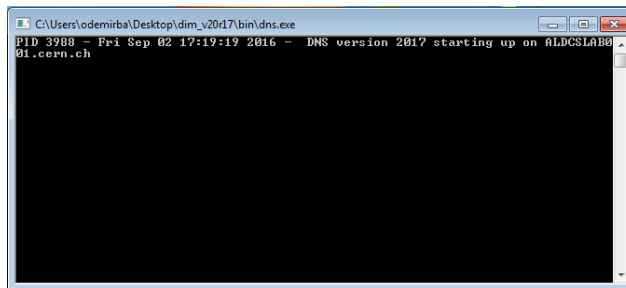


Figure 8: DIM server window

2.4 Developing the Monitoring Application

For the last part of the project, designing a monitoring application, a user interface was designed on WinCC OA, a SCADA software from Siemens, used at CERN by all experiments control systems. A framework named FwDim allowing WinCC OA to receive data from DIM server was used. This framework has a user interface and all of the connection settings and connection between data in DIM servers and clients on WinCC OA can be configured with this interface. One of the FwDim windows is shown in figure 9. Archival of data was activated for datapoints, to show temperature and humidity trends. At the end, monitoring application plotting data recieved via was designed as shown in figure 10.

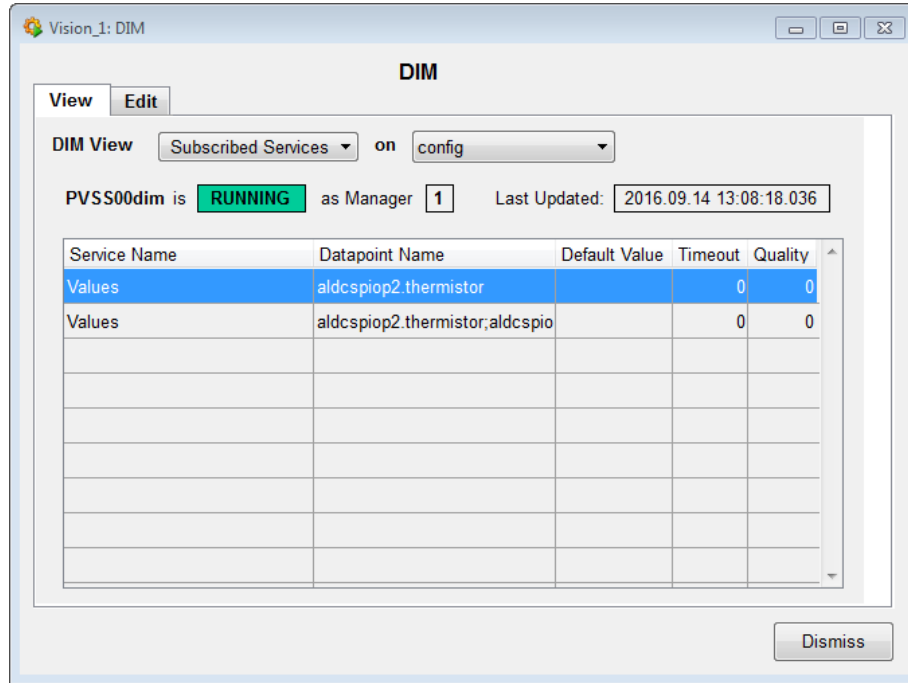


Figure 9: The FwDim window showing situation of DIM connection and value-datapoint connections

