CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF ELECTRICAL ENGINEERING
DEPARTMENT OF MEASUREMENT

Time-synchronous Datalogger and Graphical Interface for Geomagnetic Observatories

Časově synchronizovaný datalogger a grafické rozhraní pro geomagnetické observatoře

Master's Thesis

Study Programme: Kybernetika a robotika
Study Branch: Senzory a přístrojová technika

Supervisor: Ing. Michal Janošek, Ph.D.

Lukáš Pavelka
Praha 2020

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Pavelka** Jméno: **Lukáš** Osobní číslo: **434882**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra měření**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Senzory a přístrojová technika**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Časově synchronizovaný datalogger a grafické rozhraní pro geomagnetické observatoře

Název diplomové práce anglicky:

Time-synchronous Datalogger and Graphical Interface for Geomagnetic Observatories

Pokyny pro vypracování:

Sestavte na základě existujícího HW časově synchronní datalogger včetně grafického rozhraní pro využití na geomagnetických observatořích / opakovacích stanicích – konkrétně pro stanici POLOM / Dobruška. Využijte platformu Rasberry Pi a databázi (My)SQL.
Data budou ukládána do databázové struktury, ke které bude přistupovat prezentační software s grafickým uživatelským rozhraním (preferenčně implementováno v LabView), komunikace bude realizována formou klient-server po LAN. Implementujte filtraci příchozích dat, jejich přepočet na fyzikální jednotky a označení dat přesnými časovými značkami (s využitím GPS/PPS resp. NTP). Věnujte pozornost zálohování zpracovaných dat a správné funkci zařízení při výpadcích napájení (krátkodobých i dlouhodobých).

Seznam doporučené literatury:

[1] JANKOWSKI, Jerzy; SUCKSDORFF, Christian: Guide for magnetic measurements and observatory practice v4.6. 2012.
[2] OLSEN, Nils, et al.: In-flight calibration methods used for the Oersted mission. 2001.
[3] MONK, Simon: Raspberry Pi Cookbook. O'Reilly Media, Inc., 2013.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Michal Janošek, Ph.D., katedra měření

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **10.01.2019** Termín odevzdání diplomové práce: **07.01.2020**

Platnost zadání diplomové práce: **20.09.2020**

Ing. Michal Janošek, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Datum převzetí zadání

Podpis studenta

Declaration
I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

Prohlášení
Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Říčanech, 6. 1. 2020

Lukáš Pavelka

Acknowledgements

I would like to thank my family for their support and patience.

I am also very grateful to my supervisor Ing. Michal Janošek, Ph.D. for providing valuable advice and being supportive during work on this thesis.

Last but not least, I would like to thank to Professor Thomas Hebbeker.

Poděkování

Rád bych poděkoval své rodině a blízkým za jejich podporu a trpělivost.

Mé díky patří také vedoucímu práce Ing. Michalu Janoškovi, Ph.D. za odborné vedení, věcné připomínky, a velkou vstřícnost při konzultacích.

V neposlední řadě bych také rád poděkoval Prof. Thomasi Hebbekerovi.

Annotation
This thesis describes the development of the time-synchronous datalogger for geomagnetic observatories. The thesis also describes the development of software for data acquisition, processing and visualization. The developed datalogger is based on the Raspberry Pi computer and uses the MySQL database as its main storage medium.
The datalogger implements a filtration of raw data and uses a signal from GNSS/GPS receiver to achieve accurate timestamping. The device is equipped with a backup power system and implements both hardware and software tools to ensure the correct functionality during the short-term and the long-term power outages.
The developed software contains complete solution for datalogging. Two client program for data revision and analysis were also developed as a part of the project.

Keywords
datalogger, time synchronous, GNSS receiver, data processing, software, graphical interface, geomagnetic stations, geomagnetism

Anotace
Tato práce popisuje vývoj časově synchronního dataloggeru pro geomagnetické observatoře a softwaru pro záznam, zpracování a zobrazení geomagnetických dat. Vytvořený datalogger je založen na počítači Raspberry Pi a jako hlavní médium pro ukládání dat využívá MySQL databázi.
Zařízení filtruje příchozí data a s využitím GNSS/GPS přijímače je označuje přesnými časovými značkami. Datalogger je vybaven záložním napájením a hardwarovými a softwarovými nástroji, které zajišťují správnou funkci zařízení při krátkodobých i dlouhodobých výpadcích napájení.
Vytvořený software obsahuje kompletní programové vybavení pro datalogger a také dvě varianty prezentačního programu s grafickým rozhraním pro zobrazení a analýzu dat.

Klíčová slova
datalogger, časově synchronní, GNSS přijímač, zpracování dat, software, prezentační software, geomagnetické observatoře, geomagnetismus

Contents

Chapter 1

Introduction

The measurement of the magnetic field is used in various scientific, industrial and even medical applications. A typical application of the magnetic measurement is the geomagnetism - measurement of Earth's magnetic field. As described in Chapter 2.1 geomagnetic data serve for various global and local needs.

This thesis is focused on the logging and processing of geomagnetic data - the acquisition of data with the help of a magnetometer and magnetic sensor is therefore not covered, only briefly mentioned in Chapter 2.2.3.

Geomagnetic observatories (both permanent and temporary) are often placed in lightly populated locations and used for obtaining long-term magnetic data. Distant locations and large datasets comes with the need for a convenient and reliable device for the data acquisition and data storing (datalogger).

The main purpose of the developed datalogger device is to collect data from a sensor and to store them. It can be also used for various data processing (such as filtration, calibration of sensor offsets, etc.) and to display the actual data to the observatory staff in a graphical user interface.

1.1 Objectives of the Thesis

The goal of this thesis is to develop a time-synchronous datalogger with GUI (graphical user interface) for geomagnetic observatories/repeat stations - specifically for the station Dobruska/Polom.

Developed data-loger is based on previously created dataloggers. These devices were developed for the MAGLAB research group of Measurement Department at the FEE CTU in Prague and were described in bachelor thesis by Andrey Albershteyn [4] and Viktor Fúra [8].

The developed datalogger is based on the Raspberry Pi platform such as device created by Andrey Albershteyn [4] but introduces different solutions for data processing, timestamping, storing and mainly, is precisely synchronized with GPS signal to provide UTC-time-aligned sampling.

In this project, a client-server SQL database is used for the data storing. The de-

veloped system also implements filtering of received data, calculation of value of the magnetic field in physical units and a precise sample rate using GPS. The datalogger is constructed to be able to handle the unexpected power or communications outages (both short-term and long-term ones) and implements a solution for a regular backup of the obtained data.

A client software with GUI (implemented in both MATLAB and LabView) for the data processing and analysis is developed as part of the project. It accesses the database and processes geomagnetic data for visualization and further analysis.

The datalogger is designed to be placed at the geomagnetic station Dobruska/Polom as an alternative logging device to the current PC logger. Stored data will be used for MAGLAB researches and for seismological measurements as well - more detail on Seismological station Dobruska/Polom in Chapter 2.1.3.

The current datalogger at the POLOM station is implemented as a desktop PC with LINUX system and simply saves the 206-Hz ASCII data to files which need to be postprocessed afterwards. These data, because of their size, cannot be kept for more than few years backwards, moreover, the system is not reliable due to its dependency on local power. Also, it is asynchronous, so precise timestamps need to be generated by postprocessing. See Figure 1.1 - current "simple PC logger".

1.1.1 Description of the Developed System

The developed system consist of the Raspberry Pi based datalogger and a client software for data processing and revision. The structure of the system is shown in the Figure 1.1.

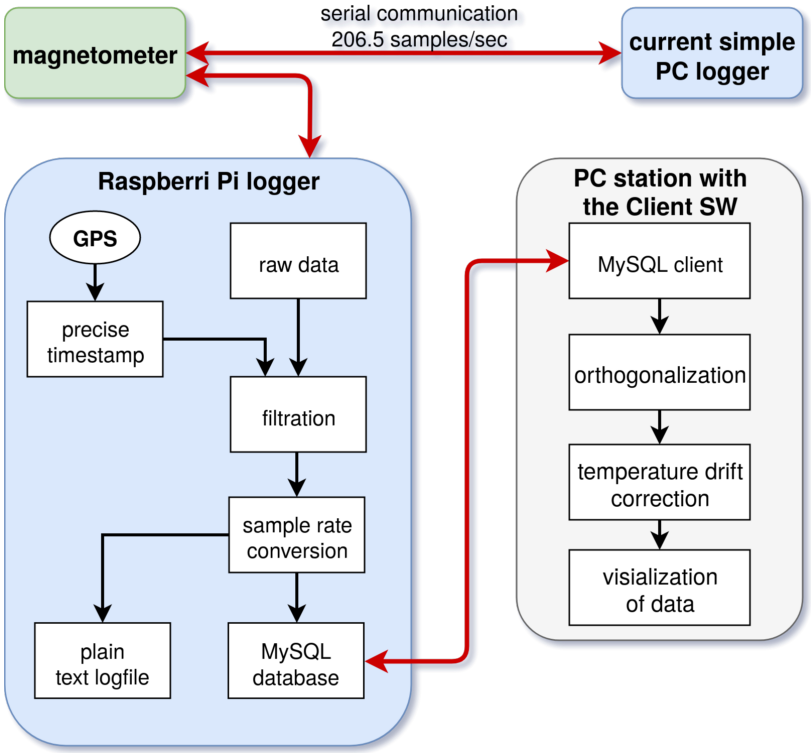

Figure 1.1: Diagram of developed system.

The developed datalogger is connected to the magnetic sensor using RS232 interface as a secondary device (the magnetometer streams data). The device reads raw data (including sensor temperature) and implements data filtration and sample rate conversion (from 206.5 Hz to 10 Hz for txt log files and to 5 Hz for the MySQL database).

Filtered data (but uncorrected in matter of offsets, orthogonalization, etc.) are then stored to the MySQL database and also to the plain text log file. The reason for saving the uncorrected data is to be able to apply different methods of correction during analysis and not to lose any information i.e. by wrongly applied coefficients or further calibrations of the device.

The client GUI software (running on a separate PC station) is connected to the MySQL database and enables to load stored data. Loaded data are then corrected (offset and gain correction, orthogonalization, temperature drift correction) and visualized. The client software serves for the data revision and a further analysis.

1.2 Available Solutions

As already mentioned, the developed datalogger shares hardware and software similarities with the datalogger [4] and further improves used technologies and their implementation.

Besides dataloggers developed by MAGLAB ([4] and [8]) there are third-party solutions as well. This chapter describes logging devices with parameters similar to the developed datalogger and their differences.

1.2.1 Data Recorder for Observatory Magnetometer

The device developed by Andrey Albershteyn [4] (Figure 1.2) is based on Raspberry Pi and has serial (RS232) interface for magnetic sensor. Device implements data processing and filtration (using Gaussian filter).

In comparison with the developed datalogger it uses only a plain text files to store data instead of SQL database. It does not implement precise time stamping of measured data and although it contains a backup battery, it is does not implement any hardware or software methods to securely handle the long-term power outages.

1.2.2 Low-power Datalogger for 1s INTERMAGNET Data

An open source datalogger based on Raspberry Pi computed developed by Achim Morschhauser et. al. from GFZ German Research Center for Geosciences (Postam, Germany) [6] shares similarities with A. Albershteyn's datalogger and with the developed datalogger.

The device (Figure 1.3) aims to offer low-power device with emphasis on modularity and flexibility. Data-logging software is implemented in C++ using object-oriented approach. Developing team is currently implementing precise timestamping.

