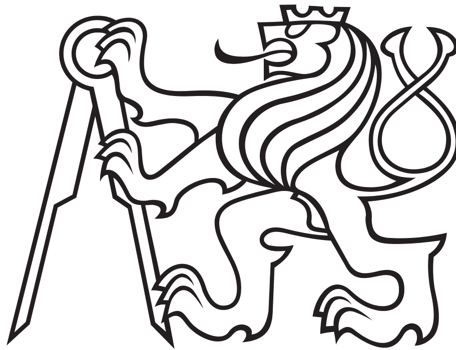
Czech Technical University in Prague

Faculty of Electrical Engineering

BACHELOR THESIS



Albershteyn Andrey

Data recorder for observatory magnetometer

Department of Measurement

Supervisor of the bachelor thesis:	Ing. Michal Janošek, Ph.D.
Study programme:	Cybernetics and Robotics
Specialization:	Sensors and measurement

Prague 2016

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra měření

Akademický rok **2015-16**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student:	**Andrey Albershteyn**
Studijní program:	**Kybernetika a robotika**
Obor:	**Senzory a přístrojová technika**
Název tématu česky:	**Záznamník dat pro observatorní magnetometr**
Název tématu anglicky:	**Data Recorder for Observatory Magnetometer**

Pokyny pro vypracování:

S pomocí modulárního systému Raspberry realizujte záznamník dat pro observatorní magnetometr. Záznamník bude přijímat data z magnetometru po sériové lince, tato data průměrovat dle standardů IAGA, opatřovat časovou značkou a hodnoty přepočítávat přes dodané kalibrační matice do fyzikálních jednotek. Data budou průběžně ukládána na interní SD kartu; implementujte „upload" dat na záložní server. Věnujte se zejména odolnosti přenosu dat proti výpadkům napájení. Za pomoci přídavného displeje realizujte jednoduchou vizualizaci přijímaných dat.

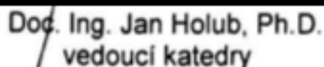
Seznam odborné literatury:

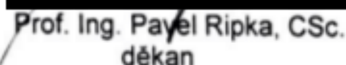
[1] JANKOWSKI, Jerzy; SUCKSDORFF, Christian: Guide for magnetic measurements and observatory practice. 1996.

[2] OLSEN, Nils, et al.: In-flight calibration methods used for the Ørsted mission. 2001.

[3] MONK, Simon: Raspberry Pi Cookbook. O'Reilly Media, Inc., 2013.

Vedoucí bakalářské práce:	Ing. Michal Janošek, Ph.D.
Datum zadání bakalářské práce:	9. prosince 2015
Platnost zadání do[1]:	30. září 2017





Doc. Ing. Jan Holub, Ph.D.
vedoucí katedry

Prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 9. 12. 2015

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

Acknowledgements

I would like to express my gratitude to my supervisor, Ing. Michal Janošek, Ph.D., for his supporting and given advice.

I hereby declare that I have completed this thesis with the topic "Data recorder for obserbatory magnetometer" independently and that I have included a full list of used references. I have no objection to the usage of this work in compliance with the act §60 Zákon è.121/2000 Sb. (copyright law).

In .... date .... signature of the author

Abstract:

This thesis aims to develop data logger based on a single-board computer Raspberry Pi for an observatory magnetometer. The device receives data from a magnetometer by the serial bus, and then data are processed in agreement with IAGA standard and transformed to physical quantity with a usage of already known calibration matrices. Obtained data are stored on an internal SD-card and uploaded to remote FTP-server. The device contains 13.8 V battery and dpi 800x480 display which is used for data visualization.

Abstrakt: Cílem této práce je návrh záznamníku dat ze sériové linky, pro použití s observatoryním magnetometrem. S pomocí single-board počítače Raspberry Pi a vícevláknového programu v prostředí Python je vytvořen přístroj pro záznam a přepočet dat přijímaných z observatorního magnetometeru včetně vizualizace s displejem 800x480 dpi. Orthogonalizace a filtrace se provádí v souladu s IAGA standardem a získaná data se ukládají na interní SD kartu, zároveň pravidelně zasílají pomocí protokolu FTP na vzdálený server. Záznamník je napájen z 13.8 V stejnosměrného zdroje a je zálohován interním akumulátorem.

Keywords: Magnetometer, Sensor, Raspberry Pi, Model 2 B, Observatory Magnetometer, Python, Data Recorder

Contents

1 Introduction **3**

1.1 Outline . . . 3

1.2 Objectives . . . 5

2 Used devices **7**

2.1 Raspberry Pi 2 Model B . . . 7

2.2 Raspberry Pi Touchscreen Display . . . 8

2.3 Magnetometer . . . 8

2.4 DC/DC converter . . . 10

3 Comparison to the state-of-the-art **11**

4 Theory **15**

4.0.1 Fluxgate types . . . 15

4.0.2 Orthogonalization . . . 16

4.0.3 Used orthogonalization . . . 17

5 Hardware part **19**

6 Software part **23**

6.1 Program for receiving information from sensor . . . 23

6.1.1 Description . . . 23

6.1.2 Program structure . . . 24

6.1.3 Configuration settings . . . 28

6.2 Program for plotting data to the display . . . 29

6.2.1 Description . . . 29

6.2.2 GUI program structure . . . 30

6.2.3 GUI program configuration settings . . . 32

6.3 GUI program for presentation purpose . . . 33

7 Tests and results **35**

8 Conclusion **37**

9 CD contents **43**

10 Appendix **45**

10.1 processor.py 45

10.2 main.py 46

10.3 saver.py 49

1. Introduction

Magnetometers are devices used for measuring the intensity and direction of magnetic fields. They are widely used for measuring Earth's magnetic field, metal detection, aerospace navigation and for orientation in space by using it like a compass, for example, when GPS [1] isn't avaiilable. Magnetometers are also used for reading magnetic tags and labels [2] and for detection of ships and submarines [3].

Data acquisition from a magnetometer provides data, which can be used in many domains in geomagnetic studies [4] and scientific research. Large data sets show trends in the magnetic field changes and can provide us information that isn't visible in a short time measurement. For example year's season fluctuations and other long term changes in magnetic field.

The data can be recorded by a specific device called data logger or data recorder. These devices should have some basic characteristics, such as being able to function for a long time period, assuring data consistency, even in case of power failure or internet disconnection, and easy access to stored data.

1.1 Outline

This thesis aims to develop a device which will be used to record data from an observatory magnetometer and transform it to easily accessible format. The project is divided into two parts. In the first part I will develop a device with hardware ports to communicate with sensors and an external computer. In the second part I will describe three programs which should provide correct data transformation and representation.

The main piece of the project is software which consist of two separate programs responsible for communication with the sensor and plotting a data graph on the display, respectively.

Now let's look more precisely at what every section stands for:

- **Used devices**

  In this section I provide some information about hardware devices which I used in this work. This section is divided into a four subsections with a description of every part:

  1. Raspberry Pi 2 Model B - description of single-board computer Raspberry Pi 2 Model B and why it was choosen.
  2. Raspberry Pi touchscreen display - display used in the project.
  3. Magnetometer - description of used fluxgate.
  4. DC/DC converter - voltage converter used to convert battery voltage level to voltage required by Raspberry Pi.

- **Comparison to the state-of-the-art**

  Comparison to other devices used to receive and store data from sensors. I will provide some examples of historically used devices and also devices used nowadays.

- **Theory**

  In this section I give brief overview of fluxgate magnetometers, their types and working principle.

  Then I explain what orthogonalization is and why we need it in this data logger.

  At the end I will explain why we need to perform data processing and how orthogonalization is implemented in the program written for this project.

- **Hardware part**

  In this part I will describe hardware part of this project, such as inner connection of the device and why each parts is used. Then I overview some achived characteristics of the device.

- **Software part**

  In this section I will describe programs which I wrote for this project.

  Firstly, I describe the main program of this project, which receives data from the sensor and performs calibration. I will provide the full description of program's cycle including communication with the sensor, communication between processes, processing, saving data and sending it to network socket etc.

  The next significant piece of software is the GUI (Graphical User Interface) program for displaying all calculated data on the screen. I will talk about the QT framework, why it is used and why it is good. Also I describe configuration settings for this program.

  At the end I will describe program for presentation, which consists of the GUI program mentioned before and part of the main program used for calibration.

- **Test runs and results**

  In this section I will show some results of this work such as format of obtained data, resulting graph and appearence of the device.

- **Conclusion**

  In the last section I will talk about the results of the project.

1.2 Objectives

The main objective of this thesis is to develop a device for reading data from the observatory magnetometer, perform orhogonalization and data processing according to IAGA standard [5] and then save it to an internal SD-card and upload to remote FTP server. The device should also contain a display which shows graphs with actual data recorded in last few hours.

The device should be able to work reliably even with unstable power supply. For example, in case of power failure device should be able to continue collecting data. Also, the internet connection safety should be assured. Thus, while the connection is unavailable, the device should continue to store data on the disk and be able to upload it to the server upon re-establishing a connection.

2. Used devices

In this section devices used in this project are shown. The main piece is microcomputer which performs all calculations and executes programs for communicating with sensor and displaying charts.

2.1 Raspberry Pi 2 Model B

Raspberry Pi is single-board micro computer of a credit-card size. It has basic attributes of desktop PC (USB ports, HDMI port, operating system), but also contains GPIO interface to communicate with peripherals. On the board there is SD card with preinstalled operating system (in this case it is Raspbian Wheezy [6], linux based system).

Even with a such small size it has enough computation power to perform online computation of received data, run linux system, work with display, communicate with remote FTP-server etc.

Table 2.1: Raspberry Pi characteristics

Chatacteristic	Value
Name	Raspberry Pi 2 Model B
Chip	Broadcom BCM2836 SoC
Core architecture	Quad-core ARM Cortex-A7
CPU	900MHz
GPU	Dual Core VideoCore IV Multimedia Co-Processor
Memory	1GB LPDDR2
Operating system	Raspbian Wheezy
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V, 2A

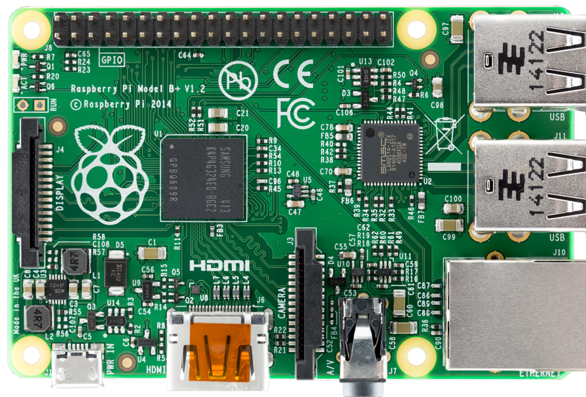


Figure 2.1: Photo of Raspberry Pi 2 Model B from the top [7]

2.2 Raspberry Pi Touchscreen Display

The main purpose of the developed program is to get raw data, perform data processing and save it to SD card. The device is also equipped with a display which allows visualization of obtained data.

