Kai Kukasch, Bastien Gouila

Room Surveillance

Internet of Things Project

Helsinki Metropolia University of Applied Sciences
Degree

Information Technology

Project Report

12/15/2016

Helsinki Metropolia University of Applied Sciences

Author(s)	Kai Kukasch Bastien Gouila
Title	Room Surveillance
Number of Pages Date	8 pages + 1 Appendix December 15th 2016
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Smart Systems
Instructor(s)	Sami Sainio, Instructor & Lecturer Joseph Hotchkiss, Instructor

Contents

1.1 Project Specifications

The purpose of this project is to design and implement an IoT (Internet of Things) application by using an embedded microcontroller and sensors. We decide to add a Raspberry Pi camera to try to add video to our room surveillance. The complete will be able to communicate with a database and to access the data from a web page.

The system's components include:

- Microcontroller (Digi ConnectCore 6 i.MX6 Single Board Computer)
- Sensors:
  - Light Dependent Resistor
  - Door Contact Switch
  - Humidity (Honeywell HIH8120)
  - Temperature (Honeywell HIH8120)
- Raspberry Pi camera
- Database

The microcomputer is the primary component in this system, we decided to install the Operating System Android 5.1 and program an application on it. It controls and recovers the data and sends them to the database. On the Android application the user is able to select the sensors that he wants to see. The camera will be added at the end as an improvement of the project if the project is finished before December,15th 2016.

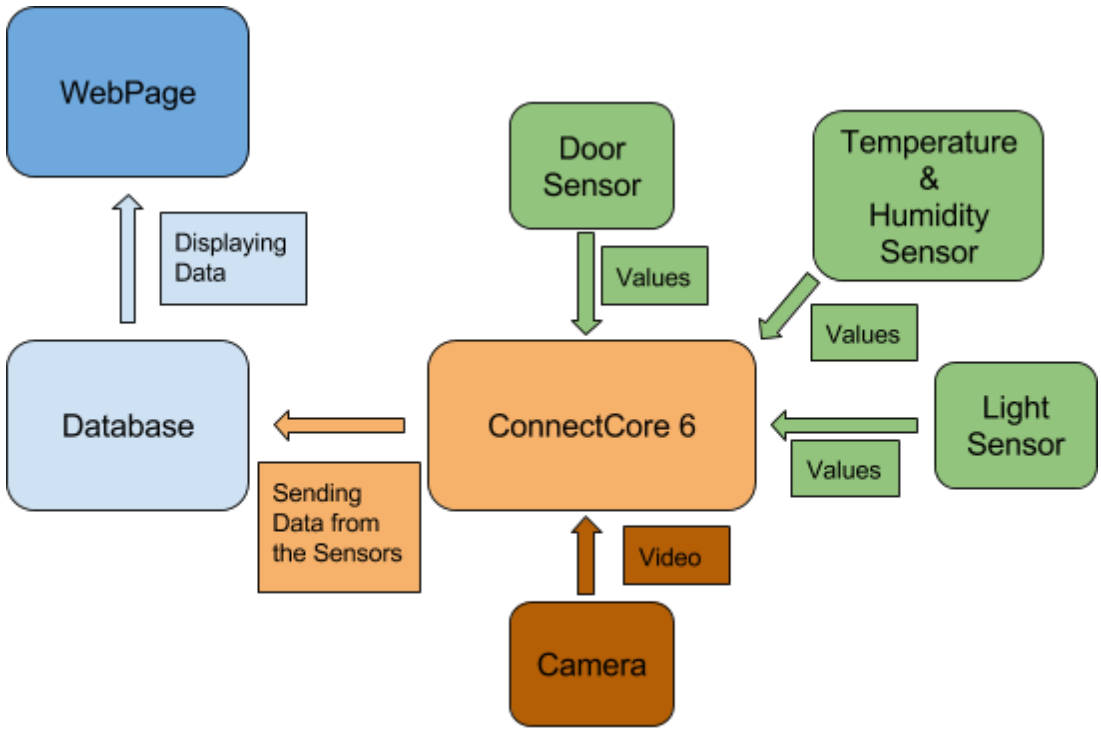

Figure 1: Project Schematic

over RS-232 was required. Hint: For the interconnection a cross-over RS-232 or null-modem cable has to be used. To program all the functions on Android we used Android Studio IDE Version 2.2.2 with the Digi ConnectCore 6 Library for Android. For flashing and debugging our program on the ConnectCore we used an USB Cable. The Android OS surface was displayed via HDMI on a computer screen and was accessed with a USB keyboard and mouse.
The door and the light sensors are connected to the GPIO port and the temperature and humidity sensor is connected to the I2C port of the ConnectCore.

