Hrvoje Cvitkušić, BSc

Monitoring System for Existing Retaining Structures

MASTER'S THESIS

to achieve the university degree of

Diplom - Ingenieur

Master's degree programme: Information and Computer Engineering

Submitted to

Graz University of Technology

Supervisor: Univ.-Prof. Mag.rer.nat. Dr.rer.nat. Alexander Bergmann

Co-Supervisor: Dipl.-Ing. BSc Reinhard Klambauer

Institute of Electronic Sensor Systems

Graz, March 2017

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit / Diplomarbeit / Dissertation identisch.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis/diploma thesis/doctoral dissertation.

…… ……

Datum / Date Unterschrift / Signature

Acknowledgments

The Monitoring System presented in this work was made under request from Dipl.-Ing. BSc Matthias Rebhan from Institute of Soil Mechanics and Foundation Engineering, TU Graz. In the behalf of the Institute, he opted for a system that could perform measurements using certain sensors controlled by Raspberry Pi. I applied for that theme since I had some experience in that field, soon we found the mutual language and our cooperation was blissful during the work on the thesis.

This Master's Thesis was entirely done at the Institute of Electronic Sensors Systems, TU Graz. There, all the equipment and materials needed for the work were at my disposal as well as budget to order the needed parts, thanks to my supervisor Univ.-Prof. Mag.rer.nat. Dr.rer.nat. Alexander Bergmann. He led me through this thesis with his extensive knowledge and experience in projects of this scope.

My special thanks go to my co-supervisor, Dipl.-Ing. BSc Reinhard Klambauer, a person who had an answer to every question I asked. Moreover, he is an expert and enthusiast in electronics and I really enjoyed having him by my side.

And above all, I would like to express my deepest thanks to my parents, Đurđica and Željko and my brother Bruno for an unconditional love, support and encouragement during my study. My parents provided me the financial and human assistance whenever I needed one and these kind words are just one drop in the sea of what they deserve for their sacrifice and support.

Graz, …… ……

Hrvoje Cvitkušić

Zusammenfassung

Stützkonstruktionen werden eingesetzt um verschiedene Bodenanpassungen zu sichern. Sie sollten in der Lage sein dem lateralen Druck des Bodens, dem des Gesteins oder anderen Substanzen Stand zu halten. Soll die Neigung eines Hanges höher als der innere Reibungswinkel des Bodens, ist eine Stützkonstruktion notwendig, die den Boden davon abhält sich zu bewegen (zu gleiten oder kriechen), da er sich nur bis zu einem gewissen Punkt senkrecht halten kann. Um sicherzustellen, dass die eingesetzten Stützkonstruktionen ihre Eigenschaften beibehalten und sich nicht auf Grund von umgebungsbedingten und geologischen Einflüssen verändern, sollten diese wiederholt überprüft werden. Viele Überwachungssysteme befinden sich heutzutage auf dem Markt aber ihre Preise sind normalerweise über dem Budget vieler Bildungseinrichtungen.

In dieser Masterarbeit präsentieren wir ein kleines, günstiges und verlässliches Überwachungssystem, welches im Sekundentakt diverse Umgebungsdaten und geologische Daten der Stützkonstruktion sammelt. Diese Daten werden von dem System über das mobile Netzwerk mit Hilfe von PubNub Cloud Serivces an den gewünschten Standort übermittelt, der normalerweise ein leistungsstarker Client-PC für die Nachbearbeitung ist. Das Hauptsteuerungsteil des Überwachungssystems ist ein Allzweckrechner in der Größe einer Kreditkarte namens Raspberry Pi, welcher mit allen angeschlossenen Sensoren kommuniziert. Raspberry Pi sammelt, bereitet auf und transportiert die Daten an den gewünschten Standort, PubNub Colud Service.

Das Design des Systems wird sicherlich mit mehreren Herausforderungen verbunden sein, insbesondere der Problematik bezüglich den Umweltbedingungen. Aufgrund der Tatsache, dass das System auf der Stützkonstruktion montiert werden soll, muss es robust genug sein um etliche Wettersituationen standzuhalten, besonders im Winter.

Außerdem, muss das ganze System in einem robusten Gehäuse untergebracht werden, da die Spitzentemperaturen von ca. -40°C, während der kalten Wintertage, bis zu ca. +60°C in der direkten Sonne im Sommer, variieren. Damit das gewährleistet

ist, wird ein System entworfen welches zusammen mit einer Vorrichtung für die Regelung der Temperatur in ein gemeinsames Gehäuse passt. Sensoren, Leitungen und andere Komponenten, die sich außerhalb des Gehäuse befinden werden viel hochwertig sein da sie automotive Standards unterliegen und UV-beständig sein müssen.

Abstract

Retaining structures are installed to enable certain adjustments in the terrain. Therefore, they need to be able to resist the lateral pressure of the soil, rocks, and other materials. During the construction work we might need to change the shape of some ground elevation beyond the internal friction angle of the soil. In order to do that we must construct a proper retaining structure to keep the soil from moving (sliding or creeping) since the soil cannot stand vertical above a certain limit. Once installed, the retaining structure should be monitored from time to time to ensure that its properties do not change due to environmental and geological effects.

There are many monitoring tools and systems on the market nowadays. However, those tools and systems are hardly affordable for many educational institutions considering their tight budget. In this paper, we will present a small, cheap, and reliable Monitoring System, which will collect various environmental data as well as the geotechnical data of the retaining structure every couple of minutes. The device will then transmit it over mobile network using PubNub cloud services to a desired location – usually a powerful client PC for post processing. The central control unit of the Monitoring Tool will be Raspberry Pi, an all-purpose credit card-sized computer that operates together with all the attached sensors and other electronic components. The main purpose of the Raspberry Pi is to collect, pre-process and transmit data to a desired location, i.e. PubNub cloud service.

Many challenges will arise during the design of the system with environmental effects being among the most problematic ones. Due to the fact that the system will be installed "on" the retaining structure, it must be robust enough to overcome severe weather conditions. As the peak temperature varies from -40°C in the winter up to +60°C under direct sun radiation, we must introduce a proper shield for the entire system. Consequently, a system is designed to fit into the steel box alongside the heating devices to maintain a stable temperature of approximately +25°C inside the box. Sensors, cables and other components used outside the box will be of higher quality since they are regulated by automotive standards and must be UV resistant.

Contents

Introduction ..... - 11 -
1.1. Thesis outline ..... - 12 -
Theoretical Background ..... - 14 -
2.1. Retaining structures ..... - 14 -
2.2. Inertial Measurement Unit ..... - 16 -
2.2.1. Accelerometer ..... - 17 -
2.2.2. Gyroscope ..... - 19 -
2.2.3. Fusing the IMU data ..... - 21 -
2.3. Rain Gauge ..... - 23 -
Hardware & Components ..... - 25 -
3.1. Overview ..... - 25 -
3.2. Cost Analysis ..... - 29 -
3.3. Raspberry Pi ..... - 30 -
3.4. Inertial Measurement Unit ..... - 32 -
3.5. Temperature Sensor ..... - 34 -
3.6. Temperature and Humidity Sensors ..... - 35 -
3.6.1. AM2302 Sensor ..... - 35 -
3.6.2. HIH6130 Sensor ..... - 36 -
3.7. Pressure Sensor ..... - 37 -
3.8. USB 3g Mobile Internet Dongle ..... - 38 -
3.9. Other components ..... - 40 -
3.10. PCB Design ..... - 42 -
3.11. Rain Gauge ..... - 46 -
Software Implementation ..... - 51 -
4.1. Setting up Raspberry Pi ..... - 51 -
4.2. Mobile Internet using 3G USB Dongle ..... - 57 -
4.3. PubNub IoT ..... - 61 -
4.4. Subsystems implementation ..... - 65 -
4.4.1. Inertial Measurement Unit ..... - 65 -
4.4.2. Internal Heat Control ..... - 67 -
4.4.3. Rain Gauge Heat Control ..... - 67 -

4.4.4. Rain Gauge Precipitation Measurement ..... - 68 -
4.4.5. Environmental Temperature Measurement ..... - 69 -
4.4.6. Environmental Relative Humidity Measurement ..... - 70 -
4.4.7. Pressure Measurement ..... - 70 -
4.4.8. Internet Connection Monitor ..... - 71 -
Testing and Results ..... - 72 -
5.1. Issues During Testing & Solutions ..... - 73 -
5.2. Testing Period Results ..... - 76 -
Conclusion ..... - 79 -
6.1. Future Work ..... - 80 -
Appendix ..... - 82 -
Internal Heat Control ..... - 82 -
Inertial Measurement Unit ..... - 84 -
Internet Connection Monitor ..... - 88 -
Humidity Measurement ..... - 90 -
Temperature Measurement ..... - 92 -
Pressure Measurement ..... - 94 -
Rain Gauge Precipitation Measurement ..... - 96 -
Rain Gauge Heat Control ..... - 99 -
Subscriber or Client Side ..... - 102 -
Glossary ..... - 104 -
Bibliography ..... - 106 -

List of figures

Figure 1: Pitch, Roll and Yaw example [5] ..... - 16 -

Figure 2: 3D Model of Accelerometer [2] ..... - 17 -

Figure 3: 3D Accelerometer and Gyroscope Model [2] ..... - 19 -

Figure 4: Model of the Complementary Filter ..... - 21 -

Figure 5: Top view of the box with all components, once opened ..... - 28 -

Figure 6: Bottom part of the box, without PCB mounted ..... - 28 -

Figure 7: Raspberry Pi 2 Model B Top and Bottom view [9] ..... - 31 -

Figure 8: Raspberry Pi 2 Model B GPIO Pin Configuration [10] ..... - 31 -

Figure 9: LSM9DS0 IMU with three 3D axis explanation[11] ..... - 32 -

Figure 10: LSM9DS0 on the Breakout Board[12] ..... - 33 -

Figure 11: Waterproof and pre-wired DS18B20 [14] ..... - 34 -

Figure 12: AM2302 on the left and HIH6130 on the right [16][17] ..... - 36 -

Figure 13: SSCMANT006BA2A3 Absolute Pressure Sensor [18] ..... - 37 -

Figure 14: Huawei E3131 USB 3g dongle with CRC9 antenna extender cable .... - 39 -

Figure 15: Schematic of the Monitoring Tool made in Cadsoft Eagle 7.6.0..... - 43 -

Figure 16: PCB in 1:1 ratio. Red lines represent voltage and signalling lines from Top layer and blue are lines from Bottom layer ..... - 44 -

Figure 17: PCB Top layer ..... - 45 -

Figure 18: PCB Bottom layer ..... - 45 -

Figure 19: Rain Gauge principle of operation [22] ..... - 48 -

Figure 20: Funnel ..... - 49 -

Figure 21: Stand ..... - 49 -

Figure 22: Lever ..... - 50 -

Figure 23: Rain Gauge ..... - 50 -

Figure 24: Changes in relative humidity [26] ..... - 74 -

Figure 25: Relative Humidity measurement, in [%] ..... - 76 -

Figure 26: Precipitation measurement, scaled to litres per square meter per hour ..- 76 -

Figure 27: Atmospheric or Barometric Pressure measurement, in [bar] ..... - 77 -

Figure 28: Temperature measurement, in [degC] ..... - 77 -

Figure 29: Comparison of IMU's X axis angle by Accelerometer and Complementary filter, in [deg] ..... - 78 -

Figure 30: Comparison of IMU's Y axis angle by Accelerometer and Complementary filter, in [deg] ..... - 78 -

Chapter 1

Introduction

Monitoring System for existing retaining structures is a system made for measuring and retrieving retaining structure's inclination and acceleration together with certain environmental data in order to monitor and explain its behaviour. It consists of two parts: the Monitoring Tool and Client or Subscriber.

The Monitoring Tool is a standalone device made for measuring and transmitting the required retaining structure's and environmental data. It is designed to be "plug and play", i.e. just plug it into 230 VAC electrical power grid and data will be received on the Client side as long as it is powered, every 2 minutes. The Monitoring Tool comes encased in the steel protecting box together with the Rain Gauge. There are four mounting panels in each corner with M10 holes. Electronic sensors measure various data, data is collected by the Raspberry Pi which is a central processing unit of the system and later it is transmitted over PubNub cloud service using mobile Internet connection. On the other side, there is a Client or Subscriber. Client is any kind of application that receives the data using valid PubNub's keys for communication. In this project, Python application collects the data and stores it into .csv files. The collected data is later processed by soil engineers.

The system was funded and made under requirements of Institute of Soil Mechanics and Foundation Engineering, TU Graz. The entire development process, from initial idea to delivery of the final product, was done under Institute of Electronics Sensor Systems, TU Graz.

1.1. Thesis outline

This master's thesis explains in detail the whole idea behind the Monitoring System; from the initial idea, through development, testing and implementation. After the short introduction chapter which gives us basic overview of the whole system, we go straight to theoretical background in chapter 2.

Theoretical background is important for later understanding of the work as we dig deeper into the engineering and scientific material with each chapter following. In the first subchapter, retaining structures and their properties are discussed from the civil engineer point of view. After that, Inertial Measurement Unit or simply put "IMU" is described. As this is an essential part of the whole system, it requires a detailed explanation – together with the measuring principle, filters for post processing are presented as well, i.e. Kalman Filter and Complementary Filter. The last but not the least to mention is Rain Gauge. Different approaches and techniques of precipitation measurement are described in this subchapter.

Chapter 3 tells us more about the whole Monitoring System from the hardware perspective. It starts with the basic overview. Later on, every major part is thoroughly described and the auxiliary parts are just mentioned. Cost analysis is also provided in this chapter. Schematic of the whole system is given, together with the PCB idea and design. The Rain Gauge design and principle of operation is explained in detail.

Chapter 4 presents the software implementation of the whole system and auxiliary subsystems. It can be considered as an application layer. Before doing any work of this kind, programing and working environment should be made accordingly. Each software application is described, together with its role in the overall system. After that, all scripts and modules created are given together with detailed explanation of each coding part which might be of interest. The full code is given in the Appendix at the end of thesis. And to make everything easier to observe, a flowchart of the application layer is given.

Chapter 5 is focused on testing and retrieving results. The Monitoring System is already implemented and data is gathered approximately every 2 minutes. Results of the measurement are presented on 2D charts, with respect to the time. However, results will be only provided, they will not be discussed together with any time derivative functions, i.e. raw data does not make any sense without proper post processing from a civil engineer but it was not part of the work. Additionally, some minor problems which arose during the testing period will be explained together with the solution to the problem. Namely, as system was designed in the lab, it was hard to predict all possible issues that might occur in different environmental conditions. And that is what testing phase helped to discover.

In chapter 6, a brief conclusion is summarizing the whole work together with possible improvements and upgrades in future work.

An Appendix comes at the end, together with a glossary and bibliography.

Chapter 2

Theoretical Background

2.1. Retaining structures

"*Material is retained if it is kept at a slope steeper than it would eventually adopt if no structure were present. Retaining structures include all types of walls and support systems in which structural elements have forces imposed by the retained material*." The definition is taken from Eurocode 7 (EC7) standard. Furthermore, three main types of retaining structures can be defined by EC7: Gravity walls, Embedded walls and Composite Walls. The exact definitions will be taken from the book in [33] as it is the main literature for this subchapter.

"*Gravity walls are walls of stone or of plain or reinforced concrete having a base footing with or without a heel, ledge or buttress. The weight of the wall itself, sometimes including stabilizing masses of soil, rock or backfill, plays a significant role in the support of retained material*." [33]

"*Embedded walls are relatively thin walls of steel, reinforced concrete or timber, supported by anchorages, struts and/or passive earth pressure. The bending capacity of such walls plays a significant role in the support of the retained material, while the role of the weight of the wall is insignificant*." [33]

"*Composite retaining structures are walls composed of elements from the above two types of wall*." [33]

Having defined the principle retaining structures, let's move on to the geotechnical measurement procedures. Namely, that is in the focus of this whole work. Due to some later uncertainties in ground conditions, there is a number of geotechnical parameters which cannot be pre-calculated with high accuracy. They are subject of interactions between structure and the soil and therefore can be measured only after the deployment of retaining structure. Nowadays, geotechnical measurements are an emerging field, with constant improvements in calculation and measuring methods. The objectives of geotechnical measurements are: Instrumentation during soil exploration, Safety, Quality control, Instrumentation using the observational method and Assessment of calculations and predictions.

To get an insight in interaction between retaining structure and soil, certain parameters have to be measured and judged later in the overall analysis. Stresses in retaining structure can be determined using material laws and can be evaluated by measuring the strain. Main parameters are: Strain, Curvature(rotation), Deflection curve, Slope change, Displacement of complete structure, Temperature, Anchor load and Strut load. But to get the proper insight, multiple parameters measured with high degree of accuracy have to be fused together. For instance, the measurement of strain in concrete and steel requires high accuracy due to high elasticity of material. Moreover, the temperature has to be measured simultaneously to compensate its influences on both structure and transducer itself.

Measuring instruments and techniques are various, with tendency on emerging automatic measuring systems. An automatic measuring system is operated by microcontroller and PC. It should be as modular as possible, with many different sensors attached. Important features of such system include registration of data online, online calculation and compensation of data, an alarm in case of exceeding limits, a self-monitoring system(i.e. a watchdog), etc. However, manual measurements will always precede the deployment of automatic systems to get the initial calibration settings.

2.2. Inertial Measurement Unit

An Inertial Measurement Unit or simply IMU is an electronic device or sensor that reports linear and angular motion and sometimes even the magnetic field around the body. It is a self-contained system; with combination of accelerometer, gyroscope and sometimes magnetometer. Although mainly used in aircrafts and spacecraft, with recent development they successfully penetrated into daily used devices, like mobile phones. [1][2]

Nowadays, term IMU is usually referring to a boxed device which contains three accelerometers with their measuring axis placed orthogonal to each other, three gyroscopes in similar orthogonal order and three magnetometers. They are mainly used as inertial guidance, navigation and orientation systems.

Principle of operation is quite sophisticated and requires combined data of all subunits to get valid measurements. With accelerometers, current rate of acceleration is detected. Pitch, Roll and Yaw are rotational attributes an their change is detected by gyroscopes. However, to get the stable values for Pitch and Roll, gyroscope and accelerometer reading should be fused. If there is magnetometer inside IMU, it is used in calibration against drift in orientation. To get stable Yaw value, gyroscope and magnetometer reading are fused. Considering the fact that measurement is done in three axis by two devices, we can talk about 6 DoF or 9 DoF if there is magnetometer installed. DoF stands for Degrees of Freedom. It is sum of axis and sensors from which we retrieve the data. It could go even up to 11 DoF, in combination with barometric pressure sensor and GPS module.

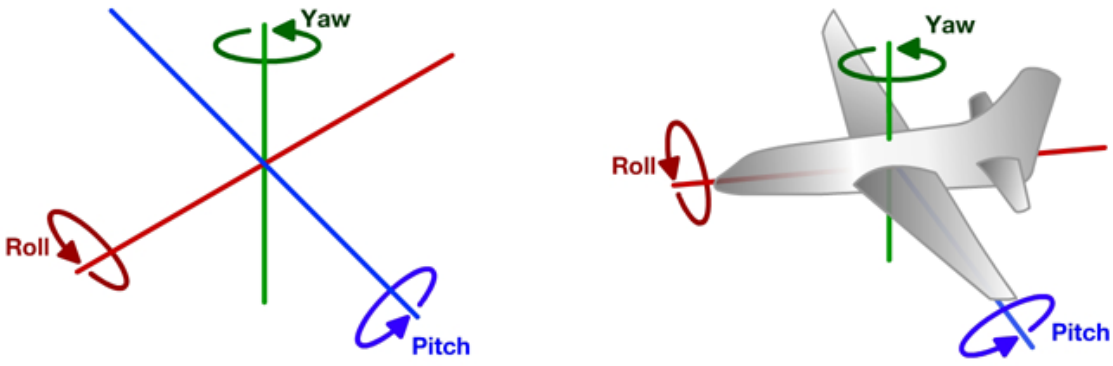


Figure 1: Pitch, Roll and Yaw example [5]

2.2.1. Accelerometer

An accelerometer is an electromechanical device used to measure static and dynamic acceleration. Static acceleration is nothing but a galaxy proper acceleration which must be deferred from "rate of change velocity" acceleration; proper acceleration on Earth is gravity and it measures $g \approx 9.81\ m/s^2$. Dynamic acceleration is product of body's movement or vibrations. Vast majority of the accelerometers on the market measure acceleration in all three axes using capacitive sensing technique. As the device vibrates, the plates move accordingly thus changing the capacitance. Acceleration can be extracted from the changes in capacitance. Other sensing techniques, like piezoresistive and piezoelectric, are less frequently used. The first micro machined accelerometer was designed in 1979 at Stanford University, yet it took almost two decades before such devices entered large volume applications. Nowadays, accelerometers are widespread in all sorts of commercial devices, with more than 100 million MEMS accelerometers being sold every year all over the world with the average price of about 4 $ per single package device. [1]

Basic principle of operation of modern day accelerometers is best described and illustrated in [2]. Here, a brief summary of [2] will be given, to understand how accelerometer works.

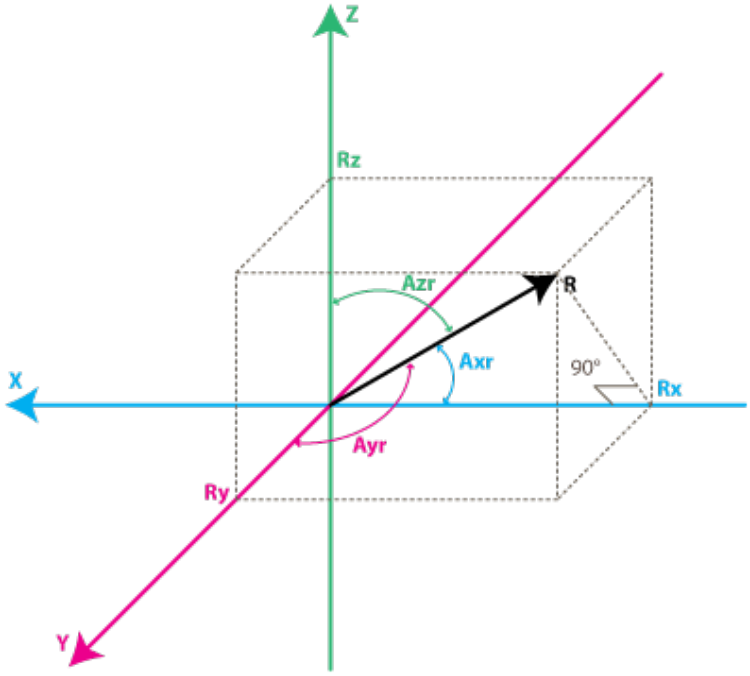


Figure 2: 3D Model of Accelerometer [2]

The 3D Model from the Figure 2 will illustrate how an accelerometer works. The coordinate system is fixed to the axes of accelerometer. The force vector is rotating around. The measured force vector is the vector R. It could be either static or dynamic acceleration, or both, it does not matter. R_x, R_y, R_z are the projection of vector R on the X, Y, Z axes. Using Pythagorean 3D theorem, a relation between R and R_x, R_y, R_z can be established. The values of Rx, Ry, Rz are the values that accelerometer will output, linearly scaled by some factor. From the information delivery point of view, accelerometers can be either digital or analog. Digital ones use I2C or SPI communication protocol and R_x, R_y, R_z can be taken as is if it is not defined different by datasheet. Analog ones should go through ADC module and getting R_x, R_y, R_z requires couple of additional operations.

At the moment, we have all three components with whom our inertial force vector is defined with. Inclination can be calculated in two directions. To calculate the inclination with respect to the ground, angle between vector and Z axis should be calculated. To calculate the inclination on specific axis, i.e. on X and Y axis, we must calculate the angle between that axis and gravitation vector. Inclination angles are marked with A_{xr}, A_{yr} and A_{zr} on the Figure 2. Looking at the triangle formed on the right hand side, inclination angle can be calculated by using arccos() function on force projection on axis vector over force vector; i.e. A_{xr} is calculated by applying arccos() on R_x over R. Another handy way to convert the accelerometer values to usable angles is via atan2() function, thoroughly described later in this work.

The main difference between accelerometer and gyroscope is that accelerometer measures the current angle at any time. It does not have to be monitored over time like gyroscope. Still, accelerometer is very noisy and optimal usage is when measuring angles over longer period of time. Another minus is that accelerometer cannot measure yaw. Yaw is the rotational attribute where device is placed on flat surface in level with ground and rotated clock wise(or anti clock wise). The Z axis readings do not change, thus there is nothing to measure. However, with combination of gyroscope and magnetometer, yaw can be measured.

2.2.2. Gyroscope

Gyroscope is a device that measures rotational motion or angular velocity. Angular velocity is rate of change of angular position of rotating body, or simply put, it is measurement of speed of rotation. Unit of measurement is degrees per second or dps, or revolutions per second or RPS. In this work, dsp unit is used. To calculate the current angle, speed of rotation has to be monitored over time. The minus is that this monitoring over time causes gyroscope to drift. Nevertheless, gyroscopes are good at measuring sharp quick movements, instead of fast spinning objects. Linear velocity and acceleration of object do not affect the gyroscope measurement. Nowadays, a typical 3 axis MEMS gyroscope has a small resonating mass inside that is shifted as the gyroscope is rotating; based on Foucault pendulum "principle". This movement is converted to electrical signal and outputted to microcontroller. Communication interface is either digital or analog. SPI and I2C protocols are used for digital communication and ADC from microcontroller reads the input from analog interface. [1]

A basic guide on how to handle the readings from modern day MEMS gyroscope is thoroughly described in [2]. Here, a part of [2] will be summarized, continuing on the Figure 2, given in the accelerometer subchapter.