On top of the Raspberry PI touchscreen display is connected, which is used to visualize processed data for a predefined time period. There are also control elements implemented on the screen allowing a user to move graph, change it time range and turning on/off auto-range mode.

Display characteristics:

Table 2.2: Display characteristics

Characteristic	Value
Name	7" touchscreen display
Screen Dimensions	194mm x 110mm x 20mm
Viewable screen size	155mm x 86mm
Screen Resolution	800 x 480 pixels
Touchscreen	10 finger capacitive touch

Photo of the display with driver and connectors:

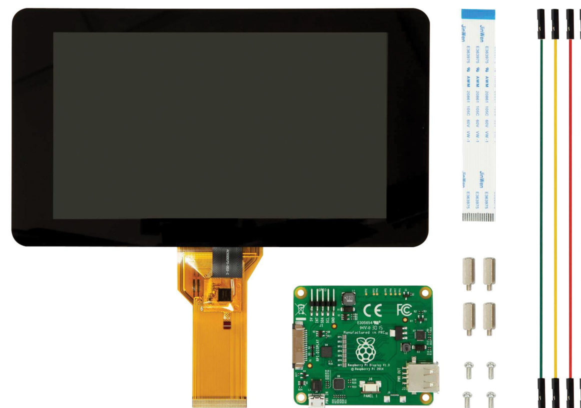


Figure 2.2: Photo of Raspberry Pi touchscreen Dispay and some other components for dispay's connection [8]

2.3 Magnetometer

Magnetometer used in this project is fluxgate parallel magnetometer with low-noise performance and high stability. The sensor consists of two parts: sensor head containing three fluxgates orthogonal to each other, and electronics which reads data from the sensor and generates data stream sent out by the serial bus.

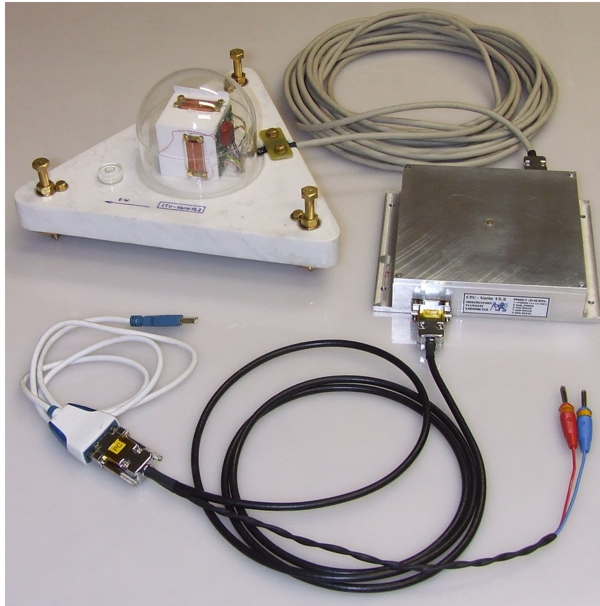


Figure 2.3: Photo of the observatory magnetometer with periphery [9]

This sensor has the serial interface which is set up to 115 200 baudrate and sends 207 samples per second. Every sample consists of three values corresponing to magnetic field intensity in each direction measured by each of fluxgates and temperature of the sensor head. Interface uses ASCII coding with one stop bit and no parity bit [9].

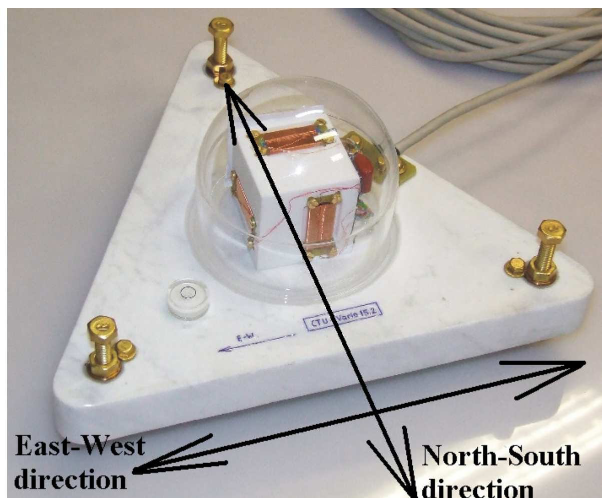


Figure 2.4: Photo of observatory magnetometer with shown directions of axes [9]

Low noise level is reached by using compensation in the Z and H axes (axes are defined in section 4.0.3). Details of the compensation and how it affects the calculations is described in section 4.0.3. All sensors are mounted on a ceramic cube which is attached to a triangular holder. As it was mentioned, temperature of the cube holder is monitored and sent in data sample.

2.4 DC/DC converter

Complete device can be powered either by a battery or by a power source. The power source is connected to the power grid and gives an output of 13.8 V. It is used power the device and charge the battery.

Battery is used to power the device in case of power failure. It has approximately the same voltage level, around 13.8 V. However, Raspberry Pi is powered by 5 V DC, hence there is a need for DC/DC converter from 13.8 V to 5 V.

Table 2.3: DC/DC converter characteristics

Chatacteristic	Value
Name	Mean Well SD-15A-5
Model	SD-15A-5
Input voltage	9.2 - 18V
Input DC current	1.9A/12VDC
Efficiency	68%
Output voltage	5 V
Output voltage regulation	4.75 - 5.5V
Output current	3 A
Power	15 W
Weight	0.68 kg
Dimensions	78mm x 51mm x 28mm

Converter Mean Well SD-15A-5 is used to convert 13.8 V to 5 V. Input of the converter is connected to the battery. Output is connected to Raspberry Pi with power LED indicator and switch to turning it off when Raspberry is shut down .

Figure 2.5: Photo of DC/DC converter Mean Well SD-15A-5 [10]

3. Comparison to the state-of-the-art

Magnetometer data loggers are already used for a long time. Main purpose of this type of devices is to collect data generated by a sensor and store it on some kind of storage device.

A lot of data recorders just record data without any calculations. The recorded data is then transformed from raw sensor output to usable physical quantities, for example components of magnetic field vector **B**.

The problem arises from the fact that raw data is not suitable for future calculations, consequently every time you need to perform transformation. Another problem is that real sensors are not ideal devices and have offsets, thus there is also need for calibration.

One very important feature of data recorders is mobility: it can be easily installed in different places. The sensor and the data recorder can easily be moved in case of some problems, for example unwanted sources of magnetic field.

Figure 3.1: The AMOS Mk 3 is an automated Observatory system deployed [11]

In Figure 3.1 data recorder from 1980s can be seen. It has large dimensions and is equipped with a lot of hardware. Such old data acquisition systems are rarely used nowadays.

However some of the old facilities are still in use. In Figure 3.2 you can see curretnly used data recorder in Budkov observatory [11]. It is commonly used in a modern magnetic observatory. It is hard to access the sotred data, because the periferals of these data recorders are outdated and are never used in modern computers.

Figure 3.2: Instruments typical of a modern magnetic observatory [11]

There is another example of data logger in Figure 3.3. It is small and has standard RS232 serial link, but its does not have any kind of indicators to display devices status.

Figure 3.3: Magnetometer sensor with data logger [12]

Raspberry Pi is very popular single-board computer and there exist other data loggers based on it. For example, Rapberry Pi has been used as data logger for observatory magnetometer [13] before, but it stores data without any calibration, conversion or data preprocessing.

Last exapmle of very good data recorder is recorder used with observatory magnetometer LEMI-025 [14]. As it can be seen in Figure 3.4, this device has compact plastic case with control buttons and SD-card slot. But in comparison to device made in this work it does not have any display to show actual information about collected data.

Figure 3.4: Data recorder fo 3-component 1-second observatory Magnetometer LEMI-025 [14]

All these devices have some disadvangates which I attempt to solve in this project.

Device, developed in this project, has all these characteristics which were mentioned before. It doesn't need external computer and any other kind of periferals. The device has internal accumulator and can be powered even in case of power supply failure, but only for a short period of time.

Communication with the devices is easy enough and uses standard communication protocols. All computed data is stored on disk and showed on display for a quick check. To download data directly from the device, computer must be connected to it through Ethernet cable. SSH protocol [15] can be used for data transfer in this case. All files are uploaded to remote FTP-server, stored in data catalog and can be easily downloaded, for example by Midnight commander [16].

The device is also equpped with touchscreen display which provide user a brief overview of obtained data.

4. Theory

4.0.1 Fluxgate types

Fluxgate is a very common sensor of magnetic flux density with a wide range of applications. They are used when there is a need for precise measurement of magnetic field. Two main types of fluxgate magnetometers exist, parallel and orthogonal. Although both types of sensors were invented almost at the same time, the parallel principle of sensor became more popular. Only in the last decade orthogonal sensors became popular due to new technology which allows us to manufacture small orthogonal fluxgates compare to parallel one [17].

In our case we are using parallel fluxgate which characteristics can be found in "Used devices" (Section 2) section. Difference between parallel and orthogonal fluxgate can be observed in Figure 4.1:

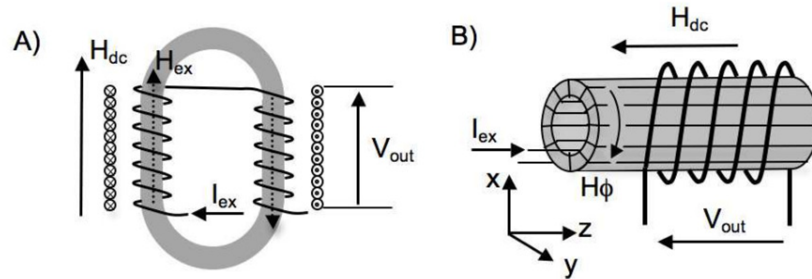


Figure 4.1: Structure of parallel (A) and orthogonal (B) fluxgates [17]

Parallel fluxgate consists of ferromagnetic ring wound with an exitation and a pick-up coils. This types of fluxgates is called parallel because measured magnetic field is parallel to exitation field. Ferromagnetic core is periodicaly saturated by excitation magnetic field generated by AC current. Saturations occures twice during one period of exitation current, consquently two voltage pulses are induced in the secondary (pick-up) coil. Fluxtage principle is explained in more details in [17].

In orthogonal fluxgate, we have ferromagnetic core and excitation toroidal coil. The pick-up coil is wound around the core. In this case exication field lies in XY plane and is orthogonal to sensed magnetic field which lies in Z direction [17].