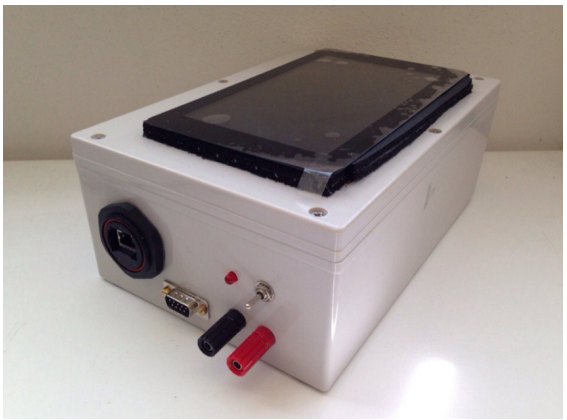

Figure 1.2: MagLab datalogger developed by A. Albershteyn; source: [4].

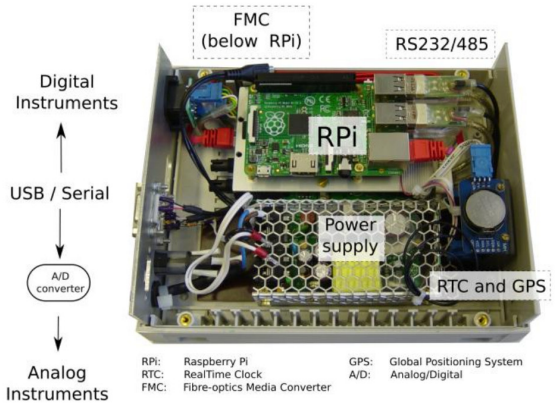

Figure 1.3: Low-pawer datalogger based on Raspberry Pi computer; source: [6].

The main advantage of the developed datalogger over the Morschhauser's device is the use of the MySQL database for the data storing and its sample rate variability

(Morschhauser's device is designed to provide only 1 Hz data). The Morschhauser's device also does not implement backup battery as a part of the datalogger and it does not implement any methods to safely handle power failures [7].

1.2.3 Portable Datalogger for Vector Magnetometer

The device was developed by Viktor Fúra and described in [8]. This device (Figure 1.4) was built with emphasis on portability and durability. Device contains GPS module for precise timestamping but it does not implement any data processing. Data are stored only in plain text file on a SD card.

This device is a portable datalogger for short-term measurements and it's not designed to be an observatory datalogger such as the developed device.

Figure 1.4: Portable datalogger developed by V.Fúra; source: [8].

Figure 1.5: Avisaro datalogger with RS232 interface; source: [12].

1.2.4 Commercial Solutions

The main reason for researchers developing their own devices for magnetic data logging is the lack of commercial solutions with suitable parameters and flexibility. Common commercial solution often lacks data processing (such as filtering), precise timestamping, portability or possibility of customization.

As an example of portable commercial logging device using RS232 interface we can use Avisaro Data Logger Box 2.0 [12] - simple portable logger for recording RS232 data on a SD card (Figure 1.5).

The Avisaro datalogger is only a simple device for data recording. It is not designed to be placed at geomagnetic observatory as a datalogger system. It does not implement any power backup system and it lacks precise sample rate support.

Chapter 2

Theory

In this part of the thesis the theoretical principles and used methods are described. This chapter also briefly describes history of geomagnetic research and the purpose of geomagnetic observatories.

2.1 Geomagnetism

History of geomagnetism and its practical use is very long as is described in IAGA Guide [1]. The first definite mention of usage of a simple compass (magnetized steel needle) in Europe comes from the year 1190. But at the same time the compass was probably already commonly used for the navigation in China.

In the 15th century the magnetic declination (an angle on the horizontal plane between the magnetic north and the geodetic north) was described. Discovery of the magnetic inclination (an angle between the Earth's magnetic field lines and the horizontal plane) followed in 1600.

In the 17th and 18th century the continuing research of the Earth's magnetic field alongside with the the discovery of new measurement methods (e.g. Coulomb's relative method allowing to measure intensity of the horizontal component of the magnetic field) led to the discovery that Earth's magnetic field varies not only with place but also with the time.

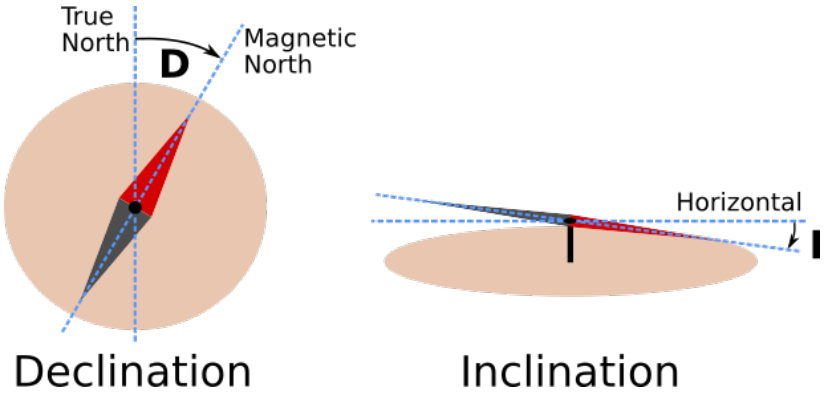

Figure 2.1: Illustration of the magnetic declination and inclination.

At the beginning of the 19th century Alexander von Humboldt laid the foundation of modern geomagnetism with his long-term measurements. He initiated a building of magnetic observatories all over the world.

Nowadays there is over 150 INTERMAGNET (the global network of observatories monitoring the Earth's magnetic field) permanent observatories and numerous of temporary or private geomagnetic stations serving both science and commercial uses.

2.1.1 Geomagnetic Field

The magnetic field of the Earth is a vector field and it is observable all over the globe, see Chapter 2. in IAGA Guide [1]. According to the Hydromagnetic Dynamo Theory [13], the main part of field is created by electrokinetic currents running close to the surface of the Earth's outer liquid core (at the depth about 2 900 km). The field on the Earth's surface varies at the different locations - at the equator the field's value is about 30 μT and about 60 μT in the polar areas.

The slow changes of the geomagnetic field (changes over periods of a year or more) are called *secular variations*. As is described in [1] these variations have dimensions of thousands of kilometers and are caused mainly by the inhomogeneity of the electric currents inside the Earth.

The smaller anomalies of geomagnetic field are mostly of crustal origin (the crust is the outermost solid shell of the Earth). Crustal anomalies can have dimension from hundred of kilometers to few meters and are caused by remanent magnetization, boulders containing big amount of magnetite (rock mineral - one of the main iron ores), etc.

Short-term variations and anomalies are mostly caused by the particle radiation from the Sun. Interaction of the solar wind (the flow of the particles from Sun) with the Earth's magnetic field creates magnetospheric and ionospheric currents and leads to magnetic storms at the Earth's surface.

Last common type of geomagnetic field variations are regular variations. These variations are related to the rotation and the orbital movement of the Earth, Moon and most importantly the Sun. As is described in [1] there are two important periodic variations: solar daily variation and lunar daily variation (the lunar magnetic effect is significantly smaller than the solar magnetic effect). The regular solar daily variation is also a function of the geomagnetic latitude, solar activity and the time of the year.

2.1.2 Geomagnetic Observatories

As was described, the main purpose of geomagnetic observatories is to monitor the variations of the geomagnetic field. The observatories are typically placed in low populated areas without large magnetic anomalies and far enough from the man-made magnetic disturbances.
In the Czech Republic there is one official INTERMAGNET observatory in the southern Bohemia near Budkov [14].

Geomagnetic data are used to study the geological structure of the Earth's crust and core. There are also many applications of geomagnetism in geology, seismology.

The precise geomagnetic data are also needed for research into current systems in the ionosphere and magnetosphere.

Observatories fulfill global and local needs for geomagnetic data. In the global scale, it is important to monitor secular variations and magnetic storms that affect large areas. From a local point of view geomagnetic observatories are important for measurement of local variations because local magnetic anomalies can lead to communicational interferences and even power failures. Geomagnetic observatories also often serve as facilities for calibration of magnetic instruments.

2.1.3 Geomagnetic Station Dobruska/Polom

The geomagnetic station Dobruska/Polom is part of the measurement station Polom located in the Orlické hory Mountains. Station was founded in 1974 in the area of the Military Geodetical Institute at Polom and its main task was to monitor seismic activity for military purposes [16].

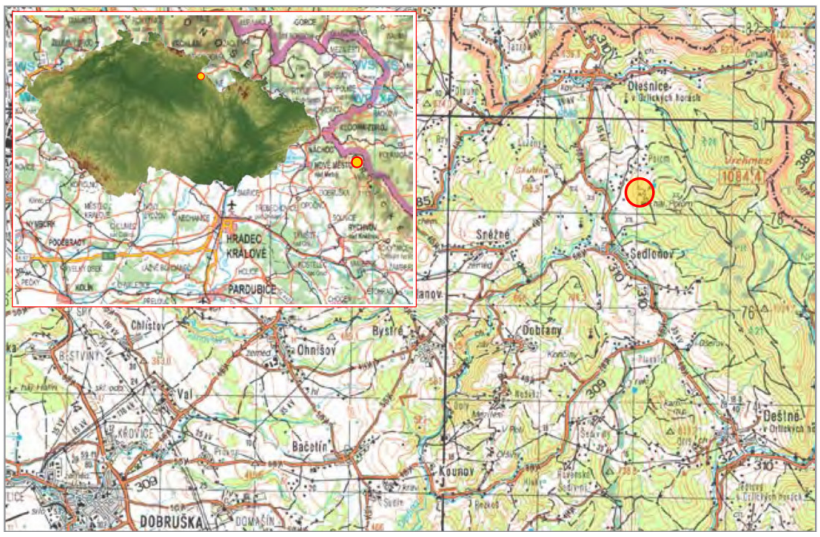

Figure 2.2: Location of the Dobruska/Polom station, source: [16].

Nowadays Polom station still serves as a military measurement station but it is also a place of multiple collaborations with scientific institutes such as Institute of Geophysics of the Czech Academy of Science or Czech Hydrometeorological Institute [16].

Seismological station Dobruska/Polom (DPC) is still the most important part of the Polom station. The station DPC is equipped with modern seismometers and provides very precise seismic data thanks to very low level of the seismic noise in the area. The station is also part of the Federation of Digital Seismic Network - network of about 200 selected high-quality broadband stations used among others for fast location of potentially damaging earthquakes [15].

Polom station also serves as a place for geodetic, geophysical (Polom is used as a GPS reference station) and hydro-meteorological measurements (in cooperation with the Czech Hydrometeorological Institute). Even an astronomical observations of extremely bright meteors are conducted at the station by automatic bolide camera (in

cooperation with the Astronomical Institute of the Czech Academy of Sciences) [16].

Figure 2.3: Measurement station at the Dobruska/Polom, source: [16].

Geomagnetic station Dobruska/Polom is a cooperation between the MAGLAB research group and the Institute of Geophysics of the Czech Academy of Science. Local geomagnetic data are used for research of magnetism conducted by the MAGLAB and the Institute of Geophysics. Local geomagnetic data are used for the research of magnetism and also as a referential data for the elimination of a geomagnetic reference in a seismic data.

Figure 2.4: Bolide camera installed at the DPC station, source: [16].

Figure 2.5: Geomagnetic station at the Dobruska/Polom.

In the Figure 2.5 we can see the sensor head in a non-magnetic hut, surrounded by white bricks for basic passive temperature stabilization.

2.2 Instruments for Measurement of the Geomagnetic Field

Magnetometers used to measure the Earth's magnetic filed fall into two categories [11]: instruments for measuring the temporal changes in the field (variometers) and instruments for measuring the absolute value of the magnetic field at a specific time instant

In the last two centuries, there have been a rapid development of the instruments for magnetic measurement. Classical magnetometers based on the observation of the magnet in the changing magnetic field were replaced by modern instruments such as fluxgates magnetometers. Currently the most commonly used magnetometers in modern magnetic observatories are the triaxial fluxgate variometer and the declination-inclination magnetometer in conjunction with an Overhauser magnetometer for absolute observations.

This chapter aims to briefly describe the most significant historical instruments and the most widely used instruments in modern geomagnetic observatories.
Variometer installed at the Dobruska/Polom observatory is of fluxgate type and is described in the Chapter 2.2.3.