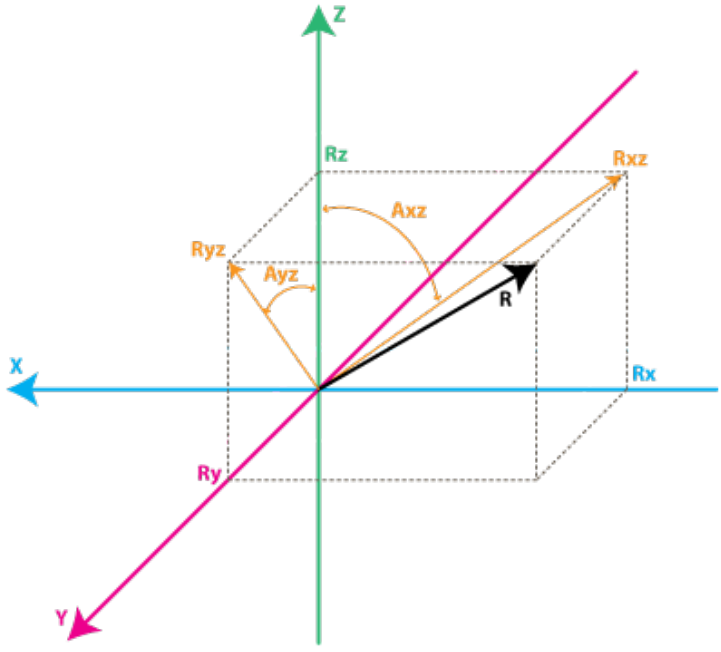


Figure 3: 3D Accelerometer and Gyroscope Model [2]

The 3D Model in the Figure 3 includes the 3 axis accelerometer model and we will try to describe what is gyroscope measuring according to that model. A 3 axis gyroscope will have three orthogonally placed gyroscopes, thus outputting on three different channels. Each channel measures the rotation around single axis. In the Figure 3, we can define R_{xz} and R_{yz} as the projection of the force vector R on the X-Z plane or Y-Z plane, respectively. Angles between R_{xz} and Z axis or R_{yz} and Z axis are defined as A_{xz} and A_{yz}, respectively. The gyroscope will output an value that is linearly related to the rate of change of these angles. For instance, if we want to measure the rotation angle around X axis, we must calculate the A_{yz} angle. We will record the A_{yz} angle at the starting time t_0 and at some time later t_1. The rate of change will be calculated using the following equation, expressed in dps:

$$Rate A_{yz} = (A_{yzt1} - A_{yzt0}) / (t_1 - t_0) \quad [dps]$$

Vast majority of the MEMS gyroscopes, even the digital ones, will not output the value already converted to dps. Many of them offer the customization options allowing the user to set by himself the sensitivity of the reading, frequency and gain as a product of the former and latter one. Also, for the sake of precision, the period between two readings must be calculated each time between two reading to avoid unnecessary drift. So, if we want to read the rotation angle from the typical customizable MEMS gyroscope, first we must set the sensitivity and frequency of the reading. Gain value is provided in the datasheet, fusing the sensitivity and frequency values. To get the rotation angle, raw reading of the gyroscope must be multiplied by gain value and summed to the gyroscope angle of that axis.

2.2.3. Fusing the IMU data

To estimate the attitude of a body, it is not enough to use only accelerometer data or only gyroscope data. Namely, the accelerometer can be trusted only when the acceleration is progressive, without any vibrations which will result in corrupted results. On the other hand, gyroscope has a drift when tracking over some period of time because it is designed to track quick sharp moves. To get the best possible body attitude estimation, the readings from both devices should be fused. Accelerometer is good for orientation in static conditions. Gyroscope is good for tilting in dynamic conditions. Data is fused(merged) using a mathematic filter. There are two major filters for this purpose: Complementary filter and Kalman filter. As the Complementary filter is used in this work, it will be thoroughly explained. The Kalman filter will be only briefly explained. Pros and cons of each are discussed in [31] and [32] and a short overview will be given later in this subchapter.

The Complementary filter is merging the data in a way that it passes the accelerometer data through low pass filter and gyroscope data through high pass filter and the outputs are summed. They are emerging nowadays, especially in hobby world, because they are simple and not much processor intensive. Filter discussed here will consider only first order filter. Of course, the deeper we go, i.e. second order, third order etc.; the better are the results. But it is questionable whether the additional increase in complexity brings any obvious gains to justify itself.

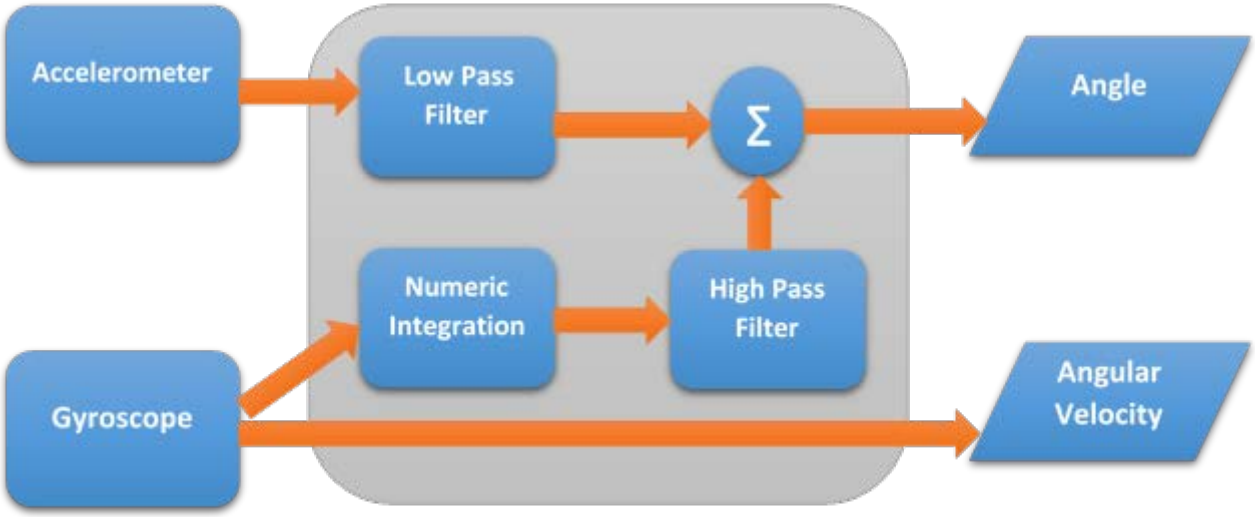


Figure 4: Model of the Complementary Filter

In the Figure 4, the Complementary filter principle of operation is illustrated. Low pass filter passes only long term changes and filter outs short term oscillations. High pass filter does totally the opposite, it passes only short term fluctuations and filter outs the ones that are steady over time. Integration means nothing but a summing the change in amount of time to a previous state. For instance, we have a body that travels with a known velocity and a clock that ticks at exact rate. To calculate the new position of a body after each tick, we add the velocity multiplied by tick(time) to previous position. And that is what integration does. The two filters are complementary to each other, thus the name "Complementary".

Now, let's dig a bit deeper in the Complementary filter. To do so, we will use the work presented in [2] and extract only the main observations. The Complementary filter is defined by the following formula:

CF_angle = CFC x (CF_angle + Gyro_rotation_rate) + ((1-CFC) * Accelerometer_angle)

CF_angle is the final angle to be used. It can be either in X or Y axis. CFC stands for Complementary Filter Constant. It is user defined, but there are some rules on how to choose the proper CFC. It is related to time constant and determines the boundary after which the accelerometer readings stop and gyroscope readings are considered. If we want to be more precise, CFC should be the fraction of time constant over sampling frequency. Gyroscope rotation rate must be multiplied by loop time. It is advised to measure loop time every time we read the gyroscope to get the accurate results. To get the accelerometer angle in degrees, raw values must be converted accordingly. The simplest way is to use atan2() function on A_{ccy} and A_{ccz} value to get X angle or on A_{ccz} and A_{ccx} to get Y angle. Atan2() function returns principle value of the tangent of other angles in radians so π must be added to get result between 0 and 2. To convert the radians to degrees, the value must be multiplied by (180/π).

Kalman filter uses a series of measurements observed over time to estimate the attitude of body, minimizing the mean squared error. It consists of two stages: prediction and correction. From theoretical point of view, it is the best possible filter but it is cumbersome to implement and processor intensive to use. As it can be seen in [31] and [32], there is no real need to implement the Kalman filter for the needs of this project because the (eventual) gains will not justify its complexity.

2.3. Rain Gauge

In this subchapter, I will give a basic overview of precipitation measuring techniques and used devices. The subchapter is made following the book in [4]. Definition of the rain gauge can be summarized as a device that collects the rain in a funnel and then measures it in some way. Precipitation is any kind of condensed atmospheric water vapour falling to the ground. It includes rain, dew, drizzle, blizzard, snow and hail.

Precipitation is a significant part of the climate system, almost important as temperature, for instance. But to measure precipitation, first we must know what precipitation really is. The first requirement is to understand the whole hydrological cycle, and that includes evaporation, water vapour, convection, condensation, clouds, soil moisture, groundwater and the origin of rivers. As of various religious dogmas, legends and superstitions, very few hypotheses about hydrological cycle were formed during history. Together with the uprising secularity among the scientists, the most notable progress in this field was made between year 1600 and 1900, in the period known as Renaissance. In 1662, Sir Christopher Wren invented the tipping bucket rain gauge. It was self-emptying mechanical device which was run by weight-driven pendulum clock. It recorded barometric pressure, temperature, rain, relative humidity and wind direction. Measurements were marked on paper tape, punching holes every 15 minutes. This is known as first Automatic Weather Station, built almost four centuries ago.

There were many different rain gauges built in the following era, with most of them being manual or mechanic. The real revolution started with electronic rain gauges. The precision of electronic rain gauges is (almost) impeccable and unattainable for their mechanic competitors. From 1960s onward, in so called "modern era", electronic rain gauges took over the prime. Tipping bucket rain gauges are the most common nowadays. Sir Christopher Wren's device was a forerunner for the modern day devices. The major difference is that modern devices are all symmetrical though the shape of the bucket may vary. Precision of the rain gauge is not continuous, especially during heavy rains or bad positioning. Sometimes

it measures more precise, sometimes less. However, some things have to be respected in order to get the better precision:

- The minimum size of the tip should be 10 mm, meaning that at least 10 ml per tip should be collected by the funnel
- To measure 0.2 mm of rainfall reliably, a funnel of at least 500 square cm is required
- Tip intervals should be 0.1, 0.2, 0.25, 0.5 and 1 mm
- Not installing near object for at least twice the size of the object
- Wind shield installation

As the water goes down the funnel, it tips in the compartment of the lever. The lever has two compartments. In the centre weight point of the lever fulcrum is located. The lever is pivoting around fulcrum once the water in the compartment over-weighs the other side of the lever. Pivoting means “switching sides” or states. There can be only two states, top left and top right. On top of the lever, a magnet is attached. Above the fulcrum point reed switch is attached. Once the lever switches sides(states), the magnet passes near reed switch and closes it. That event is recorded by microcontroller that is in charge of the whole system.

Capacitance rain gauge is also a form of tipping bucket rain gauge. The water is collected in a cylinder. Inside are electrodes which act as capacitor plates. Water acts as dielectric. Dielectric constant of water and air is known. Depth of water is measured by including the capacitor in a tuned circuit. The water should be drained periodically.

Tipping bucket is very simple device, consumes no power and very little can go wrong. A way to optimize measuring is to respect the above stated rules. There are some other things that might affect the accuracy of the measurement, and they are: evaporative loss, out splash, levelling, siting of gauge, wind effect, etc.

Other forms of electronic rain gauges include radar, optical and drop-counting rain gauges. Radar rain gauges include very small Doppler radars working at 11 GHz. They measure the spectrum of backscatter from falling rain drops, indicate their fall velocity and size and fuse those data into rainfall intensity. Optical rain gauges sense drops passing through a light beam and convert the signal into rainfall intensity.

Chapter 3

Hardware & Components

3.1. Overview

The Monitoring System is a complete system for monitoring the behaviour of retaining structures remotely. It measures retaining structure's linear acceleration and angular rate, combining both data using the Complementary filter. Additionally, it measures environmental conditions, such as air temperature, relative humidity, absolute barometric pressure and precipitation. There are two main parts of the system. One is Host part or also called the Monitoring Tool and the other part is Client or Subscriber. Host part collects and transmits data over PubNub cloud channel using mobile data. Client is subscribed to the channel and retrieves the data for storage and post processing.

The Monitoring Tool or Host part is a standalone, "plug and play" device. It comes enclosed in steel box together with the Rain Gauge, connected by wire. Once properly fixed on the retaining structure, plugging it into 230 VAC Type F female socket or Schuko socket is enough for it to start working. Looking from bird's-eye view on the closed box, there are four cables going out of it. Left one is for temperature sensor DS18B20, thick middle one is for the Rain Gauge, right one is 230 VAC plug and the one on the right side is from relative humidity sensor AM2302. On the corners, there are four L-shaped steel panels with mounting spots for fixing the box on the retaining structure, using M10 screw. Green LED, placed on the bottom,

signifies if the Monitoring Tool is connected to the Internet. Once plugged in, the Monitoring Tool should start working approximately 90 seconds after, thus illuminating the LED. On the right side of the box is female antenna connector on which UMTS/GSM omnidirectional antenna is connected.

The situation inside the box looks a bit complicated and fuzzy, so let's demystify it. On the bottom left side, grounding cable is connecting the lid with the box. There are several wire bundles on top, all ending up in different connectors on the PCB. White nylon male connectors located on the bottom of PCB, named S1, T1, T2 and T3 are for the Rain Gauge's reed switch, DS18B20 environmental air temperature sensor, DS18B20 Rain Gauge grid heating control temperature sensor and AM2302 relative humidity sensor, respectively. Robust 2-pin male green connectors named LOAD1 and LOAD2 are for the Rain Gauge grid heating and inside box heating, respectively. White nylon 2-pin male connector on the top left side is for Internet control LED light. Robust 3-pin male green connector on the top right side is for 12 VDC input, coming from AC/DC converter. Black twisted cable going from USB mobile Internet dongle to the connector on the right side of the box is antenna cable; connecting the external antenna with mobile Internet dongle.

Once all those cables are removed, PCB can be clearly seen. On the left and right side, it is fixed to the steel box using L-shaped aluminium panels. DC/DC converter is a little black cuboid, soldered on the top with label RECON - 625. Going down, the Raspberry Pi together with USB mobile Internet dongle follows. However, the Raspberry Pi is not soldered to the PCB, it is just plugged with its GPIO pins into a 40 Pin female header and additionally fixed with screws on the corners. Later on, other parts are soldered with the following labels:

- J1 - Inertial Measurement Unit LSM9DS0 with breakout board
- T4 SSC – Absolute pressure sensor SSCMANT006BA2A3
- U1 HIH6130 – Temperature and relative humidity sensor HIH6130
- Q1, Q2 – Low side switches VNN3NV04PTR-E
- Q3 – N-Channel transistor BSS138
- F1, F2 – 3 Amps fuses

Aside of the main parts, there are lots of other electronic components important for the functionality of the whole system, like various resistors, capacitors and control LEDs.

PCB is located in the middle of the box, with respect to the height. There are several other components under, but first the PCB should be removed. In order to remove the PCB, all cable bundles must be unplugged. Internet control LED and external antenna connector have to be removed. Later on, L-shaped aluminium panels must be unscrewed from the steel box and the PCB can be taken out now. On the lower side, there are three chassis mount resistors which are working as heat emitters for heating the inside of the steel box in order to maintain the constant temperature of approximately 25°C. AC/DC Converter is fixed on the bottom and on the side of the box. Cables which enter the box are also located in this bottom part and are bend in the corner so they can reach connectors on the PCB without pulling them with too much force.

The Rain Gauge is also part of the Monitoring Tool. It is connected with the box using a thick black cable with 10 wires, out of which 9 are used. Although it's called "Rain Gauge", it actually measures all kind of precipitation, including snow, sleet, blizzard etc. On top of the Rain Gauge is protective heated steel grid which heats once the temperature drops below 2°C. Temperature sensor DS18B20 is located on the grid in order to measure the temperature and switch the heating if necessary. Once the water passes through the Funnel, it fills the container which is pivoting around a central point. Container is the Lever with magnet on it and each position change is sensed by the reed switch. Knowing the volume of the container, it is rather easy to measure how much precipitation is there per unit of time. More detailed information about the Rain Gauge will follow in the upcoming chapters.

Client or Subscriber side of the Monitoring System is not a specific device. It can be any kind of device, provided it has a proper made application with subscriber key and Internet connection. For the requirements of this project, a rather simple script in Python is made and run from the PC. All it does is just connect to the PubNub channel to which the Monitoring Tool is broadcasting data, retrieve that data and store it in specific textual files for post processing. No doubt that any future upgrade of the whole system will include a mobile application for client or subscriber side.

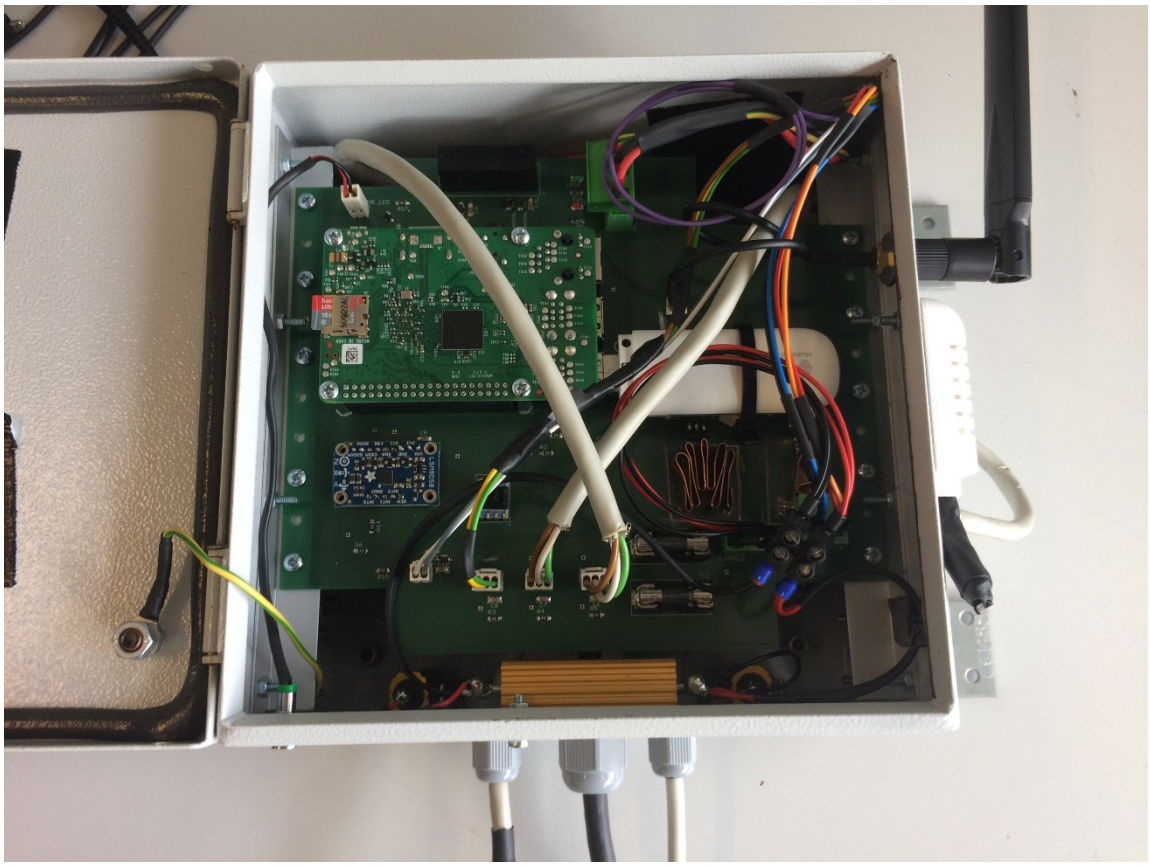


Figure 5: Top view of the box with all components, once opened

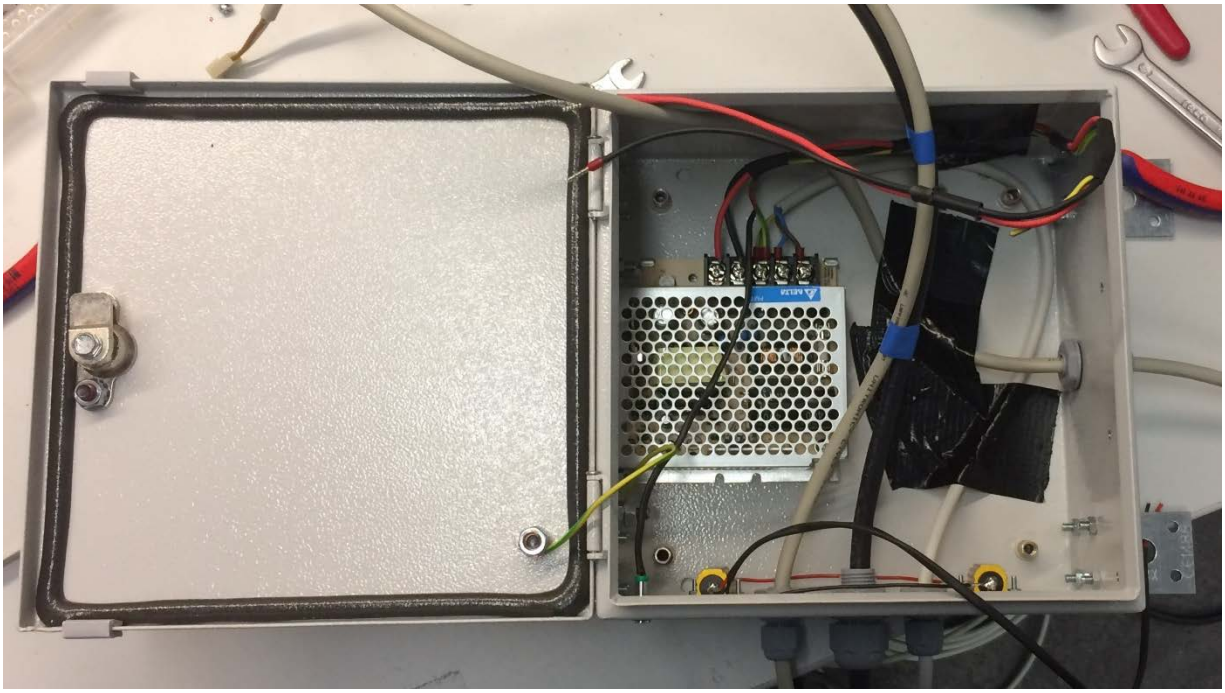


Figure 6: Bottom part of the box, without PCB mounted

3.2. Cost Analysis

When making a project such as this one, total cost analysis must be made in the planning stage. As it was stated in the introduction chapter, there are many devices already available in the shops, but they are above education institution's budget. Moreover, they are not customizable, once bought. The Monitoring System should have as close as possible accuracy comparing to its competitors on the market but it must outperform them in price and customization options after. The maximum budget was set to 800,00 Euros. However, the whole system costs around 700,00 Euros which is in the budget and every single component was compared to its competition on the market and the best one was chosen with respect to system requirements, shipping costs and price-to-value ratio. The budget was entirely spent on physical components of the Monitoring Tool and the Rain Gauge. Tools, equipment and spare electronic material needed for the project were already available in the lab so no money was spent in that direction. All the applications, software and libraries used are either open-source or available for free for public use which also reduces the costs significantly.

Mobile Internet cost is not included in the budget calculation, it depends solely on the final user of the system and it is a rather minor cost. To the best of my knowledge, that cost should not go above 10 Euros per month, using services from Austrian mobile provider inside Austria. USB 3g mobile Internet dongle is unlocked, thus available for all mobile Internet providers with normal sized SIM card. Any mobile Internet option with package of approximately 1 Gb or above is more than enough to cover the needs of mobile Internet data usage made by the Monitoring System. However, it is advised to use reliable provider with good 3g network coverage in the remote area where the Monitoring Tool will be mounted to avoid connection malfunctions or sudden drops. For the needs of this project, SIM card from Hofer Telekom is used. Network coverage is pretty good in Austria and mobile Internet package of 3 Gb costs 6,49 Euros.

3.3. Raspberry Pi

The Raspberry Pi is a series of all-purpose credit card-sized computers developed by the Raspberry Pi Foundation in the United Kingdom, introduced in 2012. With its low cost of around 35 Euros, it was primarily designed to promote STEM academic discipline in pre-university level, especially hardware understanding and programming. However, due to the small size, accessible price and high computing power, it soon became popular among electronic enthusiasts as a proper replacement for basic microcontroller. [7][8]

In this project, the Raspberry Pi 2 Model B was used as central control unit of the Monitoring Tool, paired with the 16 GB micro SD card from SanDisk. It features the BCM2836 System on Chip with 32-bit quad-core ARMv7 processor which is multi-thread-coding friendly option. Other notable properties include 1 GB of RAM, 4 USB ports, HDMI and A/V connector, micro USB connector and RJ45 Ethernet port. On the side, there are 40 GPIO pins which are used to connect sensors and other peripheral electronics and communication can be done using I2C, SPI or 1-Wire protocol.

In the stage of planning the project, various other platforms emerged together with the Raspberry Pi, like the Arduino Uno and the BeagleBone Black. However, the Raspberry Pi was chosen and advantages which support the decision will be summarized in this chapter. Once an Operating system is installed, it's enough to plug in the keyboard and mouse through USB ports, monitor through HDMI connector and 5 Volts 2 Amps power supply through micro USB port and programming can start. It doesn't require separate host and client developing environment like its competitive platforms. Memory is expendable up to 64 GB, it depends on the SD card solely. It costs roughly as much as the Arduino Uno, but is cheaper than the BeagleBone Black. Current on USB port can be up to 1.2 Amps which is sufficient for power hungry devices, like the USB 3g mobile Internet dongle used in the project. Many operating systems are supported by the Raspberry Pi, with Raspbian and Ubuntu being among the most important ones. Regarding the programming languages, C, C++, Java and Python are supported which is also enough to outperform the Arduino programing language supported by the competitive platforms. There are also some disadvantages like no battery powered real-time clock, no BIOS so it boots from the

SD card every time and only one 1-Wire bus, but those are minor things which didn't affect the decision. One more thing to be careful with are GPIO pins. Namely, physical pins on the board are not numbered equally to the Broadcom chip-specific pins. From this point on, all GPIO pins will be enumerated using Broadcom chip-specific pins numbering, i.e. GPIO pin 4 means Broadcom chip-specific pin 4 but physically, it is the Raspberry Pi's pin 7.

The Raspberry Pi is a part of the Monitoring Tool and it is located inside the steel box. On the PCB, two 40 Pin headers are stacked in order to overcome the height of the Raspberry Pi's USB and RJ45 connectors. The Raspberry Pi is then plugged upside-down in the 40 pin connector using its GPIO pins and additionally fixed to PCB with four screws in the corner for better fixation. Distance holders of 2 cm in respect to the PCB are used to keep the Raspberry Pi well fixed and connected via GPIO pins.

Figure 7: Raspberry Pi 2 Model B Top and Bottom view [9]

Pin#	NAME	NAME	Pin#
01	3.3v DC Power	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	Ground	06
07	GPIO04 (GPIO_GCLK)	(TXD0) GPIO14	08
09	Ground	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	Ground	14
15	GPIO22 (GPIO_GEN3)	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	Ground	20
21	GPIO09 (SPI_MISO)	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	(SPI_CE0_N) GPIO08	24
25	Ground	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	(I2C ID EEPROM) ID_SC	28
29	GPIO05	Ground	30
31	GPIO06	GPIO12	32
33	GPIO13	Ground	34
35	GPIO19	GPIO16	36
37	GPIO26	GPIO20	38
39	Ground	GPIO21	40

Figure 8: Raspberry Pi 2 Model B GPIO Pin Configuration [10]

3.4. Inertial Measurement Unit

The Inertial Measurement Unit used in this project is LSM9DS0 from STMicroelectronics producer. In the beginning, just the device itself was ordered but as it is rather small with only 4x4 millimetre of surface and 24 Pins on it, it was hard to solder it. At some point later, a device already soldered on a small breakout board was ordered from Adafruit Industries LLC manufacturer, for the sake of simplicity. Together with the breakout board, the IMU dimensions are 33mm x 20mm. And it is quite cheap with the price of around 25 Euros, but yet very powerful and sufficient for the needs of this project. Moreover, the power consumption is only 6.5 mA in normal operation mode with all sensors turned on. System is powered with 3.3 Volts.