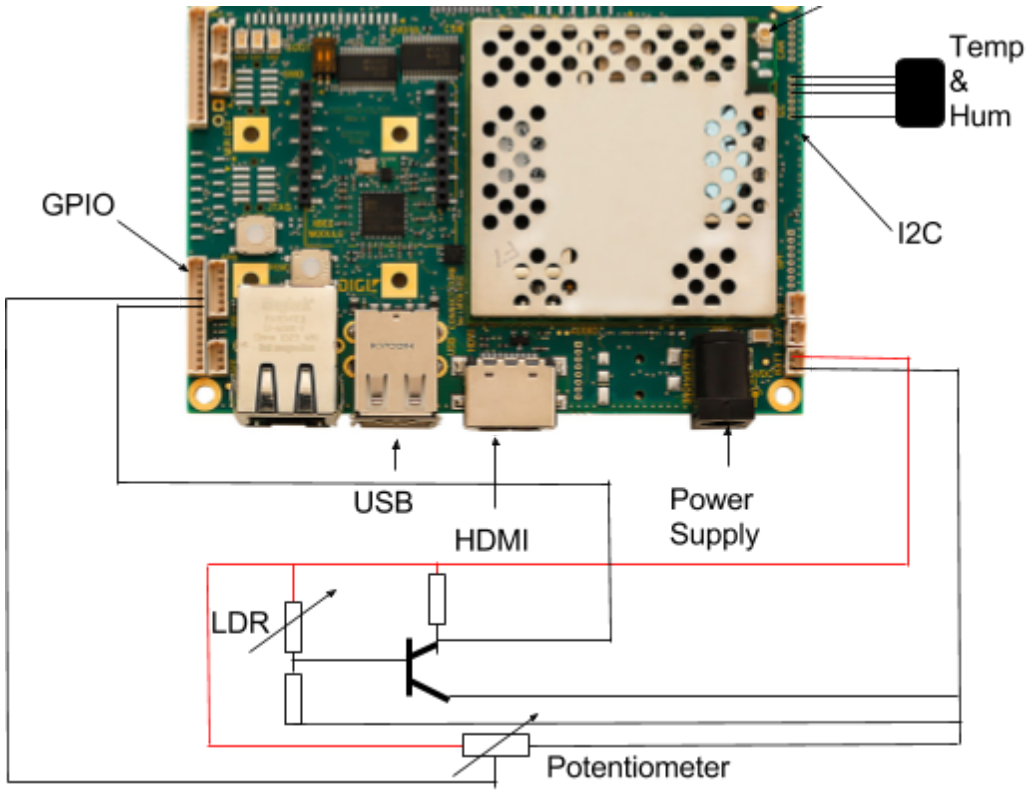

Figure 2: Contact Plan

1.2 Team

The team is composed of two students and we splitted our tasks as followed:

Bastien Gouila: Door and Light Sensor (GPIO) + Android Global application + Database + Camera.

Kai Kukasch: Temperature and Humidity sensor (I2C) + Database & web interface + Camera.

2 2 Sensors implementation

2.1 Section Overview

In this section we are going to explain how we implemented the sensors in our application and what protocols we used.

2.2 Door and Light

2.2.1 Components Overview

The idea of this sensor is detect if the door is open in the room or close by using a contact switch located on the door. In order to simplify the system, we simulate the contact with a potentiometer that we had at our disposal.