We are using parallel fluxgate of race-track type [9]. Therefore, we need three fluxgate heads directed in all three Cartesian axes, in other words, all sensors should be perpendicular to each other.

4.0.2 Orthogonalization

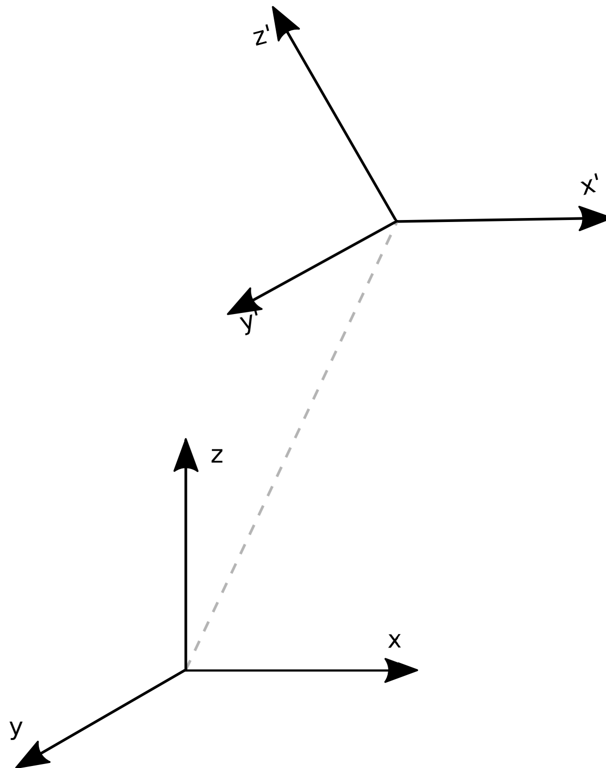


Figure 4.2: Orthogonal and non-orthogonal basis

Let us look at what orthogonalization is. Initially we receive data in the non-orthogonal magnetc sensor axes coordinate system, because sensors are not located perfectly. There is a need for transformation to an orthogonal coordinate system for easier future calculations.

This transformation is named orthogonalization. According to [18] orthogonalization is method that finds an orthonormal basis of the span of given vectors:

$$S = span\{a_1, ..., a_k\} = span\{q_1, ..., q_n\},$$

where $a_1, ..., a_k \in R^n$ are given vectors, and $q_1, ..., q_n \in R^n$ are computed orthogonal vectors.

For orthogonal vectors next condition must hold:

$$q_i^T q_j = 0 \; for \; i \neq j.$$

If this condition is valid for every pair of vectors q_i, q_j $(i, j = 1...n, i \neq j)$, then $span\{q_1, ..., q_n\}$ forms an orthogonal basis of R^n space.

In Figure 4.2 a difference between non-orthogonal space and orthogonal space is shown. If we have some vector in non-orthogonal space, it has scalar projections on every

axis. The these values of these projections are different in orthogonal space. Thus, orthogonalization transforms vector elements from one basis to another. In this case its tramsformation is performed from the non-orthogonal magnetic sensor axes coordinate system to orthogonal magnetic axes system.

4.0.3 Used orthogonalization

Obtaining of calibration coefficients is out of the scope of this thesis, only implementation of a software program with already known data is required. The orthogonalization is implemented in the software in according to [19].

$$\mathbf{B} = \mathbf{PS}(\mathbf{F} - \mathbf{O}),\ where$$

B - result value of vector intensity coresponding to sensor direction [T]
P - orthogonalization matrix [-]
S - sensivity matrix [LSB/T, LSB - least significant bit]
F - raw data vector received from the sensor, in Engeeniring Units [EU]
O - offsets vector for all three directions [LSB, LSB - least significant bit]

In matrix form:

$$\begin{bmatrix}\Delta H\\ \Delta Z\\ \Delta E\end{bmatrix} = \begin{bmatrix}p_{11} & p_{12} & p_{13}\\ p_{21} & p_{22} & p_{23}\\ p_{31} & p_{32} & p_{33}\end{bmatrix} \cdot \begin{bmatrix}\frac{1}{S_H} & 0 & 0\\ 0 & \frac{1}{S_Z} & 0\\ 0 & 0 & \frac{1}{S_E}\end{bmatrix} \cdot \begin{bmatrix}R_H - O_H\\ R_Z - O_Z\\ R_E - O_E\end{bmatrix} =$$
$$= \begin{bmatrix}p_{11} & p_{12} & p_{13}\\ p_{21} & p_{22} & p_{23}\\ p_{31} & p_{32} & p_{33}\end{bmatrix} \cdot \begin{bmatrix}\frac{1}{S_H} \cdot (R_H - O_H)\\ \frac{1}{S_Z} \cdot (R_Z - O_Z)\\ \frac{1}{S_E} \cdot (R_E - O_E)\end{bmatrix} \tag{4.1}$$

As mentioned before O_H, O_Z, O_E - are offsets of sensor [EU], S_H, S_Z, S_E - are sensitivity for every direction [LSB/T], P - is orthogonalization matrix [-], C_H, C_Z, C_E - are elements of compensation field [T].

Subindexes mean H - North-South direction, E - East-West direction and Z - vertical direction.

If we are intrested in complete value of vector intensity, not its variation, we need to add applied compensation field:

$$\begin{bmatrix}H\\ Z\\ E\end{bmatrix} = \begin{bmatrix}C_H + \Delta H\\ C_Z + \Delta Z\\ C_E + \Delta E\end{bmatrix}$$

ΔE is approximately zero in comparison to other components and compensation is not used in this axis. For example for Prague, corresponding to World Magnetic Model [20], magnetic field is:

Model Used:	WMM2015				
Latitude:	50° 6' 22" N				
Longitude:	14° 27' 25" E				
Elevation:	0.0 km Mean Sea Level				
Date	**Horizontal Intensity**	**North Comp (+ N | - S)**	**East Comp (+ E | - W)**	**Vertical Comp (+ D | - U)**	**Total Field**
2016-05-12	19,902.8 nT	19,861.0 nT	1,288.6 nT	44,798.1 nT	49,020.3 nT
Change/year	7.7 nT/yr	4.9 nT/yr	44.1 nT/yr	29.0 nT/yr	29.7 nT/yr
Uncertainty	133 nT	138 nT	89 nT	165 nT	152 nT

Figure 4.3: Magnetic field in Prague according to World Magnetic Model [20]

There is table of calibration values provided by supervisor [9] :

Table 4.1: Parameters for used magnetometer

CTU-Vario			
Axis	H (North-South)	Z (vertical)	E (East-West)
Offset [LSB]	-2338	-632	-1823
Sensitivity [LSB/T]	1.8802E+12	1.8714E+12	1.8583E+12
Compensation field [T]	2.6580E-5	3.7380E-5	0
Orthogonalization matrix [-]	$\begin{bmatrix} 1 & 0 & 0 \\ 0.003272853 & 1.000005356 & 0 \\ -0.008173561 & -0.023780785 & 1.000315443 \end{bmatrix}$		

5. Hardware part

The project is based on Raspberry Pi [21] microcomputer. Even with such small size it has enough computation power to allow communication with the sensor, data processing and plotting at the same time.

Figure 5.1: Photo of the device

Complete device, which I have built, is represented as a plastic box with the display and few connection ports such as Ethernet, RS-232 serial port and two connectors for power supply. In Figure 5.1 disconnected, unpowered device is shown.

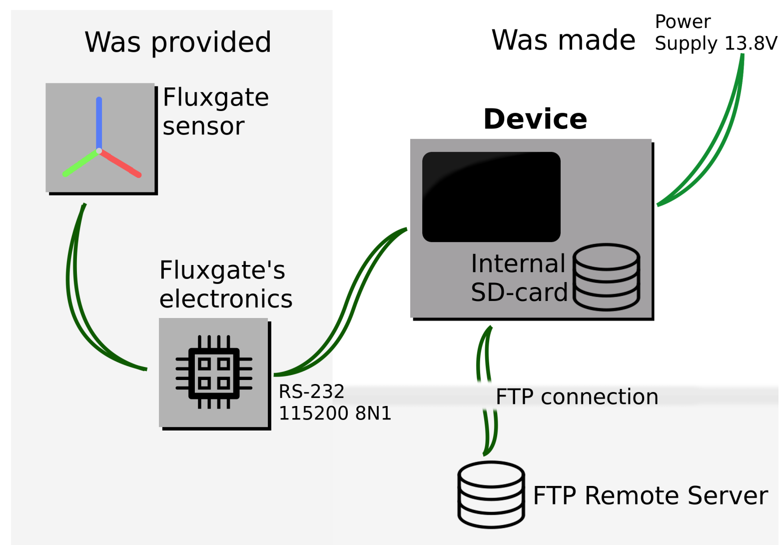


Figure 5.2: Schematic of connection device with perpherials

In Figure 5.3 schematic of the internal configuration of the instrument is shown. As it can be seen, the device contains battery for the purpose of stable work even in case of power supply problem (for example blackout).

Raspberry Pi and display are powered by 5 V DC. However, the battery supplies 13.8 V. Because of this fact the battery is connected to DC/DC converter [Section 2.4] which converts 13.8 V to 5 V. The converter is connected to Raspberry GPIO DC power pins 04 (5 V) and 06 (Ground).

The display can be powered both from the micro-USB connector and on-board pins. Because there is no need to use micro-USB, wires from display driver are connected to GPIO pins 09 (ground) and 02 (5 V). It should be noted that data bus which transfers signals between Raspberry Pi and its display is not shown on the schematic. On both boards, there is a particular connector for this interconnection.