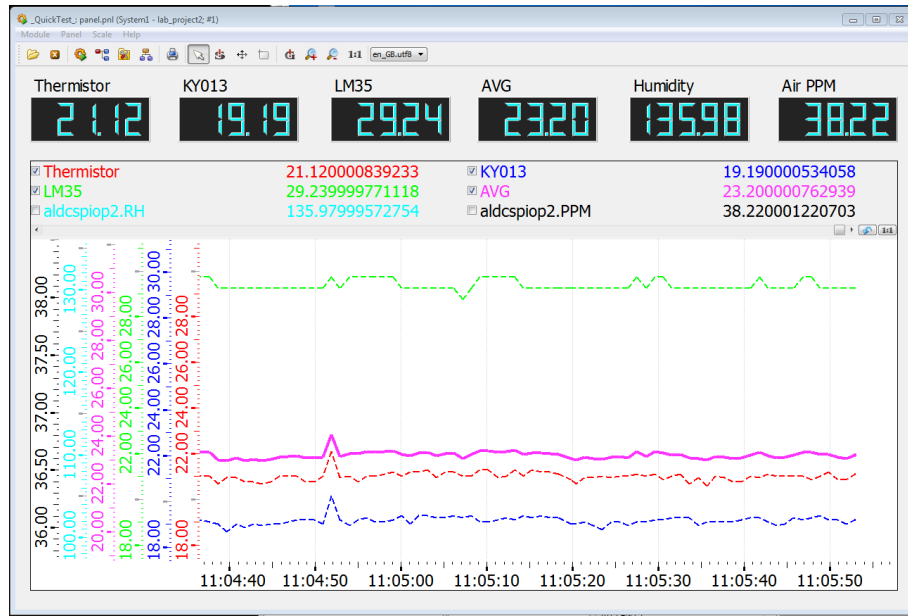


Figure 10: Monitoring application

References

- [1] 2pcs keyes ky - 013 vc temperature sensor module the arduino avr pic new — what's it worth. <http://www.terapeak.com/worth/2pcs-keyes-ky-013-vc-temperature-sensor-module-the-arduino-avr-pic-new/252335451270/>. (Accessed on 08/22/2016).
- [2] Air quality click - mikrobus board with mq-135 sensor for detecting gases that impact air quality. <http://www.mikroe.com/click/air-quality/>. (Accessed on 08/22/2016).
- [3] air-quality-click-manual-v100.pdf. <http://download.mikroe.com/documents/add-on-boards/click/air-quality/air-quality-click-manual-v100.pdf>. (Accessed on 08/30/2016).
- [4] [bella]honeywell (honeywell) and humidity sensor hih 4000 003 hih4000 003 5pcs/lot-in sensors from electronic components & supplies on aliexpress.com — alibaba group. <https://www.aliexpress.com/item/HHH-4000-003-HIH4000-003-5PCS-LOT/2052007509.html>. (Accessed on 08/22/2016).
- [5] Sensors :: Temperature :: Lm35 temperature sensor. <http://www.geeker.co.nz/sensors/temperature/lm35-temperature-sensor.html>. (Accessed on 08/22/2016).

Appendix

Python Codes

```

1 import serial
2 import sys
3 import pydim
4 import time
5 from datetime import datetime
6
7 def opsub(tag):
8     try:
9         return(v,)
10    except KeyboardInterrupt:
11        pydim.dis_stop_serving(svc)
12        sys.exit()
13
14 global v
15 v="init=0.000"
16 ser = serial.Serial('/dev/ttyACM0', 9600)
17 svc = pydim.dis_add_service('op_aldcspiop4','C:200', opsub, 0)
18
19 if not svc:
20     sys.stderr.write("Error registering service")
21     sys.exit(1)
22 pydim.dis_update_service(svc)
23 pydim.dis_start_serving('op_aldcspiop4')
24
25 while True:
26     try:
27         v = ser.readline()
28         pydim.dis_update_service(svc)
29     except KeyboardInterrupt:
30         pydim.dis_stop_serving(svc)
31         sys.exit()

```

C++ Codes

```

1 #include <sys/socket.h>
2 #include <netinet/in.h>
3 #include <netdb.h>
4 #include <arpa/inet.h>
5
6 #include <pthread.h>
7
8 #include <cstdio>
9 #include <errno.h>