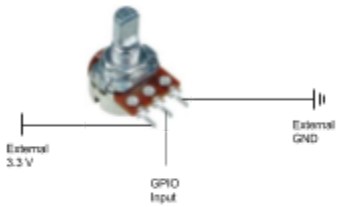

Figure 3: Wiring potentiometer to simulate a door contact switch

The variation of the resistance of the potentiometer offers us to simulate the door opening system by making a voltage variation between 3.3V and 0V.
The total resistance between the two extremity pin of the potentiometer is 10k Ohms and the middle pin connected to the GPIO input is acting like an oscilloscope probe.

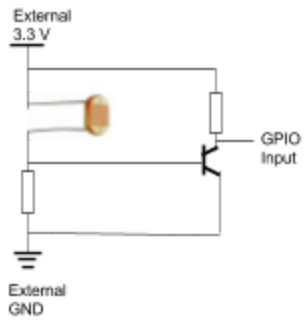

Figure 4: Wiring LDR to contact with GPIOs

The light sensor is done with an LDR. The luminosity is creating a voltage variation on the voltage bridge divider enough to activate or not the base of the transistor.
The transistor will allowed us to send a 3.3V

2.2.2 Using GPIO

For using the GPIO with the ConnectCore running with Android OS there are some parameters and signal to check with the hardware.

To use the GPIO with Android as input, you need to include the GPIO library, then, you need to create a GPIO manager and then a GPIO object. You need also to know the Android GPIO Pin number.

Example:

```
GPIO manager gpiomanager= new GPIOmanager(this);
GPIO DoorGPIO = gpiomanager.createGPIO(GPIO_Android_Pin, GPIOMode.INPUT);
```

To collect the data, you need to create a GPIO value object linked to your GPIO object and then you read the data from it.

Example:

```
GPIOValue DoorValue = DoorGPIO.getValue();
int iValueDoor = DoorValue.getValue();
```

The GPIO pin on board are located here:

Figure 5: Connect Core GPIO Port

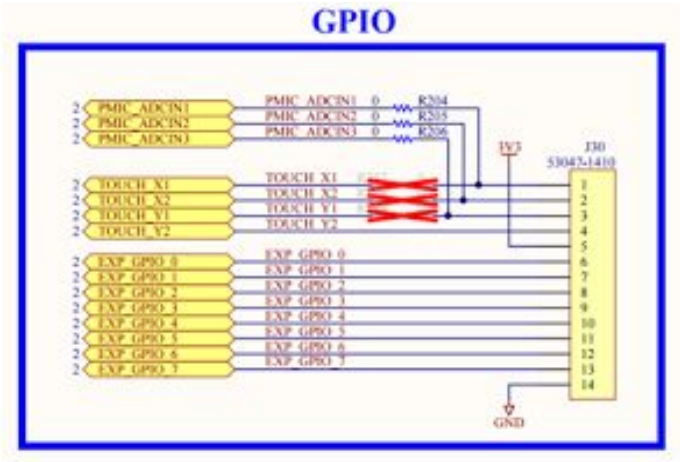

Figure 6: GPIO Port Pins

The internal GPIO pins and ports are listed here:

Signal Name	GPIO	Use
CSI0_DATA_EN	GPIO_5_20	CSI0_GPIO
EIM_A25	GPIO_5_2	CSI1_GPIO
EIM_CS0	GPIO_2_23	SPI1_IRQ_N
EIM_CS1	GPIO_2_24	EXP_GPIO_3
EIM_D23	GPIO_3_23	LVDS1_PEN_IRQ_N
EIM_D27	GPIO_3_27	XBEE_ONSLEEP_N
EIM_D28	GPIO_3_28	XBEE_RESET_N
EIM_D29	GPIO_3_29	XBEE_SLEEP_RQ
EIM_DA10	GPIO_3_10	USB_HUB_RESET_N
EIM_DA15	GPIO_3_15	CSI1_RESET_N
EIM_EB0	GPIO_2_28	EXP_GPIO_4
EIM_EB1	GPIO_2_29	EXP_GPIO_5
EIM_LBA	GPIO_2_27	PCIe_VCC_PG
EIM_OE	GPIO_2_25	AUD_PWR_EN
EIM_RW	GPIO_2_26	UART_PWR_EN
EIM_WAIT	GPIO_5_0	CSI0_RESET_N
ENET_CRS_DV	GPIO_1_25	RGMII_RESET_N
ENET_TX_EN	GPIO_1_28	RGMII_INT_N
GPIO_2	GPIO_1_2	CAN1_STBY
GPIO_4	GPIO_1_4	PCIE_DIS_N
GPIO_5	GPIO_1_5	CAN2_STBY
GPIO_9	GPIO_1_9	BT_DISABLE_N
GPIO_16	GPIO_7_11	LVDS0_PEN_IRQ_N
GPIO_18	GPIO_7_13	EXP_GPIO_6
GPIO_19	GPIO_4_5	EXP_GPIO_7
NANDF_ALE	GPIO_6_8	XBEE_IDENT
NANDF_CS2	GPIO_6_15	EXP_I2C_IRQ_N
NANDF_CS3	GPIO_6_16	EXP_I2C_GPIO
NANDF_D0	GPIO_2_0	AUD_HP_DET
NANDF_D1	GPIO_2_1	DISP_IRQ
NANDF_D2	GPIO_2_2	USER_LED0
NANDF_D3	GPIO_2_3	USER_LED1
NANDF_D4	GPIO_2_4	USER_LED2
NANDF_D5	GPIO_2_5	EXT_GPIO_0
NANDF_D6	GPIO_2_6	EXT_GPIO_1
NANDF_D7	GPIO_2_7	EXT_GPIO_2
NANDF_RB0	GPIO_6_10	PCIe_VCC_EN
SD3_DAT2	GPIO_7_6	CSI_RESET_N
SD3_DAT3	GPIO_7_7	PCIE_WAKE_N
SD3_RST	GPIO_7_8	PCIE_RESET_N
PMIC_GPIO7	PMIC_GPIO7	PWR_EN

In order to use the GPIO in our Android code we need to find the external signal linked to the GPIO external board pin.
The external signals are connected to the internal signals of the processor thanks to multiplexers.
We need to find what internal GPIO signal of the processor has been linked to this external signal.

For our application:
The door sensor is connected to the external GPIO board pin number 10. In the schematic we see that it is connected to the external signal EXT_GPIO_4. Internally, EXT_GPIO_4, is connected to the GPIO_2_28 processor signal according to the GPIO table.
The door sensor is connected to the external GPIO board pin number 9. In the schematic we see that it is connected to the external signal EXT_GPIO_3. Internally, EXT_GPIO_3, is connected to the GPIO_2_24 processor signal according to the GPIO table.

Apply this formula for Android GPIO pin = ((GPIO internal port-1)*32) + GPIO internal pin.
Door Sensor Android Pin = ((2 - 1) * 32) +28 = 60
Light Sensor Android Pin = ((2 - 1) * 32) +24 = 56

Figure 7: Table GPIO

2.3 Temperature and humidity

2.2.1 Components Overview

The sensor we use is the HIH8120 by Honeywell. It uses a Thermoset-polymer capacitive sensor to get the humidity and temperature information. The capacitive humidity sensor measures relative humidity by placing a thin strip of metal oxide between two electrodes. The metal oxide's electrical capacity changes with the atmosphere's relative humidity. The combination of humidity and temperature sensor allows to get a temperature compensated humidity measurement.

To monitor the temperature and humidity in the room we are using a 2in1 temperature and humidity sensor HIH8120 by Honeywell. For data transfer the I2C bus protocol has to be used. Before implementing the sensor in our software, the wiring has to be done like in the following figure.