LSM9DS0 IMU features 3D acceleration sensor, 3D gyroscope or angular rate sensor and 3D magnetic sensor which makes it a 9 DoF device. All sensors are digital. Inside of it, there are actually two IC's. One is L3G4200D gyroscope and other is LSM303DLMTR accelerometer and magnetometer. That's why there are two pins for chip select, CSG for gyroscope and CSXM for accelerometer and magnetometer. Measurement full scales are as following[11]:

- Linear acceleration ±2/±4/±6/±8/±16 g
- Gyroscope or angular rate ±245/±500/±2000 dps
- Magnetic field ±2/±4/±8/±12 gauss

Although magnetic sensor is available, it is not used in this project. Only gyroscope and accelerometer are used, combined with the Complementary filter.

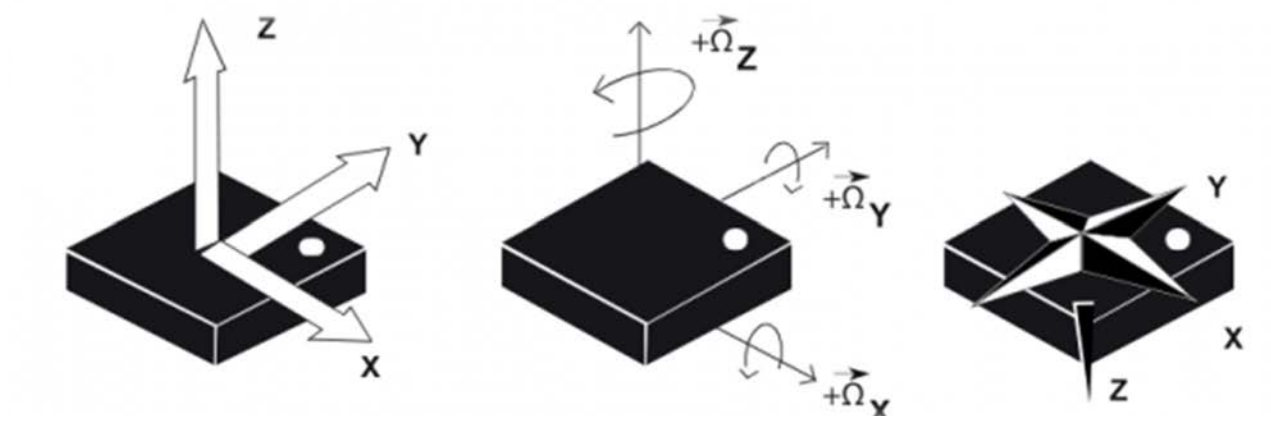


Figure 9: LSM9DS0 IMU with three 3D axis explanation[11]

Communication protocols I2C and SPI are included which makes it easier for communication with any microcontroller. Although SPI protocol is easier to implement, only I2C protocol is used in this project because it is already used for communication with some other electronic devices. And I2C requires only 2 bidirectional open drain lines for communication which is easier for later PCB design. Voltage used is +3.3 Volts. Both Raspberry Pi's and LSM9DS0's communication pins are not tolerant to higher voltage. Serial Data Line SDA and Serial Clock Line SCL are pulled up to +3.3 Volts with 2.2 kOhm resistors.

IMU came already soldered to a small breakout board. Out of 14 Pins available, only four are soldered to the PCB and are needed for this project: Vin, GND, SCL and SDA. Vin is connected to input voltage of 3.3 Volts, GND is connected to ground, SDA is connected to I2C SDA line and SCL is connected to I2C SCL line.

Accelerometer, with I2C address 0x1D, is used with the following settings:

- 6.25 Hz data rate
- X, Y, Z axis enabled
- Antialiasing filter bandwidth is 773 Hz
- +/- 2G full scale

Gyroscope, with I2C address 0x6B, is used with the following settings:

- 95 Hz output data rate with 12.5 cut-off
- X, Y, Z axis enabled
- 245 dps full scale

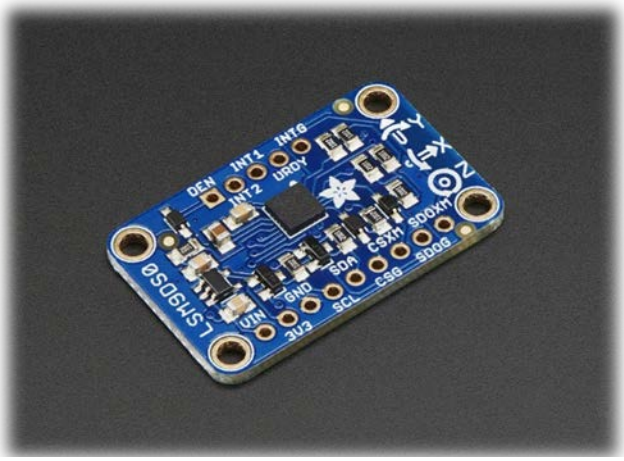


Figure 10: LSM9DS0 on the Breakout Board[12]

3.5. Temperature Sensor

Temperature sensors used in this project are two DS18B20, digital temperature sensors from Maxim Integrated Products.[13] Both came in already pre-wired and waterproof version, since they are used outdoor. One is attached on the Rain Gauge and measures temperature on the protective and heating grid and other is mounted outside the box to measure environmental air temperature.

DS18B20 sensor is fairly precise with +/- 0.5 °C accuracy in range from -10°C to 85°C. Operating temperature range is from -55°C to +125°C. On the tip of the sensor is a stainless tube, 30 mm long and 6 mm in diameter. Sensor is digital so there will not be any signal degradation even on longer distances. This is important for the one mounted on the Rain Gauge protective and heating grid – the cable is around 2 meters long with one interconnection point in the middle. Reading resolution is selectable, in the range from 9 to 12 bit. Communication protocol is 1-Wire Maxim which requires only one pin for communication. Multiple devices can be connected to the same digital pin because each device has an unique 64 bit ID.

Sensor has three wires. Red one is connected to 3.3 Volts, Black is connected to the ground and Yellow is Data, connected to physical pin 7 of the Raspberry Pi or Broadcom chip-specific GPIO pin 4. A 4.7 kOhm resistor is used as a pull up from the Data to VCC or Positive Supply Voltage line. Although each DS18B20 sensor connector is connected to its own data pin on the Raspberry Pi, only GPIO pin 4 is used because Broadcom chip supports only one 1-Wire bus. Once the PCB is turned, it can be seen that data pins are in short-cut to GPIO pin 4 and line to GPIO pin 17 is scratched.

Figure 11: Waterproof and pre-wired DS18B20 [14]

3.6. Temperature and Humidity Sensors

For the measurement of temperature and relative humidity, two different sensors were used in this project. One is AM2302, which is used for environmental relative humidity measurement and the other one is HIH6130 which is used for temperature and relative humidity measurement inside the steel box in which the Monitoring Tool is packed.

3.6.1. AM2302 Sensor

Temperature and humidity sensor AM2302 is a slightly modified version of DHT22 sensor. Compared to the latter one, it is encased and pre-wired, produced by Adafruit Industries LLC manufacturer. [15] With price of around 15 Euros, it is a good bargain considering the fact that is made for outdoor usage with wide operating temperature and humidity, in range from -40°C to +80°C and from 0% to 100% RH.

It features a capacitive humidity sensor and temperature dependent resistor or thermistor. It is temperature compensated and already calibrated, with the digital output signal. Measurement accuracy is +/-0.5°C and +/-2% RH. Sensitivity or resolution is 0.1 °C and 0.1% RH. Communication protocol is 1-Wire which requires only one wire and one microprocessor's digital pin for communication. It's different from Maxim/Dallas 1-Wire bus and thus incompatible so it must be used on different Broadcom's GPIO pin.

For connection, there are three wires available. Red one is connected to 3.3 Volts, Black is connected to the ground and Yellow is Data, connected to the physical pin 13 of the Raspberry Pi or Broadcom chip-specific GPIO pin 27. A 1 kOhm resistor is used as a pull up from the Data to VCC or Positive Supply Voltage line. As mentioned in previous chapter, Broadcom chip supports only one 1-Wire bus of its kind so new GPIO pin was introduced to operate with AM2302. It's pin 27.

The sensor is positioned on the side of the steel box, hitched with M4 screw. It is used only for environmental relative humidity measurement. Environmental temperature measurement is discarded and considered corrupted because the steel

box is heated from the inside and thus it makes temperature reading incorrect. For outside environmental temperature, DS18B20 from the previous chapter is relevant.

Figure 12: AM2302 on the left and HIH6130 on the right [16][17]

3.6.2. HIH6130 Sensor

Temperature and relative humidity sensor HIH6130 is a digital output-type sensor produced by Honeywell manufacturer. Energy efficiency and small SMD Surface Mount Device package combined with the price of approximately 13 Euros makes it an optimum choice for the needs of this project. Package is only 6 mm long and 5 mm wide with 8 legs later soldered to the PCB. [17]

With 14 bit ADC, resolution is 0.04% RH and 0.025°C. Accuracy is +/-4% RH and +/- 1°C over a compensated range from 5°C to 50°C and from 10% RH to 90% RH. This compensated range is of interest because sensor is used inside the heated steel box which protects the whole Monitoring Tool and has a stable temperature of around 25°C and significantly lower RH than environment. Main purpose of this sensor is to control temperature and relative humidity level inside the steel box and depending on the temperature level, inside-box heating is switched. It's powered with 3.3 Volts and drains only 650 µA in full operation. I2C protocol is used for communication and address is 0x27. SDA and SCL lines are pulled high with 2.2 kOhm resistors. Alarm outputs are not used. Voltage line is decoupled with capacitors.

3.7. Pressure Sensor

Pressure sensor used in this project is SSCMANT006BA2A3, a board mount sensor produced by Honeywell manufacturer. It is part of TruStability SSC Standard Accuracy Silicone Ceramic series, already pre calibrated on the temperature range from -20°C to +85°C. As it works as absolute pressure sensor, an internal vacuum reference is built inside and output value is proportional to the absolute pressure. [18] With the price of around 40 Euros, it definitely beats the competitive sensors in price-to-value ratio although is a bit above the initial planned budget for this type of sensor. It comes in the SMT package, with 8 legs later soldered to the PCB but only four of them are important for proper sensor operation. Length is 10 mm and width is 13,3 mm. Communication protocol is I2C with SDA and SCL lines pulled high using 2.2 kOhm resistors. A decoupling capacitor is added to VCC line. From the name label, it is easy to extract the properties of the sensor:

- SSC – Standard Accuracy, Compensated/Amplified
- M – SMT package
- AN – Pressure port is single axial barbed port
- T – Optionally: liquid media on Port 1, no diagnostics
- 006BA – Absolute pressure, in range from 0 to 6 bar
- 2 – I2C communication protocol, address is 0x28
- A – Transfer function with respect to the input voltage and pressure, 10% to 90% of VCC, 2^{14} counts (digital)
- 3 – Supply voltage is 3.3 VDC

Figure 13: SSCMANT006BA2A3 Absolute Pressure Sensor [18]

3.8. USB 3g Mobile Internet Dongle

In order to access the Internet over mobile network, many possible options were considered, from various modems which should be soldered to PCB to USB dongles. Important requirements were low power consumption, low or none current bursts in some operation modes, SIM card friendly and option for external antenna. Huawei E3131 HiLink 3g USB dongle was chosen as optimal solution. It's paired with external UMTS antenna and cable with small CRC9 connector. All three parts cost approximately 35 Euros.

Comparing to the competitive options, it is rather simple to use, just insert SIM card inside, plug it in the USB port of the Raspberry Pi and configure the necessary software to make it operational. Possible disadvantage was a rather big size of 85.40 mm in height, 27.00 mm in width and 12.45 mm in depth because the Monitoring Tool's protective box size was still unknown. But as a box of 20x20 cm was chosen, dongle size was not an issue any more. Second very important thing to concern was power consumption. As the Monitoring Tool will be placed on a remote location, (possibly) far from the mobile provider's transceiver station, power consumption will rise as system is trying to maintain stable connection with the mobile Internet provider transceiver station. It is known fact that transmitting/receiving power is proportional with distance between communicating sides. Thus, USB dongle's power consumption will be greater with distance from the mobile provider transceiver station. Maximum allowed power consumption by device plugged in the USB port of the Raspberry Pi is 500 mA. As USB dongles tend to pull a bit more in bursts when starting up and establishing connection, let's reduce that level to 400 mA, just out of precaution. The selected USB dongle fits that requirement as well. From power consumption measurements done in [19], we can clearly see that maximum power consumption is 258 mA in heavy upload using 2g network or 188 mA in heavy upload using 3g network.

Additional safety mechanism to avoid current bursts when powering on the USB 3g dongle is added, in simple form of Raspberry Pi's delayed boot. As it was mentioned before, the Monitoring Tool is made to be "plug and play". Once it is connected to the power grid, all components are powered. It means that Raspberry

Pi and USB dongle will start at almost same time, with latter one being powered by the first one. An unwanted situation that might easily happen is that Raspberry Pi enters into "continuous reboot mode". If the Raspberry Pi gets sudden current drops while booting, i.e. by USB dongle from USB port, it will stop the booting process and restart it. But when restarting the Raspberry Pi, its USB port will also experience the power drop thus the USB dongle will restart as well. And that is a continuous reboot loop which might happen if both devices start at the same time. To avoid that, a simple software fix was made in the Raspberry Pi's kernel to delay the operating system boot for 20 seconds. Now, once the Monitoring Tool is powered on, the current will rush through the Raspberry Pi and power on the USB dongle. After approximately 20 seconds, USB dongle will already be in idle state and Raspberry Pi's operating system will start the boot process. The exact steps are explained in later chapters, together with "free space problem" which is addressed in software chapter.

Figure 14: Huawei E3131 USB 3g dongle with CRC9 antenna extender cable

3.9. Other components

Main components of the Monitoring System were described in previous chapters. Now, let's go through the other used components, equally important for the overall system functionality as the previous ones.

Electronic components include various capacitor, resistors, switches, diodes etc. Capacitors, tantalum and electrolytic, were mostly used for decoupling. They are attached between VCC and GND line to try keeping the voltage as constant as possible, i.e. to "smooth" the sudden changes in voltage or simply called noise. In a DC circuit, there is no problem of short circuit since capacitor functions like an open circuit. They must be soldered as close as possible to the element they are "protecting". Tantalum capacitors must be soldered with caution because polarity is important. Resistors are mainly used as pull up or pull down or together with LED as current protectors. Here is the full list of electronic components used on the Monitoring Tool:

- Tantalum capacitors, 1206 SMD package: 22µF x3, 100µF x1
- Electrolytic capacitors, 1206 SMD package: 100nF x8, 220nF x1
- Resistors, 1206 SMD package: 100 Ohm x2, 200 Ohm x2, 680 Ohm x1, 1kOhm x3, 2.2kOhm x2, 2.4kOhm x2, 4.7kOhm x3, 100kOhm x3
- Chassis mount heat emitting resistors: 7.5 Ohm x2, 10 Ohm x1
- DC/DC converter R-625.0P by Recom Power, x1
- AC/DC converter PMT-12V50W1AA by Delta Electronics PMT, x1
- LED control lights, 1206 SMD package, x2
- LED control light, wired with thread, x1
- Fuses 3 Amps with fuse holders 5x20 mm, x2
- Zener diode 5.6 V, 1206 SMD package, x1
- BSS138LT1G N-channel MOSFET, SOT 23 package, x1
- Low side switch VNN3NV04PTR-E by STMicroelectronics, SOT 223, x2
- 40 pins header, 2.54 mm pitch, x2
- 2 and 3 pin nylon connectors, male and female, 2.54 mm pitch, x5 sets
- 2 and 3 pin connectors, male and female, 7.62 mm pitch, x3 sets

Other components used on the Monitoring Tool, apart from above stated electronic ones, are following:

- UV resistant 3-wire cable for 230 VAC, 5m
- UV resistant 10-wire cable, 3m
- UV resistant Schuko male plug, x1
- Steel box with lock, 20 x 20 x 12 cm, GL 66 by Hofmann, x1
- Aluminium L profile 15 x 15 x 200 mm, x2
- Steel L profile with M4 and M10 holes, x4
- Cable glands, M12 x3, M16 x1
- Universal 10A connector strip, x8
- Distance holders of 25 mm with M3 thread, x4
- Plastic tie strips, 3mm wide, x4
- Screws, nuts and washers, M2, M3, M4, M5

Components used for the Rain Gauge and their function will be explained later, here is just the list of them:

- Custom designed and 3D printed Funnel, Stand and Lever
- Plexiglas plates, x4
- Reed Switch x1, with NdFeB magnets x2
- Temperature sensor DS18B20, x1
- Heating pad, 12V 10W, x1
- Universal 10A connector strip, x8
- Cable gland, M16, x1
- Protective steel mesh, x1
- Threaded rod, M5, 18 cm long, x4
- Bolt screw, M4, 6 cm long, x1
- M4 lock nuts, x2
- M5 lock nuts, x8
- M5 nuts, x8
- M3 screws and nuts sets, x6

3.10. PCB Design

In the early stage of the project, electronic components were connected using prototyping board. After all the components were made to work together on the prototyping board, it was time to switch to the PCB. PCB design, including schematic, was done in CadSoft Eagle 7.6.0 software. First, a schematic of the whole electronic circuitry was done. Later on, steel box was selected and PCB dimensions had to match the size of the box with some extra place in the corners for wires and better air flow inside. PCB dimensions are 180x165 mm, with four rectangle shaped gaps in the corner. On the edges, there are several M3 holes to fix the PCB on the L shaped aluminium panel which is later fixed to the steel box. PCB costs around 85 Euros which is a bit over planed budget. Truth to be said, all the electronic components used could be fitted to a smaller and thus cheaper PCB. However, for the sake of simplicity, a PCB of this dimensions was designed to match the dimensions of the steel box and thus be properly fixed. Looking at the opened box from the bird-eye view, PCB and box look like an unity and it is pleasing to the eye.

PCB is made out of two copper layers, the top and the bottom one. On the top one, main voltage and signalling lines are passing and all surface mount components are soldered there. The bottom layer is serving as auxiliary route for voltage and signalling lines which cannot be connected in one piece due to some obstructions along their way, i.e. other lines. Both copper layers are grounded and tracks that can be seen are signalling and voltage lines. Private library was created for most of the components, including Device, Package and Symbol attributes. Some components already have open source library from [20] and [21] and they were just “moved” into Private library for easier manipulation. Design grid is set to 0.25 mm. Voltage lines are 1.016 mm thick and signalling lines are 0.4064 mm thick. Top and bottom layers are connected with vias. Two cooling rectangles under low side switches were not named accordingly so they were not recognized thus not made by machine. Instead, that area was manually scratched to separate it from top grounding copper layer. Additionally, improvised passive copper cooling was soldered on those rectangles to better dissipate the unwanted heat.

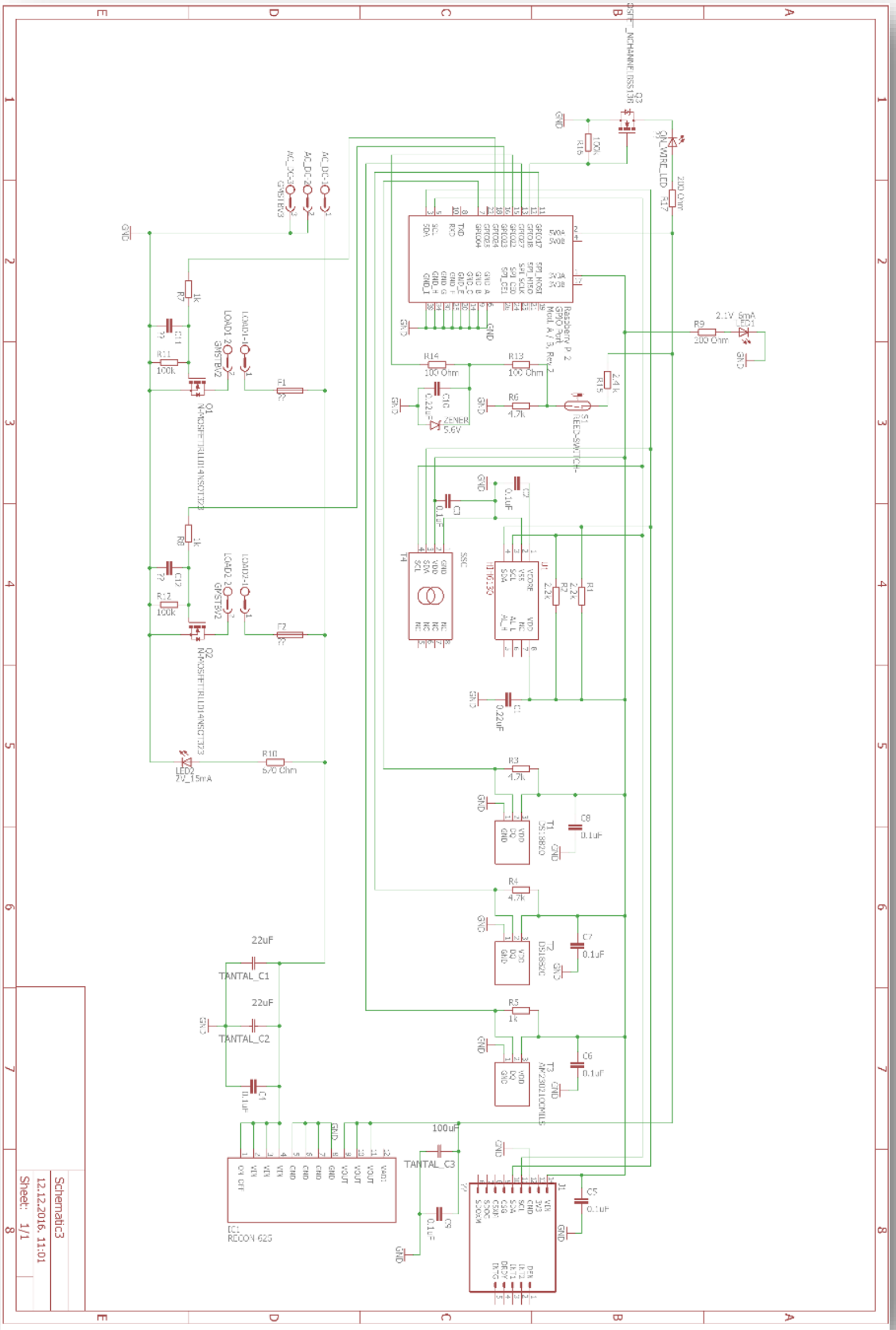


Figure 15: Schematic of the Monitoring Tool made in Cadsoft Eagle 7.6.0

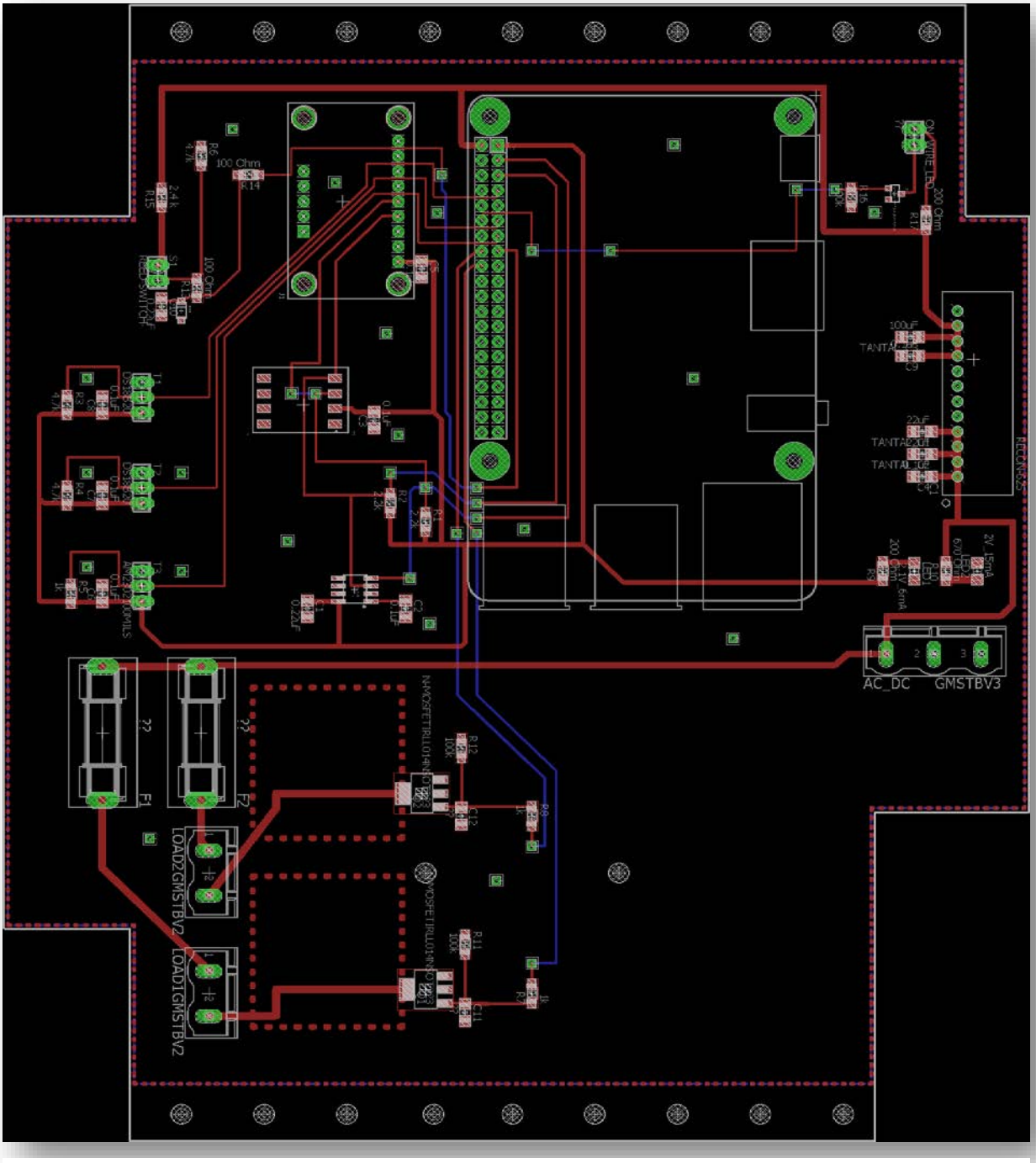


Figure 16: PCB in 1:1 ratio. Red lines represent voltage and signalling lines from Top layer and blue are lines from Bottom layer