2.2.1 Historical Magnetometers

The torsion magnetometers are a classical magnetometers commonly used since the 19th century. As is described in [1] on page 54, torsion magnetometers are based on the observation of a magnet in a changing magnetic field. The magnet is suspended by a thin fibre and the torque of the magnetic field on the suspended magnet is compensated by the torque of the suspension fibre.

2.2.1.1 Declinometer

The Declinometer is a classical instrument for declination measurement. This instrument employs a magnet suspended from a long torsionless fibre so it is free to align itself in the direction of the horizontal magnetic field (as described in [11] on page 26). The system with the magnet is placed on top of a non-magnetic theodolite.

The principle of a declination measurement is quite simple as is shown in the Figure 2.6. To calculate the declination D, we need to take the theodolite reading A of magnetic meridian and the theodolite reading B of an azimuth mark with known azimuth A_Z: $D = A - (B - A_Z)$.

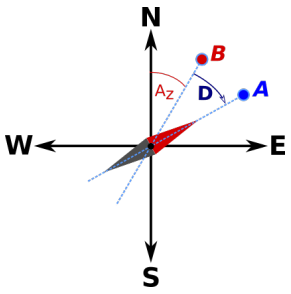

Figure 2.6: Theoretical principle of the declinometer.

2.2.1.2 Torsion Variometers

As is described in [1] in Chapter 4.4, torsion magnetometers with suspended and balanced magnets were the most common type of variometer for more than a hundred years.

The main principle is shown in the Figure 2.7: the light beam from the lamp (placed at the focal distance of the lens L) goes through the lens L to the variometer mirror VM (attached to the suspended magnet), reflects from there back through the lens L and a cylindrical lens CL and focuses at the photo-paper wrapped around rotating drum D where it forms a sharp spot. The light spot draws the magnetogram at the moving photo-paper. The base-line of the magnetogram is produced by another light beam reflecting from a fixed mirror FM and drawing a straight line on the photo-paper.

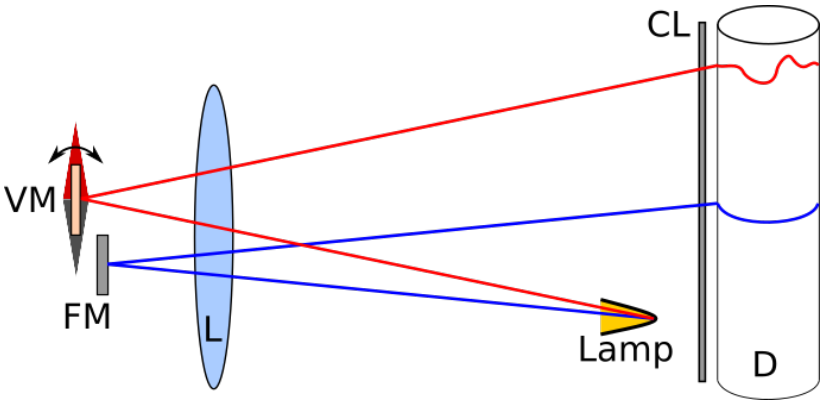

Figure 2.7: Main principle of the torsion variometer. Source: [1]

The analog form of recording was the main shortcoming of the classical torsion variometer but the introduction of the phototransformers helped to overcome this difficulty.

The Torsion Photoelectric Magnetometer (TPM) uses a classical suspended magnet system as a detector but it implements phototransformer as a recorder as is described in [11] in Chapter 4.7.3. The light beam is reflected onto a pair of fototransformers which transform the angle of deviation into a voltage. The voltage output is then amplified and also fed to a negative feedback winding which acts to keep the mirror stationary. The current in the negative feedback circuit is a measure of the strength of the magnetic field component [11].

2.2.2 Modern Instruments

In this chapter, the most common modern instruments for measuring magnetic variations and absolute value of the magnetic field are described.

2.2.2.1 Fluxgate Magnetometers

The fluxgate mechanism is based on the non-linear characteristics of ferromagnetic materials in the sensing elements of the instrument. All fluxgate sensors uses an easily saturable core made of a material with high permeability. Around the core there are two windings: an excitation coil and a pick-up coil.

The fluxgate sensor principle will be described for a sensor with a linear twin core (Figure 2.8) which is the most common type of fluxgate. Described function applies also for a ring core fluxgate sensor and for race-track fluxgate sensor (Figure 2.9).

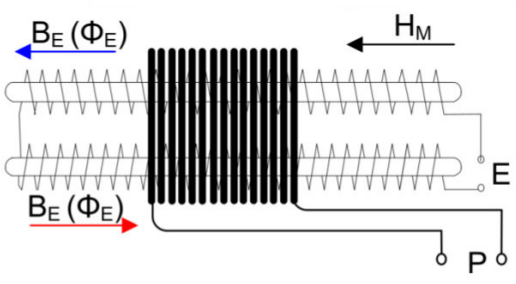

Figure 2.8: Twin core fluxgate sensor schema. Source: [26].

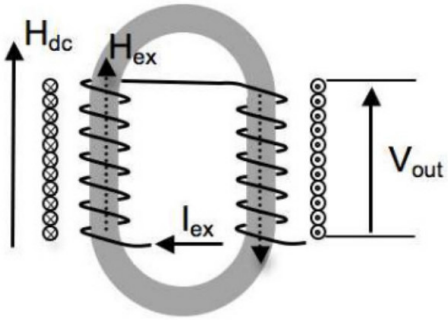

Figure 2.9: Race-track fluxgate sensor. Source: [25]

As is shown in the Figure 2.8 the excitation winding of the sensor is placed around each core. The cores are wound so they are excited in opposite directions which means that the induced voltage produced by the excitation winding is canceled by the phase reversal. The output signal is obtained from a pick-up winding encircling both cores.

As is described in [11] Chapter 4.2.2, when the excitation current I is zero, the core is not saturated and its relative permeability μ_r is maximum. The core concentrates the the ambient magnetic field to be measured, producing a magnetic flux. When the current I is fed into the excitation winding it creates a magnetic field, the core is saturated. It causes the drop of the core permeability (close to permeability of vacuum) and collapse of the flux.

The core is saturated alternatively in opposing directions by sine or square excitation signal (or other shapes for power saving). After the collapse, the flux recovers in the next half cycle of excitation waveform and it is once again at a high level until the core is saturated in the opposite direction.

The pick-up coil detect these flux changes. In the absence of an external magnetic field, the saturations are symmetrical and the output signal has only odd harmonics. The presence of an external magnetic filed (to be measured) disturbs this symmetry and

causes creating even harmonics with a dominant second harmonic [11]. The pick-up coil will detect even harmonics in output signal (odd harmonics are canceled thanks to the phase reversal of the opposite winding of the excitation coils) which are a measure of the measured magnetic field.
The triaxial fluxgate sensor employs three fluxgate sensors orthogonal to each other which allows to measure vector value of magnetic field.

The fluxgate sensors are currently the most common type of magnetic variometers. The fluxgate sensor can be also employed to conduct an absolute magnetic measurements. The instrument called Declination-Inclination Magnetometer (DIM) consist of the fluxgate sensor mounted on top of the non-magnetic theodolite and is commonly used to measure magnetic declination and inclination. The principle of measurement is described in [11] in Chapter 4.3.

2.2.2.2 Overhauser Magnetometer

The Overhauser magnetometer is a proton magnetometer with high sensitivity, lower power consumption and no DC polarization [11]. It is a very common instrument for absolute measurement of the magnetic field.

Function of a proton magnetometer is based on the principle that protons are spinning on an axis aligned with the magnetic field. Standard proton precision magnetometer uses a solenoid to create a strong magnetic field around a fluid rich in hydrogen atoms. That causes that protons contained in the fluid are aligned in the direction of the field. When the field is interrupted, protons start to realign themselves with am ambient field. During the realignment protons precess at an angular frequency ω directly proportional to the magnetic intensity of ambient B field [11]: $\omega = \gamma_p B$, where γ_p is a gyromagnetic constant.

Magnetic dipoles are produced by the spin of charged particles precessing around the magnetic field direction [11]. Created magnetic field is detected and the value of magnetic intensity B is determined.

The *Overhauser magnetometer* uses free radical fluid and it aligns protons with a low power radio-frequency (RF) field instead of a magnetic field created by solenoid. The RF polarization also allows for concurrent measurement with maximum practical rate of 5 readings per second [11] (Chapter 4.4.3 Overhauser magnetometers).

Figure 2.10: High precision Overhauser magnetometer GEM GSM-19. Source: [24]

2.2.3 Magnetometer at the Dobruska/Polom Station

The magnetometer installed at the Dobruska/Polom station was manufactured at the CTU using low-noise high-stability fluxgate sensors [5]. As is described in [19] race-track fluxgate sensors with planar oval core (cut by pico-second UC-laser) were used. The triaxial sensor head is made from MACOR machinable ceramics and it is further fixed on a marble plate.

The sensor at the Dobruska/Polom station is oriented in "UVZ" orientation: the two horizontal axes are oriented $\pm 45°$ from the local meridian. This means that both horizontal axes are measuring roughly the same magnetic field [5].

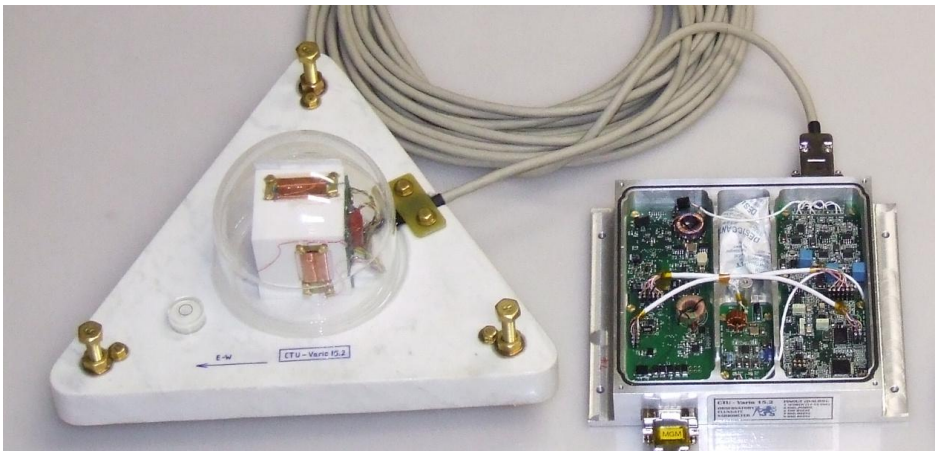

Figure 2.11: Fluxgate variometer installed at Dobruska/Polom station. Source: [19].

The electronics uses 24-bit 200 Hz A/D converter with simultaneous sampling and all digital processing is done in FPGA. The noise performance of the complete measurement system is below $6\,\mathrm{pT}/\sqrt{\mathrm{Hz}}$ @; 1Hz [19]. FPGA also serves for streaming the data to the serial line.

2.3 Sample Rate Conversion and Low-Pass Filtering

The sample rate conversion and filtering is an important part of the magnetic data processing. It is common to sample magnetic data with fast rate and then use a digital filter to improve the signal and to convert the sample rate. With a use of suitable low-pass filter we can achieve an improvement of signal resolution, decrease of the bandwidth and also the noise and the aliasing reduction. Analog filters should be used at minimum, because they will always add instability, noise and temperature dependence of their transfer function.

The commonly used low-pass filters are described in following chapters. Considered filters are compared in Chapter 2.3.5 and selection of the most suitable filter is discussed.

2.3.1 Successive Sample Averaging with Decimation

The Successive Sample Averaging (also called Boxcar filtering or Moving Average filter) is a basic low-pass filter. It provides the noise reduction and can be also used for a signal decimation. As is described in [10] in the Successive Sample Averaging (SAS) implementation every output sample represents an average value of M consecutive input samples.