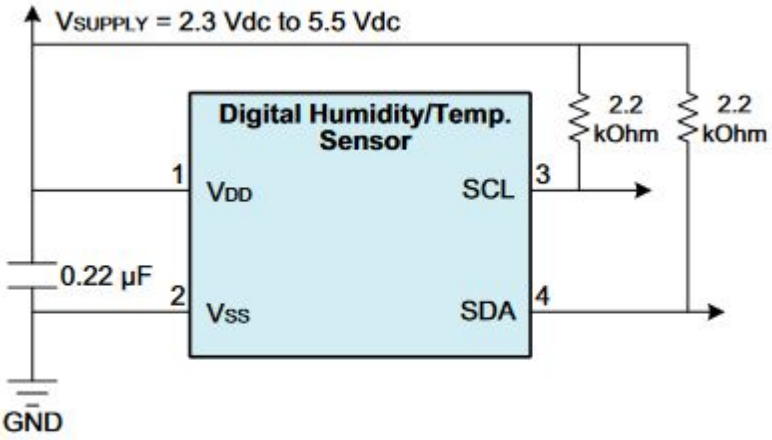

Figure 8: Wiring HIH-8120

(Source:http://www.farnell.com/datasheets/1863542.pdf?_ga=1.263477260.1012292472.1477913532)

2.2.2 Implementing the I2C Connection

For getting the sensor data over I2C, we used the Digi ConnectCore Library. More specifically we needed the libraries: `com.digi.android.i2c.I2C` and `com.digi.android.i2c.I2CManager`. To open the I2C connection we first needed two instances, one of the class I2CManager and I2C, which are from the imported libraries. The I2CManager class is to used create the I2C Interface with the correct device. With the I2C Interface the I2C Connection can be opened with the I2C library function `open()`. For getting the measurement data from the sensor, a temperature and humidity measurement request has to be sent from the ConnectCore to the HIH8120 as described in the following schematic of the sensors datasheet:

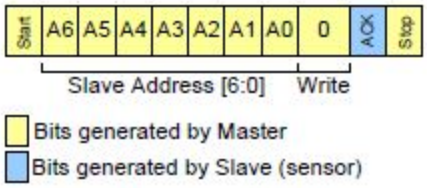

Figure 9: Datagram I2C Writing Sequence (Source: Technical Notes, I2C Communication with the Honeywell HumidIcon™ Digital Humidity/Temperature Sensors)

The request can be made of writing the 7 bit sensors slave address followed by a logic zero, what stands for a writing command. For reading the data the master (ConnectCore) has to send the slave address followed by a logic “1” as shown in the next figure from the datasheet. The data bytes will be send from the sensor after this command.

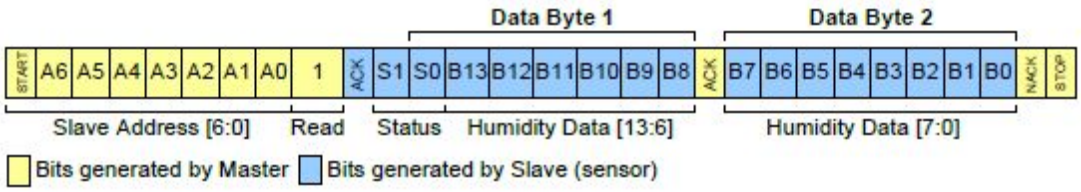

Figure 10: Datagram I2C Reading Sequence (Source: Technical Notes, I2C Communication with the Honeywell HumidIcon™ Digital Humidity/Temperature Sensors)

In our implementation we used the I2C library functions `void write(int deviceAddress, byte[] data)` and `byte[] read(int deviceAddress, int numBytes)`. After receiving the telemetry the data byte array has to be splitted and mathematically converted to its final value. The following conversion formulas can be taken from the HIH8120 datasheet:

[1] Conversion Formulas:

14 bit ADC output for humidity to %RH:

$$\text{Humidity (\%RH)} = \frac{\text{Humidity\_14\_bit\_ADC\_output}}{2^{14}-2} \times 100$$

14 bit ADC output for temperature conversion to °C:

$$\text{Temperature (C°)} = \frac{\text{Temperature\_14\_bit\_ADC\_output}}{2^{14}-2} \times 165 - 40$$

Figure 11: Conversion Formulas HIH-8120

(Source:http://www.farnell.com/datasheets/1863542.pdf?_ga=1.263477260.1012292472.1477913532)

3 ConnectCore using Android

3.1 Section Overview

In this section we are going to explain how we have done our application on the ConnectCore. To do our application we started with the a Getting started application provided by Digi. This is the easiest way to do because of all the digi libraries are already included.