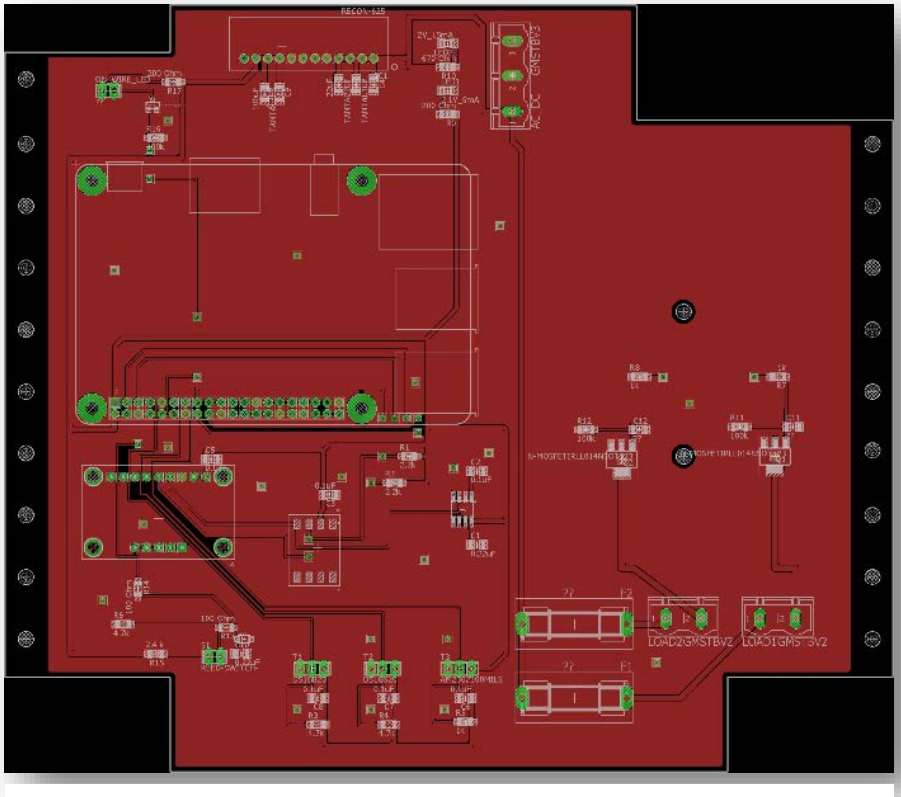


Figure 17: PCB Top layer

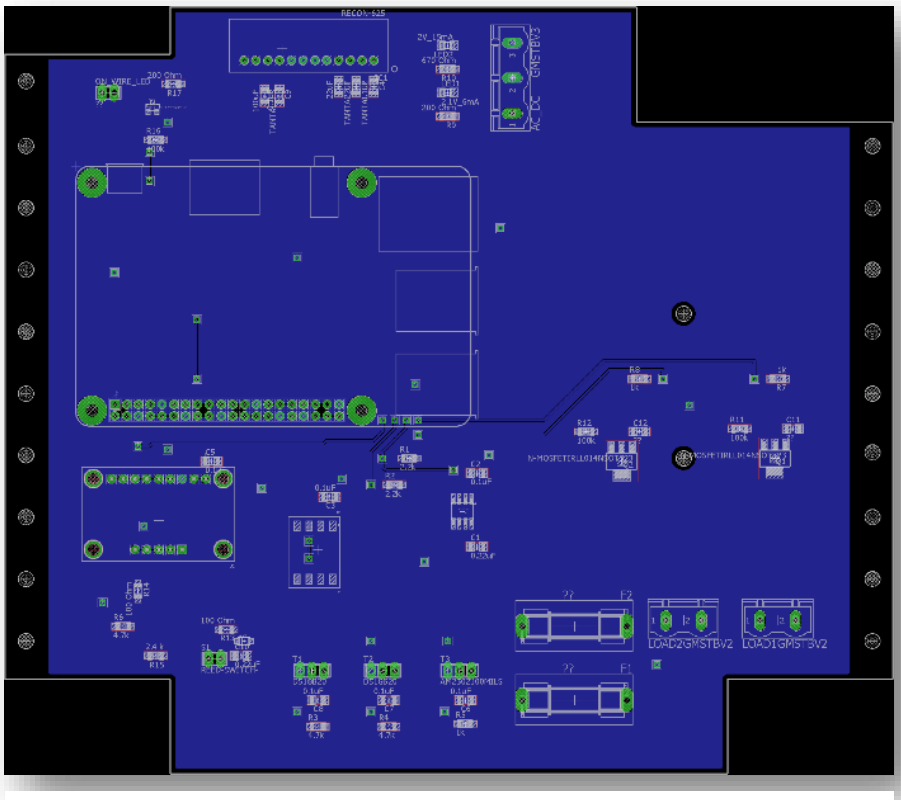


Figure 18: PCB Bottom layer

3.11. Rain Gauge

The Rain Gauge is a device for measuring precipitation. It measures all kinds of precipitation, i.e. any kind of condensed atmospheric water vapour falling to the ground. Various different approaches were already described in previous chapter. Now, we will focus on the one that was custom made for the needs of this project. It is a slightly modified version of the tipping bucket rain gauge. Water is not stored, it flushes out through a drain holes on the bottom. Another difference is the shape, it is cuboid instead of cylindrical shaped bucket.

The device is fully custom made, following the examples of the ones made for the market and the ideas described in [4], [22] and [23]. It measures 160 mm in width, 90 mm in length and approximately 180 mm in height. Height measurement is not exact, it depends on how much the top and the bottom nut are tight with respect to the threaded rod. It can oscillate up to 5 mm. The device consists of three main parts and several auxiliary ones.

Main parts are the Funnel, the Lever and the Stand. All parts are custom designed and printed in 3D printer using UV resistant all-weather polymer ASA. Design was done using Autodesk Tinkercad, an easy-to-use and intuitive online 3D CAD design tool for simple projects like this one. Funnel is top part of the Rain Gauge, measuring 160 mm in width, 90 mm in length and 63 mm in height. It looks like flipped 4-sided pyramid with rectangular base. Inside, it is “empty” and that empty space works as a funnel. Rectangular top of the Funnel measures area of 75.64 square cm and narrow cylindrical drain measures 4 mm in diameter. The Stand is 160 mm wide, 90 mm long and 60 mm high. As the name suggests, it is a bottom part of the Rain Gauge and serves as a stand for the whole Rain Gauge, drain for the water and fulcrum around which the Lever is pivoting. It has two vertical stands which are holding the fulcrum, i.e. M4 threaded rod. The Lever is “middle” part of the Rain Gauge. It measures 105 mm in width, 30 mm in length and 40 mm in height. It has a 5 mm in diameter cylindrical hole in the central weight distribution point and it is pivoting around that point over fulcrum. There is no “middle” state, it can stay only in one position, i.e. top left or top right. On the both sides, there are two compartments with volume of 14.5 cubic cm. Considering the weight of the whole

Lever together with water and pivoting angle, only 7.94 cubic cm of water is enough to out-weight the counterpart and turn the Lever to another side. As mentioned before, it can end up only in one side, there is no middle state.

Other auxiliary parts are protective grid, heater, temperature sensor, magnets, reed switch, universal 10A connector strip, Plexiglas housing, threaded rods, calibration screws, fulcrum screw and nuts. Protective grid is on top of the Funnel. It is steel mesh which protects the Funnel entrance from various types of dirt that might choke it. Heater is a 10W heating pad, fixed to the protective grid. It heats the grid once the temperature drops below 2°C and melts the snow which might block the Funnel entrance. And snow is also a kind of precipitation so once it is melted, it can be measured by the Rain Gauge. Temperature sensor DS18B20 is fixed to the protective grid. It is connected to the Broadcom chip-specific GPIO pin 4 and measures the temperature to determine whether it is necessary to switch on the heater. Magnets are glued to the Lever. As the Lever is pivoting, they move together and excite(close) the reed switch once they pass nearby. The reed switch is connected to the Broadcom chip-specific GPIO pin 22. Inside the Rain Gauge, it is fixed to the Plexiglas housing in the height level which is maximum to the magnets on the Lever. Maximum height that magnets on the Lever can achieve is when the Lever is in the “middle” position between two sides(states). But as there are only two finite states for the Lever, that middle point is just a quick pass. Each time the magnets pass near the reed switch, the reed switch closes and software interrupt routine from the Raspberry Pi records that event. One pass equals to the one compartment full of water, i.e. to 7.94 cubic cm of water. Plexiglas housing sits between the Funnel and the Stand. It is protecting the interior of the Rain Gauge but as it is transparent, it allows us to see what is going on inside the Rain Gauge. On the front and the back are two plates, 150 mm wide, 130 mm high and 4 mm thick. On the sides are two plates, 72 mm wide, 130 mm high and 4 mm thick. The Funnel and the Stand have 4 mm wide and 5 mm deep channel shaped bearings on all four sides so Plexiglas plates can fit perfectly and form a nice unity. Universal connector strip with 8 positions is fixed on the housing plate from the inside. It serves as interconnection point between electric/electronic devices of Rain Gauge and the Monitoring Tool. All Rain Gauge’s device wires meet at that connector strip. Their

counter part is an UV resistant, 3 m long cable bundle which leads to the Monitoring Tool. Reed Switch has 2 wires, DS18B20 has 3 wires and heater has 4 wires or 2 per pole. Inside the Monitoring Tool, these wires are plugged in the appropriate connectors on the PCB. Four threaded rods, M5, with length of 180 mm are holding the whole Rain Gauge united. They pass through the M5 cylindrical holes in the corners of the Stand and the Funnel and are fixed with nuts under the Stand and above the protective grid. Calibration screws are M5 screws located on the Stand. They control the pivoting angle of the Lever. Fulcrum screw, M4, goes from one side of the vertical stand to the another side, passing through the Lever's central weight distribution point. Around the fulcrum screw, the Lever pivots.

Principle of operation is rather easy to understand. Precipitation water is passing through the Funnel and tipping down. Right under the Funnel is the Lever. The Lever can stand only in two sides, top right or top left. That means that other part of the Lever is standing above the fulcrum and water is tipping inside it. Once the water inside that compartment reaches the volume of 7.94 cubic cm, it out-weights the other side of the Lever, the Lever turns and water of the compartment drains through holes of the stand. In the meantime, the other compartment is already standing above the fulcrum and water is tipping inside of it. Once the Lever switches sides(states), magnets pass near the reed switch and excite(close) it. That event is recorded by software implemented interrupt routine. One event means 7.94 cubic cm of precipitation per 75.64 square cm of area. The measurement is expressed in litres per square meter of area in one hour. One compartment equals 1.2516 l/m3.

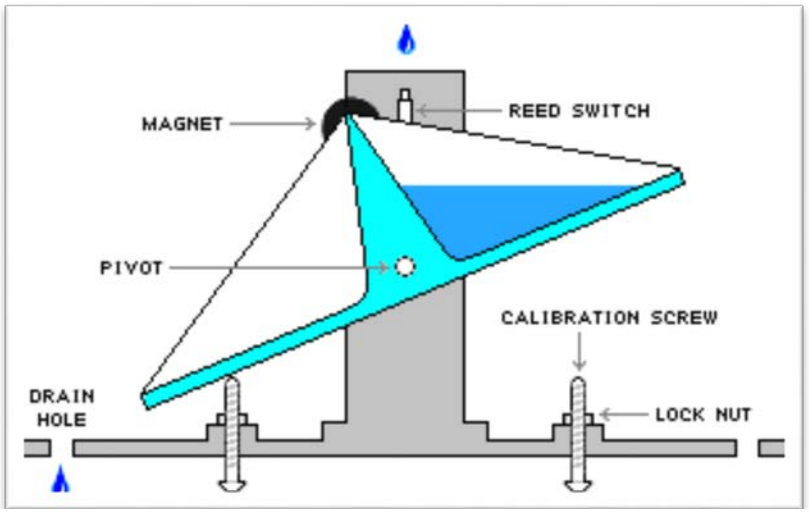


Figure 19: Rain Gauge principle of operation [22]

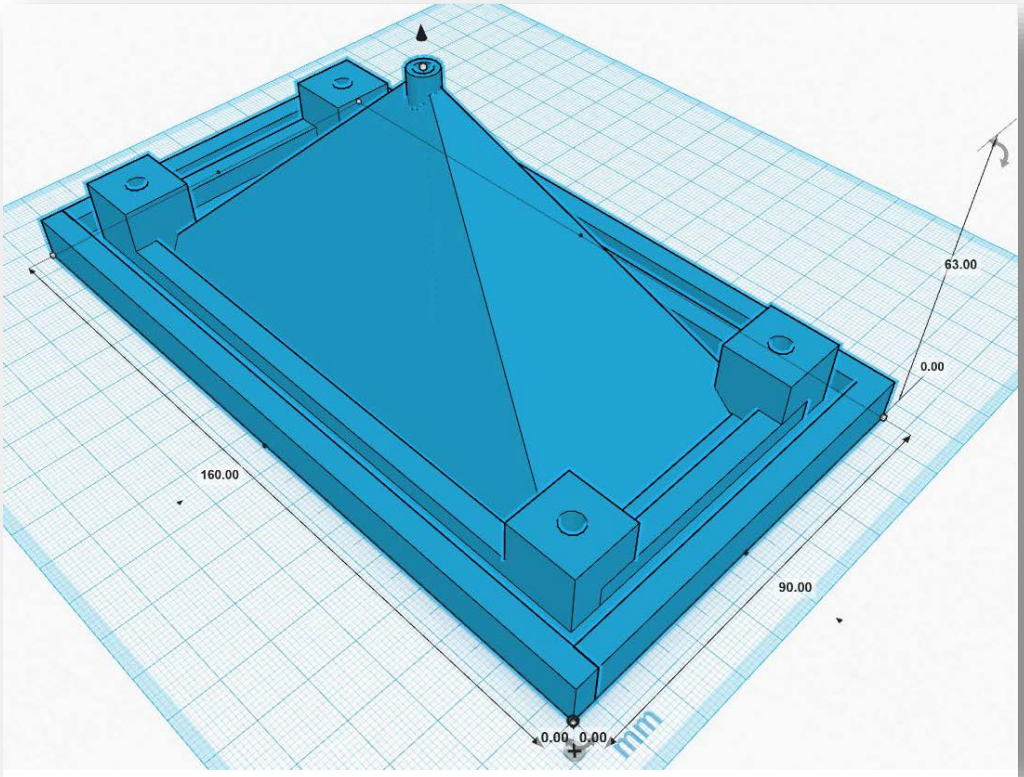


Figure 20: Funnel

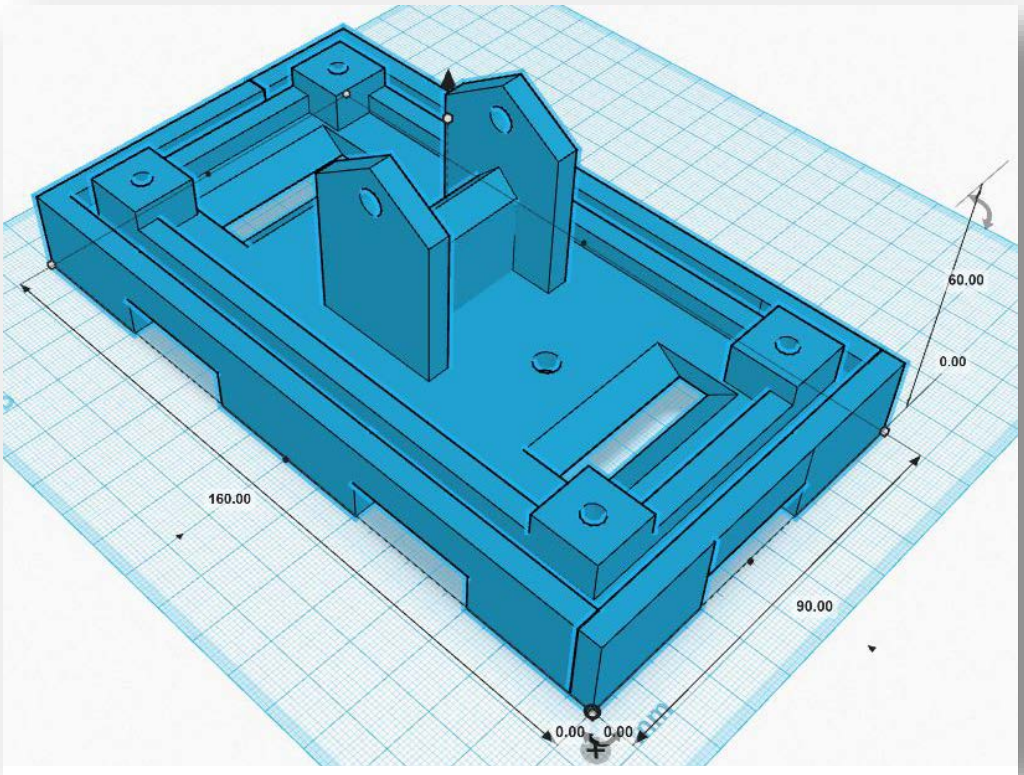


Figure 21: Stand

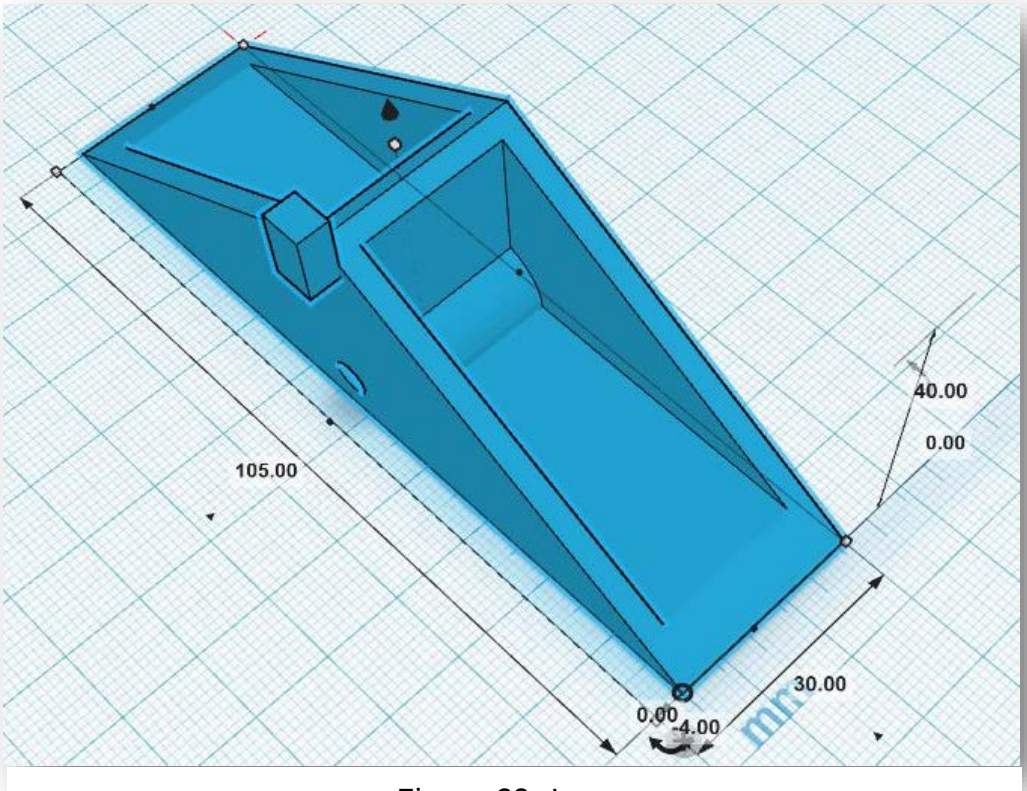


Figure 22: Lever

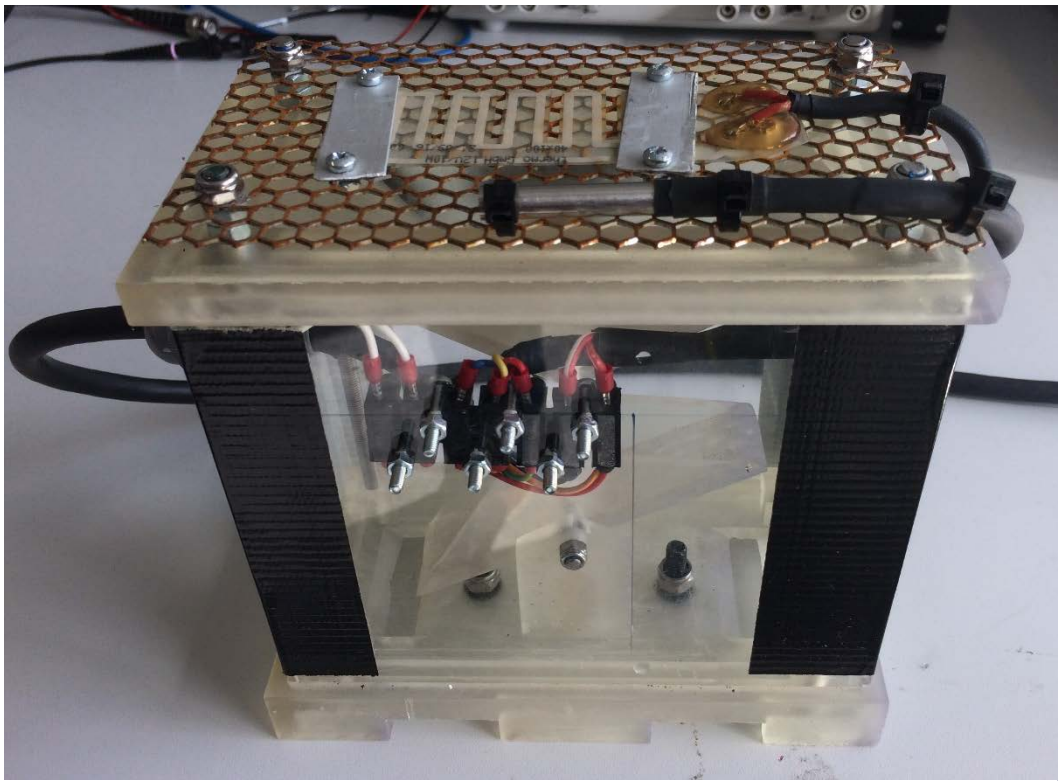


Figure 23: Rain Gauge

Chapter 4

Software Implementation

In the stage of planning the project, analysis of possible hardware and software solutions was done. Two main "software tools" had to be chosen in advance: the programing tool and the data stream network.

A programing tool used in this project is Python 27. It is a high level, general purpose programming language. As the author of this project, I had the right to use any programing tool. Python 27 was the optimal choice considering the fact that I already poses considerable knowledge and experience in Python 27, there are plenty available libraries and scripts which already interact with most of the components used in this project and moreover, it has a huge online community willing to help and support with tips and advices if something goes wrong. In addition, Raspberry Pi Foundation recommends Python, although any language which will compile for ARMv6 (Pi 1) or ARMv7 (Pi 2) can be used, like C, C++, Java, Scratch, and Ruby. [34]

Data stream network used in this project is Pubnub. The service such as this one provides a real-time network and stream of data. A huge advantage in IoT is that the user can focus on the core "interest", one does not have to think on how to exchange the data between communicating parties. Pubnub can be easily integrated in Python 27 applications and other platforms, it has a large base of use cases [35], network and stream are stable and free package of 300 000 messages per month is enough for the needs of this project. Moreover, I had some previous experience with Pubnub and considering everything mentioned here, Pubnub is the optimal solution.

4.1. Setting up Raspberry Pi

Central processing unit of the Monitoring Tool is the Raspberry Pi 2 Model B. The device itself was already explained thoroughly, but now an insight in how to set it up will be given. The chapter is following the work in [27], since the scope of this project and the one in [27] are similar in some manner.

Once ordered, the device arrived totally empty. First step was to write the operating system to the micro SD card. Raspbian Jessie was chosen as the optimal operating system. It is a version or distribution of Debian Linux Operating system, optimized for the Raspberry Pi. Operating system was downloaded as an Image file from the official Raspberry Pi site, at `http://www.raspberrypi.org/downloads/`. Later on, it was written or installed as disk image to the micro SD card using Win32DiskImager, an open source application.

With a Raspbian disk image on a micro SD card, it is time to power on the Raspberry Pi. Software Configuration Tool appears first and there we configure that all the micro SD card storage is available to the operating system. It requires a reboot and after that we have to insert the default credentials to log in. The default credentials are still in use, they were not changed during the development and after the delivery of the whole Monitoring System.

Username: `pi`

Password: `raspberry`

At this moment, we have full access to the operating system and all of its features. There are several applications and software packages already preinstalled so they should be updated to the latest version. Of course, Internet connection is required for this step. To establish Internet connection, the Ethernet port should be connected to the router(or any other socket) using RJ45 connector.

Find the latest available software versions: `sudo apt-get update`

Upgrade current software to that version: `sudo apt-get upgrade`

Now, we have an operating system updated to the latest version and Internet connection. The next step is to connect the PC and the Raspberry Pi to enable the

remote access. The PC, with Windows 7 operating system, is used as a development or host computer and the Raspberry Pi as a client. From the remote access application's perspective, PC is client and the Raspberry Pi is server. Important benefit of this way of working is that only one keyboard, mouse and monitor would be enough for both devices. The current IP address given by the network administrator should be static with some other minor updates to the file `/etc/network/interfaces`, so the final version looks like this:

```
iface eth0 inet static
address 129.27.131.22
netmask 255.255.0.0
gateway 129.27.131.1
```

One mode of remote access is already preinstalled on the Raspberry Pi and runs on start up. It is SSH or Secure Socket Shell protocol which is used for secure operation of network services over an unsecure network. From the SSH point of view, the Raspberry Pi acts as a server and PC as a client. To communicate using SSH, an open source applications named Putty.exe and WinSCP.exe are installed on the PC. With Putty, we have a command line connection via SSH and we can do (almost) everything just like we are working directly on the Raspberry Pi. WinSCP or Windows Secure Copy Protocol lets us transfer the files between the PC and the Raspberry Pi securely. For remote login to the Raspberry Pi, the default credentials previously mentioned are used.

The third and most used way of remote access in this project is TightVNC. It is software for Virtual Network Computing, free for personal use. What makes it better than Puttty is remote GUI environment. Putty lets us use only remote command line from the Raspberry Pi, but with TightVNC we have a remote GUI environment which is quite easier to use and navigate. It is a bit more power demanding than basic SSH but the Raspberry Pi is fine with that since we haven't done any cumbersome (graphical) operations. TightVNC does not have a file transfer feature so WinSCP is still the only option for that operation. TightVNC must be installed on both platforms, with the PC being client and the Raspberry Pi being

server. For the PC, there is an installation package on the official website and for the Raspberry Pi, a simple command should be executed:

`sudo apt-get install tightvncserver`.

Now, we must ensure that TightVNC starts at the boot of the Raspberry Pi, otherwise there is no full remote access. In the file `/etc/rc.local`, commands that run with the start-up are stored so we must append the following line to that file:

```
su - pi -c '/usr/bin/tightvncserver :1'
```

From this point on, every time we power on the Raspberry Pi, TightVNC is run. To connect from the PC, we must run the TightVNC Viewer and the address of the Raspberry Pi should be appended with `:1`, which means that we want to start the desktop session. Password is `raspberr`. The password does look a bit awkward(or at least unfinished), but TightVNC allows only 8 character password so that is the best we could do. To sum it up, the credentials for TightVNC Viewer are the following:

Remote Host: `129.27.131.22:1`

Password: `raspberry`

Communication with PC and with Internet is working. Now, we must enable other communication protocols, mainly for communication with sensors. 1-Wire communication over 1-Wire Bus is enabled by default. However, it is not fully configured since there is no BIOS that checks the hardware upon boot. That being said, we must append the /boot/config.txt which does the “BIOS job”:

```
dtoverlay=w1-gpio
```

Once there is a device connected to the GPIO, with a simple command LKM or Loadable Kernel Module is added or removed to the Linux kernel. On the example of the DS18B20 temperature sensor used in this project, it looks like this:

```
sudo modprobe w1-gpio
```

```
sudo modprobe w1-therm
```

Now, when we enter the `/sys/bus/w1/devices` directory, DS18B20 temperature sensor is listed with ID looking like this: `cd 28-xxxx`, where “xxxx” matches the unique serial number of the device. Namely, all the 1-Wire devices connect to the

same GPIO and 1-Wire bus. To distinct between them, each has a hard coded, unique serial number from the manufacturer to avoid the collision.