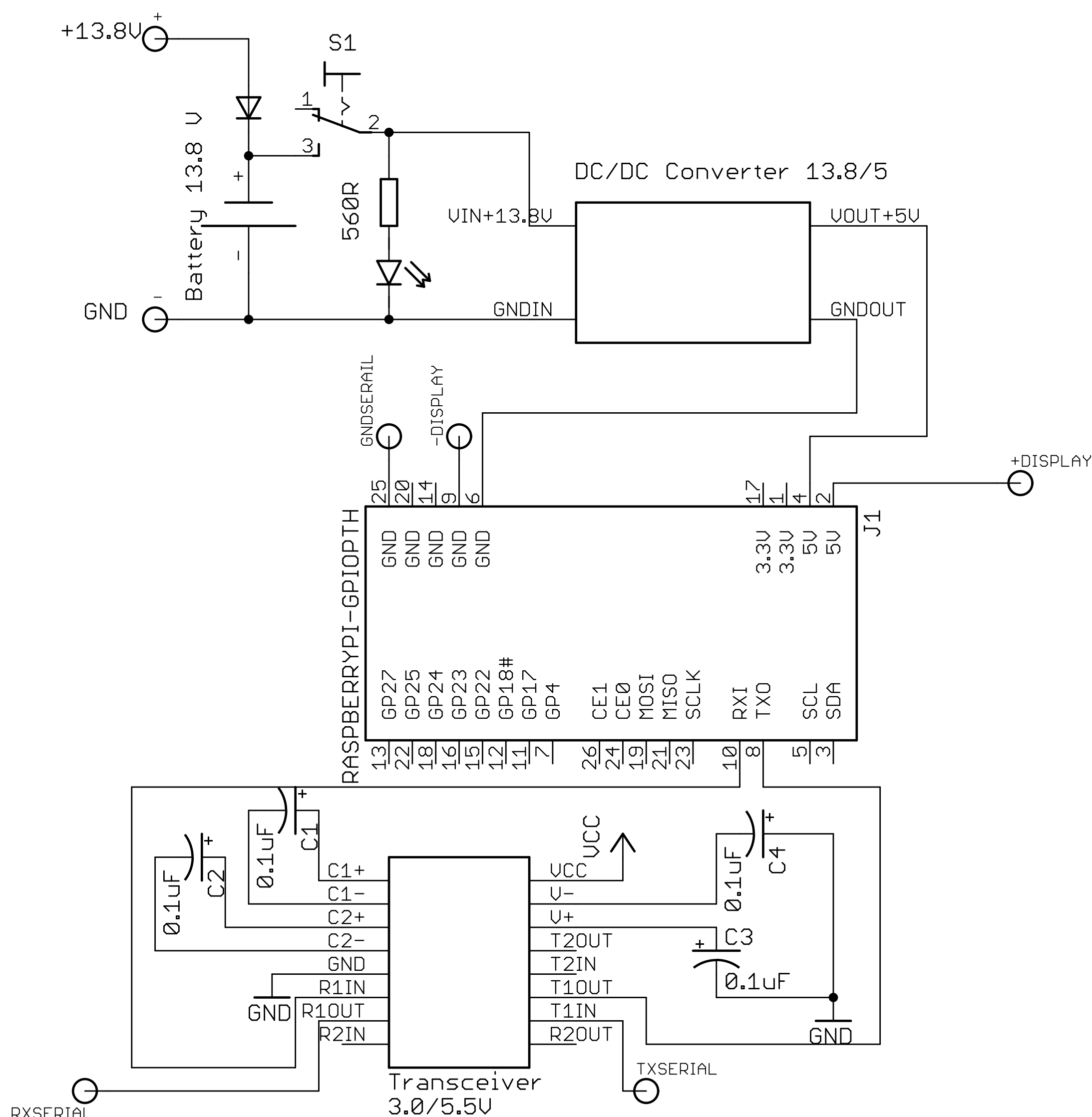


Figure 5.3: Inner connection of the device.

Communication of a magnetometer electronics and Raspberry Pi is carried out by the

serial link. There is 5 V signal on the output of RS-232 port of the sensor but GPIO serial pins are designed for 3 V communication. Therefore, there is a need for digital transceiver between mentioned voltage levels.

I chose to make it based on the MAXIM3232 chip. It is true RS-232 transceiver from 5 to 3 V which use four 0.1 μF external capacitors. I connected it using standard wiring represented as an example in datasheet MAXIM3232, page 12.

Because the battery is connected to Raspberry through DC/DC converter, there is switch to disable powering of the converter in case Raspberry Pi is turned off.

The device can be powered by internal battery or power source. If a power supply is connected it charges the battery and powers the device. If power supply is not connected energy is taken from the battery. Because input of power supply is connected in parallel with the battery, there is a need for a diode. Otherwise, the battery will give some amount of current to the power source which is unwanted.

In Figure 5.4 side panel of device is shown with all parts marked:

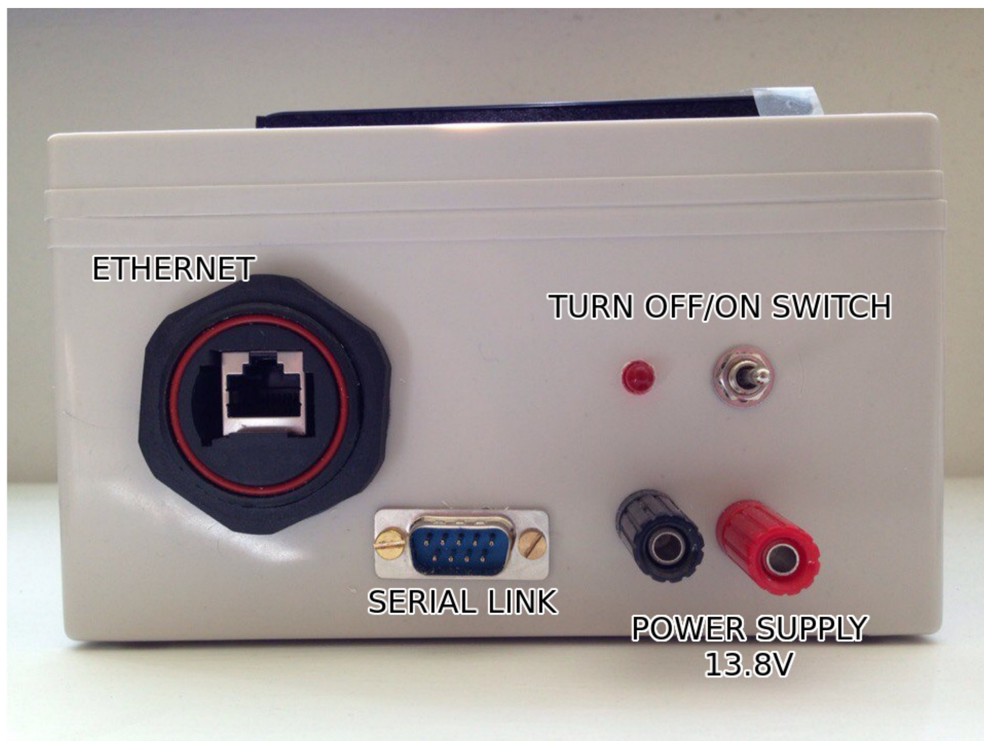


Figure 5.4: Devices connectors.

In the table 5.1 some power characteristics of the device are shown. For power saving there is a screensaver mode in-built in Raspbian operating system. The device turns off the screen after ten minutes of inactivity.

Table 5.1: Table of device power characteristics

Characteristic	Value
Voltage	13.8 DC Volts
Current without display	$\approx$ 0.43 A
Current with display	$\approx$ 0.75 - 0.80 A
Estimated working time using battery without display	$\approx$ 4 - 5 hours

Table 5.2: Battery Characteristics

Model	NPG 2.2-12, 12V2.2Ah	
Type	Voltage Regulation	Inital Current
Stand-by use	13.5 $\sim$ 13.8 V	No Limit
Cycle use	14.4 $\sim$ 15.0 V	0.66 A Max

6. Software part

6.1 Program for receiving information from sensor

6.1.1 Description

Firstly, let's look at the main program I have developed which is responsible for communications with the sensor by the RS-232 serial bus, receiving data samples, performing calibration and storing obtained values on a disk with respective timestamps.

The program is separated into three different processes. First "main" process is a main flow of the program, it communicates with the sensor and sends requests for raw data. The second process, "processor", is responsible for data averaging and orthogonalization. The last one, the third process saves obtained data to files and sends it to predefined network socket. Processes are connected by two pipelines, and they are independent, only in case of terminating the main process, other processes are killed.

This program has a few test functions for debugging. One of them is that program can indicate every received sample by changing state of one of the GPIO pins, available on Raspberry Pi. This pin can be chosen in the configuration file, by default it is pin 4. This functionality can be used for confirming that all samples sent by sensor electronics are received by Raspberry Pi.

Another option is that program can be run in "virtual" mode and does not need a real sensor. In this mode program generates random data and uses it as an input stream. It can be used for verification that calibration is working correctly and it also can test program functionality.

Finally, the data flow of the whole program cycle can be monitored by turning on logging procedure. The program contains information messages in different places to inform a user about currently performed task. It is possible to turn on/off this option in the configuration file. Logging is written using built-in "logging" library and can be set up to different levels of message importance.

The program is written using numerical library NumPy [22], which allows easy use of matrix operations and works with big data arrays. GPIO library is also used to provide a communication interface with on-board pins. In case debug mode is turned on (thus some of the GPIO pins are used for samples indication) it is necessary to install GPIO library to communicate with those pins; otherwise this library isn't used and can be skipped.

The code is written and tested in Python 3.2, so it should be compatible with all Python 3.* versions.

6.1.2 Program structure

Now let us overview the program structure. As mentioned previously, the program consists of three independent processes. The need for independent processes was found during testing: possible data drop-out due to heavy load dit not allow a single-process program. Also the CPU load is low in this case: 30 - 40 %. In Figure 6.1 a simplified diagram of program structure and data flow can be seen.

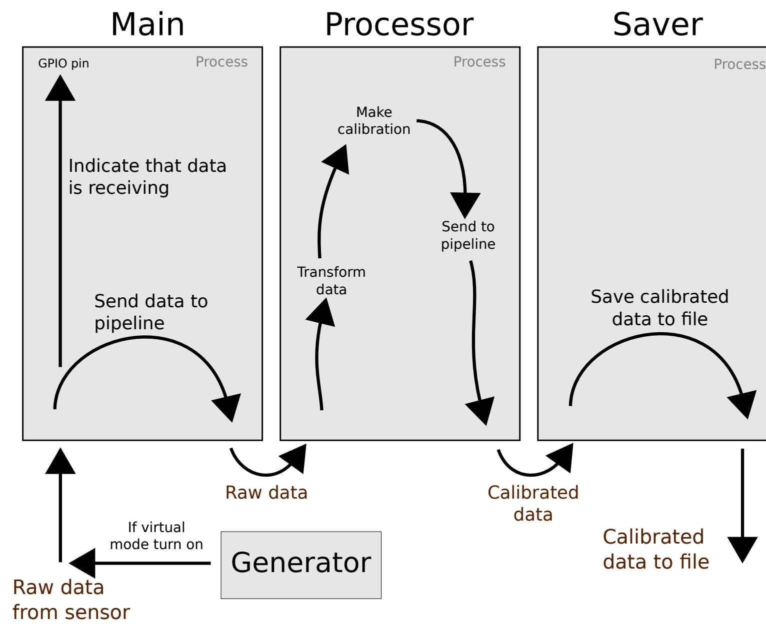


Figure 6.1: Structure of main program that communicats with sensor

List of files:

1. **main.py** - This file is main input point to the program. Runs as a separate process and performs data reading from the serial port and then sends it to a pipeline to the following calibration.

2. **saver.py** - contains only one function, which is run as a separate process and performs data saving.

3. **calibration.py** - contains a few utility functions, which are used in "processor" for data processing and calibration.