```

```

10 #include <fcntl.h>
11 #include <unistd.h>
12 #include <termios.h>
13 #include <sys/ioctl.h>
14 #include <iostream>
15 #include <stdlib.h>
16 #include <dis.hxx>
17
18 using namespace std;
19
20 #define BAUDRATE B9600 /* 9600 Bits Per Second – it must be the same as
    serial baud of Arduino */
21 #define MODEMDEVICE "/dev/ttyACM0" /* Serial Data Source */
22 #define _POSIX_SOURCE 1 /* POSIX compliant source */
23 #define FALSE 0
24 #define TRUE 1
25
26 #define DIMSERVER "ALDCSLAB001.cern.ch" /* Dim Server Adress */
27 #define BUFFERSIZE 255000 /* Buffer Size */
28
29 #define TIME 0
30 #define MIN 70
31
32 volatile int STOP=FALSE;
33
34 struct SerialData {
35     float thermistor;
36     float KY013;
37     float LM35;
38     float AVG;
39     float RH;
40     float PPM;
41 } streamData;
42
43 /* The function that splits string parameter and sets SerialData struct
    values */
44
45 void splitSerialDataToStruct(char data[]) {
46     char * pch;
47     pch = strtok (data, "&");
48     pch = strtok (NULL, "&");
49     streamData.thermistor = atof(pch);
50     pch = strtok (NULL, "&");
51     pch = strtok (NULL, "&");
52     streamData.KY013 = atof(pch);
53     pch = strtok (NULL, "&");
54     pch = strtok (NULL, "&");
55     streamData.LM35 = atof(pch);
56     pch = strtok (NULL, "&");

```

```

57 pch = strtok (NULL, "=");
58 streamData.AVG = atof(pch);
59 pch = strtok (NULL, "=");
60 pch = strtok (NULL, "=");
61 streamData.RH = atof(pch);
62 pch = strtok (NULL, "=");
63 pch = strtok (NULL, "=");
64 streamData.PPM = atof(pch);
65 }
66
67 main() {
68     int fd, c, res;
69     struct termios oldtio, newtio;
70     char buf[BUFFERSIZE];
71
72     cout << "Opening Port..." << endl;
73
74     fd = open(MODEMDEVICE, ORDWR | O_NOCTTY);
75     if (fd < 0) {
76         perror(MODEMDEVICE);
77         exit(-1);
78     }
79
80     cout << "Port is opened" << endl;
81
82     tcgetattr(fd, &oldtio); /* save current port settings */
83
84     bzero(&newtio, sizeof(newtio));
85     newtio.c_cflag = BAUDRATE | CRTSCTS | CS8 | CLOCAL | CREAD;
86     newtio.c_iflag = IGNPAR;
87     newtio.c_oflag = 0;
88
89     /* set input mode (non-canonical, no echo,...) */
90     newtio.c_lflag = 0;
91
92     newtio.c_cc[VTIME] = TIME; /* inter-character timer unused */
93     newtio.c_cc[VMIN] = MIN; /* blocking read until 5 chars
94     received */
95
96     tcflush(fd, TCIFLUSH);
97     tcsetattr(fd, TCSANOW, &newtio);
98     cout << "Ready to read serial data" << endl;
99
100     DimService service("Values", "F:6", &streamData, 24);
101     DimServer::setDnsNode(DIMSERVER);
102     DimServer::start("MonitoringData");
103     cout << "DIM Server Connection Activated" << endl;
104
105     cout << "Sending Data..." << endl;

```

```
105 while (1) { /* loop for input */
106     res = read(fd, buf, BUFFERSIZE); /* returns after 5 chars have been
    input */
107     buf[res]=0; /* so we can printf... */
108     //printf(":%s:%d\n", buf, res);
109     splitSerialDataToStruct(buf);
110     memset(buf, 0, sizeof(buf)); /* Clear Buffer */
111     //cout << "asd: " << streamData.AVG << endl;
112     service.updateService(); /* Update service with new values */
113     usleep(500);
114 }
115 tcsetattr(fd, TCSANOW, &oldtio);
116 }
```