SPI and I2C protocols are disabled by default. SPI is not used in this project so it is not of interest. To enable I2C, first we need to load the necessary kernel support modules. In the Advanced Options section of the Raspberry Pi Software Configuration Tool, we must enable the `ARM I2C` interface and also set the `I2C Kernel Module` to load by default. After the changes, reboot is necessary. Later on, in the file `/boot/config.txt`, we append the following lines:

```
dtparam=i2c1=on
```

```
dtparam=i2c_arm=on
```

Now, we load the tools to work with the I2C devices with the following command:

```
sudo apt-get install i2c-tools
```

At this moment, I2C communication is enabled and fully configured. Provided that all the I2C devices are powered and connected accordingly, we can see the devices and the hexadecimal addresses they are using. With the following command, we get the I2C addresses map:

```
sudo i2cdetect -y 1
```

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- 1d -- --
20: -- -- -- -- -- -- -- 27 -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- 6b -- -- -- --
70: -- -- -- -- -- -- -- 77
```

As we can see from the I2C address "table", the occupied addresses are 0x1d which is the address of the accelerometer of IMU LSM9DDS0, 0x27 is the HIH6130 temperature and humidity sensor, 0x6b is the gyroscope of IMU LSM9DS0 and 0x77 is the SSCMANT006BA2A3 pressure sensor. As we are using Python for programming, we need to install the appropriate modules to work with I2C communication using the following line:

```
sudo apt-get install python-dev python-smbus
```

4.2. Mobile Internet using 3G USB Dongle

Internet connection is a must, in order to transmit the data from the remote location. As there would be no cable Internet option in the remote operating areas, the optimal solution was to implement the Internet connection using options from the Mobile Internet provider. Looking retrospectively, enabling the Mobile Internet feature was one of the main challenges of this whole project. Huawei E3131 HiLink 3g USB dongle together with Hofer Telekom SIM card was used to establish the Internet connection. Reading the online articles, forums and blogs, it was soon concluded that this is a quite challenging task. The only similar works were the ones in [28] and [29] so this chapter is closely leaned to those blog articles.

Once the USB dongle is plugged in the USB port, Rasbian Operating system is immediately asking for the authentication to mount the USB device. This is a good security feature from the Operating system's point of view, but it is annoying and time consuming to always enter the credentials, especially when only one USB device will be used, totally monitored by the end owner, like in our case. In order to mount the arbitrary USB device automatically, without authentication, the mounting policy file `/usr/share/polkit-1/actions/org.freedesktop.udisks2.policy` must be edited in the following way:

```
<action id="org.freedesktop.udisks2.filesystem-unmount-others">
<defaults>
      <allow_any>yes</allow_any>
      <allow_inactive>yes</allow_inactive>
      <allow_active>yes</allow_active>
</defaults>
```

From this point onward, our USB dongle will mount automatically, without any authentication needed. The other problem which emerged is the considerable free memory space on the Huawei E3131 HiLink 3g USB dongle. In the datasheet of the dongle, it is noted that it comes together with the Windows driver on the hidden disk partition. Although its primary use is to be the Mobile Internet dongle, Operating

system does not know that. It mounts the dongle as the storage device since there is some amount of free memory on the dongle. To avoid this scenario, we must install the `usb-modeswitch` program, using the following command:

```
sudo apt-get install usb-modeswitch
```

The program works automatically for most of the Huawei devices so we do not have to do anything more, it will mount the device as USB Internet modem.

Now, the USB dongle mounts as Internet modem, without authentication needed so the next step is to connect it to the Internet. For this task, a shell script called `Sakis3g` will be used. It came as a result of a project, run by Sakis Dimopoulos, to make an easy all-round solution for establishing a Mobile Internet connection using any kind of modem and operator combination. Although the author abandoned the project some years ago, the script still works like a charm and it saved a whole bunch of time then manually creating an Internet establishing script would. To make it work, first we need to download it from some online repository and convert it to executable using the following command:

```
chmod +x sakis3g
```

Later on, we must make it run automatically, without authentication needed. In other words, it must act as a system default application with super user rights. To do so, it must be moved to `/opt` directory and executable file to `/usr/bin` directory:

```
sudo mkdir -p /opt/sakis3g/
```

```
sudo mv sakis3g /opt/sakis3g
```

```
sudo chown root:root /opt/sakis3g/sakis3g
```

```
sudo ln -s /opt/sakis3g/sakis3g /usr/bin
```

In the `visudo` script, no authentication property must be appended in this way:

```
pi ALL = NOPASSWD: /opt/sakis3g/sakis3g
```

In this step, the `ppp` program must be installed to enable data connection:

```
sudo apt-get install ppp
```

Now, we are ready to connect to the Internet. All we have to do is to provide the appropriate attributes needed by the `Sakis3g` program. That can be extracted from the mobile Internet provider's instruction manual. In the project, Mobile Internet provider is Hofer Telekom. On their website, all the necessary data is available for the users. Sakis3g can be run in two modes: interactive mode or by providing the necessary attributes when called from the console. For the first use, the simplest way is the interactive mode. It is user friendly with nice GUI. There, we do the following steps to connect to the Internet, already knowing from the provider's manual that our APN is `webaut` and credentials for authentication are empty:

```
sudo sakis3g --interactive
```

```
Connect with 3g -> Custom APN -> webaut -> (blank) -> (blank)
```

If it all worked well, a notification that Internet connection is established pops out on the screen. And our USB dongle's LED light is either green or blue, meaning that we are connected to the 2g or 3g network, respectively. The next step would be to automatize this. First we generate the `sakis3g success report`. There we can see all the attributes needed to establish a connection and we simply copy the necessary parts of that report to the `Sakis3g` configuration file `/etc/sakis3g.conf`:

```
USBDRIVER="option"
USBINTERFACE="0"
APN="CUSTOM_APN"
CUSTOM_APN="webaut"
APN_USER=""
APN_PASS=""
MODEM="12d1:1506"
```

From now on, every time we want to establish an Internet connection, all we have to do is to type the following command:

```
/opt/sakis3g/sakis3g --sudo "connect"
```

`Sakis3g` is truly an amazing program. It does all the Internet connecting job by itself. However, it comes with a big oversight – it is not possible to establish the

connection again, once it is lost. Namely, `Sakis3g` establishes the Internet connection with ease, once it is run. But if the connection breaks at some point, it does not re-establish it again. And that is a big issue. Luckily, there is an another program, named `UMTSkeeper` which does exactly that job, in cooperation with `Saksis3g`. It is a free software under the terms of the GNU General Public License and HESSLA Hacktivismo Enhanced-Source Software License Agreement. First idea of solving the "re-establishing connection" issue was to run the `Sakis3g` on the start-up. That would establish the Internet connection once the Monitoring Tool is powered on. It would be paired with the script that would constantly check the Internet connection and once it is down, it would reboot the Raspberry Pi so the Internet connection would be re-established using `Sakis3g` on the start-up. But this idea was abandoned once the `UMTSkeeper` was discovered. The program tries to re-establish the Internet connection once it is down by constantly calling `Sakis3g`. No matter whether connection breaks due to our fault, i.e. dongle disconnected, or due to the mobile Internet provider's error, i.e. traffic congestion, it will try to re-establish the connection anyway. Installation procedure is straightforward, just download it from the online repository and place in the same folder where `Sakis3g` is, i.e. in `/opt/sakis3g`. It should run automatically after the boot and run constantly in the background to use its full potential. To enable that, we must append the following command to the `/etc/rc.local` file:

```
sleep 20

sudo /opt/sakis3g/umtskeeper --sakisoperators "USBDRIVER='option' USBINTERFACE='0' APN='CUSTOM_APN' CUSTOM_APN='webaut' APN_USER='user' APN_PASS='pass' MODEM='12d1:1506'" --sakisswitches "--sudo --console" --devicename 'Huawei' --log --silent --monthstart 8 --nat 'no' --httpserver &>> /opt/sakis3g/error.log) &
```

Delay of 20 seconds after the boot is necessary to avoid the possible "constant reboot" issue. It was thoroughly explained in previous chapters.

4.3. PubNub IoT

Internet of Things or IoT is a rapidly emerging field in the IT world. IoT products are deployed massively in personal and commercial domains. A definition of IoT, simply put, is an array of network connected physical devices embedded with electronic sensors and actuators working on collecting and exchanging data in between. This project can be considered as an IoT project. The Monitoring Tool is equipped with various electronic sensors which collect the data and stream it over Mobile Internet connection to an end user. In this subchapter, data stream network PubNub will be explained.

PubNub set out in 2009 to develop a DSN or Data Stream Network for developers to build real-time apps as easily as building a web page. The PubNub Data Stream Network provides global cloud infrastructure and key building blocks for real-time interactivity.[30] From our point of view, all we have to do to transmit and receive the data is to have the adequate publishing and subscribing applications built using PubNub's SDK and proper Internet connection. PubNub will handle the data format, traffic and other possible issues. On the Monitoring Tool, there are 8 independent python 27 scripts that transmit the data over PubNub. On our PC there is one python 27 script that is subscribed to the PubNub channel and collects the incoming data from the Monitoring Tool.

To start working with the PubNub Iot, first we have to sign up for their service. For the needs of this project, we will use the non-commercial, free user option. It allows us to transmit up to 300 000 messages per month which should be enough. As mentioned before, 8 scripts are publishing the data, out of which 5 publish the message each 120 seconds, one publishes 4 messages each 120 seconds, one publishes message each 90 seconds and the last one publishes the message each hour. On average, that would be 7464 messages daily and that does not exceed the monthly limit. The messages are short, ranging from 20 to 90 characters per message so they do not generate a significant data traffic and the basic Mobile Internet package of 3 Gb from Hofer Telekom is more than sufficient for this task. Once we signed up, we got the unique publish and subscribe keys of our account.

The next step is to install the PubNub on our machines. Python 27 is the programing language used in this project so we chose the PubNub Python SDK to be installed. The version of Python SDK used in this project is V3.x.x. As PubNub releases new SDK versions from time to time, this should be updated in order to function properly. To install the PubNub Python SDK on PC and on Raspberry Pi, Python 27 should be installed first together with `pip` package. And later, we just type the following command and it is all set:

```
pip install pubnub
```

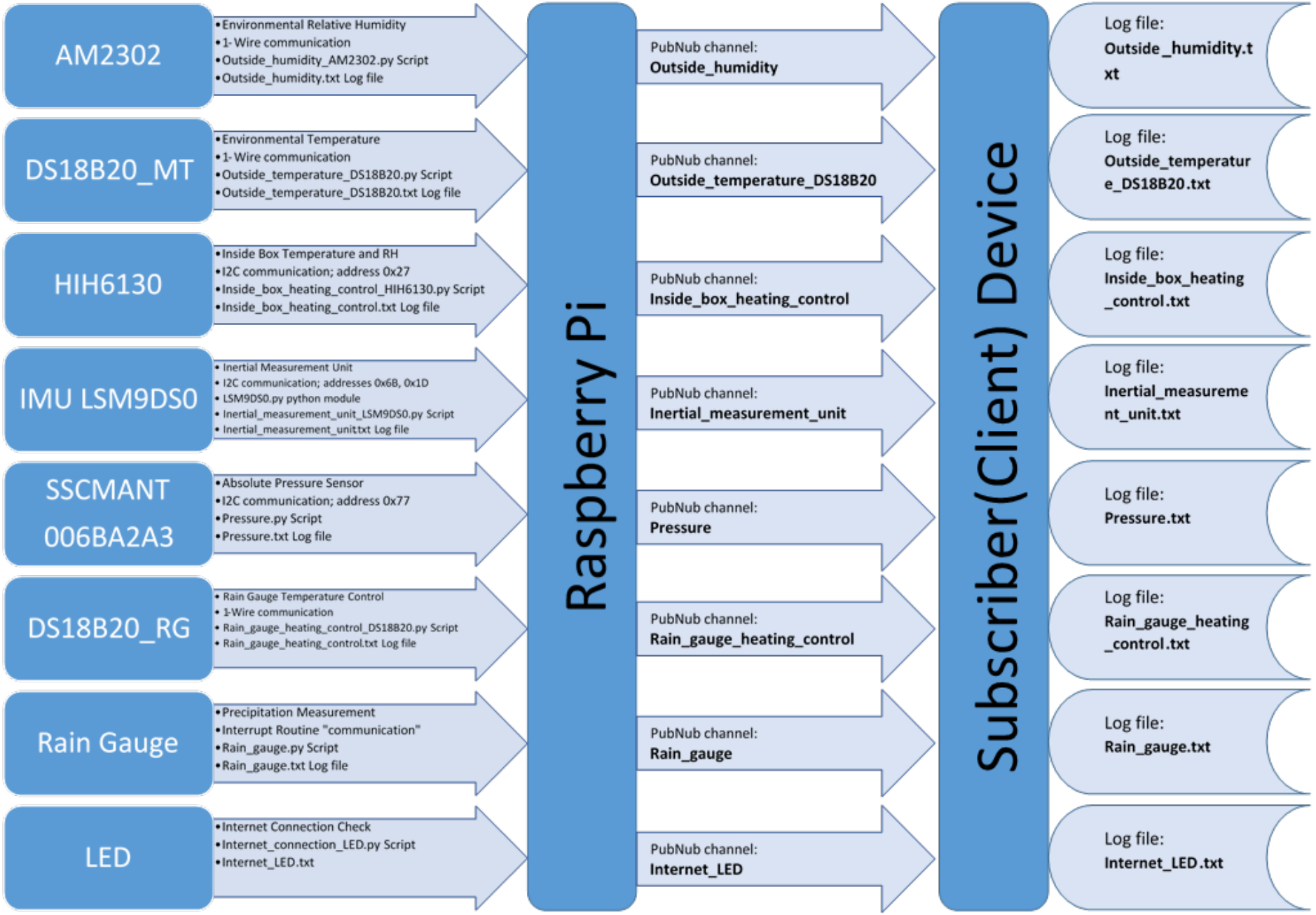
On the PubNub website, there are many tutorials on how to publish and receive the messages. In this project, all the messages will be published by Raspberry Pi and received by an application running on PC. As it can be seen in the codes provided in Appendix, publishing "routine" is always the same. The only different thing is the Channel. PubNub lets us define many channels on the same account, using the same publish and subscribe keys. In this project, 8 channels are used to differ the messages. In the Flowchart Diagram, it can be seen which channel corresponds to which script. Publishing "routine" algorithm goes like this: 1) import PubNub library to the Python script, 2) provide the adequate publish and subscribe keys, 3) select the channel for communication, 4) define the calback function and 5) call the callback function on message you want to publish on the channel. For the example of Python code, let's take the "Internet_connection_LED.py" script to see how it looks like(of course, keys are hidden):

```
1.  from pubnub import Pubnub
2.
3.  pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
4.
5.  channel = 'Internet_LED'
6.
7.  def callback(message, channel):
8.      print(message)
9.
10. pubnub.publish(channel, message)
```

Once the message is published, it is taken over by the PubNub's streaming service and it "floats" in the channel. To read the message, an application using adequate subscribe and publish keys paired with the proper channel should be made.

The message is nothing but a string of characters and it can be received by any other application, not necessarily a Python application. Having said that, in the future, any kind of mobile or any other application can be made to read the streamed messages. In this project, a simple application to retrieve the messages and store them was made in Python 27. Application is subscribed to the PubNub account and to all of the 8 channels. It fetches the messages and stores them in proper log files, based by their content and channel. The full code can be found in the Appendix part, but the subscription "routine" is the same for all. Here is an example on how to subscribe to one of the channels and store the messages in the log file; the subscription "routine" will match the publishing "routine" from the previous paragraph:

```
1.  from pubnub import Pubnub
2.
3.  pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
4.
5.  channel6 = 'Internet_LED'
6.
7.  def callback_channel6(message, channel):
8.      print(message)
9.      with open('Internet_LED.txt', 'a') as logfile:
10.         logfile.write(message)
```


AM2302
•Environmental Relative Humidity
•1-Wire communication
•Outside_humidity_AM2302.py Script
•Outside_humidity.txt Log file
DS18B20_MT
•Environmental Temperature
•1-Wire communication
•Outside_temperature_DS18B20.py Script
•Outside_temperature_DS18B20.txt Log file
HIH6130
•Inside Box Temperature and RH
•I2C communication; address 0x27
•Inside_box_heating_control_HIH6130.py Script
•Inside_box_heating_control.txt Log file
IMU LSM9DS0
• Inertial Measurement Unit
• I2C communication; addresses 0x6B, 0x1D
• LSM9DS0.py python module
• Inertial_measurement_unit_LSM9DS0.py Script
• Inertial_measurement_unit.txt Log file
SSCMANT 006BA2A3
•Absolute Pressure Sensor
•I2C communication; address 0x77
•Pressure.py Script
•Pressure.txt Log file
DS18B20_RG
• Rain Gauge Temperature Control
• 1-Wire communication
• Rain_gauge_heating_control_DS18B20.py Script
• Rain_gauge_heating_control.txt Log file
Rain Gauge
•Precipitation Measurement
•Interrupt Routine "communication"
•Rain_gauge.py Script
•Rain_gauge.txt Log file
LED
•Internet Connection Check
•Internet_connection_LED.py Script
•Internet_LED.txt
Raspberry Pi
PubNub channel:
Outside_humidity
PubNub channel:
Outside_temperature_DS18B20
PubNub channel:
Inside_box_heating_control
PubNub channel:
Inertial_measurement_unit
PubNub channel:
Pressure
PubNub channel:
Rain_gauge_heating_control
PubNub channel:
Rain_gauge
PubNub channel:
Internet_LED
Subscriber(Client) Device
Log file:
Outside_humidity.txt
Log file:
Outside_temperature_DS18B20.txt
Log file:
Inside_box_heating_control.txt
Log file:
Inertial_measurement_unit.txt
Log file:
Pressure.txt
Log file:
Rain_gauge_heating_control.txt
Log file:
Rain_gauge.txt
Log file:
Internet_LED.txt

4.4. Subsystems implementation

The Monitoring System consists of several, so called, subsystems. Subsystem can be defined as a Python script which controls a specific electronic sensor or some other electronic element of the whole system. Apart of some specific function, each subsystem generates a message(or report) of its actions together with timestamp. The message is stored in the appropriate log file on the Raspberry Pi and also published using specific PubNub channel. Every subsystem starts with the boot of Raspberry Pi, delayed by 90 seconds in order to let the operating system stabilize itself and establish the Internet connection. All subsystem's Python scripts and their log files are located in the `/home/pi/Monitor_Tool` directory, together with `Monitor_Tool.sh` bash script which starts them with the boot, delayed by 90 seconds. The purpose of the delay is to let the operating system stabilize itself and establish the Internet connection. Once started, they operate in the background all the time. This was done by appending the following lines to the `/etc/rc.local` file:

```
sleep 90

sudo /home/pi/Monitor_Tool/Monitor_Tool.sh) &
```

There are 8 various subsystems implemented. On the Flowchart Diagram on the previous page, it can be seen what is the purpose of each of them. Here, functionality of each subsystem will be explained and the full Python code is located in the Appendix part.

4.4.1. Inertial Measurement Unit

As the title suggests, this subsystem deals with Inertial Measurement Unit LSM9DS0. It retrieves the data from the IMU's accelerometer and gyroscope, implements the Complementary Filter, publishes the data over PubNub, stores the data to the log file and sleeps for 120 seconds. Before coding anything, first we have to install Python modules `smbus.py` and `LSM9DS0.py`. First one is for I2C communication and the latter one maps the IMU's registers. Having that installed,

we can proceed with coding. PubNub publishing “routines” does not require any additional explanation since they were thoroughly explained in previous subchapter.

Now, we must initialise accelerometer and gyroscope with proper settings. For accelerometer, the following settings are used: 6.25 Hz data rate, X, Y, Z axis enabled, antialiasing filter bandwidth is 773 Hz, +/- 2G full scale. For gyroscope, the following settings are used: 95 Hz output data rate with 12.5 cut-off, X, Y, Z axis enabled, 245 dps full scale. Before entering the loop, all angle values must be set to zero. After we enter the loop, first we read the registers. Then we deal with gyroscope and accelerometer reading before combining them with the Complementary filter. For gyroscope, first we calculate the loop time or how much time passed between two readings, then convert the readings to degrees per second and calculate the angles from the gyroscope. For the accelerometer, first we convert the readings to degrees and scale them in -180 to 180 degrees range. Then we implement the Complementary filter by fusing the final readings of gyroscope and accelerometer in order to filter out the gyroscope’s drift and accelerometer’s noise. For the output, there are four messages generated. The messages express accelerometer’s X angle, accelerometer’s Y angle, Complementary filter’s X angle and Complementary filter’s Y angle, respectively. Each message is published over PubNub and stored in the log file.

Pyhon Script: `Inertial_measurement_unit_LSM9DS0.py`

Log File: `Inertial_measurement_unit.txt`

PubNub Channel: `Inertial_measurement_unit`

Message Example:

```
2017-02-07 00:50:29.828843   -93.1468221879 deg Acc_X angle

2017-02-07 00:50:29.915175   149.264512802 deg Acc_Y angle

2017-02-07 00:50:29.995041   -91.8014598064 deg CF_X angle

2017-02-07 00:50:30.065213   152.519593957 deg CF_Y angle
```

4.4.2. Internal Heat Control

In this subsystem, the title is also quite suggestive. This subsystem controls the temperature and humidity inside the Monitoring Tool steel box and switches the heaters if necessary. Temperature and relative humidity sensor HIH 6130 is used to monitor the conditions inside the box. Once the temperature drops below 25°C, the heating is switched on. Three heat emitting chassis mount resistors are installed on the lower side of the steel box, with 24 Watts of power all together. Python module `smbus.py` is necessary because HIH 6130 uses I2C communication protocol. Additionally, Adafruit's library containing `HIH6130.io` is also needed. It maps the registers of the sensor together with already implemented functions.

Algorithm is quite straight forward – read the temperature, switch the heaters on if temperature is below 25°C, publish the report message, sleep for 120 seconds and repeat. In python script, first we set the GPIO pin 23 to output mode because it controls the low side switch on which heaters are connected. Then we read the temperature and relative humidity and set the GPIO pin 23 high if temperature is below 25°C. Message containing current temperature, relative humidity and heaters state is published over PubNub and stored in the log file.

Pyhon Script: `Inside_box_heating_control_HIH6130.py`

Log File: `Inside_box_heating_control.txt`

PubNub Channel: `Inside_box_heating_control`

Message Example: `2017-02-17 13:11:12.837960  RH: 20.19 %  Temperature: 26.1 degC  Inside box heating OFF`

4.4.3. Rain Gauge Heat Control

On top of the Rain Gauge, there is a steel grid installed, together with heating pad and DS18B20 temperature sensor. Steel grid serves as a protection from dirt and other things that may block the Funnel. Snow and ice might collect on the grid and also block the Funnel and that is why 10 Watts heating pad is installed on the grid. Algorithm is simple, almost the same as in the previous subchapter. Read the

temperature, switch the heater on if temperature is below 2°C, publish the report message, sleep for 120 seconds and repeat. Python script is also quite similar to the one in previous chapter, just the temperature sensor and GPIO pin are different, here is pin 24 used. Temperature sensor is DS18B20. To read its value, first we must enable the 1-Wire communication in Raspbian and read the data from the sensor's location in `/sys/bus/w1/devices`.

In the sensor's datasheet, the format of the reading is explained. It takes couple of operations to convert the reading to decimal temperature in degrees Celsius. Reading is in 16-bit "LSB MSB" format. First we rotate the data and drop the space in between so we get the "MSBLSB" format. The last 4 bits tell us the value of temperature after decimal point or fractions. The least possible fraction is 1/16 or 0.0625°C. First 4 bits of the reading are the sign bit, they should be omitted. Inner 8 bits give the temperature. If the reading in decimal is below 128, it means that temperature is positive or above 0°C and we take it as is, adding the fraction part. If the reading in decimal is above 128, it means that temperature is negative or below 0°C. In this case, we must drop the MSB and convert the other 7 bits to decimal format. To that number, we add the complement of fractions part of the reading. And a "minus" in front because this is the negative or below 0°C temperature. If the temperature drops below 2°C, the heater is switched on.

Pyhon Script: `Rain_gauge_heating_control_DS18B20.py`

Log File: `Rain_gauge_heating_control.txt`

PubNub Channel: `Rain_gauge_heating_control`

Message Example: `2017-02-11 14:40:25.138906 Rain gauge DS18B20 reporting: 2.9375 degC Rain gauge heating is ON.`

4.4.4. Rain Gauge Precipitation Measurement

Precipitation is measured using the Rain Gauge. The principle of operation is explained in detail in "hardware" chapter. Basically, there are two compartments on the Lever. And there are only two states in which it can be, meaning that only one compartment can be on top. Once the compartment on top is full enough with

water to overweight its bottom part, it switches to other side. During that move, a magnet mounted on the Lever closes the reed switch. The reed switch is connected to GPIO pin 22 and controlled by interrupt subroutine. Each closing of the reed switch adds 0.00794 litres of water to the precipitation count. 0.00794 litres is the volume of one compartment. At the start of the full hour, a message containing the volume of precipitation scaled to litres per square meter is published over PubNub and stored in log file.

In python script, first we define the volume of the compartment, area of the Funnel and scale it to square meter. Then we set the GPIO pin 22 to input, activate the internal pull up resistors and turn off the annoying GPIO warnings. Interrupt routine is activated on a falling edge, i.e. the move from +3.3V to GND, and calls the interrupt event function. All the function does is adding one compartment volume to the precipitation counter. Additionally, bounce time of 600 ms is added to avoid the possible switch de-bouncing effect. Every full hour, a message containing timestamp and volume of the precipitation in litres per square meter is published and rain counters are set to zero.

Pyhon Script: `Rain_gauge.py`

Log File: `Rain_gauge.txt`

PubNub Channel: `Rain_gauge`

Message Example: `2017-02-04 22:00:01.216782  5.00630517024 litres of precipitation per square m in last hour`

4.4.5. Environmental Temperature Measurement

Environmental temperature is measured using DS18B20 temperature sensor. Sensor's principle of operation is already explained in paragraph 4.4.3. All this subsystem does is it measures the environmental temperature, publishes the reading on the PubNub channel, stores the message in the log file, sleeps for 120 seconds and repeat.

Pyhon Script: `Outside_temperature_DS18B20.py`

Log File: `Outside_temperature_DS18B20.txt`

PubNub Channel: `Outside_temperature_DS18B20`

Message Example: `2017-01-19 13:23:17.961958    7.4375 degC`

4.4.6. Environmental Relative Humidity Measurement

Relative humidity is measured using AM2302 sensor. It is an encased version of DHT22 sensor so the same python libraries can be used for this sensor. Although this sensor reads temperature as well, this reading is not used because it is corrupted. As it was explained before, the sensor is placed on the steel box which is heated so the temperature reading is always couple of degrees Celsius above the real environmental temperature. The function of this subsystem is as simple as it can be – read the relative humidity, publish the reading on the PubNub channel, store the message in the log file, sleep for 120 seconds and repeat. Python library `Adafruit_DHT` provides the functions for reading the sensor data.

Pyhon Script: `Outside_humidity_AM2302.py`

Log File: `Outside_humidity.txt`

PubNub Channel: `Outside_humidity`

Message Example: `2017-02-11 05:08:20.498793    65.4 % RH`

4.4.7. Pressure Measurement

Absolute pressure is measured using SSCMANT006BA2A3 sensor. Same as for temperature and humidity reporting subsystems, this subsystem also has a simple function – read the absolute pressure, publish the reading over PubNub channel, store the message in the log file, sleep for 120 seconds and repeat. As this sensor uses I2C communication protocol, python library `Adafruit_GPIO.I2C` is used as it has already implemented I2C reading functions. The reading format is explained in detail in sensor's datasheet. Couple of operations have to be done to turn the raw reading into pressure in bars.

In python script, first we define the basic sensor properties. Reading is stored in 14 bit format, which is 16384 decimal counts. 10% of this number is 1638 and 90% of this number is 14746. Raw reading should be in between those boundaries. Sensor's maximum pressure is 6 bars and minimum is 0 bars. Using `Adafruit_GPIO.I2C` function, we get the sensor reading in unsigned 16 bit little endian format. First 2 MSB are neglected to get 14 bit reading and using pressure function from datasheet, we get the absolute pressure in bar.

Pyhon Script: `Pressure.py`

Log File: `Pressure.txt`

PubNub Channel: `Pressure`

Message Example: `2017-01-20 16:26:53.266635  0.985047299359 bar`

4.4.8. Internet Connection Monitor

This subsystem controls if the Internet connection is active. Every 90 seconds Internet connection is checked by pinging `www.google.com`, status of the connection is published on the PubNub channel and message is stored in the log file. Moreover, LED placed on the Monitoring Tool steel box is illuminated green if there is an active Internet connection. LED is connected to BSS138 N-channel transistor which is controlled by GPIO pin 18. During the testing period of four weeks, Internet connection was active for more than 99% of time, with couple of connection drops not longer than couple of minutes each. That can be considered as a very good result, meaning that more than 99% of the messages will be streamed over PubNub. The missing ones will be still saved to log file so they can be extracted from there if necessary.

Pyhon Script: `Internet_connection_LED.py`

Log File: `Internet_LED.txt`

PubNub Channel: `Internet_LED`

Message Example: `2017-01-16 13:58:38.547480  Internet connection is ON. Outside LED is GREEN.`

Chapter 5

Testing and Results

The whole Monitoring System was planned, designed, made and initially tested inside the lab, enjoying comfort conditions. Once it was working properly, it was time to test it in the real outdoor conditions. Inside the lab, it was not possible to simulate the outdoor weather conditions, precipitation, etc. Testing was done during winter time, January-February 2017 in Graz, Austria. In the first week of outdoor testing, all flaws and errors that were concealed in the lab testing conditions, came to light. Once fixed, the device operated properly for the next 4 weeks, earning itself a "green light", meaning that all the problems are solved, device is fully operational and it is ready to be delivered to the final user.

In the following subchapters, testing results will be presented together with previously mentioned issues and their fix. Results will not be commented, they will be given as it is. The idea behind this project was to design a standalone system for collecting and delivering certain sensor data. Post processing of the data and drawing conclusion is a job of a civil engineers.

5.1. Issues During Testing & Solutions

The Rain Gauge is made to measure precipitation. Water is passing through the Funnel and tipping into one of the compartments of the Lever. Once the compartment is full enough to over-weight the other part of the Lever, the Lever pivots around the fulcrum and switches side(state). Meanwhile, magnets on it excite the reed switch and software interrupt routine marks the event. But how much exactly is "full enough"? In the design phase, the Lever's mass was unknown. It depends upon the material, magnets and quality of printing. It was estimated to weigh about 40g, but just as precaution, the compartment volume was made to over-weight the Lever even if it weighs 60g. Therefore, each compartment has a volume of exactly 14.5 cubic cm. Additionally, two calibration screws were added under each side of the Lever to adjust the pivoting angle. Once the Rain Gauge was assembled, calibrating screws were adjusted and volume to over-weight was exactly measured. Volume of the compartment to over-weight the other part of the Lever is exactly 7.94 cubic cm. This was used later in the software implementation, so each interrupt event of the Rain Gauge is expressed as 7.94 cubic cm of precipitation.

In the PCB design phase, heat dissipation rectangles for low side switches were not labelled accordingly and machine did not make them. Therefore, they were made by hand, simply by scratching a rectangle on the top layer around low side switch and disconnecting it from the whole top layer. Low side switches are controlling the inside box heat resistors and the Rain Gauge heating pad. They feature a thermal shutdown option. If their inner temperature goes above 150°C [25], shutdown occurs and there is no other way to switch them on except repowering the whole system, i.e. unplugging from the electrical power grid and plugging back again. During testing, they were ON for longer period of time as it was winter time and that resulted in often shutdowns. Obviously, the heat dissipation rectangles were not big enough. Therefore, an improvised copper-made heat sinks with good ventilating properties were soldered above the heat dissipating rectangles to enhance the heat dissipation. That was the solution for thermal shutdown.

In the planning phase, it was decided that part of the system that will operate outdoors, i.e. the Monitoring Tool, has to be encased in airtight steel box. It will

protect the system against humidity, dirt, corrosion, insects and other unwanted stuff. According to that, an IP66 standard steel box was ordered. System was placed inside and during test phase, it was noticed that considerable amount of water is collected on the bottom of the box. That was totally unwanted state. Why was that so? As mentioned previously, system inside the box incorporates heat emitting resistor in order to maintain the stable temperature of approximately 25°C inside. But as outdoor temperature during testing phase was way below 25°C, there was always a significant difference between inside box temperature and outdoor temperature. And there was no air exchange between. That resulted in higher pressure inside the box which lead to higher relative humidity comparing to the environmental temperature. And the final result was condensed water on the sides of the box. In order to overcome this problem, isobaric heating solution was introduced. On the bottom of the box, two M5 holes are drilled to enable air flow between environment and inside box. Therefore, the inside-box and environmental pressure are the same but as it is warmer inside the box, the relative humidity inside the box is significantly lower. The results from testing also confirm that – comparing the environmental relative humidity measurement with the one inside the box. That means that no water is condensed inside the box any more. This solution is well explained in the Figure 24.

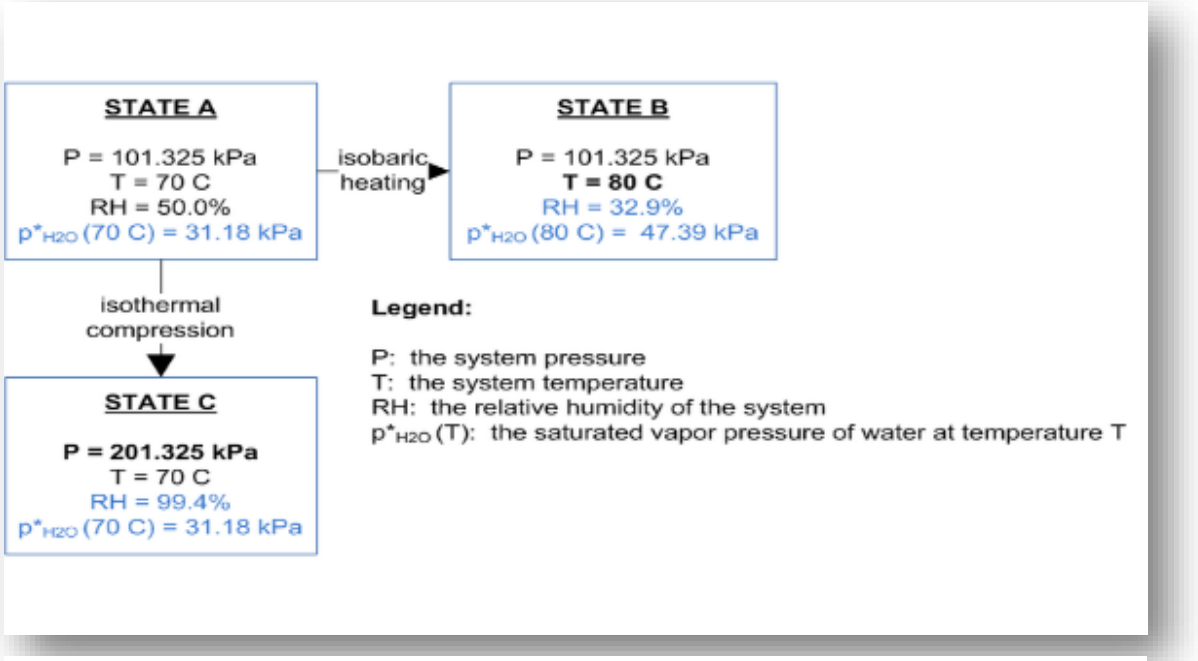


Figure 24: Changes in relative humidity [26]

Optimal heating power for inside box heating was hard to calculate or estimate. As it was mentioned in the previous paragraph, the box is not airtight any more. Now it exchanges air with the environment. That means that additional heating power is needed in order to maintain the stable temperature of approximately 25°C inside the box. The raw calculation of the required heating power considered only air inside the box. Density of air multiplied with box volume gives us air mass in kg. Air mass multiplied with specific heat capacity of air multiplied with temperature differential gives us required energy in kJ. Power in Watts can be considered as Joules per second. Applying the proper values for all variables in the previous equation, result for worst case scenario is following – with 10W of heating power air inside the box should be heated by 25°C in 29 seconds. But that turned out to be totally wrong. First of all, the steel box is more heat conducting than isolating. Therefore, it is hard to keep it on the stable temperature because it exchanges heat with environment thus getting cooler. And the heat dissipating chassis mount resistors were mounted on the steel box so significant heat was also exchanged with environment instead with inside box. Additionally, there are holes in the bottom for air exchange with environment. Trials and errors method during testing phase was the only solution to find optimal heating power. First, two chassis mount resistors of 7.5 Ohm in series were placed on the bottom of the box. Powered by 12 Volts, the dissipated heat was 9.6 Watts or 4.8 Watts per resistor. But as that wasn't enough, another 10 Ohm chassis mount resistor was added in parallel with the previous ones. Powered by 12 Volts, the dissipated heat on the latter resistor is 14.4 Watts. All together that is 24 Watts of heating. And from the testing results, we can conclude that it is enough even in cold winter nights.

During the testing phase, it was noticed that Raspberry Pi is working slower from time to time, becoming more sluggish. A lot of data is processed, the buffers are not properly cleaned and RAM is filling up. SD card maybe keeps corrupting. An easy solution was to give fresh start to the system from time to time. Every day at 00:05, Raspberry Pi is rebooted. In order to do that, `/etc/crontab` file must be edited. The following line is appended to the file:

```
5 0   * * *      root       reboot
```

5.2. Testing Period Results

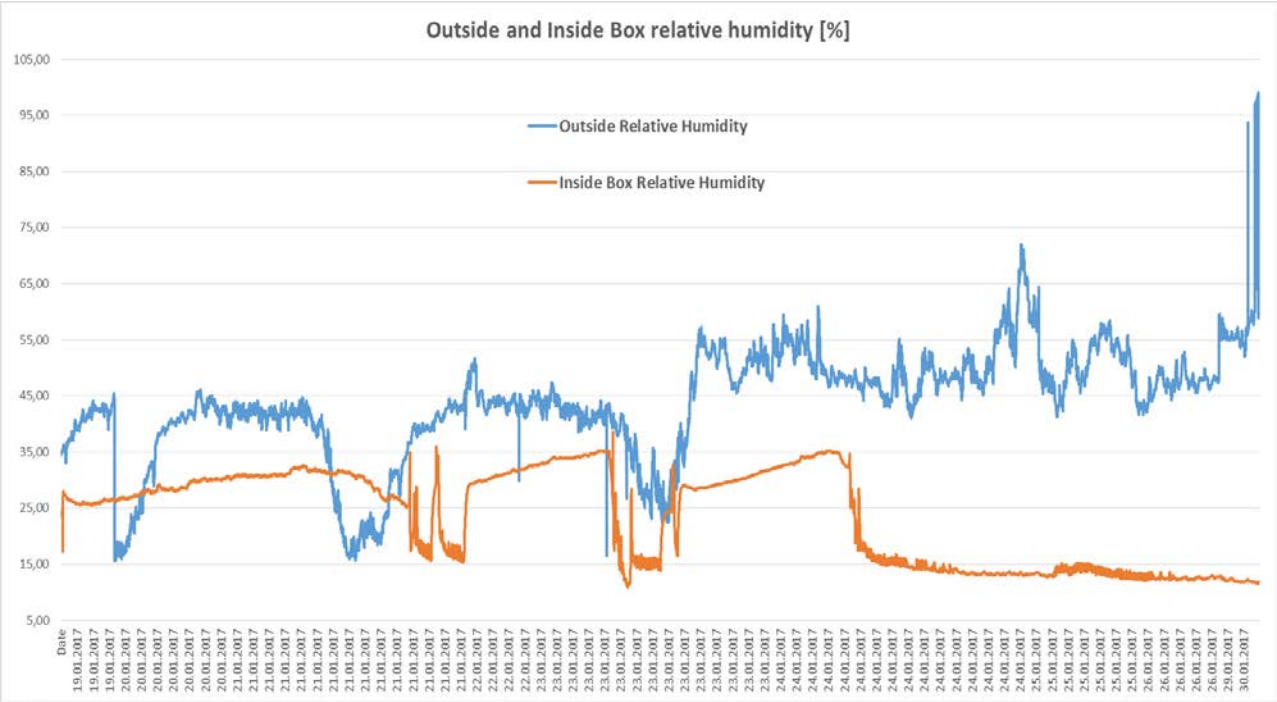


Figure 25: Relative Humidity measurement, in [%]

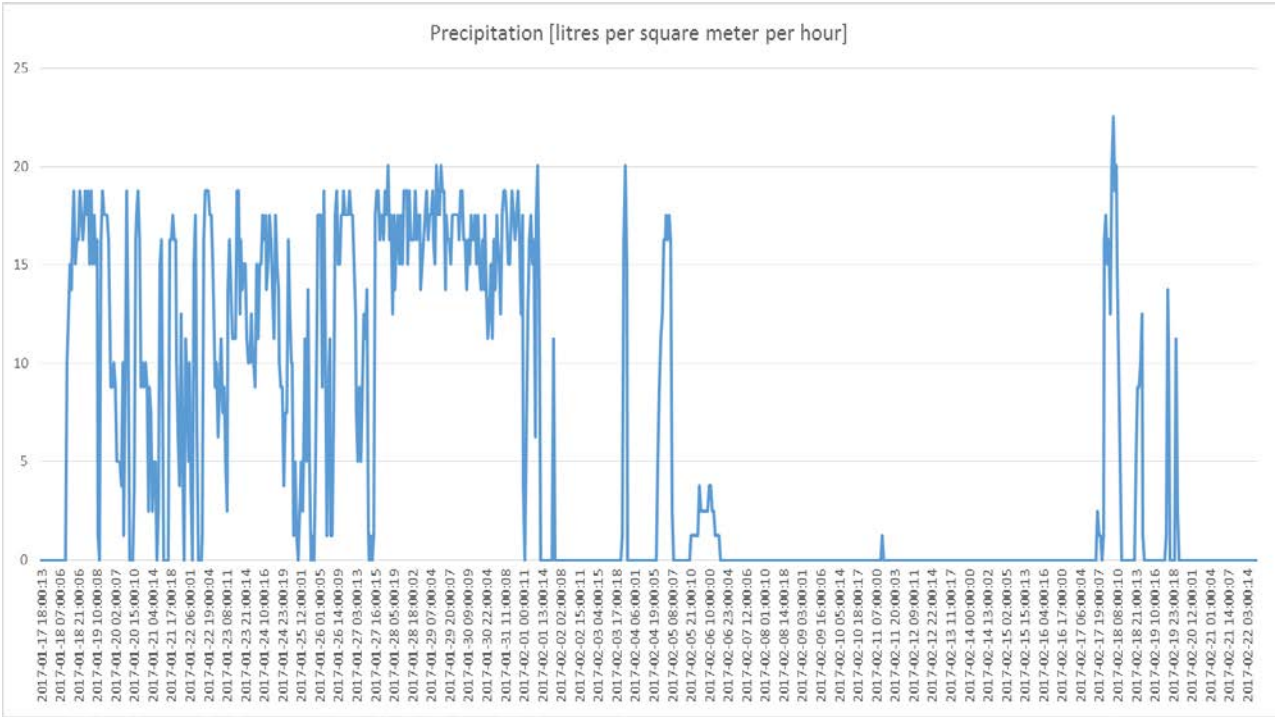


Figure 26: Precipitation measurement, scaled to litres per square meter per hour

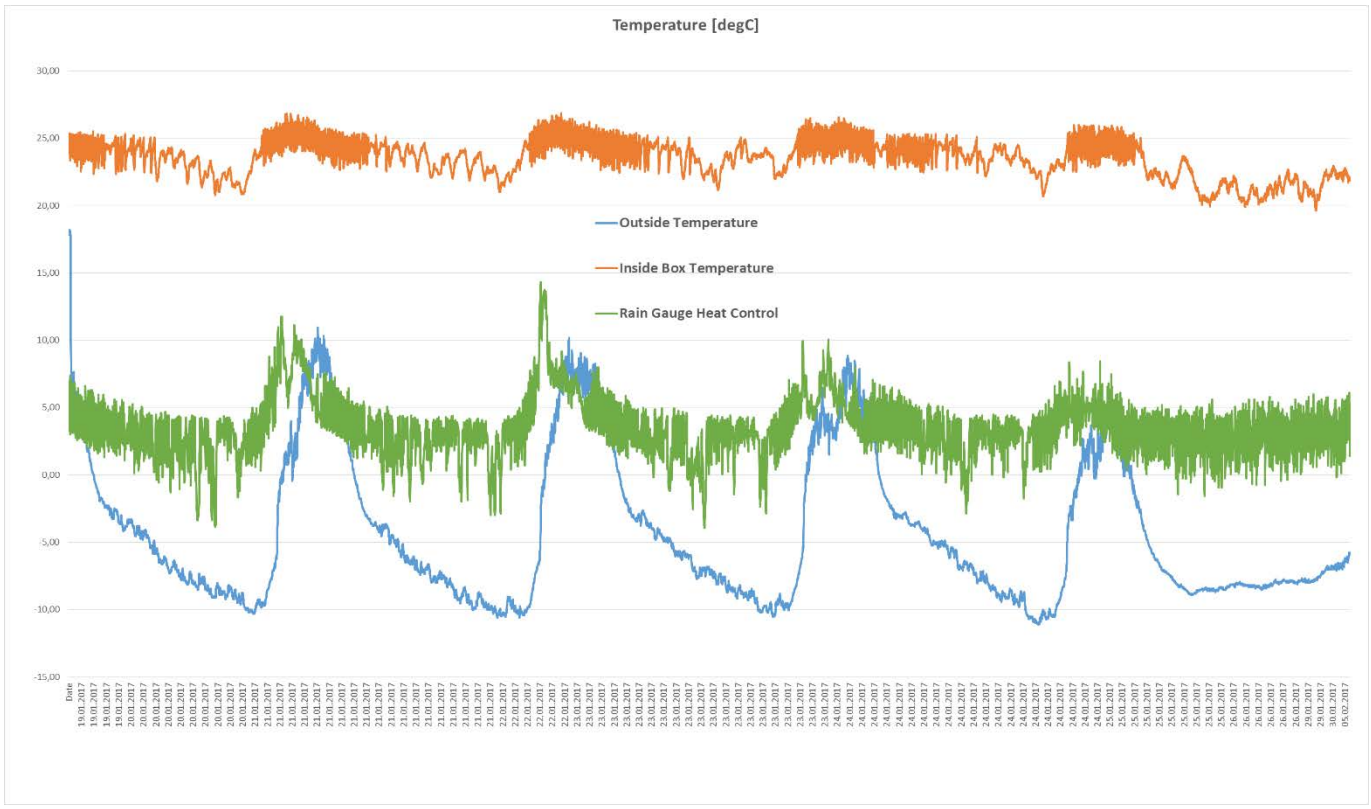


Figure 28: Temperature measurement, in [degC]

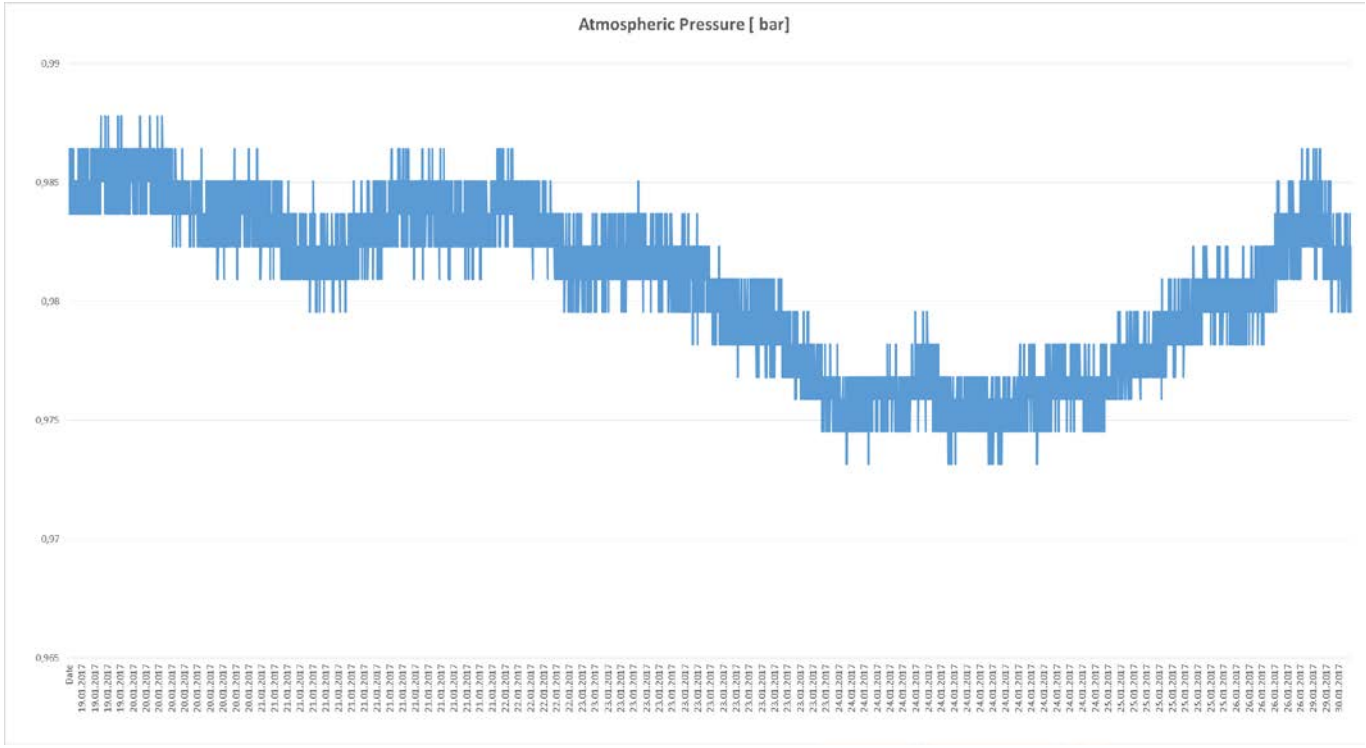


Figure 27: Atmospheric or Barometric Pressure measurement, in [bar]

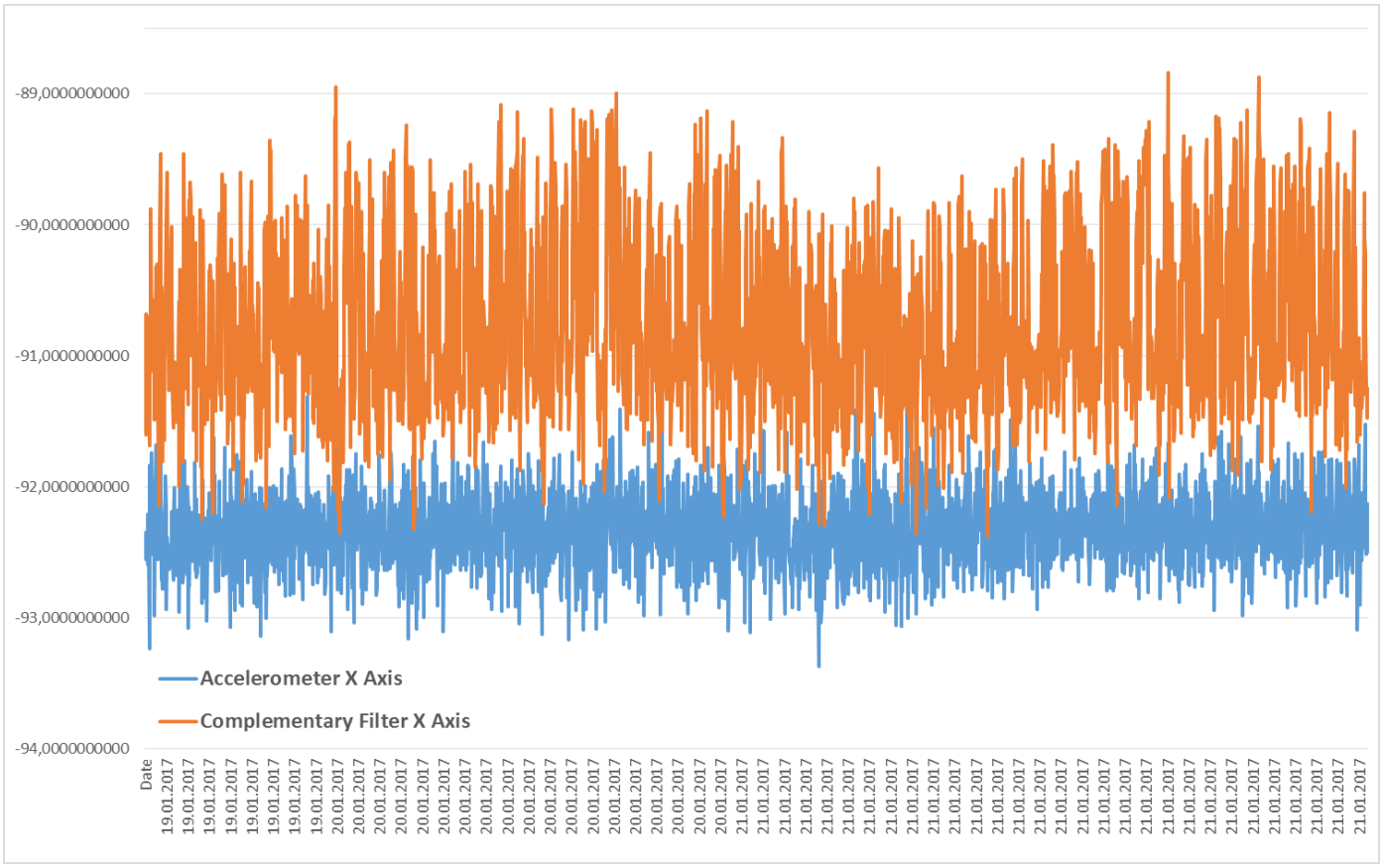


Figure 29: Comparison of IMU's X axis angle by Accelerometer and Complementary filter, in [deg]

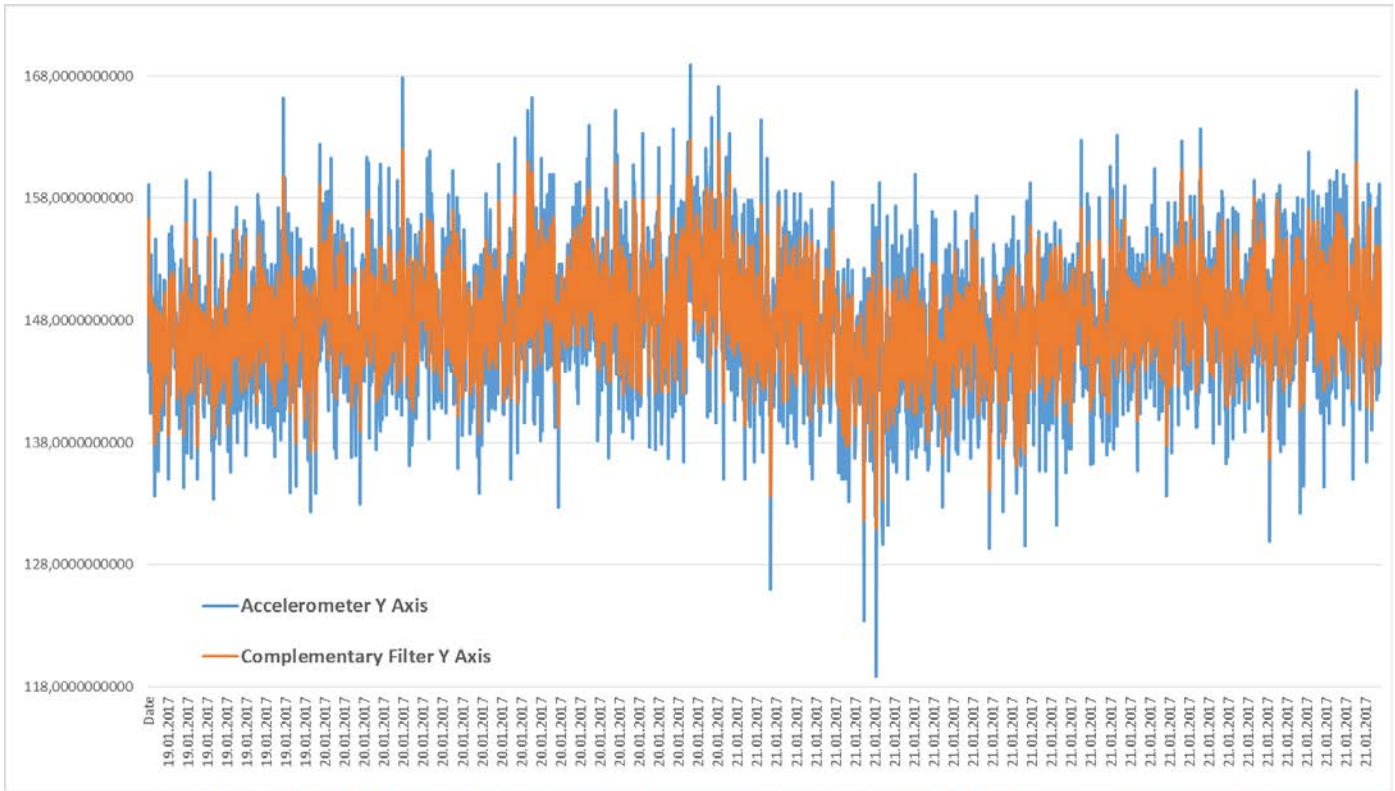


Figure 30: Comparison of IMU's Y axis angle by Accelerometer and Complementary filter, in [deg]

Chapter 6

Conclusion

The Monitoring System is standalone system made for observing the behaviour of retaining structure. Once the Monitoring Tool is mounted on the retaining structure and plugged to the 230 VAC power grid, it immediately starts with collecting the various sensor data and transmitting it to the final user over mobile Internet connection using PubNub stream service. Sensor data can be split into retaining structure's properties, i.e. inclination, and environmental properties. Proper fusion of those data will give us an insight in retaining structure's behaviour.

To the best of my knowledge, no similar work has been done yet and published in some form since the devices of this type are mainly for commercial use. One of the main ideas behind this system is to make it cheaper than its rivals. It is still unknown if the Monitoring System can outperform its commercial rivals in precision and reliability, but it surely outperforms them in price and customization options.

The project was very challenging, yet interesting. Considering the fact that this is my very first time to do a whole project of this scope, the Monitoring System can be considered a job-well-done, in my humble opinion.

In this Thesis all the stages of the Monitoring System development were thoroughly explained, together with all the encountered challenges, issues and their solutions. Just by following the work presented in this thesis, one should be able to understand the idea behind every part of the system and reproduce the fully operational version by himself with credits to the author.

6.1. Future Work

The Monitoring System is far from a finished project. Although it works well and meets all the requirements set prior to the execution, there are many possible improvements in all directions.

From the hardware perspective, the Monitoring Tool behaves well as a standalone device. Possible upgrade should go in direction of mobility, i.e. battery pack. Solar panel system together with rechargeable battery pack would be a huge improvement to the system because it could be installed in the remote places where electrical grid is not available. But that would require significant reduction in power consumption of certain devices inside the Monitoring Tool, especially heaters and USB mobile Internet dongle.

From the software perspective, potential improvements are countless. The ones that ought be done from time to time are software updates. For instance, PubNub issues a new SDK or Software Development Kit every little while. Current version for Python is V3. Software subsystems which use the services from PubNub should be updated to the newest version once it is issued in order to function properly. Automatic update is not a solution since new SDK requires certain adjustments in the programming code.

Subscriber or Client options and platforms are numerous. The one which is made as a part of the Monitoring System is simple Python27 script for Windows. All it takes to read data with some other device, like mobile phone, is a proper mobile application with PubNub publisher and subscriber key from the Monitoring System.

The Inertial Measurement Unit is quite precise in its measurements. Maybe even "too much" precise, e.g. it senses every little vibration even though it is set to the lowest possible sampling rate. What might be done in a future work is some kind of filter for "false" vibrations. Additionally, inertial measurement unit might be more precisely calibrated in certain angle range. For instance, if the retaining structure is leaned by 30°, the inertial measurement unit should be as precise as possible in the angle range of 25° to 35°, it does not have to be precise in the full angle spectrum.

The Monitoring Tool is made to be a standalone, plug-and-play device. It is just enough to mount it and plug it into electric power grid and it will do all the job by itself. From the testing results, it can be seen that Internet connection is ON for more than 98% of the time. As the Monitoring Tool was planned to collect sensor data and transmit it over PubNub and Internet connection is ON for majority of the time, it was concluded that some minor non-transmitted and internally stored data can be neglected. Therefore, no SSH tunnel over mobile Internet network was made to communicate with the Monitoring Tool because there was no need for that. And it is rather complicated to achieve since it mostly depends upon the security permissions from the mobile network provider. That being said, the only way to communicate with the device is to have physical access, but SSH tunnel would really be useful considering the fact that device will be mounted on some (far) remote place.

The Rain Gauge's accuracy can be improved as well. Currently, the Rain Gauge measures litres per hour and then resets the counter every full hour. Therefore, it is unable to represent the small-scale precipitation with temporal variability. Solution to this would be higher sampling rate and dynamic Rain Gauge precision. That problem, together with plausible solution, is addressed in [24].

Another issue should be addressed to the Rain Gauge and that is its size. The Rain Gauge made in this project is rather small with insufficient funnel area to reach proper accuracy. As it is stated in [4], to measure 0.2 mm of rainfall reliably it requires a funnel of at least 500 cm2 with tipping intervals of 0.1, 0.2, 0.25, 0.5 and 1 mm. The area here is way smaller and tipping intervals are unknown because they were not measured. This Rain Gauge was made to match the size of the box so they could be mounted together. But if it is considered not accurate enough, then a new rain gauge should be made following the observations in [4] and error correction in [24].

Appendix

Internal Heat Control