4. **processor.py** - contains one function, which is run as a separate process. This function perform data processing and send obtained values by pipeline to "saving" process.

5. **reader.py** - this file contains functions for serial communication setup and functions for communication with the device.

6. **generator.py** - contains class, which is used as a generator of random numbers. It is used only in debug mode when we do not have real sensor and we use this generator to test program functionality.

7. **settings.py** - this file contains two classes which are responsible for program and data processing settings 6.1.3.

There are three main files which should be discussed in detail. "main.py" is the main file of the program which runs other processes. This file contains only one function "init_sensor" which checks if initialization of sensor is successful and starts it with "CN" command.

Let us overview what happens when the application is started. At first, the program gets some configuration settings from the settings file. Then, program checks which mode is chosen; if arguments of the program contain word "virtual" then the program runs in virtual mode. In this case instead of using the real sensor, "Generator" object is used. This "virtual sensor" creates random data used only for testing. Note that it does not behave like a real magnetometer.

The pipeline connected between the main and calibration processes is created. After that data handling process is run.

The main loop of the main process receive data from a sensor and send it to the pipeline. However, in case debug mode is on (it can be turned on/off in the config file) the program also indicates every sample by changing state of GPIO pin.

Next vital process is "processor" which performs all calculations in the program. File "processor.py" contains only one function and definition of Gaussian distribution represented as a numpy [22] array, which later will be used for averaging of collected samples.

Function "process_data" (name of a function in file "processor.py") is run as a separate process from the main file with a few parameters including binding pipeline. First of all, some variable initializations are performed. Then new pipeline is created which will be connected to this process and process which saves data.

After that main loop is run. In the main loop the process receives samples from the pipeline and if a buffer is already full it performs orthogonalization and sends calculated samples to next process. Otherwise, it add new sample to the buffer.

The buffer is used for finding mean value before orthogonalization. Resulting samples are generated every second, but raw samples received from the sensor are generated 207 times per second. Thus, at first, program accumulates some amount of data and then start performing all calculations.

Let us overview implemented data processing. In "processor" two functions are called, "find_mean" and "calibrate".

```
# Calculate mean value and make calibration
mean_value = find_mean(data_set, gauss)
result = calibrate(mean_value)
```

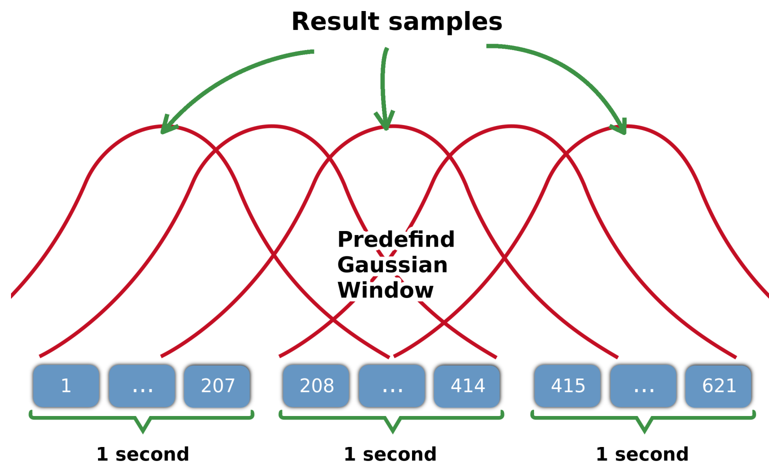


Figure 6.2: Applience of Gaussian Window [5]

The first function receives data set of samples obtained by the sensor and defines an array ("window") with Gaussian distribution. Then the Gaussian array is multiplied by data set and arithmetic mean is found every second according to IAGA standard, by using numpy library [22].

```
def find_mean(data, gauss):
    return np.mean(gauss[np.newaxis, :].T*data, axis=0)*2
```

In Figure 6.2 diagram of this process can be seen. After applying a window with Gaussian distribution on data set, program finds mean value and returns it as the result.

Next important function is "calibrate". It receives data sample generated by the previously mentioned function as parameter. Only first three componentes are taken from the sample, fourth component is temperature which is not needed in conducted calculation. Then we get calibration settings from the configuration file and perform calculation by method, described in the capitol 4.1. At the end we add temperature to resulting array.

```
def calibrate(data):
    raw = data[0:3]
    comp = np.array(cal_config['comp'])
    ofs  = np.array(cal_config['ofs'])
    sen  = np.array(cal_config['sen'])
    P = np.array(cal_config['ort_mat'])
    eu = raw - ofs
    B = np.array(1/sen*eu)
    M = np.dot(P, B)
    M = M + comp
    M = list(M)
    return M.append(data[3])
```

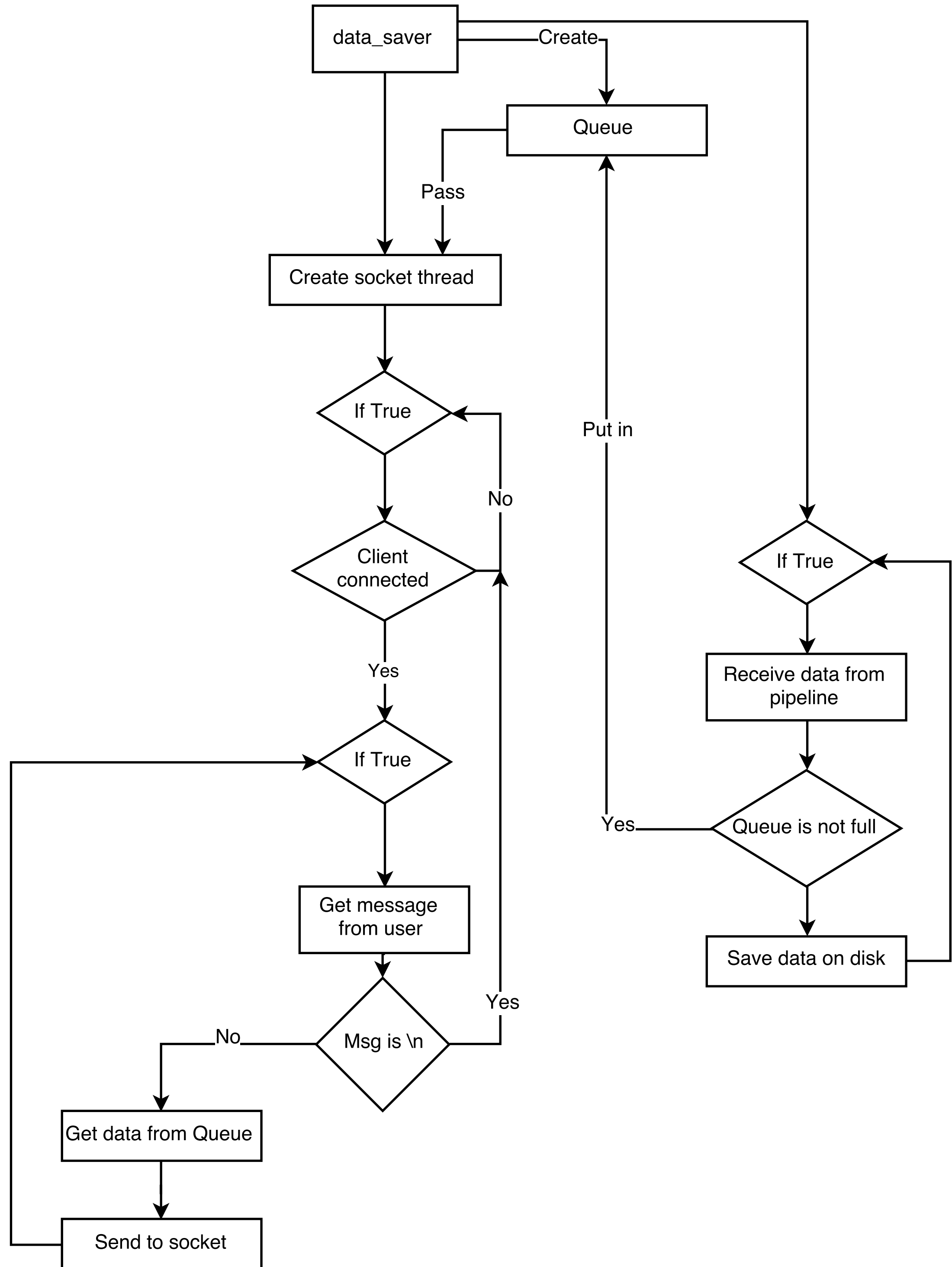


Figure 6.3: Structure diagram of "saver" process. "Msg" - received message

All processes have simple structure and can be easily understood. However, "saver" has more complicated construction. In Figure 6.3 structure diagram of this process can be senn:

Main function "data_saver" creates data queue and one more thread which is responsible for communication with clients by network socket. After that in while loop function starts receiving data from the pipeline, putting it into queue and save it to disk.

Socket thread runs a loop which is waiting for a client to connect. If a connection is detected one more loop is started. In this loop program receives one symbol from user and if it is the EOL symbol ("\n") loop breaks. Otherwise, the thread gets data from the queue and sends it to the socket.

6.1.3 Configuration settings

In this section the description of the configuration file can be found. Program has two configuration classes. One of them is responsible for program configuration: path to data file storage, debug mode, serial communication settings, commands for communication with the sensor, etc. The second class is responsible for data processing settings, such as offsets, sensitivity, orthogonalization and compensation field.

Program configuration:

1. **samples** - the number of samples received per second.
2. **debug** - Turn on or turn off debug mode. If debug is on the program will indicate every sample on debug pin by sending impulses and all program actions will be logged.
3. **debug_pin** - GPIO pin which is used to indicate samples received from the sensor. This pin will change its state before and after receiving data sample.
4. **path** - path where the program will storing all its files.
5. **port** - port for serial communication. The sensor should be connected to this port. By default this port is '/dev/ttyAMA0'.
6. **baudrate** - serial communication speed. By default, speed is set to 115 200 baudrate.
7. **timeout** - timeout for serial communication. By default, the value is 3 seconds.
8. **start_cmd** - byte string, which is sent to the sensor to start it. In other words, command to start data stream.
9. **stop_cmd** - byte string, which is sent to the sensor to stop it.
10. **file_name_format** - string, which defines a format of names of files with obtained data.

Calibration settings:

1. **comp** - compensation field.

2. **ofs** - offsets.
3. **sen** - sensitivity.
4. **ort_mat** - orthogonalization matrix.

6.2 Program for plotting data to the display

6.2.1 Description

One of the next steps of this work is to implement visualization of received and processed data. The device is equipped with the color touchscreen display which is used for displaying variations in the magnetic field in the last few hours.

Main program (Chapter 6.1) saves calibrated data on disk and in the same time send it to localhost socket. One of the thread in "saver" process is waiting for a client to connect to specified localhost port and start transmitting requests for new data.