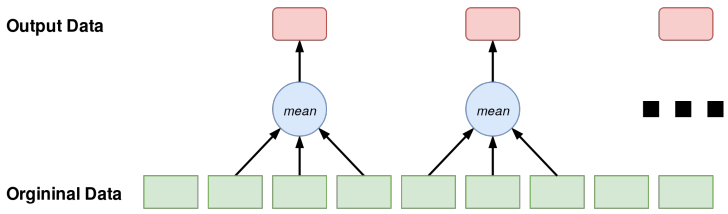

Figure 2.12: SAS filter with decimation.

The frequency characteristic of Successive Sample Averaging have very sharp nulls at the frequency $\frac{\text{sample rate}}{\text{number of averaged samples}}$ and its harmonics - as is shown in the Figure 2.13.

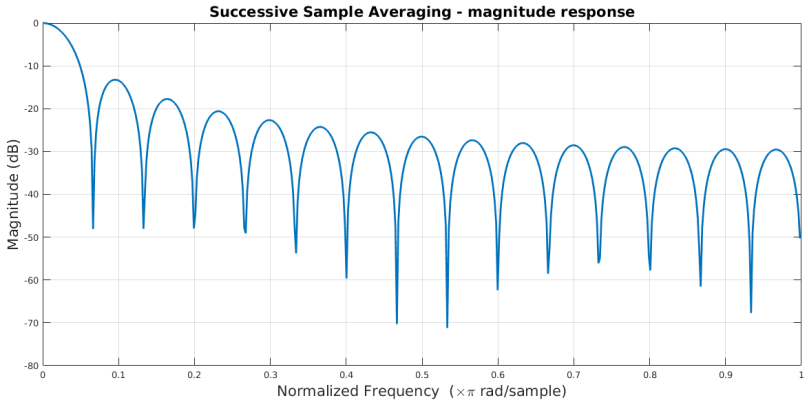

Figure 2.13: Frequency characteristic of Successive Sample Averaging.

2.3.2 Gaussian Low-Pass Filter

The Gaussian filter is a common low-pass filter for magnetic data processing and is even recommended by INTERMAGNET. In the principle, it is similar to the Successive Sample Averaging. The main difference is that Gaussian filter uses multiplication of input samples with coefficients given by Gaussian distribution function, instead of assigning the same weight for every input sample as the Successive Sample Averaging does (basically the Successive Sample Averaging is Gaussian filter with all coefficients equal to 1.0). In turn, this filter has a sharper roll-off.

As is explained in IAGA Guide [1], *7.1 Sapmpling rate* (pages 139-141) the Gaussian filter can be described with the Fourier transform pair of functions $f(t)$ and $F(\omega)$.

$$f(t) = e^{\left(-\frac{t^2}{2 \cdot \tau^2}\right)} \tag{2.1}$$

$$F(\omega) = \tau \cdot e^{\left(-\frac{\omega^2 \tau^2}{2}\right)} \tag{2.2}$$

Functions $f(t)$ and $F(\omega)$ have the shape of the Gaussian distribution.

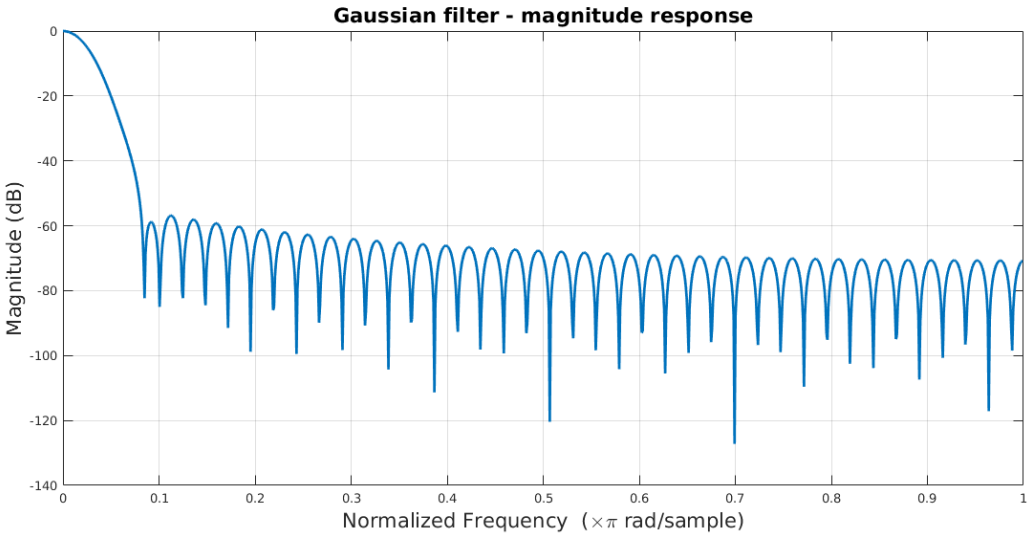

Figure 2.14: Magnitude response of a Gaussian filter.

The Gaussian filter provides better rejection in stop band, wider transition band and higher steepness of transition band comparing to Successive Sample Averaging.

2.3.3 Low-pas FIR filter

The Finite Impulse Response (FIR) filter is a digital filter with finite duration of its impulse response. The FIR filter difference equation is defined as:

$$y[n] = \sum_{i=0}^{N} b_i \cdot x[n-i] \tag{2.3}$$

where N is number of filter coefficients, $x[n-i]$ is the input delayed by i samples, b_i is the i-th coefficient of the filter and $y[n]$ is the filtered output at discrete time n.

The output of the FIR filter depends only on the input values which leads to its finite impulse response [17].

FIR filters are simple to design and are guaranteed to be BIBO (bounded input - bounded output) stable. One of the main advantages of a FIR filter is its constant delay on all frequencies. The group delay τ of a FIR filter with symmetrical coefficients is defined as:

$$\tau = \frac{N-1}{2} \tag{2.4}$$

where N is number of filter coefficients and τ is a delay of the signal in samples.

In the case of FIR filter with 276 coefficients (selected FIR filter) the delay of signal is 137.5 samples.

The main disadvantage of a FIR filter is a need for high order of the filter when the steepness of transition band is required.

2.3.4 Low-pas IIR filter

Unlike the FIR filter, the output of an Infinite Impulse Response (IIR) depends on both input and output values. The feedback loop of the filter is responsible for the infinite impulse response [17]. The general equation for an IIR filter is:

$$y[n] = \frac{1}{a_0}\left(\sum_{i=0}^{P} b_i \cdot x[n-i] - \sum_{j=1}^{Q} a_j \cdot y[n-j]\right) \tag{2.5}$$

where P is the number feedforward coefficients of the filter, $x[n-i]$ is the input delayed by i samples, b_i is the i-th feedforward coefficient of the filter, Q is the number of feedback coefficients, $y[n-j]$ is the output of the filter delayed by j samples, a_j is the j-th feedback coefficient of the filter and $y[n]$ is the filtered output at discrete time n.
In the most IIR filter designs the coefficient a_0 is equal to 1 ($a_0 = 1$).

An IIR filter is commonly used for high-speed applications because it requires significantly lower number of coefficients (which means fewer multiplication operations) than FIR filter of similar filtering properties.
The main disadvantage of an IIR filter is its nonlinear phase. This leads to the different delay for each frequency component. Also, the filter stability is not not guaranteed (as shown in the Figure 2.15.)

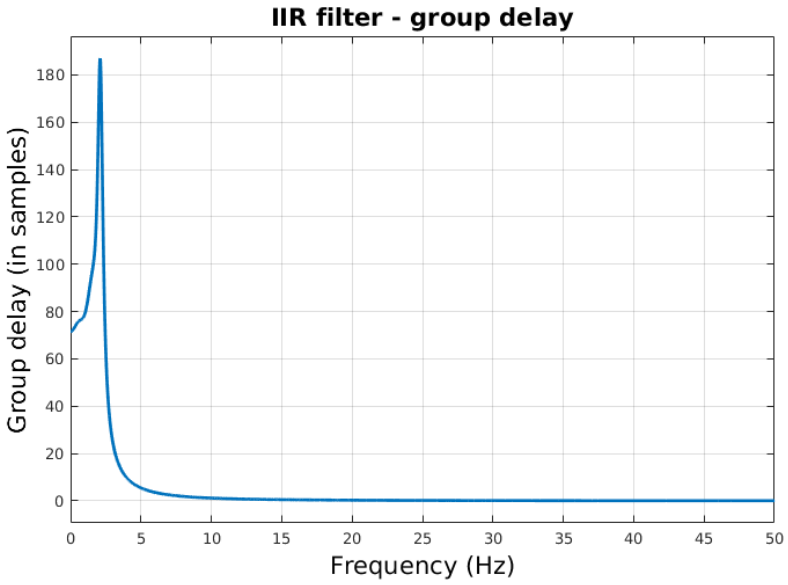

Figure 2.15: Example of a group delay of an IIR filter.

The group delay τ of an IIR filter is defined as [18]:

$$\tau(\Omega) = -\frac{d(\Phi(\Omega))}{d\Omega} \tag{2.6}$$

where Ω is the frequency, $\Phi(\Omega)$ is the phase spectrum (defined as a normalized continuous function of Ω) and τ is the delay of signal in samples.
There are some options for IIR filter delay compensation[18] but the compensation is never full and therefore it is not very suitable for "on the fly" filtering.

2.3.5 Filter Selection

As already mentioned, this project requires using a low-pass filter for data smoothing and noise reduction.

The primary purpose of the developed datalogger is the processing and storing of the geomagnetic data.
Most of the geomagnetic field variations have very low frequencies (lower than 1 Hz) and geomagnetic signal often suffers from a man-made magnetic noise (such as 50 Hz noise caused by the power distribution system).

Based on the data from long-term measurements, geomagnetic signal in Dobruska/Polom station suffers from the strong white noise. It is mainly caused by the full-filed configuration, which is implemented at the time instead of a true variometer - the A/D converter dominates the noise floor, and leads to flat spectrum of recorded signal. Long-term geomagnetic signal from Dobruska/Polom and its spectrum are shown in the Figure 2.16.

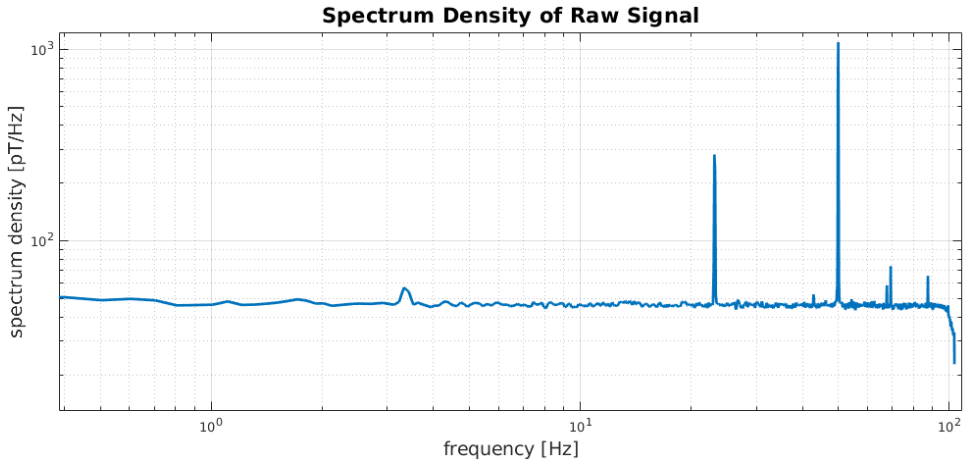

Figure 2.16: Spectrum density of the long-term geomagnetic data from Dobruska/Polom (March 2018).

Based on the long-term geomagnetic data, the required properties of the low-pass filter were set as follows:

Required parameters:

1. steep roll-off
2. cut-off frequency at circa 3 Hz
3. efficient filtration of the 50 Hz noise
4. predictable delay of the filter
5. reasonable processing power requirements

Predictable delay of the filter is required to achieve a precise sample rate: with use of the GPS it is necessary to be able to determine the delay of the signal so it can be

compensated and timestamps are synchronized.