```
1.  #!/usr/bin/python
2.
3.  #Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com
4.
5.  import sys
6.  import RPi.GPIO as GPIO
7.  from HIH6130.io import HIH6130
8.  import time
9.  import datetime
10. from pubnub import Pubnub
11.
12. #GPIO pin which is used to activate the heat emitting resistors over low side switch

13. heater_pin = 23 #Broadcom pin 23 (On board soldered pin 16)
14.
15. #neglect the warning from GPIO
16. GPIO.setwarnings(False)
17.
18. #use the GPIO notaion from RPi's Broadcom processor
19. GPIO.setmode(GPIO.BCM)
20.
21. #set the controlling GPIO pin to output
22. GPIO.setup(heater_pin, GPIO.OUT)
23.
24. #Setting up the initial settings for Pubnub communication
25. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
26. channel = 'Inside_box_heating_control'
27. def callback(message, channel):
28.     print(message)
29.
30. #loop that will perform the heating control
31. while True:
32.     #read the values from the HIH6130 humidity and temperature sensor located inside
     the Monitor Tool box
33.     try:
34.         rht = HIH6130()
35.         rht.read()
36.     except:
37.         pass
38.
39.     #if the device is not connected i.e. there is not such device on the i2c bus.
40.     if rht.fault is 2:
41.         #turn OFF the heating
42.         GPIO.output(heater_pin, GPIO.LOW)
43.
44.         #report output format
45.         measurement = '\n {0:} \t Inside box HIH6130 reporting: Something is wrong..
    . Could not find i2c device. Inside box heating OFF'.format(datetime.datetime.now())

46.         print measurement
47.         message = measurement
```

```
48.
49.         #save output to the log file
50.         with open('/home/pi/Monitor_Tool/Inside_box_heating_control.txt', 'a') as lo
    gfile:
51.             logfile.write(message)
52.
53.         #publish the output on the pubnub channel
54.         pubnub.publish(channel, message)
55.         sys.exit(0)
56.
57.     #if the device is visible on the i2c bus but for some reason, we can't read the
    data from it
58.     if rht.fault is 3:
59.         #turn OFF the heating
60.         GPIO.output(heater_pin, GPIO.LOW)
61.
62.         #report output format
63.         measurement = '\n {0:} \t Inside box HIH6130 reporting: Something is wrong..
    .  Could not read from i2c device located at 0x27. Inside box heating OFF'.format(da
    tetime.datetime.now())
64.         print measurement
65.         message = measurement
66.
67.         #save output to the log file
68.         with open('/home/pi/Monitor_Tool/Inside_box_heating_control.txt', 'a') as lo
    gfile:
69.             logfile.write(message)
70.
71.         #publish the output on the pubnub channel
72.         pubnub.publish(channel, message)
73.         sys.exit(0)
74.
75.     #if inside box temperature is below 25 degC, activate the inside box heating
76.     if rht.t < 25:
77.         #turn ON the heating
78.         GPIO.output(heater_pin, GPIO.HIGH)
79.         #report output format
80.         measurement = '\n {2:} \t RH: {0} %\t Temperature: {1} degC \t Inside box he
    ating ON'.format(rht.rh, rht.t, datetime.datetime.now())
81.     else:
82.         #turn OFF the heating
83.         GPIO.output(heater_pin, GPIO.LOW)
84.         #report output format
85.         measurement = '\n {2:} \t RH: {0} %\t Temperature: {1} degC \t Inside box he
    ating OFF'.format(rht.rh, rht.t, datetime.datetime.now())
86.
87.     print measurement
88.     message = measurement
89.
90.     #save output to the log file
91.     with open('/home/pi/Monitor_Tool/Inside_box_heating_control.txt', 'a') as logfil
    e:
92.             logfile.write(message)
93.
94.     #publish the output on the pubnub channel
95.     pubnub.publish(channel, message)
96.
97.     #get some rest
98.     time.sleep(120)
```

Inertial Measurement Unit

```
1.  #!/usr/bin/python
2.
3.  #   Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com
4.  #   Followed the example of BerryIMU guide of ozzmaker.com
5.  #   Getting usable angles using a Complementary filter.
6.
7.  import smbus
8.  import time
9.  import math
10. from LSM9DS0 import *
11. import datetime
12. from pubnub import Pubnub
13. bus = smbus.SMBus(1)
14.
15. #Setting up the initial settings for Pubnub communication
16. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
17. channel = 'Inertial_measurement_unit'
18. def callback(message, channel):
19.     print(message)
20.
21. #initial values which are subject to change, with respect to our needs
22. RAD_TO_DEG = 57.29578
23. M_PI = 3.14159265358979323846
24. G_GAIN = 0.00875  # [dps/LSB]  +-245 dps
25. CFC =  0.4      # Complementary filter constant
26.
27. #functions to write data to accelerometer, magnetometer and gyroscope
28. def writeACC(register,value):
29.     bus.write_byte_data(ACC_ADDRESS , register, value)
30.     return -1
31.
32. def writeMAG(register,value):
33.     bus.write_byte_data(MAG_ADDRESS, register, value)
34.     return -1
35.
36. def writeGRY(register,value):
37.     bus.write_byte_data(GYR_ADDRESS, register, value)
38.     return -1
39.
40. #functions to read data from XYZ axis of accelerometer
41. def readACCx():
42.     acc_l = bus.read_byte_data(ACC_ADDRESS, OUT_X_L_A)
43.     acc_h = bus.read_byte_data(ACC_ADDRESS, OUT_X_H_A)
44.     acc_combined = (acc_l | acc_h <<8)
45.     return acc_combined  if acc_combined < 32768 else acc_combined - 65536
46.
47. def readACCy():
48.     acc_l = bus.read_byte_data(ACC_ADDRESS, OUT_Y_L_A)
49.     acc_h = bus.read_byte_data(ACC_ADDRESS, OUT_Y_H_A)
50.     acc_combined = (acc_l | acc_h <<8)
51.     return acc_combined  if acc_combined < 32768 else acc_combined - 65536
52.
53. def readACCz():
54.     acc_l = bus.read_byte_data(ACC_ADDRESS, OUT_Z_L_A)
55.     acc_h = bus.read_byte_data(ACC_ADDRESS, OUT_Z_H_A)
56.     acc_combined = (acc_l | acc_h <<8)
57.     return acc_combined  if acc_combined < 32768 else acc_combined - 65536
58.
```

```
#functions to read data from XYZ axis of magnetometer
def readMAGx():
    mag_l = bus.read_byte_data(MAG_ADDRESS, OUT_X_L_M)
    mag_h = bus.read_byte_data(MAG_ADDRESS, OUT_X_H_M)
    mag_combined = (mag_l | mag_h <<8)
    return mag_combined  if mag_combined < 32768 else mag_combined - 65536

def readMAGy():
    mag_l = bus.read_byte_data(MAG_ADDRESS, OUT_Y_L_M)
    mag_h = bus.read_byte_data(MAG_ADDRESS, OUT_Y_H_M)
    mag_combined = (mag_l | mag_h <<8)
    return mag_combined  if mag_combined < 32768 else mag_combined - 65536

def readMAGz():
    mag_l = bus.read_byte_data(MAG_ADDRESS, OUT_Z_L_M)
    mag_h = bus.read_byte_data(MAG_ADDRESS, OUT_Z_H_M)
    mag_combined = (mag_l | mag_h <<8)
    return mag_combined  if mag_combined < 32768 else mag_combined - 65536

#functions to read data from XYZ axis of gyroscope
def readGYRx():
    gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_X_L_G)
    gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_X_H_G)
    gyr_combined = (gyr_l | gyr_h <<8)
    return gyr_combined  if gyr_combined < 32768 else gyr_combined - 65536

def readGYRy():
    gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_Y_L_G)
    gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_Y_H_G)
    gyr_combined = (gyr_l | gyr_h <<8)
    return gyr_combined  if gyr_combined < 32768 else gyr_combined - 65536

def readGYRz():
    gyr_l = bus.read_byte_data(GYR_ADDRESS, OUT_Z_L_G)
    gyr_h = bus.read_byte_data(GYR_ADDRESS, OUT_Z_H_G)
    gyr_combined = (gyr_l | gyr_h <<8)
    return gyr_combined  if gyr_combined < 32768 else gyr_combined - 65536

#initialise the accelerometer
writeACC(CTRL_REG1_XM, 0b00100111) # "0010" 6.25 Hz data rate, "0" Continuous update, "111" z,y,x axis enabled
writeACC(CTRL_REG2_XM, 0b00100000) # "00" Anti aliasing filter default 773 Hz, "100" +/- 2G full scale, "00" Self test disable, "0" SPI not important

      #initialise the magnetometer
      writeMAG(CTRL_REG5_XM, 0b01110000) # "0" Temp disable, "11" high resolution, "100" Data rate 50 Hz, "00" no interrupt
      writeMAG(CTRL_REG6_XM, 0b01100000) # "0" nothing, "11" +/- 12 gauss full scale, "00000" noting
      writeMAG(CTRL_REG7_XM, 0b00000000) # "00" Normal default mode with xyz reset, "0" internal filter bypassed, "00" nothing, "0" Low power off, "00" Continuous-conversion mode

      #initialise the gyroscope
      writeGRY(CTRL_REG1_G, 0b00001111) # "0000" Data rate and Bandwidth 95 Hz 12.5 Cutoff, "1" Normal power mode, "111" all axis enabled
      writeGRY(CTRL_REG4_G, 0b00000000) # "0" Continuos update, "0" LSB lower address, "00" 245 dps, "0" nothing, "00" Self test disable, "0" SPI not important

      gyroXangle = 0.0
      gyroYangle = 0.0
      gyroZangle = 0.0
      CFangleX = 0.0
      CFangleY = 0.0
```

```
115.
116.       a = datetime.datetime.now()
117.
118.       while True:
119.
120.           #Read the accelerometer, gyroscope and magnetometer values
121.           ACCx = readACCx()
122.           ACCy = readACCy()
123.           ACCz = readACCz()
124.           GYRx = readGYRx()
125.           GYRy = readGYRy()
126.           GYRz = readGYRz()
127.           MAGx = readMAGx()
128.           MAGy = readMAGy()
129.           MAGz = readMAGz()
130.
131.           #calculate looptime(LT). How long have passed between gyro reads
132.           b = datetime.datetime.now() - a
133.           LT = b.microseconds/(1000000*1.0)
134.           a = datetime.datetime.now()
135.
136.           #convert gyro raw reading to degrees per second
137.           #raw reading times sensitivity gives us dps value or how fast it is rotat
    ing in degrees per second. sensitivity is in dps
138.           rate_gyr_x =  GYRx * G_GAIN
139.           rate_gyr_y =  GYRy * G_GAIN
140.           rate_gyr_z =  GYRz * G_GAIN
141.
142.           #calculate the angles from the gyro.
143.           gyroXangle+=rate_gyr_x*LT
144.           gyroYangle+=rate_gyr_y*LT
145.           gyroZangle+=rate_gyr_z*LT
146.
147.           #convert accelerometer values to degrees
148.           #with atan2 we get the principal value of the tangent of the other angles
    , expressed in radians
149.           #atan2 is similar to atan, except it returns values in range of (-PI,PI)
    as opposed to (-PI/2,PI/2) that is returned by atan. also, it takes 2 arguments
    instead of one. two values are converted to angles in the full range of 360 degrees.
               #adding PI to the radians gives us result from 0 to 2. And that times (18
    0/Pi) are degrees
150.           AccXangle =  (math.atan2(ACCy,ACCz)+M_PI)*RAD_TO_DEG
151.           AccYangle =  (math.atan2(ACCz,ACCx)+M_PI)*RAD_TO_DEG
152.
153.           #convert the values to -180 and +180
154.           AccXangle -= 180.0
155.           if AccYangle > 90:
156.               AccYangle -= 270.0
157.           else:
158.               AccYangle += 90.0
159.
160.           #complementary filter used to combine the accelerometer and gyro values
161.           #somehow we must get rid of gyro drift and accelrometers noise
162.           CFangleX=CFC*(CFangleX+rate_gyr_x*LT) +(1 - CFC) * AccXangle
163.           CFangleY=CFC*(CFangleY+rate_gyr_y*LT) +(1 - CFC) * AccYangle
164.
165.           #output the X angle from the accelerometer
166.           measurement = '\n {0:} \t {1} deg Acc_X angle'.format(datetime.datetime.n
    ow(), AccXangle)
167.           print measurement
168.           message = measurement
169.           #save the output to the log file
170.           with open('/home/pi/Monitor_Tool/Inertial_measurement_unit.txt', 'a') as
    logfile:
```

```
171.                 logfile.write(message)
172.             #publish the output on the pubnub channel
173.             pubnub.publish(channel, message)
174.
175.             #output the Y angle from the accelerometer
176.             measurement = '\n {0:} \t {1} deg Acc_Y angle'.format(datetime.datetime.n
    ow(), AccYangle)
177.             print measurement
178.             message = measurement
179.             #save the output to the log file
180.             with open('/home/pi/Monitor_Tool/Inertial_measurement_unit.txt', 'a') as
    logfile:
181.                 logfile.write(message)
182.             #publish the output on the pubnub channel
183.             pubnub.publish(channel, message)
184.
185.             #output the X angle from the complementary filter
186.             measurement = '\n {0:} \t {1} deg CF_X angle'.format(datetime.datetime.no
    w(), CFangleX)
187.             print measurement
188.             message = measurement
189.             #save the output to the log file
190.             with open('/home/pi/Monitor_Tool/Inertial_measurement_unit.txt', 'a') as
    logfile:
191.                 logfile.write(message)
192.             #publish the output on the pubnub channel
193.             pubnub.publish(channel, message)
194.
195.             #output the Y angle from the complementary filter
196.             measurement = '\n {0:} \t {1} deg CF_Y angle'.format(datetime.datetime.no
    w(), CFangleY)
197.             print measurement
198.             message = measurement
199.             #save the output to the log file
200.             with open('/home/pi/Monitor_Tool/Inertial_measurement_unit.txt', 'a') as
    logfile:
201.                 logfile.write(message)
202.             #publish the output on the pubnub channel
203.             pubnub.publish(channel, message)
204.
205.             #get some rest. and it also makes the output more readable
206.             time.sleep(120)
```

Internet Connection Monitor

```
11. #!/usr/bin/python
12. 
13. #Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com
14. 
15. import sys
16. import RPi.GPIO as GPIO
17. import time
18. import datetime
19. import socket
20. from pubnub import Pubnub
21. 
22. #remote server which will be our reference point to determine whether internet conne
    ction works
23. remote_server = "www.google.com"
24. 
25. #GPIO pin which controls the LED
26. LED_pin = 18 #Broadcom pin 18 (On board soldered pin 12)
27. 
28. #neglect the warning from GPIO
29. GPIO.setwarnings(False)
30. 
31. #use the GPIO notaion from RPi's Broadcom processor
32. GPIO.setmode(GPIO.BCM)
33. 
34. #set the LED controlling pin to output
35. GPIO.setup(LED_pin, GPIO.OUT)
36. 
37. #Setting up the initial settings for Pubnub communication
38. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
39. channel = 'Internet_LED'
40. def callback(message, channel):
41.     print(message)
42. 
43. #function which tests if there is an active internet connection and returns us True
    or False,
44. def is_connected_to_internet():
45.     try:
46.         host = socket.gethostbyname(remote_server)
47.         s = socket.create_connection((host, 80), 2)
48.         return True
49.     except:
50.         pass
51.     return False
52. 
53. 
54. while True:
55. 
56.     if is_connected_to_internet() is True:
57.         #report the output
58.         message = '\n {0:} \t Internet connection is ON. Outside LED is GREEN.'.form
    at(datetime.datetime.now())
59. 
60.         #switch the LED ON
61.         GPIO.output(LED_pin, GPIO.HIGH)
62. 
63.     else:
64.         #report the output
```

```
65.         message = '\n {0:} \t Internet connection is OFF. Outside LED is OFF.'.forma
    t(datetime.datetime.now())
66.
67.         #switch the LED OFF
68.         GPIO.output(LED_pin, GPIO.LOW)
69.
70.     print message
71.
72.     #save the output to the log file
73.     with open('/home/pi/Monitor_Tool/Internet_LED.txt', 'a') as logfile:
74.             logfile.write(message)
75.
76.     #publish the report on the pubnub channel
77.     pubnub.publish(channel, message)
78.
79.     #get some rest
80.     time.sleep(90)
```

Humidity Measurement

```
1.  #!/usr/bin/python
2.
3.  #Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com
4.
5.  import sys
6.  import io
7.  import Adafruit_DHT as dht #nalazi se u /home/pi/Adafruit_Python_DHT/Adafruit_DHT/co
    mmon.py
8.  import time
9.  import datetime
10. from pubnub import Pubnub
11.
12. #Setting up the initial settings for Pubnub communication
13. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
14. channel = 'Outside_humidity'
15. def callback(message, channel):
16.     print(message)
17.
18. #pin on which data bus is connected to
19. pin = '27'
20.
21. #continuous loop for collecting and publishing data
22. while True:
23.     #read the humidity and temperature data from the AM2302 sensor
24.     #we will consider only the humidity value. the temperature sensor we already hav
    e and the temperature value of this sensor
25.     #might be questionable due to the fact that the sensor is hard mounted on the me
    tal casing of Monitor Tool. and the metal
26.     #casing is heated from the inside so the temperature values might not match the
    real temperature values.
27.     humidity, temperature = dht.read_retry(dht.AM2302, pin)
28.
29.     timestamp = 'Timestamp: ', datetime.datetime.now()
30.
31.     #if the received values are good
32.     if humidity is not None and temperature is not None:
33.         #output the report
34.         measurement = '\n {1:} \t {0:0.1f} % RH'.format(humidity, datetime.datetime.
    now())
35.         print measurement
36.         message = measurement
37.
38.         #save the output to the log file
39.         with open('/home/pi/Monitor_Tool/Outside_humidity.txt', 'a') as logfile:
40.             logfile.write(message)
41.
42.         #publish the output on the pubnub channel
43.         pubnub.publish(channel, message)
44.
45.     #if the received values are corrupted
46.     else:
47.         #output the report
48.         measurement = '\n {0:} \t Something is wrong... '.format(datetime.datetime.n
    ow())
49.         print measurement
50.         message = measurement
51.
52.         #save the output to the log file
```

```
53.         with open('/home/pi/Monitor_Tool/Outside_humidity.txt', 'a') as logfile:
54.             logfile.write(message)
55.
56.         #publish the output on the pubnub channel
57.         pubnub.publish(channel, message)
58.
59.     #get some rest
60.     time.sleep(120)
```

Temperature Measurement

```
1.  #!/usr/bin/python
2.
3.  #Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com
4.
5.  import os
6.  import sys
7.  import time
8.  import datetime
9.  from pubnub import Pubnub
10.
11. #Setting up the initial settings for Pubnub communication
12. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
13. channel = 'Outside_temperature_DS18B20'
14. def callback(message, channel):
15.     print(message)
16.
17. os.system('modprobe w1-gpio')
18. os.system('modprobe w1-therm')
19.
20. temperature_sensor = '/sys/bus/w1/devices/28-000007c6bc2b/w1_slave'
21.
22. while True:
23.
24.     def raw_read():
25.         if os.path.exists(temperature_sensor) == False:
26.             measurement = '\n {0:} \t Something is wrong... '.format(datetime.dateti
    me.now())
27.             print measurement
28.             message = measurement
29.             with open('/home/pi/Monitor_Tool/Outside_temperature_DS18B20.txt', 'a')
    as logfile:
30.                 logfile.write(message)
31.             pubnub.publish(channel, message)
32.             sys.exit(0)
33.
34.         file = open(temperature_sensor, 'r')
35.         lines = file.readlines()
36.         file.close()
37.         return lines
38.
39.     def read_temperature():
40.         lines = raw_read()
41.         if lines[0].strip()[:-4] == '00 00 00 00 00 00 00 00 00 : crc=00':
42.             measurement = '\n {0:} \t Something is wrong... '.format(datetime.dateti
    me.now())
43.             print measurement
44.             message = measurement
45.             with open('/home/pi/Monitor_Tool/Outside_temperature_DS18B20.txt', 'a')
    as logfile:
46.                 logfile.write(message)
47.             pubnub.publish(channel, message)
48.             sys.exit(0)
49.
50.         time.sleep(0.5)
51.         temp_output = lines[1].find('t=')
52.         if temp_output != -1:
53.             #get the LSB and MSB from raw data. count also one sign for the space in
     between
```

```
54.             temperature_hex_inverted = lines[0][:5]
55.
56.             #rotate it properly, so now we have MSBLSB format. and also drop the spa
    ce in between
57.             temperature_hex_proper = temperature_hex_inverted[3:] + temperature_hex_
    inverted[:2]
58.
59.             #get the last hex sign, it tells us the value of temperature after decim
    al point. or simply said fractions.
60.             temperature_dec_fractions = int(temperature_hex_proper[3:].strip(), 16)

61.
62.             #4 bits are decimal fractions, so the least possible value is 0.0625. di
    vide 1/16.
63.             fractions = temperature_dec_fractions * 0.0625
64.
65.             #strip the sign hex bit and fractions hex bit. two inner hex bits gives
    us the temperature. convert them to decimal.
66.             temperature_inner_hex = temperature_hex_proper[:3]
67.             temperature_inner_hex = temperature_inner_hex[1:]
68.             temperature_inner_dec = int(temperature_inner_hex, 16)
69.
70.             if temperature_inner_dec < 128:
71.                 #above zero degC temperature
72.                 #temperature is sum of decimal part and fraction. expressed in degC.