3.2 User Interface

3.2.1 Main Activity

With the main activity the user is able to enable the sensor thanks to the sensor activity runnable with the “Sensor button”.

The user is also able to look at the camera thanks to the “Camera” button.

The “Database” button allows to do a direct acquisition and to send the data to the database.

The main activity is doing the data acquisition and the data sending in another thread.

Figure 12: Android Application Surface

3.2.2 Other activities

The sensor activity is just a simple activity with four switches, one per sensor. If the user enable or not the switches, the values returned from the activity are enabling or disabling the data acquisition of the corresponding sensor.

The Camera activity is the activity which is displaying the Camera video flow and is taking a picture when you go out of the activity. We wanted first to use a Raspberry Pi camera because we saw that it was possible to integrate in Digi documentation. The problem was that there was no documentation about it, so we decide to use an USB camera. With the ConnectCore, some devices, like the camera or the HDMI display are not immediately working and you need to repower the ConnectCore.

3.3 Collecting and sending data

3.3.1 Collecting Data

```
<uses-permission android:name="com.digi.android.permission.GPIO"/>
<uses-permission android:name="com.digi.android.permission.I2C" />
<uses-permission android:name = "android.permission.INTERNET"/>
<uses-permission android:name = "android.permission.CAMERA"/>
<uses-feature android:name = "android.hardware.camera"/>
```

To use all the functionalities of the ConnectCore, you every time need to add the uses permissions in the Android Manifest of your application.

To collect the data you can see the code example that we provide you in the first appendix.

We are collecting data every second and displaying it on the main activity.

3.3.2 Sending data

For our database we used the IoT platform ThingSpeak. Sending the sensor data to ThingSpeak was done with a HTTP Connection, what was the simplest way. Sending data was done by a simple HTTP GET function with our channel's API key, for example: `GET https://api.thingspeak.com/update? api_key={API_KEY} &field1={FIELD1_VALUE}&field2={FIELD2_VALUE} …`

The data fields and graphs have to be configured on the ThingSpeak website.

In the free ThingSpeak subscription the number of data messages was limited, so we decided in our application send data every 20 seconds.

The ConnectCore telemetry is displayed on four charts, one per sensor on the ThingSpeak channel number 171576, which can be accessed under the following URL Link: https://thingspeak.com/channels/171576