The processing power is an important filter parameter because of HW limitation of the Raspberry Pi computer - data processing will be running "on the fly" and each filter iteration have to be finished before the arrival of the new data sample.

To select the most suitable low-pass filter a series of Matlab simulations with real long-term Dobruska/Polom data was conducted.

2.3.5.1 Tested Filters

At the first round of the filter selection following low-pass filters were compared: Successive Sample Averaging (theory in Chapter 2.3.1), Gaussian filter (theory in Chapter 2.3.2), FIR filter (theory in Chapter 2.3.3) and IIR filter (theory in Chapter 2.3.4).

Tested **Successive Sample Averaging filter** has following parameters:

Parameter	**Value**
Output sample rate	10 Hz
Number of averaged samples	30

Table 2.1: SAS filter parameters.

The **Gaussian filter** was described by following parameters:

Parameter	**Value**
Output sample rate	10 Hz
Cut-off frequency	2 Hz

Table 2.2: Gaussian filter parameters.

Coefficients of Gaussian filter were computed in MATLAB with use of following script:

```
function [Ci]=gaussCoeffs(inputPeriod,outputPeriod,cutOfFreq)
%Generate Gaussian Filter Coefficients
% inputs:
%   input period ... original sample period [sec]
%   output period ... sample period after filtering [sec]
%   cut-off frequency ... cut-off point (-3 dB) [Hz]
% output:
%   C ... full array of Gaussian coefficients

    dt = inputPeriod;           %original sample period
    valuesTime = outputPeriod;  %data period after filtering
    omega = 2*pi*cutOfFreq;
    tau = 0.83255461/omega;
    tmax = sqrt(-2*log(0.01)*tau^2);
```

```
    t = 0:dt:floor(tmax/dt)*dt;
    f = exp(-((t/tau).^2)./2);
    imax = length(f)-1;
    k = f(1)+2*sum(f(2:end));
    C = (1/k).*f;
    Ci = [flip(C(2:end)),C];
end
```

Tested **IIR filter** can be re-created with MATLAB *designfilt* function:

```
iirFilt = designfilt('lowpassiir','FilterOrder',6, ...
          'PassbandFrequency',2,'PassbandRipple',0.1, ...
          'SampleRate',206.5);
```

The IIR filter than have following parameters:

Parameter	**Value**
Number of filter coefficients	7
Cut-off frequency (-3 dB)	2.2 Hz
Passband frequency	2 Hz
Passband ripple	0.1
Attenuation of 50 Hz noise	-190 dB
Filter delay	frequency dependent

Table 2.3: IIR filter parameters.

The last of tested filters was the **FIR filter**. It can be re-created with MATLAB *designfilt* function:

```
firFilt=designfilt('lowpassfir','PassbandFrequency',2*2/206.5,...
          'StopbandFrequency',4*2/206.5,'PassbandRipple',0.3, ...
          'StopbandAttenuation',70,'DesignMethod','equiripple');
```

The FIR filter had following parameters:

Parameter	**Value**
Number of filter coefficients	276
Cut-off frequency (-3 dB)	2.5 Hz
Passband frequency	2 Hz
Stopband frequency	4 Hz
Passband ripple	0.3
Attenuation of 50 Hz noise	-80 dB
Filter delay	137.5 samples

Table 2.4: FIR filter parameters.

2.3.5.2 Filter Comparison

The Successive Sample Averaging as a simple form of low-pass filter is not efficient enough for purposes of this project as can be seen in comparison of the spectral density

of filtered signal (Figure 2.18).

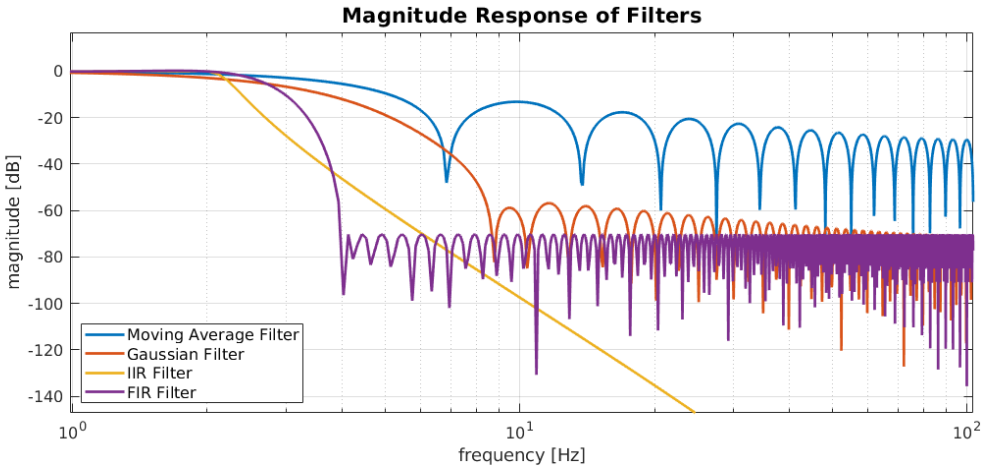

Figure 2.17: Filter comparison - magnitude response.

Even though required processing power is very low and the filter has only a small delay, SAS filter is not suitable for this project. As can be seen in the Figure 2.17 the magnitude response of the SAS filter does not meet the demands set by project requirements.

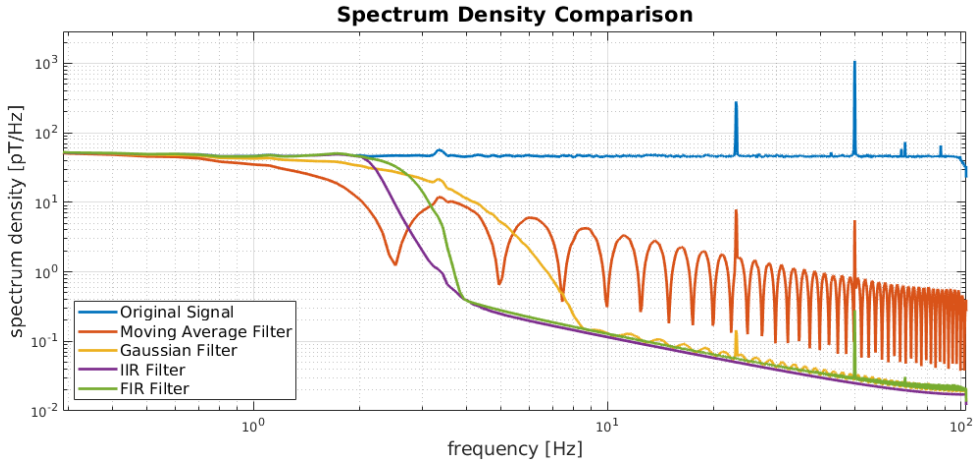

Figure 2.18: Filter comparison - spectral density of signal.

The Gaussian filter's biggest advantages are its constant delay and low number of coefficients. As was described in Chapter 2.3.2 it is very common filter for the processing of magnetic data. Despite its qualities the Gaussian filter is not suitable for this project: because of its gradual roll-off. As is shown in Figure 2.17 and in Table 2.5, although cut-off frequencies of Gaussian, IIR and FIR filters are similar, roll-off of the Gaussian filter is not steep enough.

The Gaussian filter is very useful in processing of signal with $1/f$ attenuation of the specter where its effective steepness is increased. As is shown in the Figure 2.16, typical geomagnetic data that will be processed by the developed datalogger has very flat spectrum without $1/f$ attenuation.

Parameter	Gauss Filter	IIR filter	FIR filter
Number of filter coefficients	83	7	276
Cut-off frequency (-3 dB)	2 Hz	2.2 Hz	2.5 Hz
Passband frequency	-	2 Hz	2 Hz
Stopband frequency	-	-	4 Hz
Passband ripple	-	0.1	0.3
Attenuation of 50 Hz noise	-80 dB	-190 dB	-80 dB
Filter delay	constant	frequency dependent	137.5 samples

Table 2.5: Comparison of the Gaussian, the IIR and the FIR filter with similar filtration abilities.

After elimination of the SAS and the Gaussian filter as not suitable enough it was necessary to decide between IIR and FIR filter and select the most suitable filter for thesis requirements.

The IIR filter seems to be more suitable than the FIR filter. As is shown in Figure 2.17 it has steeper roll-off and greater signal attenuation in lower frequencies. And although the feedback loop of IIR filter requires more processing power than strictly feedforward loop of same number of coefficients, the extremely low number of IIR filter coefficients (almost 40 times less than FIR filter of similar filtering quality) results into very low processing power requirements.

Despite all of its qualities the main disadvantage of the IIR filter is its delay. As was described in Chapter 2.3.4 the group delay of IIR filter depends on the signal spectrum and it is not constant nor predictable.

Although the spectrum of geomagnetic signal from Dobruska/Polom seems to be very stable through long-term measurements, assumption of constant delay (even based on the long-term measurements) would lead to phase distortion of higher-frequency components of interest, which would then render the recorded data useless for comparison with other stations.

This project aims to deliver precise sample rate of the filtered data with the use of GPS module. In the light of this goal the IIR filter is not suitable.

After thorough discussion the FIR filter was selected as the most suitable filter for data processing. Magnitude response of selected filter is shown in the Figure 2.19 and its parameters are stated in Table 2.5. The selected filter's roll-off is not as steep as the IIR filter's but the FIR filter is still efficient enough to process signal data with required quality.

The delay of the FIR filter is constant (137.5 samples) which will enable to implement precise sample rate.

The number of FIR filter coefficients was determined as compromise between filter roll-off and processing power requirements. The selected FIR filter was tested and it is able to process geomagnetic data "on the fly" without affecting the datalogger performance.

List of parameters selected FIR filter can be found in the Chapter 2.3.5.1 in the Table 2.4.

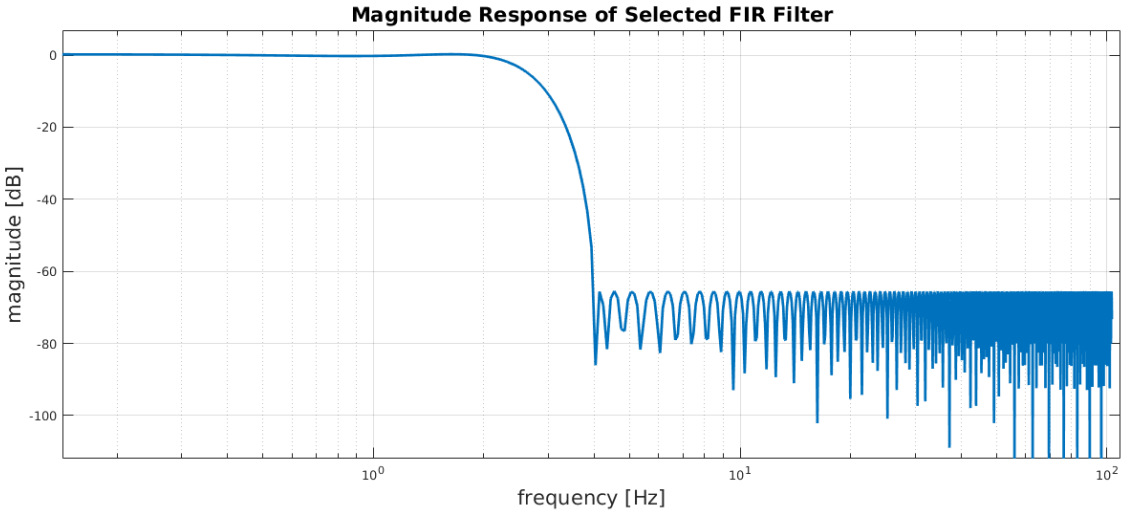

Figure 2.19: Magnitude response of the selected FIR filter.

2.4 Data Correction based on Sensor Calibration

As described in the Chapter 1.1.1 recorded raw geomagnetic data from the magnetometer are filtered (signal filtering is described in the Chapter 2.3) and then stored into the database (the database structure is described in the Chapter 3.2.1) without any data correction. The reason to store uncorrected data is to be able to work with original data and to change parameters of correction in the view of the needs of various experiments.