73.                 temperature_final = temperature_inner_dec + fractions
74.
75.             else:
76.                 #below zero degC temperature
77.                 #convert inner decimal number to binary
78.                 temperature_inner_bin = bin(temperature_inner_dec)
79.
80.                 #cut the MSB. we have to cut first three bits to cut MSB because dat
    a format is "0bxxxxx"
81.                 temperature_dec = int(temperature_inner_bin[3:], 2)
82.
83.                 #final temeperature in degC with complement of temperature and fract
    ions.
84.                 #and multiply with (-1) to get negative values
85.                 temperature_final = (float)((127 - temperature_dec) + (1.0 - fractio
    ns)) * (-1)
86.
87.             measurement = '\n {0:} \t {1} degC'.format(datetime.datetime.now(), temp
    erature_final)
88.             print measurement
89.
90.             message = measurement
91.             #save the output to the log file
92.             with open('/home/pi/Monitor_Tool/Outside_temperature_DS18B20.txt', 'a')
    as logfile:
93.                 logfile.write(message)
94.             #publish the output on the pubnub channel
95.             pubnub.publish(channel, message)
96.
97.     read_temperature()
98.     time.sleep(120)
```

Pressure Measurement

```
1.  #!/usr/bin/python
2.
3.  #Made by: Hrvoje Cvitkusic. December, 2016. hrvoje.cvitkusic@gmail.com
4.
5.  import time
6.  import Adafruit_GPIO.I2C as I2C
7.  import datetime
8.  from pubnub import Pubnub
9.
10. #Setting up the initial settings for Pubnub communication
11. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
12. channel = 'Pressure'
13. def callback(message, channel):
14.     print(message)
15.
16. #sensor data is stored in 14 bits, for later calculation, 2 to the power of 14 is 16
    384 decimal counts. 2**14 = 16384
17.
18. #Minimum raw data reading of the sensor is 10% of the full raw data reading - 14 bit
     decimal counts - according to the datasheet
19. raw_reading_min = 1638  #it's 10% of 2**14
20.
21. #Maximum raw data reading of the sensor is 90% of the full raw data reading - 14 bit
     decimal counts - according to the datasheet
22. raw_reading_max = 14746 #it's 90% of 2**14
23.
24. #Minimum pressure reading of the sensor expressed in bars - according to the datashe
    et
25. pressure_bar_min = 0.0
26.
27. #Maximum pressure reading of the sensor expressed in bars - according to the datashe
    et
28. pressure_bar_max = 6.0
29.
30. #Pressure sensor is located on the hex address (0x28)
31. sensor = I2C.get_i2c_device(0x28)
32.
33. while True:
34.
35.     #read 16 bit data in decimal counts
36.     reading_16bit_decimal = sensor.readU16BE(0)
37.     time.sleep(1)
38.
39.     #we are reading a 16 bit value. the reading sometimes (in less than 10% cases) r
    eturns wrong values which are always lower
40.     #than 4500 decimal counts so that value we simply neglect and move forward to th
    e next reading.
41.     if reading_16bit_decimal > 4500:
42.
43.         #convert 16 bit value in decimal counts to 16 bit value in binary
44.         reading_16bit_binary = bin(reading_16bit_decimal)
45.
46.         #neglect first 2 MSB and convert it to 14 bit binary value - according to th
    e datasheet. first two MSB are just status
47.         #bits and they are not important to us. the pressure data is stored in other
     14 bits.
48.         #because of python's notaion, we neglect first 4 binary counts. two of them
    are "0b" or "1b" and other two are MSB.
```

```
49.         #for instance, from 16 bit binary value "0b1011011100011010", we neglect fir
    st 4 counts and we get the 14 bit value
50.         #"11011100011010" which we later convert to pressure. that is our pressure d
    ata.
51.         reading_14bit_binary = reading_16bit_binary[4:]
52.
53.         #convert 14 bit binary value to decimal value
54.         reading_14bit_decimal = int(reading_14bit_binary, 2)
55.
56.         #get pressure value in bars using the equation from the sensor's datasheet
57.         pressure = (((reading_14bit_decimal - raw_reading_min) * (pressure_bar_max -
     pressure_bar_min)) / (raw_reading_max - raw_reading_min)) + pressure_bar_min
58.
59.         #this is the final output. pressure is expressed in bar.
60.         measurement = '\n {0:} \t {1:} bar'.format(datetime.datetime.now(), pressure
    )
61.
62.         print measurement
63.
64.         message = measurement
65.
66.         #save reading to a log file
67.         with open('/home/pi/Monitor_Tool/Pressure.txt', 'a') as logfile:
68.                 logfile.write(message)
69.
70.         #publish the reading on the pubnub channel
71.         pubnub.publish(channel, message)
72.
73.         time.sleep(120)
```

Rain Gauge Precipitation Measurement

```
1.  #!/usr/bin/python
2.
3.  #Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com
4.  #Followed the example of Fred Sonnenwald from January, 2014.
5.
6.  import RPi.GPIO as GPIO
7.  import time
8.  import datetime
9.  from pubnub import Pubnub
10.
11. #although I am always mixing the words "rainfall" and "precipitation" in the comment
    s, the true measured object is
12. #PRECIPITATION i.e. any kind of condensed atmospheric water vapor, in forms of drizz
    le, rain, sleet, snow, graupel, hail...
13.
14. #volume of one compartment of the lever in litres
15. compartment_volume = 0.00794
16.
17. #area of the funnel expressed in square meters. area in square mm divided by 10 to t
    he power of 6 to get square m
18. funnel_area = (52.0 * 122.0) / 1000000.0
19.
20. #ratio between funnel area in square m and one whole square m. just to see how many
    "funnel areas" can fit into one square m.
21. #when using the same ratio for rain volume, we get the following - one compartment o
    f the lever equals 1.2516 litres of
22. #precipitation per square m
23. ratio = 1.0 / funnel_area
24.
25. #GPIO pin on which one side of the reed switch is connected to
26. control_pin = 22 #GIPO22
27.
28. #neglect the warning from GPIO
29. GPIO.setwarnings(False)
30.
31. #use the GPIO notaion from RPi's Broadcom processor
32. GPIO.setmode(GPIO.BCM)
33.
34. #set the control pin as input and activate the internal pull up resistors. now, the
    pin is connected to +3.3V over the internal
35. #pull up resistor. the other part of the reed switch is connected to the ground (GND
    ). when the lever goes from one side to the
36. #other, the magnet attached to it passes near the reed switch and activates it. reed
     switch closes the circuit to the GND and
37. #our GPIO pin senses that there has been a change.
38. GPIO.setup(control_pin, GPIO.IN, pull_up_down = GPIO.PUD_UP)
39.
40. #Setting up the initial settings for Pubnub communication
41. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
42. channel_used = 'Rain_gauge'
43. def callback(message, channel):
44.     print(message)
45.
46. #set the initial values
47. rain = 0
48. first_enter = 0
49.
```

```
50. #function to which we will refer after the "rain event" is detected on the GPIO pin
51. def rain_event(channel):
52.     #global variable rain which measures the amount of percipitation
53.     global rain
54.
55.     #volume of the rain is increased by the volume of the compartment
56.     rain = rain + compartment_volume
57.
58.     #output of the function that indicates us that there is some precipitation
59.     measurement = '\n Rain gauge reporting: Timestamp: {0:} \t  It is raining! ' .fo
    rmat(datetime.datetime.now())
60.     print measurement
61.
62. #this is a GPIO interrupt routine, taken from the Adafruit GPIO library. other possi
    ble solution is to use continuous loop but
63. #that's too demanding for the whole system and unnecessary for something that doesn'
    t occur for a longer period of time.
64. #as our pin is in pull up mode, we are waiting for a falling edge i.e. the move from
     +3.3V to 0V. after that happend, we call the
65. #function refered as callback function. the bouncetime prevents switch debounce effe
    ct - a fast shift between two states in the
66. #event of closing/opening the switch. so, once the switch is detected, it won't rais
    e any event for the next 600 ms.
67. GPIO.add_event_detect(control_pin, GPIO.FALLING, callback = rain_event, bouncetime =
     600)
68.
69. #loop that will perform the rain reporting
70. while True:
71.
72.     #get the current time
73.     timestamp_current = datetime.datetime.now()
74.
75.     #first enter to the loop after powering on the whole system
76.     if first_enter is 0:
77.
78.         #the hour which is ending and in which we are monitoring the rainfall
79.         time_monitored = timestamp_current
80.
81.         #increased hour so that we now when the monitored hour has expired
82.         #loop is here to protect us from switching to an hour chart of the new day i
    .e. transition from 23.59 to 00.00
83.         if time_monitored.hour is 23:
84.             time_increased = 0
85.         else:
86.             time_increased = time_monitored.hour + 1
87.
88.         #after this, we will never enter into this first "if". it's here just for th
    e initial powering on of the whole system.
89.         first_enter = 1
90.
91.     #the main if of the whole loop. it compares if we entered in the next hour. once
     we entered i.e. when the current timestamp
92.     #matches with the last hour increased by one, then we report the rainfall for th
    e previous hour, reset the rain meter to 0,
93.     #publish the message over pubnub and increase the time for comparison by one hou
    r.
94.     if timestamp_current.hour is time_increased:
95.
96.         #rain ratio
97.         rain = rain * ratio
98.
99.         #output the amount of rain for the previous hour. liters per square m2 in la
    st hour.
```

```
100.                measurement = '\n {0:} \t {1: } litres of precipitation per square m
    in last hour' .format(datetime.datetime.now(), rain)
101.
102.                #reset the rain counter to zero
103.                rain = 0
104.
105.                print measurement
106.
107.                message = measurement
108.
109.                #publish the reading on the pubnub channel
110.                pubnub.publish(channel_used, message)
111.
112.                #save reading to a log file
113.                with open('/home/pi/Monitor_Tool/Rain_gauge.txt', 'a') as logfile:
114.                        logfile.write(message)
115.
116.                #publish the reading on the pubnub channel
117.                pubnub.publish(channel_used, message)
118.
119.                #increased hour so that we now when the monitored hour has expired
120.                #loop is here to protect us from switching to an hour chart of the ne
    w day i.e. transition from 23.59 to 00.00
121.                if time_increased is 23:
122.                    time_increased = 0
123.                else:
124.                    time_increased = time_increased + 1
125.
126.            #this loop should return a value each full hour so a bit of sleep won't d
    o any harm
127.            time.sleep(20)
```

Rain Gauge Heat Control

```
1.  #!/usr/bin/python
2.
3.  #Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com
4.
5.  import os
6.  import sys
7.  import RPi.GPIO as GPIO
8.  import time
9.  import datetime
10. from pubnub import Pubnub
11.
12. heater_pin = 24 #Broadcom pin 24 (On board soldered pin 18)
13. GPIO.setwarnings(False)
14. GPIO.setmode(GPIO.BCM)
15. GPIO.setup(heater_pin, GPIO.OUT)
16.
17. pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
    xxxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx")
18. channel = 'Rain_gauge_heating_control'
19. def callback(message, channel):
20.     print(message)
21.
22. os.system('modprobe w1-gpio')
23. os.system('modprobe w1-therm')
24.
25. temperature_sensor = '/sys/bus/w1/devices/28-000007c69025/w1_slave'
26.
27. while True:
28.
29.     def raw_read():
30.         if os.path.exists(temperature_sensor) == False:
31.             GPIO.output(heater_pin, GPIO.LOW)
32.             measurement = '\n {0:} \t Rain gauge DS18B20 reporting: Something is wro
    ng... Rain gauge heating is OFF'.format(datetime.datetime.now())
33.             print measurement
34.             message = measurement
35.             with open('/home/pi/Monitor_Tool/Rain_gauge_heating_control.txt', 'a') a
    s logfile:
36.                 logfile.write(message)
37.             pubnub.publish(channel, message)
38.             sys.exit(0)
39.
40.         file = open(temperature_sensor, 'r')
41.         lines = file.readlines()
42.         file.close()
43.         return lines
44.
45.     def read_temperature():
46.         lines = raw_read()
47.         if lines[0].strip()[:-4] == '00 00 00 00 00 00 00 00 00 : crc=00':
48.             GPIO.output(heater_pin, GPIO.LOW)
49.             measurement = '\n {0:} \t Rain gauge DS18B20 reporting: Something is wro
    ng... Rain gaguge heating is OFF'.format(datetime.datetime.now())
50.             print measurement
51.             message = measurement
52.             with open('/home/pi/Monitor_Tool/Rain_gauge_heating_control.txt', 'a') a
    s logfile:
53.                 logfile.write(message)
54.             pubnub.publish(channel, message)
```

```
55.             sys.exit(0)
56.
57.         time.sleep(0.5)
58.         temp_output = lines[1].find('t=')
59.         if temp_output != -1:
60.             #get the LSB and MSB from raw data. count also one sign for the space in
     between
61.             temperature_hex_inverted = lines[0][:5]
62.
63.             #rotate it properly, so now we have MSBLSB format. and also drop the spa
    ce in between
64.             temperature_hex_proper = temperature_hex_inverted[3:] + temperature_hex_
    inverted[:2]
65.
66.             #get the last hex sign, it tells us the value of temperature after decim
    al point. or simply said fractions.
67.             temperature_dec_fractions = int(temperature_hex_proper[3:].strip(), 16)

68.
69.             #4 bits are decimal fractions, so the least possible value is 0.0625. di
    vide 1/16.
70.             fractions = temperature_dec_fractions * 0.0625
71.
72.             #strip the sign hex bit and fractions hex bit. two inner hex bits gives
    us the temperature. convert them to decimal.
73.             temperature_inner_hex = temperature_hex_proper[:3]
74.             temperature_inner_hex = temperature_inner_hex[1:]
75.             temperature_inner_dec = int(temperature_inner_hex, 16)
76.
77.             if temperature_inner_dec < 128:
78.                 #above zero degC temperature
79.                 #temperature is sum of decimal part and fraction. expressed in degC.

80.                 temperature_final = temperature_inner_dec + fractions
81.
82.             else:
83.                 #below zero degC temperature
84.                 #convert inner decimal number to binary
85.                 temperature_inner_bin = bin(temperature_inner_dec)
86.
87.                 #cut the MSB. we have to cut first three bits to cut MSB because dat
    a format is "0bxxxxx"
88.                 temperature_dec = int(temperature_inner_bin[3:], 2)
89.
90.                 #final temeperature in degC with complement of fractions and tempera
    ture
91.                 temperature_final = (float)((127 - temperature_dec) + (1.0 - fractio
    ns)) * (-1)
92.
93.             if temperature_final < 2:
94.                 GPIO.output(heater_pin, GPIO.HIGH)
95.                 measurement = '\n {0:} \t Rain gauge DS18B20 reporting: {1} degC \t
    Rain gauge heating is ON.'.format(datetime.datetime.now(), temperature_final)
96.             else:
97.                 GPIO.output(heater_pin, GPIO.LOW)
98.                 measurement = '\n {0:} \t Rain gauge DS18B20 reporting: {1} degC \t
    Rain gauge heating is OFF.'.format(datetime.datetime.now(), temperature_final)
99.
100.                print measurement
101.
102.                message = measurement
103.                #save the output to the log file
104.                with open('/home/pi/Monitor_Tool/Rain_gauge_heating_control.txt',
     'a') as logfile:
```

```
105.                         logfile.write(message)
106.                     #publish the output on the pubnub channel
107.                     pubnub.publish(channel, message)
108.
109.             read_temperature()
110.             time.sleep(120)
```

Subscriber or Client Side

```
#!/usr/bin/python

# Made by: Hrvoje Cvitkusic. January, 2017. hrvoje.cvitkusic@gmail.com

import sys
import time
from pubnub import Pubnub

pubnub = Pubnub(publish_key = "pub-c-xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx", subscribe_key = "sub-c-xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxx")
channel1 = 'Outside_temperature_DS18B20'
channel2 = 'Outside_humidity'
channel3 = 'Inside_box_heating_control'
channel4 = 'Pressure'
channel5 = 'Inertial_measurement_unit'
channel6 = 'Internet_LED'
channel7 = 'Rain_gauge_heating_control'
channel8 = 'Rain_gauge'

def callback_channel1(message, channel):
    print(message)
    with open('Outside_temperature_DS18B20.txt', 'a') as logfile:
        logfile.write(message)

def callback_channel2(message, channel):
    print(message)
    with open('Outside_humidity.txt', 'a') as logfile:
        logfile.write(message)

def callback_channel3(message, channel):
    print(message)
    with open('Inside_box_heating_control.txt', 'a') as logfile:
        logfile.write(message)

def callback_channel4(message, channel):
    print(message)
    with open('Pressure.txt', 'a') as logfile:
        logfile.write(message)

def callback_channel5(message, channel):
    print(message)
    with open('Inertial_measurement_unit.txt', 'a') as logfile:
        logfile.write(message)

def callback_channel6(message, channel):
    print(message)
    with open('Internet_LED.txt', 'a') as logfile:
        logfile.write(message)

def callback_channel7(message, channel):
    print(message)
    with open('Rain_gauge_heating_control.txt', 'a') as logfile:
        logfile.write(message)

def callback_channel8(message, channel):
    print(message)
    with open('Rain_gauge.txt', 'a') as logfile:
        logfile.write(message)

```

```
69. pubnub.subscribe(channel1, callback_channel1)
70. pubnub.subscribe(channel2, callback_channel2)
71. pubnub.subscribe(channel3, callback_channel3)
72. pubnub.subscribe(channel4, callback_channel4)
73. pubnub.subscribe(channel5, callback_channel5)
74. pubnub.subscribe(channel6, callback_channel6)
75. pubnub.subscribe(channel7, callback_channel7)
76. pubnub.subscribe(channel8, callback_channel8)
77.
78. #dummy loop
79. counter = 0
80.
81. while True:
82.     #do nothing
83.     counter = counter + 1
84.     counter = 0
85.     time.sleep(20)
```

Glossary

Abbreviation	Description
AC/DC Converter	Alternating Current to Direct Current Converter
ADC	Analog to Digital Converter
ASA	Acrylonitirle Styrene Acrylate polymer
BE	Big Endian
BIOS	Basic Input Output System
CAD	Computer Aided design
DC/DC Converter	Direct Current to Direct Current Converter
DoF	Degree of Freedom
dps	Degrees per Second
DSN	Data Stream Network
GB	Gigabyte
GND	Ground
GPIO	General Purpose Input Output
HDMI	High Definition Multimedia Interface
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IMU	Inertial Measurement Unit
IoT	Internet of Things
LE	Little Endian
LED	Light emitting diode
LKM	Loadable Kernel Module
LSB	Least Significant Bit
MEMS	Micro-Electro-Mechanical System
MSB	Most Significant Bit
PC	Personal Computer
PCB	Printed Circuit Board
RAM	Random-access Memory
RH	Relative Humidity
RPS	Revolutions per second

SCL	Serial Clock Line in I2C communication
SCP	Secure Copy Protocol
SD Card	Secure Digital Card
SDA	Serial Data Line in I2C communication
SDK	Software Development Kit
SMD	Surface Mount Device
SMT	Surface Mount Technology
SPI	Serial Peripheral Interface
SSH	Secure Socket Shell protocol
STEM	Science, Technology, Engineering, Mathematics
UMTS	Universal Mobile Telecommunications System
USB	Universal Serial Bus
VAC	Voltage Alternating Current
VCC	Positive Supply Voltage
VDC	Voltage Direct Current

Bibliography

[1] Franck Chollet and Haobing Liu. *A (not so) short Introduction to Micro Electromechanical Systems*. Version 5.3, 2016. http://memscyclopedia.org/introMEMS.html

[2] Starlino article - guide. *A Guide to using IMU(Accelerometer and Gyroscope Devices) in Embedded Applications*. December, 2009. http://www.starlino.com/imu_guide.html

[3] Dieter Uckelmann, Mark Harrison, Florian Michahelles. *Architecting the Internet of Things.* Springer 2011. ISBN 978-3-642-19156-5

[4] Ian Strangeways. *Precipitation: Theory, Measurement and Distribution*. Cambridge University Press, 2007.

[5] Picture taken from: *The quadcopter: control the orientation.* May, 2012. http://theboredengineers.com/2012/05/the-quadcopter-basics/

[7] Mirjana Maksimović, Vladimir Vujović, Nikola Davidović, Vladimir Milošević and Branko Perišić. *Raspberry Pi as Internet of Things hardware: Performances and Constraints.* IcETRAN, June 2014.

[8] *Ultimate guide to Raspberry Pi.* Computer Shopper, Issue 324. Pages 104 – 119, February 2015. http://micklord.com/foru/Raspberry%20Pi%20Pages - %20from%20Computer%20Shopper%202015-02.pdf

[9] Picture taken from: *Introducing the Raspberry Pi 2 - Model B.* Adafruit Industries LLC. https://learn.adafruit.com/introducing-the-raspberry-pi-2-model-b

[10] Picture taken from: https://element14.com/community

[11] *LSM9DS0 iNEMO inertial module*. Datasheet by STMicroelectronics, 2013. http://www.st.com/resource/en/datasheet/lsm9ds0.pdf

[12] Picture taken from: Adafruit 9-DoF LSM9DS0 Breakout Board. Adafruit Industries LLC. https://www.adafruit.com/product/2021

[13] *DS18B20 datasheet*. Maxim Integrated Products, 2008. https://cdn-shop.adafruit.com/datasheets/DS18B20.pdf

[14] Picture taken from: Waterproof DS18B20 Digital temperature sensor. Adafruit Industries LLC. https://www.adafruit.com/product/381

[15] *AM2302 datasheet*. Adafruit Industries LLC. https://cdn-shop.adafruit.com/-datasheets/Digital+humidity+and+temperature+sensor+AM2302.pdf

[16] Picture taken from: AM2302 temperature and humidity sensor. https://www.adafruit.com/product/393

[17] *HIH6130 datasheet*. Honeywell, May 2015. https://sensing.honeywell.com/h -oneywell-sensing-humidicon-hih6100-series-product-sheet-009059-6-en.pdf

[18] *SSC Series Pressure Sensors datasheet*. Honeywell, August 2014. http://www.mouser.com/ds/2/187/honeywell-sensing-trustability-ssc-series-standard-740340.pdf

[19] Josef Gajdusek. *Modem power consumption*. Atx Blog, September 2016. http://blog.atx.name/modem-power-consumption/

[20] Cadsoft Eagle library, Element14 Community. https://www.element14.com/community/

[21] Cadsoft Eagle library, Adafruit Eagle Library. December 2015. https://github.com/adafruit/Adafruit-Eagle-Library

[22] *Tipping Bucket Rain Gauge*. WeatherShack Education Center, 2002-2017. https://www.weathershack.com/static/ed-tipping-bucket-rain-gauge.html

[23] G. Bastin, B. Lorent, C. Duque, M. Gevers. *Optimal Estimation of the Average Areal Rainfall and Optimal Selection of Rain Gauge Locations*. Water Resources Research, Vol. 20, No. 4, Pages 463-470, April 1984.

[24] Emad Habib, Witold F. Krajewski, Anton Kruger. *Sampling Errors of Tipping-bucket Rain Gauge Measurements*. Journal of Hydrologic Engineering. March/April 2001.

[25] *VNN3NV04P-E Power MOSFET datasheet*. STMicroelectronics, 2013. Doc ID 15626 Rev. 5.

[26] Picture taken from Wikipedia. Changes in Relative Humidity.png. Author: Darryl Wolanski, May 2008. https://en.wikipedia.org/wiki/Relative_humid - ity#/media/File:Changes_in_Relative_Humidity.png

[27] Malcolm Maclean. *Raspberry Pi: Measure, Record, Explore*. May, 2016. https://leanpub.com/RPiMRE/read#leanpub-auto-measure

[28] Matthew Lawrence. *Connect Raspberry Pi to a 3g network automatically during its boot*. August, 2013. https://lawrencematthew.wordpress.com/2013/08/07/connect-raspberry-pi-to-a-3g-network-automatically-during-its-boot/

[29] Terence Eden. *3g Internet on Raspberry Pi – Success!*. July, 2012. https://shkspr.mobi/blog/2012/07/3g-internet-on-raspberry-pi-success/

[30] PubNub. January 2017. https://www.pubnub.com/company/

[31] Robottini Article. *Kalman Filter vs Complementary Filter*. September, 2011. http://robottini.altervista.org/kalman-filter-vs-complementary-filter

[32] G. Perez Paina, D. Gaydou, J. Redolfi, C. Paz, and L. Canali. *Experimental comparison of Kalman and Complementary filter for attitude estimation*. August, 2011.

[33] Ulrich Smoltczyk. *Geotechnical Engineering Handbook: Elements and Structures.* Volume 3ed. May, 2003. Wilhelm Ernst & Sohn Verlag fur Architektur und technische Wissenschaften, Berlin, Germany.

[34] Raspberry Pi Foundation. *FAQ and recommendations*. https://www.raspberrypi.org/help/faqs/

[35] Pubnub. *Solutions and Use cases*. https://www.pubnub.com/solutions/