Because the program is using multithreading concept, multiple clients can be connected at the same time.

In the configuration file a correct port which can be used to open a new socket must be chosen. GUI program automatically tries to connect to specified port and log the whole process of connection for troubleshooting purpose.

If the connection is successful, program starts collecting data for offsets calculation. Because received data ranges are very narrow, compared to their mean value, they can not be plotted on one graph; they will be represented by three straight lines. Hence, offset adjustment is required.

Offsets are calculated by taking average value of some number of received samples. This number, size of the buffer, can be changed in the configuration file. Thus, at first, the program is collecting samples to fill up the buffer and after it is full, offsets are calculated and a loop starts to send obtained samples to the GUI thread.

It happens only in case if the program is run the first time. When offsets are calculated, they are saved to file (filename is specified in the configurations), and next calculations of offsets are performed with 24 hours period by taking average value from the measurements obtained during the last day. Thus, if the program runs not the first time, it loads offsets and start displaying data without delay.

The graph shows three lines which correspond to values of H, Z and E sensors (directions of sensors can be seen in Figure 2.4). The graph is in scope mode, thus, at the beginning graph starts to fill in with the data until specified time point (for instance, 12 hours) on the **x**-axis (time axis) is reached. Then plot starts to move left as it receives new data samples.

In left-top conrner the legend is displayed. It contains offsets for every data stream. The legend is updated periodically with new offsets (because new offsets are calculated

every 24 hours the legend is updated with the same period).

The program is written using the QT Framework [23]. It is big collection of libraries with wide possibility to create advanced graphic user interface applications. This framework has bindings library for Python, which allows using the whole set of available functions. Also, QT applications can be run without any desktop environment, which is suitable for this project because desktop functionality is not required and can be omitted.

6.2.2 GUI program structure

The program is based on MVP pattern [24] which consists of three main parts: Model, Presenter, and View. Every module is responsible for specified purpose: Model for data representation, View for user interface and Presenter for communications between other modules. Let us consider each parts separately.

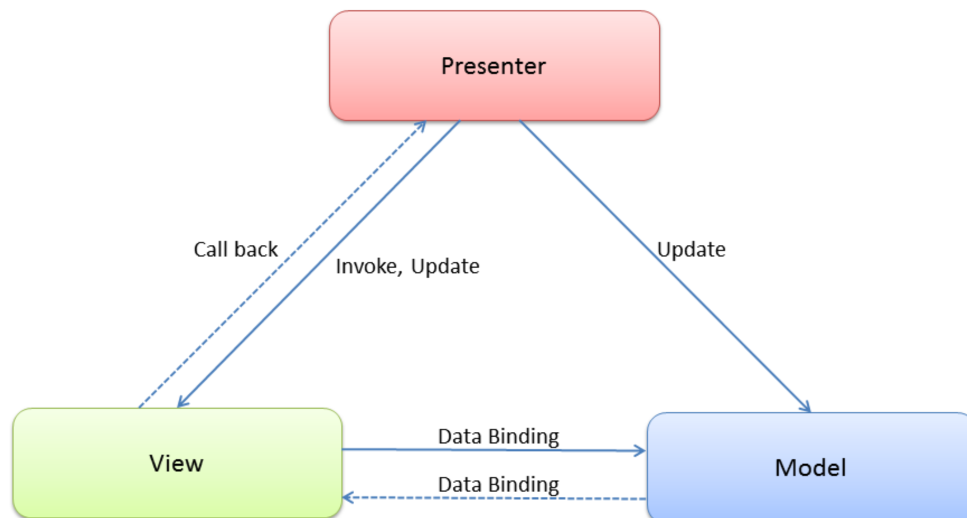


Figure 6.4: MVP programming pattern [25]

Model is a separate thread represented by one class named "Model". This class contains function "run" which is the main loop of this thread. All other functions are secondary and used in the main loop, except "stop" and "getQueue", which stop thread and return data queue, respectively.

First of all, in "run" function program tries to connect to local server socket which address is set in the configuration file. After starting the main loop of the Model, the

program starts trying to get samples from local server and put it in the data queue, if it is not full. Otherwise, it repeats until the queue has free space or thread is stopped.

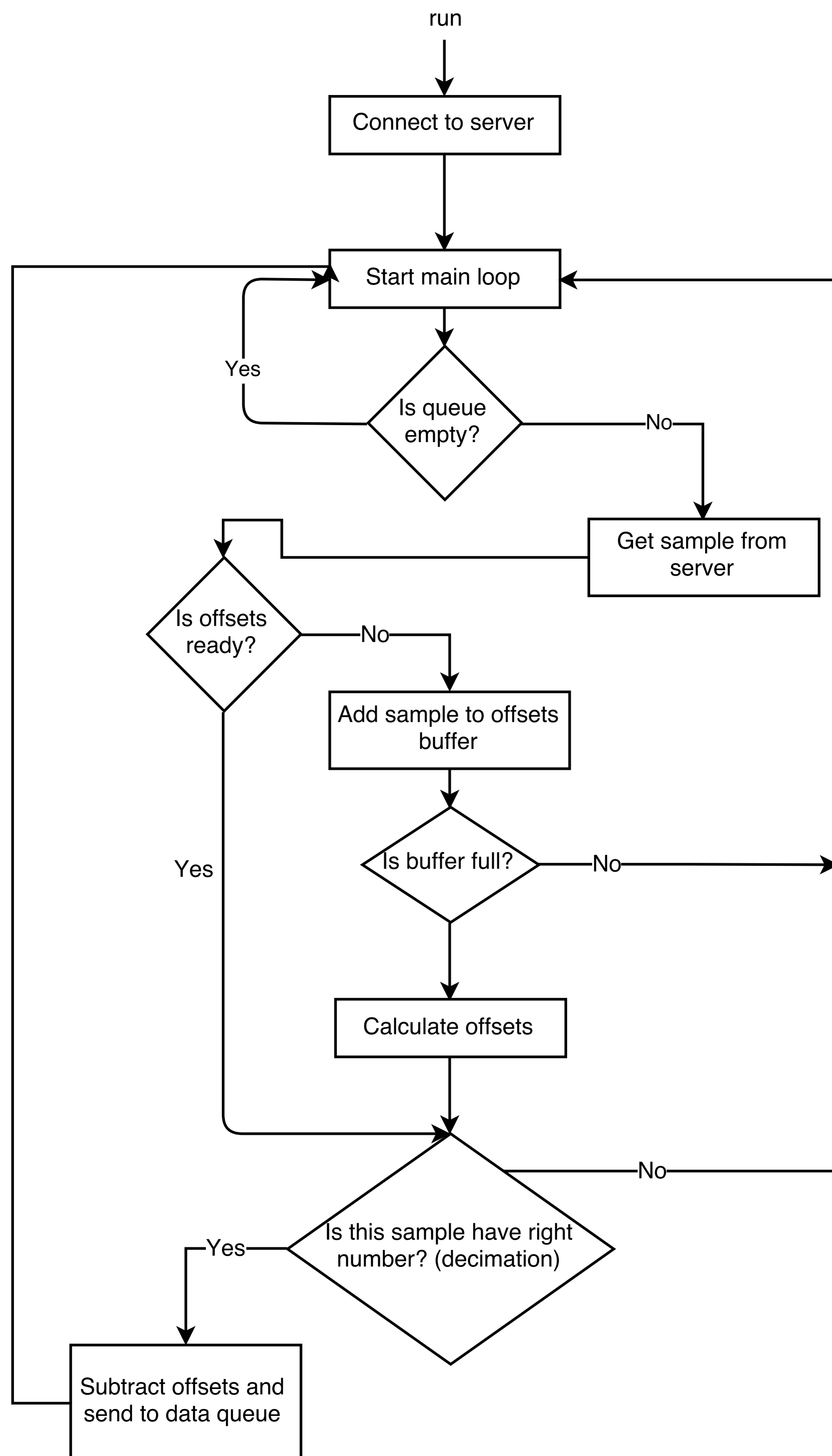


Figure 6.5: Structure of process represented data in GUI application

In case the queue is empty, the program gets a raw sample and takes data items, without timestamps. Then programs tries to load offsets from the file which is specificated in the settings. If loading is failed (because the file doesn't exist), program starts to fill

sample buffer for calculating offsets. Otherwise, the program loads offsets from the file and continues.

After the sample is received program checks if decimation is set up and if received sample number is demanded. If not, program returns to step one, where it receives sample, otherwise it subtract offsets from the sample and sends it to the data queue.

Because offsets are calculated every 24 hours, in every loop cycle program checks if there is a need for calculating new offsets.

Next part of the program is Presenter. It acts upon the Model and the View: it receives signals from the View and performs predefined reaction on them. For example, when the View is closed it sends "quit" signal which is handled by the Presenter. The Presenter then sends a stop command to the Model thread.

The Presenter also performs initialization of other signals between the View and the Model. These signals are handled by the View and perform data transfer between the Model and the View.

The last significant part of the program is the View. It is an interface which visualizes data. In this case, it is GUI (Graphical User Interface) based on QT framework [26].

The View contains two classes: "view" and "plot_ui". These class are responsible for window initialization, such as setting title, showing window in fullscreen, creating widgets layout, etc. In the function "initScene" initialization of Plot object, which is the main widget of the application, is called.

PyQtGraph library [27] was used for plotting data. At first, class calculates how many samples does it need to show a full graph. The time range is set up in the configuration file and determines range of x-axis in hours. However, the program needs to know the amount of samples displayed, hence we need to convert the dimensions of x-axis from hours to number of samples.

6.2.3 GUI program configuration settings

In this section a description of all configurations settings can be found. The GUI program and the main program are two different entities and they are completely independent; they have separate settings files.

Program defined settings in "settings.py":

1. **time_axis_range** - this value sets how wide is x-axis (time axis). For example, if the value is set to 12, x-axis will be 12 hours long.
2. **off_filename** - filename where the program will store offsets. If there is need to calculate new offsets, the file can be deleted, and after that program should be run. This action will force the program to calculate new offsets instead of waiting to end of the period. Some amount of time will be needed to fill up offsets buffer and then calculate them.

3. **o_buffer** - this value defines size is offset buffer. Offsets are calculated by taking arithmetic mean from samples added to this array.
4. **p_offsets** - if there is a need for decimation while calculating offsets this setting can be set to some value which will be the period of taking the sample for offset. For example, if p_offsets is equal 5, every 5-th sample will be added to the offset buffer.
5. **host** - host IP address. The program gets samples from a network socket. Therefore, some IP address should be specified. It does not have to be local IP, global IP can also be used.
6. **port** - this is the port which program will use to listen for new data.
7. **time_format** - this value specifies time format which is used to show time marks on the x-axis.
8. **min_stack_size** - this value is used for the offsets buffer size if the value specified for time axis is too small. For example, if time_axis_range = 1/60 (one minute), the program will use the value from min_stack_size because offsets buffer size is calculated from the size of the x-axis. When calculated value is smaller than min_stack_size default value is used. It is made because in case of a small x-range the graph looks uninformative. This setting can be set to some small value and will not affect time axis.