The data correction (such as orthogonalization and temperature drift compensation) is therefore implemented in the Client software (Chapter 3.3). In this chapter, implemented methods of data correction are described.

2.4.1 Orthogonalization, Offset and Sensitivity Correction

The triaxial fluxgate sensor consist of 3 sensors that are orthogonal in theory. In the case of real sensor that is not true because the sensors are not perfectly located. To transform non-orthogonal coordinate system of sensor to orthogonal coordinate system the transformation called orthogonalization is needed.

There are other necessary corrections and transformations as well. The triaxial fluxgate sensor suffers from offset that needs to be compensated. The magnetometer output is also in engineering units ([EU]) so it needs to be transformed to Tesla unit with the use of the sensitivity coefficients.

To transform non-orthogonal sensor data we will use orthogonalization as is described in [2]. This transformation also implements the offset and sensitivity correction:

$$\mathbf{B} = \mathbf{P} \cdot \mathbf{S^{-1}} \cdot (\mathbf{F} - \mathbf{o}) + \mathbf{C_0} \tag{2.7}$$

where:

- $\mathbf{B}$ ... orthogonalized and corrected sensor output [T]
- $\mathbf{S}$ ... sensitivity matrix [eu/T]
- $\mathbf{P}$ ... orthogonalization matrix [-]
- $\mathbf{F}$ ... raw data from magnetometer [eu]
- $\mathbf{o}$ ... offsets of magnetometer [eu]
- $\mathbf{C_0}$ ... compensation field in variometer mode [T]

The equation (2.7) in matrix form:

$$\begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{S_x} & 0 & 0 \\ 0 & \frac{1}{S_y} & 0 \\ 0 & 0 & \frac{1}{S_z} \end{pmatrix} \cdot \begin{pmatrix} F_x - o_x \\ F_y - o_y \\ F_z - o_z \end{pmatrix} + \begin{pmatrix} C_x \\ C_y \\ C_z \end{pmatrix} \tag{2.8}$$

Calibration parameters and orthogonalization matrix were recorded by scalar calibration (similarly as in [19]):

$$\mathbf{P} = \begin{pmatrix} 1 & 0 & 0 \\ -0.002625330292398 & 1.000003446173634 & 0 \\ 0.004190527854485 & 0.007839927070293 & 1.000039596867103 \end{pmatrix} \tag{2.9}$$

Axis	X	Y	Z
Orientation	North-South -45°	Z (vertical)	East-West -45°
Offset [eu]	3434.5	-85.1	2585.3
Sensitivity [eu/T]	$-1.227934 \cdot 10^{11}$	$1.216069 \cdot 10^{11}$	$1.222938 \cdot 10^{11}$
Compensation field [T]	0	0	0

Table 2.6: Parameters of used magnetometer.

Sensitivity parameter for axis X has negative value because this sensor has an opposite orientation than other sensors in triaxial fluxgate sensor.

As described in [4] if we are interested in the absolute value of intensity of the magnetic field (and not only in the variation), we need to add applied compensation field C_0 (C_x, C_y, C_z are values of compensation field) to orthogonalized geomagnetic data.

In the case of Polom/Dobruska sensor there is a zero compensation field currently, as the sensors runs in "full-field" mode. The offset coils, although implemented in the sensor head, will be utilized after temperature stabilization of the sensor head will be finished.

2.4.2 Numerical Correction of Temperature Drifts

The variometer sensor (and its electronics) at the Dobruska/Polom station is operating at ambient temperatures only with a passive temperature stabilization by non-magnetic bricks in sensor vicinity. The season and diurnal changes of temperature causes a temperature drifts which need to be corrected to improve the measurement.

As is described in [5] the temperature drift in a fluxgate magnetometer is caused by multiple effects: by the temperature of the excitation tank capacitor, by the dimensional expansion of the feedback/pick-up coil or due to the expansion of the triaxial holder material and its base.

To obtain the temperature coefficients of the sensor the long-term measurement at Dobruska/Polom station and Budkov observatory was conducted. Readings were then compared and the actual variometer drifts in all three axes were determined [5].

To obtain compensated values we will use following equations:

$$\begin{pmatrix} B_X \\ B_Y \\ B_Z \end{pmatrix} = \begin{pmatrix} B_x - \alpha \cdot \Delta\theta \\ B_y - \beta \cdot \Delta\theta \\ B_z - \gamma \cdot \Delta\theta \end{pmatrix} \tag{2.10}$$

where $\Delta\theta$ [°C] is the temperature delta, α, β, γ [T · K^{-1}] are drift constants, B_x, B_y, B_z [T] are orthogonalised and corrected geomagnetic values (obtained from Equation 2.7) and B_X, B_Y, B_Z [T] are final compensated values.

The calibration of sensor sensitvities, orthogonalities and offsets was conducted at reference temperature $\theta_0 = 4.5\,°\mathrm{C}$. The temperature delta $\Delta\theta$ from Equation 2.10 can be computed as difference between actual temperature θ and reference temperature θ_0:

$$\Delta\theta = \theta - \theta_0 \tag{2.11}$$

The calculated values of drift constants α, β and γ [T · K^{-1}] were following:

$$\alpha = 1.2\,\mathrm{nT}\cdot\mathrm{K}^{-1}, \quad \beta = -3.2\,\mathrm{nT}\cdot\mathrm{K}^{-1}, \quad \gamma = 1.8\,\mathrm{nT}\cdot\mathrm{K}^{-1} \tag{2.12}$$

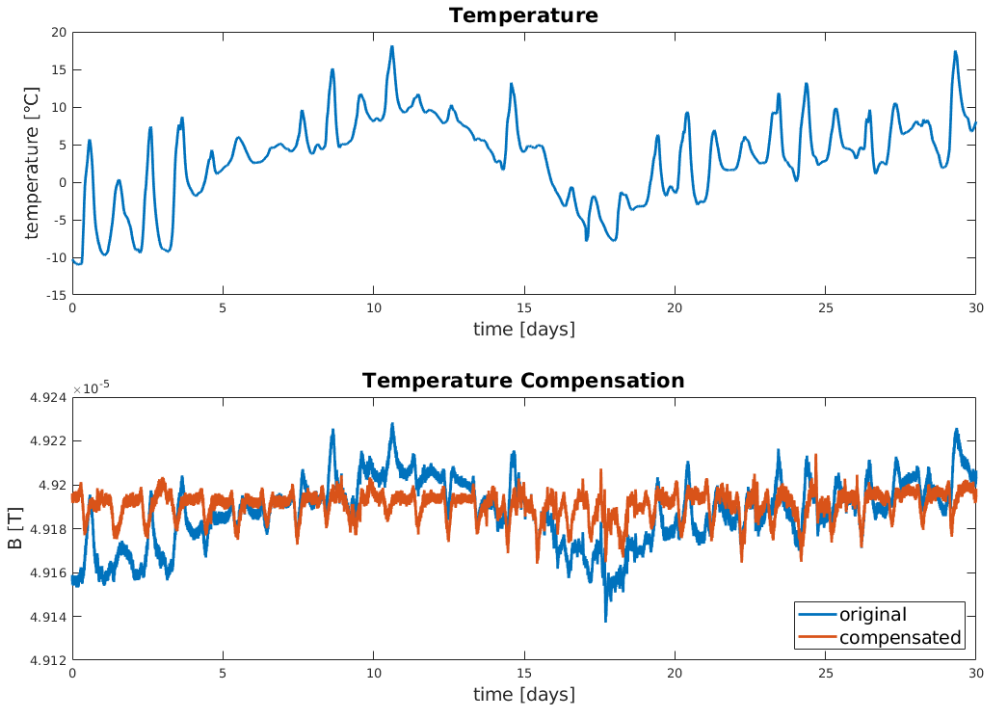

Figure 2.20: Long-term temperature measurement at Polom/Dobruska and the difference between long-term Polom/Dobruska scalar values without (blue) and after (red) temperature compensation. DPC station March 2018.

Chapter 3

Development of the Datalogger

Development of the datalogger was split into 3 part: assembly of the datalogger hardware; developing the software for datalogger (programs for data acquisition, data processing and also for device control) and developing the Client software (an application with graphical interface for data correction and analysis).

3.1 Hardware Development

The key component of the developed datalogger is a mini Linux computer Raspberry Pi 2B equipped with several interface ports including RS232 interface for connecting to the magnetic sensor.

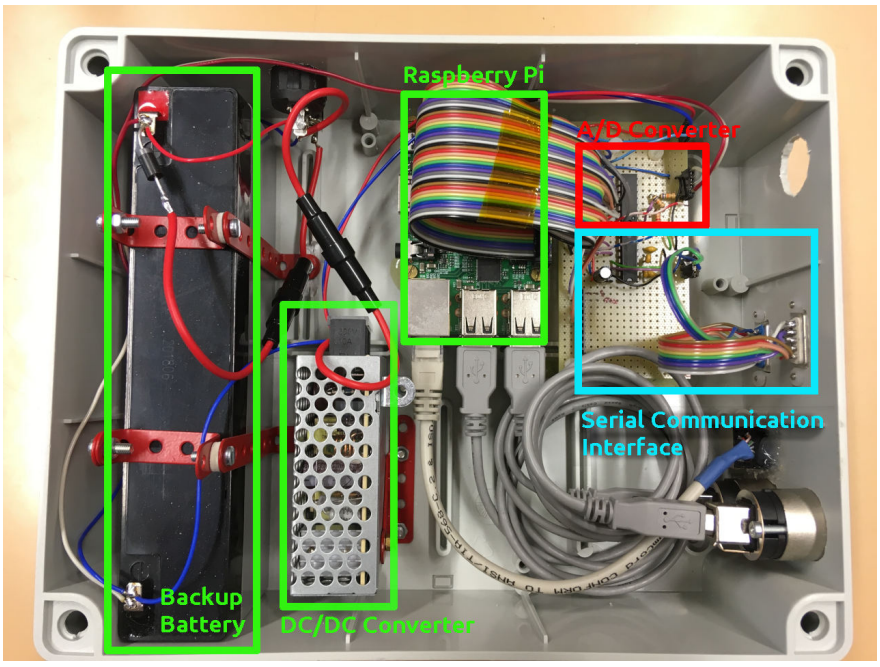

Figure 3.1: Detail of data-logger structure.

The Rapberry Pi 2B is a single board computer with 900MHz Broadcom BCM2837 Arm7 Quad Core Processor and 1GB RAM. It has an ethernet port, 4 USB ports and 40 GPIO pins. Used Raspberry Pi is running Raspbian GNU/Linux 9 (stretch) OS and has the 4.14.71-v7+ version of kernel.

It was not necessary to use specialized RTOS (real-time OS) to achieve constant sample rate of filtered and processed data (the optimization of datalogger application was efficient enough). Thanks to use of the regular OS, the device is more flexible and more accessible for users.

The device is also equipped with a backup battery and power control system for a case of power failure. In the case of short-term power failure datalogger is powered from the battery and measurement is uninterrupted.

In the case of long-term power failure the monitoring software and the power control circuit will turn off the device after the capacity of the battery is depleted. The control circuit will turn on the datalogger again after the power supply is restored.