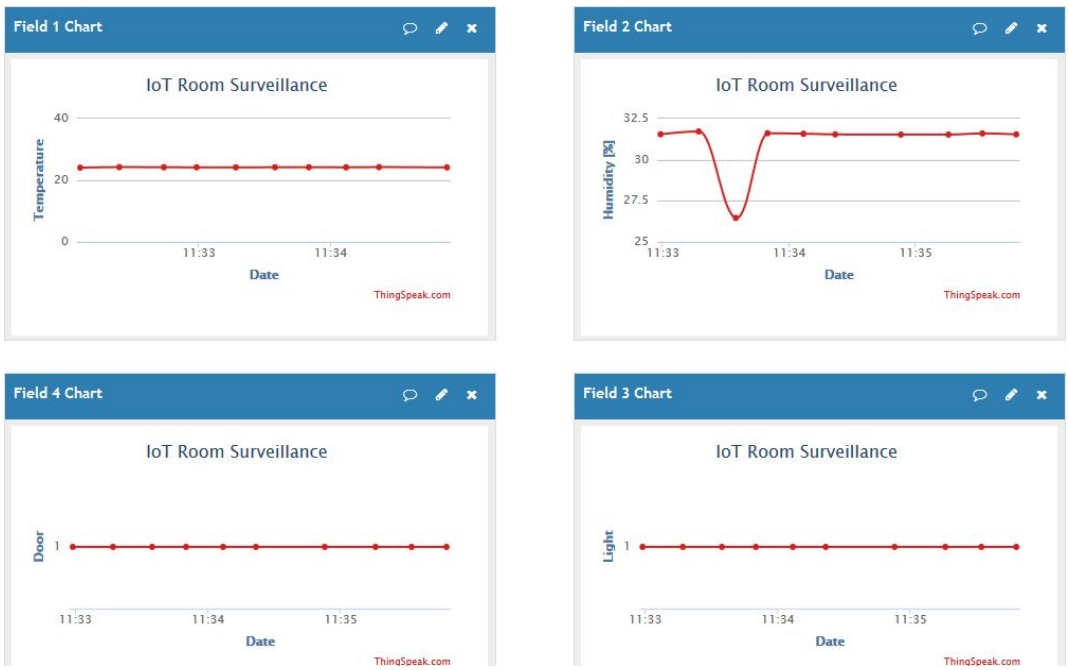

Figure 13: Our ThingSpeak Channel data

4 Implementation challenges

4.1 Section Overview

In this section we are going to explain what difficulties you can be faced when you are using the ConnectCore with Android.

4.2 Android and the ConnectCore

The difficulty of this project was to use the ConnectCore with Android and especially its Android libraries which the implementations were not clear. There was a lack of documentation, for example, the equation Android pin number for the GPIO wasn't clearly explained and there was no example for the Raspberry Pi Camera.

4.3 ConnectCore and I2C protocol

Implementing the I2C Connection was a little bit difficult at the beginning. The issues were that we didn't know how the process was to receive the I2C data with the Digi ConnectCore library function - it would have been easier to understand, if there is an example in the ConnectCore JavaDoc. Instead we used the Digi ConnectCore sample program for I2C to understand the process. Matters were complicated further by the fact that in the Honeywell HIH8120 datasheet was no hint about the number of the default slave address of the sensor. The default address (0x27) was standing in a Technical Notes sheet somewhere else on the Honeywell website.

4.4 Android and the Database

First we tried to use the database created for the IoT project but the use of "volley", the database library recommended by the backend group wasn't easy to implement and wasn't working. With the project deadline becoming closer and closer we decided to use ThingSpeak instead.

References

1. http://ftp1.digi.com/support/documentation/55001809-02-ENG_1P.PDF - Hardware documentation of the ConnectCore
2. http://www.digi.com/resources/documentation/digidocs/90001945-13/landing_pages/android_index.htm Digi documentation for using Android on the ConnectCore
3. https://drive.google.com/open?id=0B3aGVuJIwgt9Rnh1YW4zNVNOOVU Full android code of our application
4. https://github.com/rubenmoral/android-camerasample
5. http://www.farnell.com/datasheets/1863542.pdf?_ga=1.263477260.1012292472.1477913533 Datasheet Honeywell HIH8120

Appendix 1: Collecting and sending data

```
double temp =0;
double hum =0;
int door =0;
int light =0;
try {
    GPIOValue DoorValue;
    GPIOValue LightValue;

    if (bSensorEnableValues[0]) {
        DoorValue = DoorGPIO.getValue();
        door = DoorValue.getValue();
        if(String.valueOf(door).equals("1")){
            TxtViewDoorData.setText("Open");
        }
        else{
            TxtViewDoorData.setText("Close");
        }
    }
    if (bSensorEnableValues[1]) {
        temp = readValueT();
        TxtViewTempData.setText(String.valueOf(temp + "°C"));
    }
    if (bSensorEnableValues[2]) {
        LightValue = LightGPIO.getValue();
        light = LightValue.getValue();
        if(String.valueOf(light).equals("1")){
            TxtViewLightData.setText("Off");
        }
        else{
            TxtViewLightData.setText("On");
        }
    }
    if (bSensorEnableValues[3]) {
        hum = readValueH();
        TxtViewHumidityData.setText(String.valueOf(hum) + "%");
    }

}catch (final GPIOException e) {
    bUpdate = false;
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            displayError("Error setting GPIO value: " + e.getMessage());
        }
    });
}
try {
    URL url = new URL("https://api.thingspeak.com/update?api_key=Y64N1A63PRYL402B&field1=" +temp+"&field2="+hum+"&field3="+light+"&field4="+door);
    HttpURLConnection urlConnection = (HttpURLConnection) url.openConnection();
    urlConnection.getInputStream();
    urlConnection.disconnect();
} catch (Exception e) {
}
```