6.3 GUI program for presentation purpose

This program is a combination of the two previously described programs. Essentially it consists of GUI program described in the previous section (Chpater 6.2) and orthogonalization took from the first program (Chapter 6.1). The purpose of the program is presentation of the observatory data in Kelčany [28].

It is made because the remote server with magnetometer at the Kelčany observatory [28] provides data samples which are already represented as an average value of all samples collected in a period of one second but without applied orthogonalization on it.

The given program is a modification of the GUI program (Chapter 6.2). Changes are applied to Model class which now receives samples until samples buffer is full. The formula 4.1 is then applied to perform orthogonalization and transformation from engineering units to nT (nano-Tesla).

The configuration file was also changed. It contains settings for the GUI program which are described in section 6.2.3 and orthogonalization settings, such as compensation field, offsets, sensitivity and orthogonalization matrix.

7. Tests and results

There is representation of work of the device. In Figure 7.1 graph of collected data in a last 12 hours is shown. The data was remotely obtained from the observatory Kelčany [28]

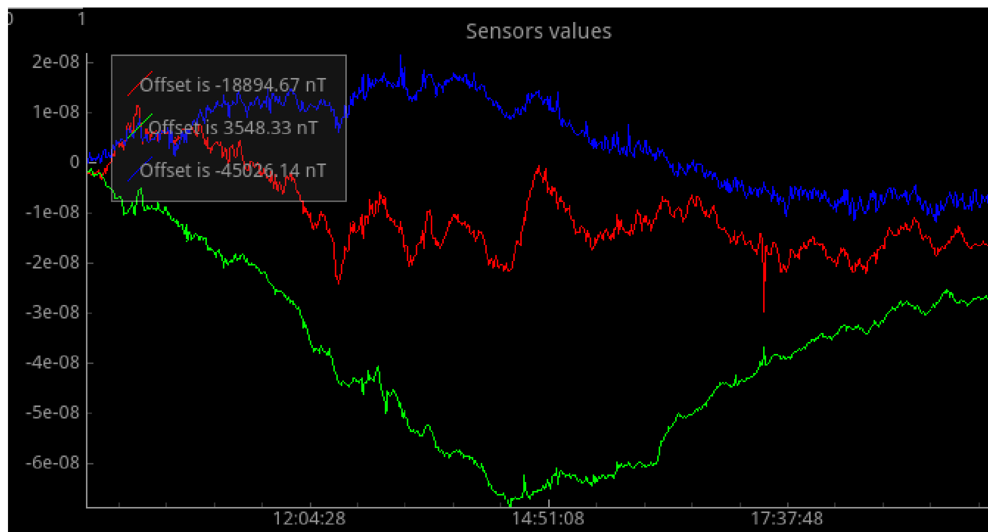


Figure 7.1: Screenshot of the display with data recorded in a last 12 hours

In Figure 7.2 the working device with recorded data from an attached magnetometer is shown. The data is noisy because the trial was run in the laboratory.

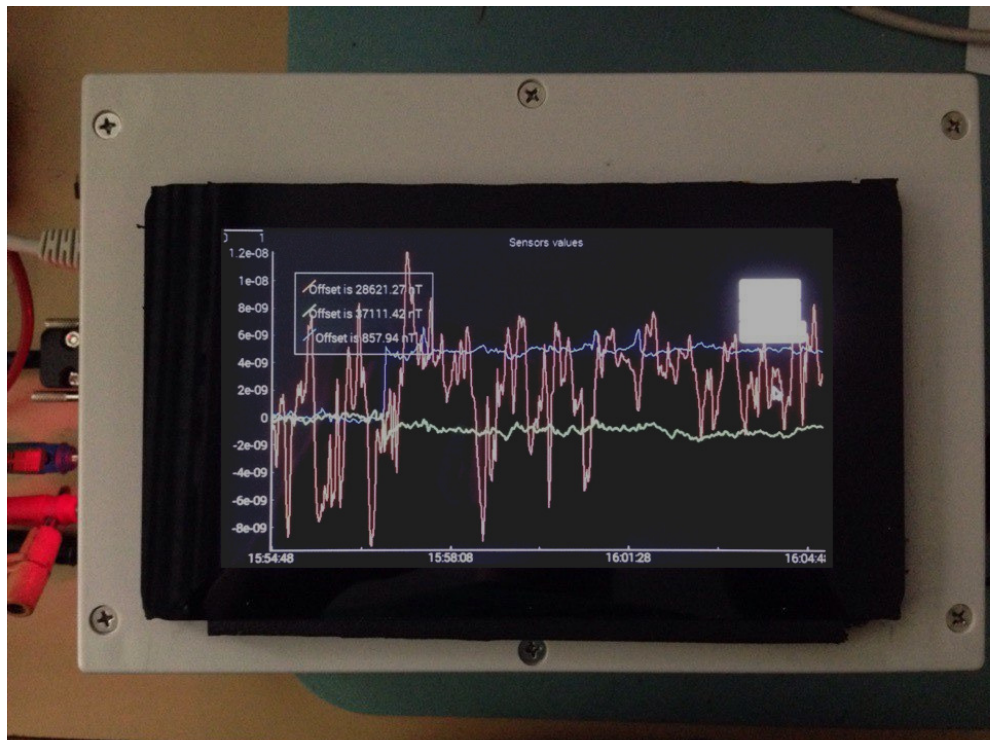


Figure 7.2: Photo of the device with recorded data

There is example of data saved on disk and sent to FTP-server, every line is represent one second sample. First three components are values of flux density [T] in every direction and fourth component is temperature dependent signal (in a raw format, calibration was not provided):


```
[[2.687517901e-05, 4.020859125e-05, 2.864210491e-06, 5355661.3762318091]]
[[2.687523858e-05, 4.020941633e-05, 2.864372418e-06, 5354517.6652289182]]
[[2.687520694e-05, 4.020929451e-05, 2.863581765e-06, 5355942.2798640989]]
[[2.687522087e-05, 4.020885831e-05, 2.864254739e-06, 5354436.5794289177]]
[[2.687526896e-05, 4.020845596e-05, 2.864223554e-06, 5355283.0826091589]]
[[2.687526815e-05, 4.020914851e-05, 2.864313906e-06, 5355692.7078081947]]
[[2.687519589e-05, 4.020854741e-05, 2.864268183e-06, 5355440.3208371103]]
[[2.687521406e-05, 4.020918601e-05, 2.863392786e-06, 5354719.3055286761]]
[[2.687526172e-05, 4.020863121e-05, 2.863977543e-06, 5354886.5044245804]]
[[2.687521947e-05, 4.020905604e-05, 2.864000686e-06, 5354836.4953571102]]
```


Photo of the device from the side with ports 7.3.

Figure 7.3: Photo of the device

8. Conclusion

The objective was to get a working device to collect and store data received from an observatory magnetometer.

As a result of this project I made a device based on Rapberry Pi installed inside a portable box with touchscreen display.

The software part of the project consist of two programs writen in Python and run inside of Raspbian operating system [6].

To assure maximal efficiency and consistency of data, the program has multiprocessing structure allows simultaneious communication with the sensor, saving data to disk and sending it to a plotting program without delays.

In accordance to IAGA [5] standard was implemented orthogonalization and averaging of data for obtaining values of magnetic field flux density vector components. Thus, in result, data are stored in physical units (concretely nano-Tesla) after applied orthogonalization and offset calibration.

Obtained data is stored on an internal SD card in a predefined catalog structure. Also every thirty minutes a script is run which connects to remote FTP-server, scans it for all already uploaded data and upload new records there, if available.

In combination with remote storage, the device contains an internal battery which prevent data loss in case of power failure or internet disconnection up to 4 hours.

There are few things which can be further improved in a future development of the recorder: adding a hardware RTC clock to keep information about time during long-term power outage and also the used 13.8 - 5 V converter could have better efficiency.

Bibliography

[1] Official U.S. Government information about the Global Positioning System (GPS) and related topics. http://www.gps.gov/.

[2] Pavel Ripka. Advances in fluxgate sensors. *Sensors and Actuators A: Physical*, 106(1):8–14, 2003.

[3] DT Germain-Jones. Post-war developments in geophysical instrumentation for oil prospecting. *Journal of Scientific Instruments*, 34(1):1, 1957.

[4] HANS Aschenbrenner and GEORGE Goubau. Eine anordnung zur registrierung rascher magnetischer störungen. *Hochfrequenztechnik und Elektroakustik*, 47(6):117–181, 1936.

[5] Jerzy Jankowski and Christian Sucksdorff. IAGA - Guide for magnetic measurements and observatory practice. 1996.

[6] Raspbian - free operating system based on Debian optimized for the Raspberry Pi hardware. https://www.raspbian.org/.

[7] Raspberry Pi 2 model B. http://www.wikiwand.com/en/Raspberry_Pi.

[8] Raspberry Pi touchscreen display. http://cz.farnell.com/raspberry-pi/raspberrypi-display/raspberry-pi-7inch-touchscreen/dp/2473872.

[9] JANOŠEK, M.; PETRUCHA, V.; VLK, M. Low-noise magnetic observatory variometer with race-track sensors. In: IOP Conference Series: Materials Science and Engineering. IOP Publishing, 2016. p. 012026.

[10] DC/DC měnič MEAN WELL SD-15A-5. http://www.gme.cz/dc-dc-menic-mean-well-sd-15a-5-p332-448.

[11] Ivan Hrvoic and Lawrence R Newitt. Instruments and methodologies for measurement of the earth's magnetic field. In *Geomagnetic Observations and Models*, pages 105–126. Springer, 2011.

[12] Fluxgate magnetometer. http://www.space.dtu.dk/english/-/media/Institutter/Space/English/instruments_systems_methods/3-axis_fluxgate_magnetometer_model_fgm-fge/FGEFluxgateMagnetometerManual.ashx, 2014.

[13] Veronika Barta Dóra Bán László Bányai József Bór Árpád Kis Dávid Koronczay István Lemperger János Lichtenberger Attila Novák Sándor Szalai Judit Szendrői Eszter Szűcs Viktor Wesztergom Dániel Piri, Tamás Nagy. Universal Raspberry PI based data logger developed for the NCK geophysical obsevatory - IAGA division 5. Observatory, instruments, suveys and analyses. *Hungarian National Report on IUGG*, 2011-2014.

[14] Lviv of institute for space recearch - LEMI 025. http://www.isr.lviv.ua/lemi025.htm.