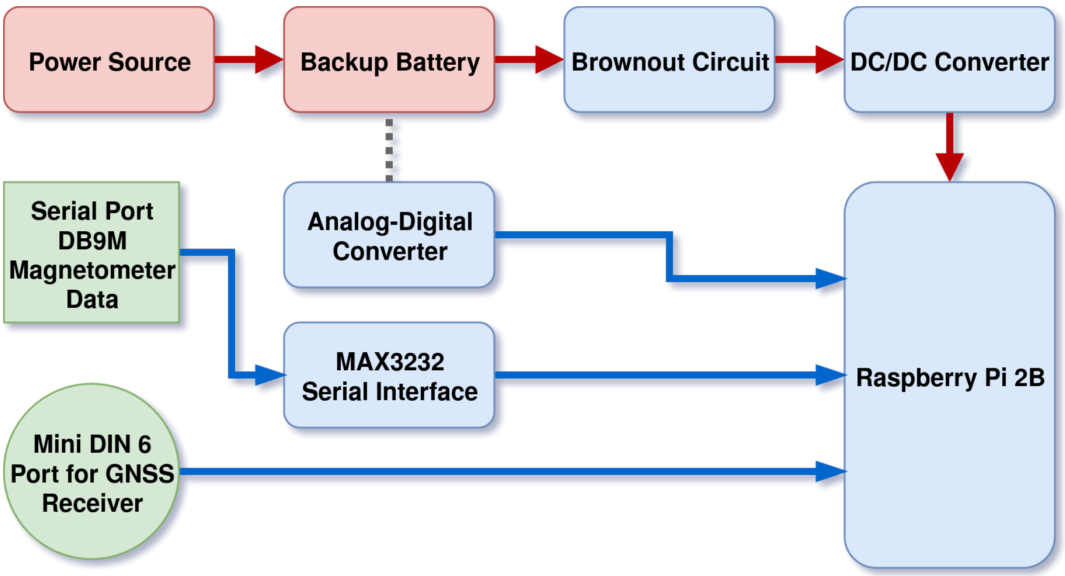

Figure 3.2: Diagram of device's hardware.

Developed datalogger implements Global Navigation Satellite System (GNSS) receiver to achieve precise sample rate by the use of PPS signal.

3.1.1 Power and Backup System

Datalogger is primary powered by the external regulated 13.8 V DC power supply (Fusion PS101). This voltage is then converted with DC/DC converter (Meanwell 15 W Single Output DC-DC Converter SD-15 series) to 5 V required by Raspberry Pi.

Backup battery (12V 2.3Ah Westinhouse WA1223E) is connected as parallel device to the 13.8 V power supply and is constantly charged. In case of short-term power failure the battery provides the backup power supply.

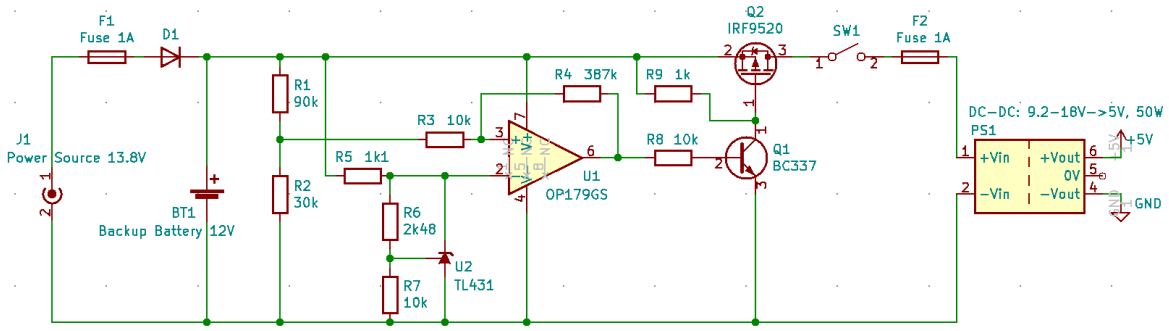

Figure 3.3: Scheme of power and backup circuit.

The developed datalogger is equipped with an analog brownout circuit to control the power source and to prevent depletion of the battery below the recommended voltage limit.

The circuit (Figure 3.3) cuts off the battery in the case of battery voltage dropping below 11.6 V. The voltage 11.6 V was set as a limit value to prolong the life cycle of the battery.
In the case of power supply restoration (that means the battery voltage will be at least 12.8 V) the circuit turns-on the power supply and the datalogger is running again.

The analog control circuit consist of an operational amplifier used as the voltage comparator with voltage reference. On the imput of the comparator the voltage divider (dividing by four) is implemented. The hysteresis of comparator is set by resistors to thresholds 2.9 V (11.6 before divider) and 3.2 V (12.8 V before divider).
The output of the comparator controls transistors.

3.1.2 Interfaces of Datalogger

The developed device has 4 USB ports for optional external memory devices (two external and two internal - inside the device box) , one LAN port for ethernet connection and the serial port for geomagnetic sensor connection.

The datalogger will be operated as a listener device to maintain the original structure of the data acquisition system. To achieve that, the device has both input and output serial port and can only read from the serial line (TX wire is not connected).

The serial interface is implementing MAX 3232 MAXIM line driver/receiver connected to Raspberry Pi GPIO pins dedicated to serial communication (implementation is similar to the example in [3] on page 286).

Device is also equipped with multichannel AD converter (MCP3008) connected to the Raspberry Pi's SPI interface. Purpose of this AD converter is to measure backup battery voltage. Data from the converter are periodically read by the program monitoring the power of battery and this information is used for fail-safe shutting down of the device (Chapter 3.2.8.1).

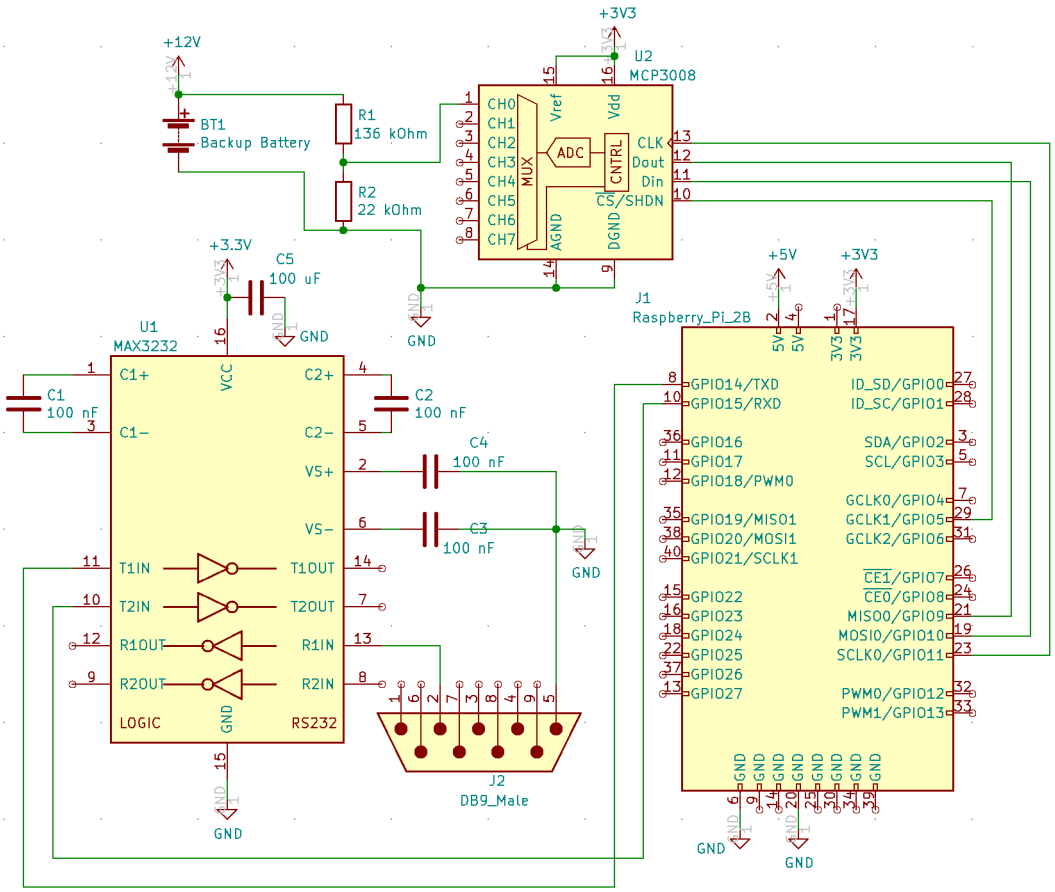

Figure 3.4: Serial communication interface and AD converter wiring.

3.1.3 GNSS Reciever

To achieve the precise sample rate of geomagnetic data the datalogger is equipped with a GNSS (global navigational satellite system) receiver Navilock NP-8004P. This receiver is based on u-blox 8 chipset and can generate precise PPS signal. The GNSS receiver was set to generate PPS signal with 10 Hz frequency (required sample rate of processed geomagnetic data).

The PPS signal is used to trigger the callback function of the datalogger application that selects and timestamps filtered data (detail description of timestamping is in chapters 3.2.5.3 and 3.2.5.4).

GNSS receiver has a MD6 connector which allows to access the pin 6 with the PPS signal. The pin 6 is connected to the Raspberry Pi's GPIO port to serve as the interrupt trigger. As is shown in the Figure 3.6 a pull-up resistor is used. The PPS signal is accessible (alongside with 3 currently unused GPIO pins, GND pin, 3.3V pin and 5V pin) from 10 pin connector.

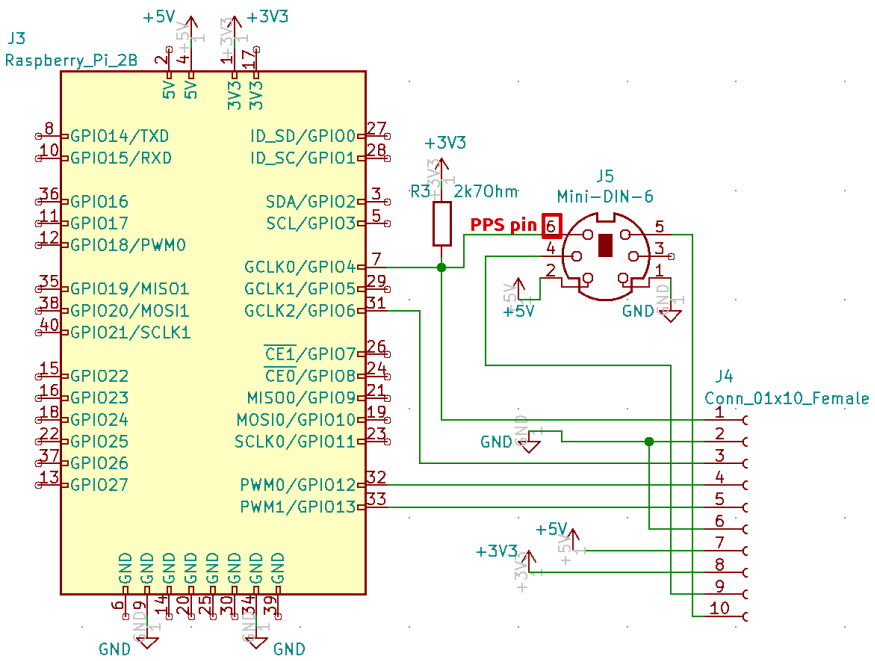

Figure 3.5: Interconnection of the Raspberry Pi and the GNSS receiver.