[15] Network Working Group of the IETF, January 2006, RFC 4251, The Secure Shell (SSH) Protocol Architecture.

[16] GNU Midnight Commander is a visual file manager, licensed under GNU General Public License and therefore qualifies as Free Software. https://www.midnight-commander.org/.

[17] Mattia Butta. Magnetic Sensors - Principles and Applications. Chapter 2. 2012.

[18] Orthogonalization: the Gram-Schmidt procedure. https://inst.eecs.berkeley.edu/~ee127a/book/login/l_vecs_orth.html, 2014.

[19] Nils Olsen, Torben Risbo, Peter Brauer, Jose Merayo, Fritz Primdahl, and Terry Sabaka. In-flight calibration methods used for the ørsted mission. 2001.

[20] NOAA, National centers for enviromental information. World Magnetic Model. http://www.ngdc.noaa.gov/geomag-web/#igrfwmm.

[21] Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. https://www.raspberrypi.org/.

[22] NumPy is the fundamental package for scientific computing with Python. http://www.numpy.org/.

[23] Qt Framework and Tools. https://www.qt.io/qt-framework/.

[24] The Model-View-Presenter (MVP) Pattern. https://msdn.microsoft.com/en-us/library/ff649571.aspx.

[25] Model-View-Controller. https://blogs.msdn.microsoft.com/ukadc/2010/07/06/model-view/, 2010.

[26] Qt Framework 5.5 Documentation. http://doc.qt.io/qt-5/.

[27] PyQtGraph library Documentation. http://www.pyqtgraph.org/documentation/.

[28] Geomagnetic observatory of Kelčany. http://measure.feld.cvut.cz/groups/maglab/geomagnetic/index.php.

[29] Putty - a free SSH and Telnet client. http://www.putty.org/.

[30] PySide Project Documentation. https://pyside.readthedocs.org/en/latest/.

[31] Python 3.* Documentation. https://docs.python.org/3/.

[32] *MONK, Simon. Raspberry Pi Cookbook. O'Reilly Media, Inc., 2013.*

List of Figures

2.1 Photo of Raspberry Pi 2 Model B from the top [7] 7
2.2 Photo of Raspberry Pi touchscreen Dispay and some other components for dispay's connection [8] 8
2.3 Photo of the observatory magnetometer with periphery [9] 9
2.4 Photo of observatory magnetometer with shown directions of axes [9] . . 9
2.5 Photo of DC/DC converter Mean Well SD-15A-5 [10] 10

3.1 The AMOS Mk 3 is an automated Observatory system deployed [11] . . 11
3.2 Instruments typical of a modern magnetic observatory [11] 12
3.3 Magnetometer sensor with data logger [12] 12
3.4 Data recorder fo 3-component 1-second observatory Magnetometer LEMI-025 [14] 13

4.1 Structure of parallel (A) and orthogonal (B) fluxgates [17] 15
4.2 Orthogonal and non-orthogonal basis 16
4.3 Magnetic field in Prague according to World Magnetic Model [20] 18

5.1 Photo of the device 19
5.2 Schematic of connection device with perpherials 19
5.3 Inner connection of the device. 20
5.4 Devices connectors. 21

6.1 Structure of main program that communicats with sensor 24
6.2 Applience of Gaussian Window [5] 26
6.3 Structure diagram of "saver" process. "Msg" - received message 27
6.4 MVP programming pattern [25] 30
6.5 Structure of process represented data in GUI application 31

7.1 Screenshot of the display with data recorded in a last 12 hours 35
7.2 Photo of the device with recorded data 35
7.3 Photo of the device 36

9. CD contents

```
./
├── logger ............................ Main program
├── GUI ............ GUI program used for visualization
├── stand .............. Program used for presentation
├── scripts ........................... Useful scripts
│   ├── connect ........ Set local network with the device
│   ├── disconnect ............... Discard local settings
│   ├── internet ........................ Share internet
│   └── id_rsa.pub ................... Public key. SSH.
├── data ...................... Example of saved files
│   └── 14052016
│       ├── 07:35.txt
│       ├── 08:00.txt
│       ├── 09:00.txt
│       └── 10:00.txt
└── BP_Albershteyn_2016.pdf ...................... Thesis
```

10. Appendix

List of main functions.

10.1 processor.py

```
def process_data(pipeline, samples, path='./'):
    '''
    Process data from sensor. Accordingly get n samples and
    calculate average value from this samples. Then use
    Gauss window and finally make
    processing.

    Args:
        pipeline: pipeline where from this function will
        receive samples
        samples: number of samples per second
        path: path where we will save our files
    '''

    logger = logging.getLogger(__name__)

    # Data set [[H], [Z], [E], [T]]
    buffersize = 2*samples + 1
    data_set = np.zeros((buffersize, 4))
    # Number of samples
    number_of_samples = 0
    # Get time for every second saving
    firsttime = datetime.datetime.now()

    # Create pipeline for communication with 'saver'
    parent_conn, child_conn = Pipe(True)
    # Create new process for data saver.
    data_saver_proc = Process(target=data_saver,
            args=(child_conn, path))
    data_saver_proc.start()

    # Start main cycle
    try:
        while True:
            # Get data from pipeline
            data = pipeline.recv()

            # Get current time
            currtime = datetime.datetime.now()
```

```
        if data:
            # If array isn't full add new line
            if number_of_samples < buffersize:
                data_set[number_of_samples] = data
                number_of_samples += 1
    else:
        # Otherwise make roll and calculate
        # calibrated values
        data_set = np.roll(data_set, 1, axis=0)
        data_set[0] = data

        mean_value = find_mean(data_set, gauss)
        result = calibrate(mean_value)
        # Send calibrated data to 'saver' process.
        if currtime.microsecond < firsttime.microsecond:
            # Calculate mean value and make calibration
            if config['debug']:
                logger.info(
                "Result is [{:1.9f}, {:1.9f}, {:1.9f}]"
                .format(result[0], result[1], result[2]))
            parent_conn.send(result)

        firsttime = currtime
except KeyboardInterrupt:
    logger.info('Keyboard interrupt in process
    \'processor\'.')
    sys.exit(0)
```

10.2 main.py

```
# Load the logging configuration
logging.config.fileConfig('logging.ini')
logger = logging.getLogger(__name__)

# Path where should be stored all received
# and calculated data
path = config['path']
# Number of samples per second
samples = config['samples']

# Initialize port communication with sensor
# Virtual mode mean that we don't have a real sensor
# and we will just
# generate random numbers. Its mode used for testing.
virtual = False
```

```
if 'virtual' in sys.argv:
    logger.info('Run in virtual mode.')
    virtual = True
    inpoint = Generator()
else:
    logger.info('Run in normal mode.')

    while True:
        try:
            inpoint = init_sensor()
        except KeyboardInterrupt:
            logger.error('Exiting...')
            sys.exit(1)

        # A few reading

        if not readline(inpoint):
            logger.error('Error while reading
            from sensor.  Try to connect
            to sensor.')
            inpoint = init_sensor()
        else:
            break


    readline(inpoint)

logger.info('Sensor successfully initialized.')

# GPIO setup
# This section is for testing
if config['debug']:
    import RPi.GPIO as GPIO

    debug_pin = config['debug_pin']
    GPIO.setmode(GPIO.BCM)
    # Broadcom pin-numbering scheme
    GPIO.setup(debug_pin, GPIO.OUT)
    GPIO.output(debug_pin, GPIO.LOW)

# Create pipeline
logger.info('Creating pipeline.')
parent_conn, child_conn = Pipe(True)

#======================================
# Start another process, which will be
# calculate data and save it to file
#======================================
```

```
logger.info('Creating data processer process.')
data_processor = Process(target=process_data,
        args=(child_conn, samples, path))
data_processor.start()


#=====================================
# Run main cycle
#=====================================
logger.info('Running main cycle.')
try:
    while True:
        # Debug
        if config['debug']:
            GPIO.output(debug_pin, GPIO.LOW)
            time.sleep(0.0001)
            GPIO.output(debug_pin, GPIO.HIGH)
        # Debug end

        try:
            # Get data from the sensor
            data = parse_data_string(inpoint.readline())
            parent_conn.send(data)
            # print('Main is runnign')

        except SerialException as e:
            logger.error('Main cycle: Error
            occured while reading from port.')
            logger.error('Excpetion: ' + str(e))
            inpoint = init_sensor()
            continue
        except ValueError:
            logger.error('Wrong format of data.
            Can\'t parse the string. Contunue.')
            continue


        # Debug
        if config['debug']:
            GPIO.output(debug_pin, GPIO.HIGH)
            time.sleep(0.0001)
            GPIO.output(debug_pin, GPIO.LOW)
        # Debug end

except KeyboardInterrupt:
    if not virtual:
        inpoint.flush()
        inpoint.close()
```

```
        logger.info('Exiting.')
        time.sleep(1)
        if config['debug']:
            GPIO.cleanup()
        sys.exit(0)
```

10.3 saver.py

Part of "saver.py" file.

```
def data_saver(pipeline, path='./'):
    '''
    This function is run as a process and its
    save data getted from pipeline.

    Args:
        pipeline: pipeline
        path: path where to save data
    '''
    logger = logging.getLogger(__name__)

    # Queue between threads
    d_queue = queue.Queue(maxsize=1)

    # Start thread which will send data to socket.
    socket_th = SocketCommunication(d_queue)
    socket_th.start()

    current_time = datetime.datetime.now()
    chours = datetime.datetime.now().hour
    fname = time.strftime(
        config['file_name_format']) + '.txt'

    save_path = path + '/' + \
     datetime.date.today().strftime('%d%m%Y') + '/'
    if not os.path.exists(save_path):
        os.makedirs(save_path)

    if config['debug']:
        logger.info('Path to save ' + str(save_path))

    try:
        while True:
            data = list()
            # Get data
```

```
        data.append(pipeline.recv())

        if not d_queue.full():
            d_queue.put(data)

        #logger.info('Save calibrated data.')
        ctime = datetime.datetime.now()
        if ctime.hour < current_time.hour:
            current_time = ctime
            save_path = path + '/' + \
            datetime.date.today().strftime('%d%m%Y') + '/'
            if not os.path.exists(save_path):
                os.makedirs(save_path)

        if ctime.hour != chours:
            chours = ctime.hour
            fname = datetime.datetime.now().strftime(
            config['file_name_format']) + '.txt'
            if config['debug']:
            logger.info('Create new file with name ' \
                + str(fname))

        if config['debug']:
        logger.info('Complete save path is ' \
            + str(save_path))
        save_data(data, path=save_path, filename=fname)

except KeyboardInterrupt:
    logger.info('Keyboard interrupt
    in process \'saver\'.')
    socket_th.stop()
    socket_th.join()
    sys.exit(0)
```