The frequency of the PPS signal can be set in the GNSS evaluation software *u-center* (downloadable from Navilock Driver Disk or ublox website).
To set PPS signal parameters the Configuration view window needs to be selected. Then it is possible to set the frequency,

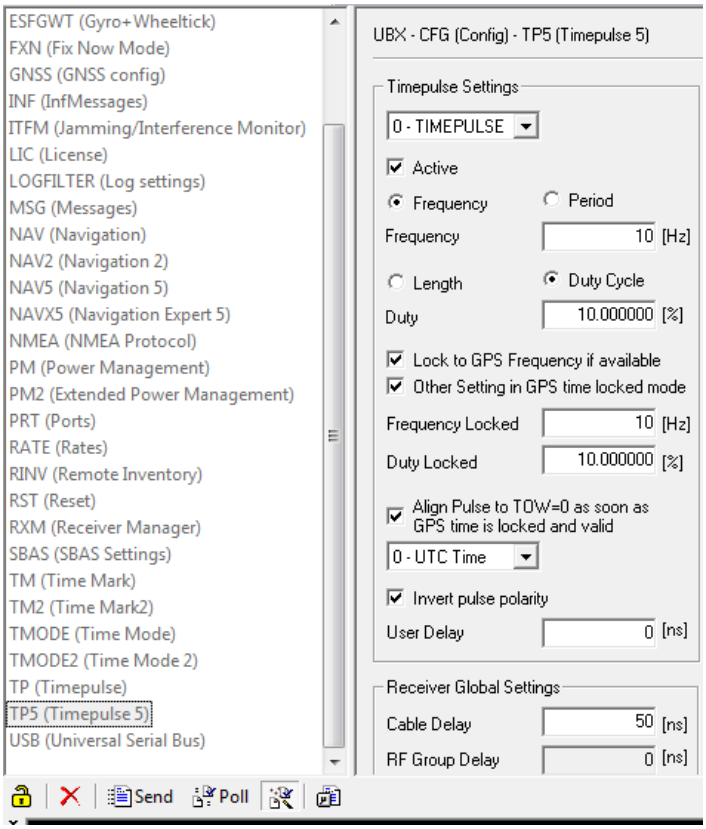

Figure 3.6: Setup of the PPS signal in *u-center* software.

3.2 Software Development

This chapter is dedicated to the description of programs and scripts created during the development of the datalogger. It also describes errors and challenges that occurred during the development and solutions resolving these problems.

All programs were written with the speed limitation in mind - every operation such as reading and parsing of geomagnetic data or filtering and saving of data to database has to be finished before new data arrives to avoid overflowing of internal buffers and enable real-time data display. Because of this limitation, every part of code was speed-tested and optimized to be as time effective as possible.

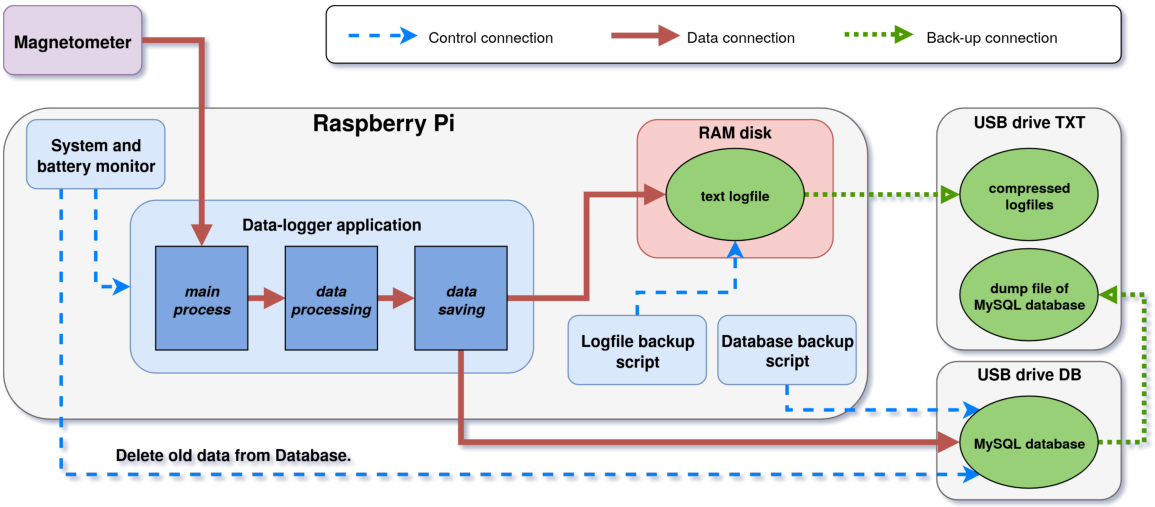

Figure 3.7: Datalogger application structure.

Developed software consist of datalogger application, system and battery monitoring script and data backup scripts. The structure and dependency of programs is shown in the Figure 3.7.

Datalogger application is the main software that reads, filters, converts and stores geomagnetic data. It is written in Python an executed on the Raspberry Pi. Application is thoroughly described in chapters 3.2.2, 3.2.3, 3.2.4, 3.2.5 and 3.2.6.

System and battery monitoring script periodically checks the system status, status of the datalogger application and backup battery voltage. It launches thedatalogger application and closes it in the case of upcoming power failure (it also correctly shuts down the system). Program saves value of battery voltage and system status info into MySQL database (table *DeviceStatus*) and program also deletes old data from database. Program implementation is described in Chapter 3.2.8.

Backup scripts are responsible for regular backup of saved data to the backup memory (Chapter 3.2.7).

All developed software can be found on attached CD. The Folder structure of the CD is described in the Chapter 9.

3.2.1 SQL Database

The SQL database is used as the main data storage system for this project. It was decided to use database because of its organized structure which allows advanced data selections.

3.2.1.1 Stored Data and Database Design

The database serves to store 5 Hz data from the sensor with timestamps alongside with the location info, sensor parameters and processing methods details.

Stored magnetic field data are filtered and down-sampled but they are not processed in terms of offset and gain calibration or orthogonalization (this part of processing is done in the Client software). Storing of uncorrected data (filtered and decimated raw data) and the sensor parameters brings the possibility of method changes and further data analysis.

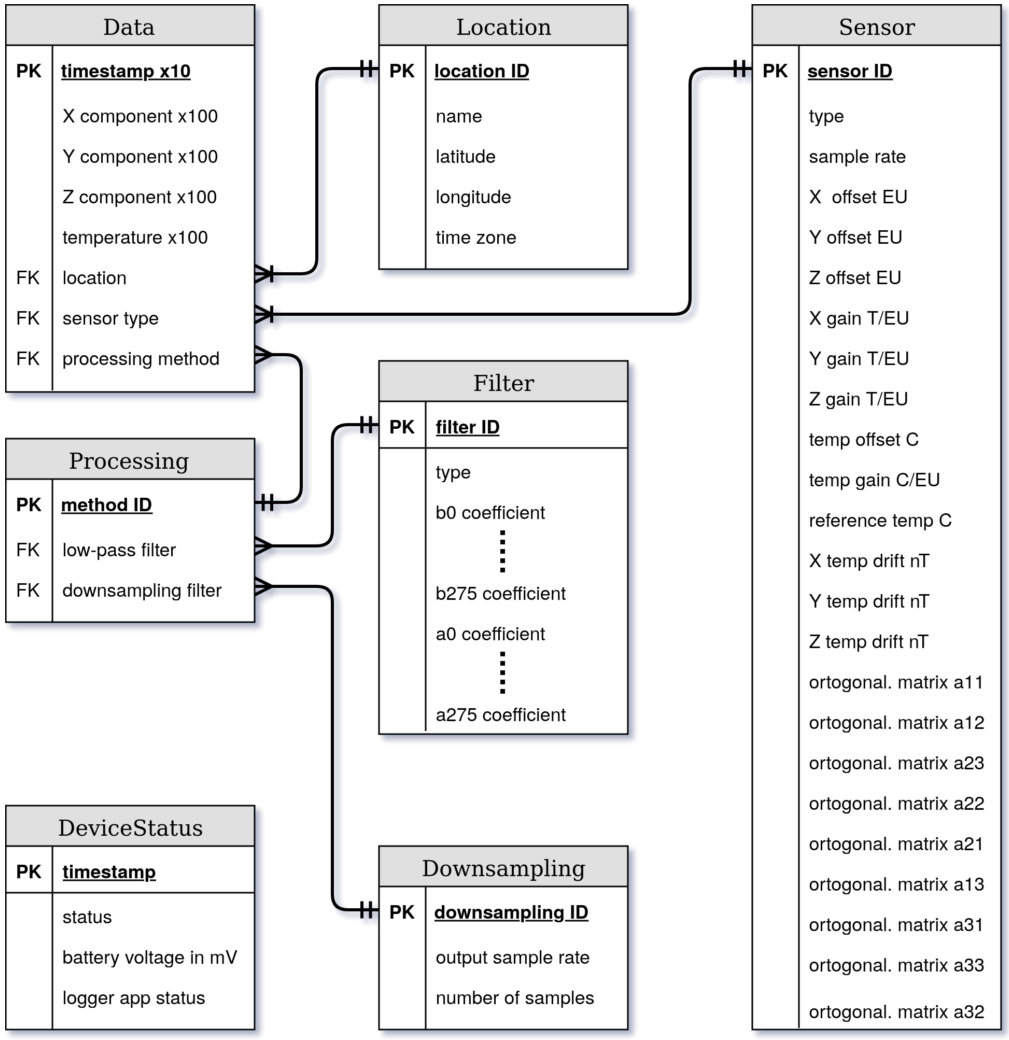

Figure 3.8: Model of the database in crow-feet notation.

The model of the database is shown in the Figure 3.8. The database contains tables for storing geomagnetic data and additional information as described before and it

also implements table for storing the data about datalogger device (such as status, battery voltage, etc).

Location and sensor details (such as sample rate, offset and gain values, etc.) are stored to provide complete information about the measurement setup.

The main table of the database is the table *Data*. It contains data from the sensor and foreign keys from tables containing information about location, sensor parameters, etc.

Timestamp in unix time (multiplied by 10 and then converted to integer) is used as a primary key. Datatype of *timestamp x10* is unsigned bigint.
Data from the sensor are multiplied by 100 and saved as integers to minimize the size of database.

Table Data is created with following SQL code:

```
drop table if exists `Data`;
create table `Data` (
`TimeStampx10` bigint unsigned not NULL,
`Xx100` int not NULL,
`Yx100` int not NULL,
`Zx100` int not NULL,
`Tempx100` int not NULL,
`Location` int not NULL,
`SensorType` int not NULL,
`ProcessingMethod` int not NULL,
primatry key (`TimeStampx10`),
foreign key (`Location`) references Location(`LocationID`),
foreign key (`SensorType`) references Sensor(`SensorID`),
foreign key(`ProcessingMethod`)referencesProcessing(`MethodID`)
);
```

The table *DeviceStatus* stores information about datalogger status (status of the device, baterry voltage, datalogger app status). Unix timestamp is used as the primary key similarly to the *Data* table, but because of larger intervals between records (minutes instead of 0.1 second) there is no need for preserving values on the right side of decimal point. Timestamp is therefore rounded to whole seconds and saved as unsigned Integer.

The size of the database is mainly effected by number of records in the table *Data*. One record line has size of 27 bytes. The data is kept only for 3 months and starts to be owerwritten.

It was determined during the testing that the whole database containing 3 months of data with 5 Hz sample rate will have circa 4.3 GB. Size of used external USB flash drive is 64 GB.

3.2.1.2 MySQL Database Implementation

For implementation of designed database was chosen the MySQL - an open-source relational database management system. An open-source fork of the MySQL called MariaDB was used because of its support on the Raspberry Pi platform.

The MySQL server runs on the Raspberry Pi computer and it is stored on the external USB drive to reduce wear-out of system SD card. USB drive is auto mounted and its directory is */media/db/*.

The MySQL database was moved to the new directory with use of following code:

```
sudo systemctl stop mariadb
sudo rsync -av /var/lib/mysql /media/db
sudo mv /var/lib/mysql /var/lib/mysql.bak
```

After the database was moved, the MySQL configuration file */etc/mysql/my.cnf* was edited and old directory (*/var/lib/mysql*) was changed to the new one (*/media/db/mysql*). Then the database was restarted.

Data from the magnetometer are uploaded from Python datalogger application (detail description of implementation is in the Chapter 3.2.6.2). Data are filtered and decimated but not corrected and compensated. Processing of data is done in the client software (Chapter 3.3).

3.2.2 Datalogger Application

The datalogger application consist of three separate processes interconnected by the Multiprocessing [20] library. It was necessary to use a multiprocessing and to divide the application to multiple parts because of the speed requirement. Sample rate of incoming geomagnetic data in combination with relatively long time needed for data filtering and saving would cause too big workload for a single process running on Raspberry Pi.

The multi process parallelism was chosen instead of the threading parallelism because multiprocessing uses separate memory spaces and it allows better use of CPU.

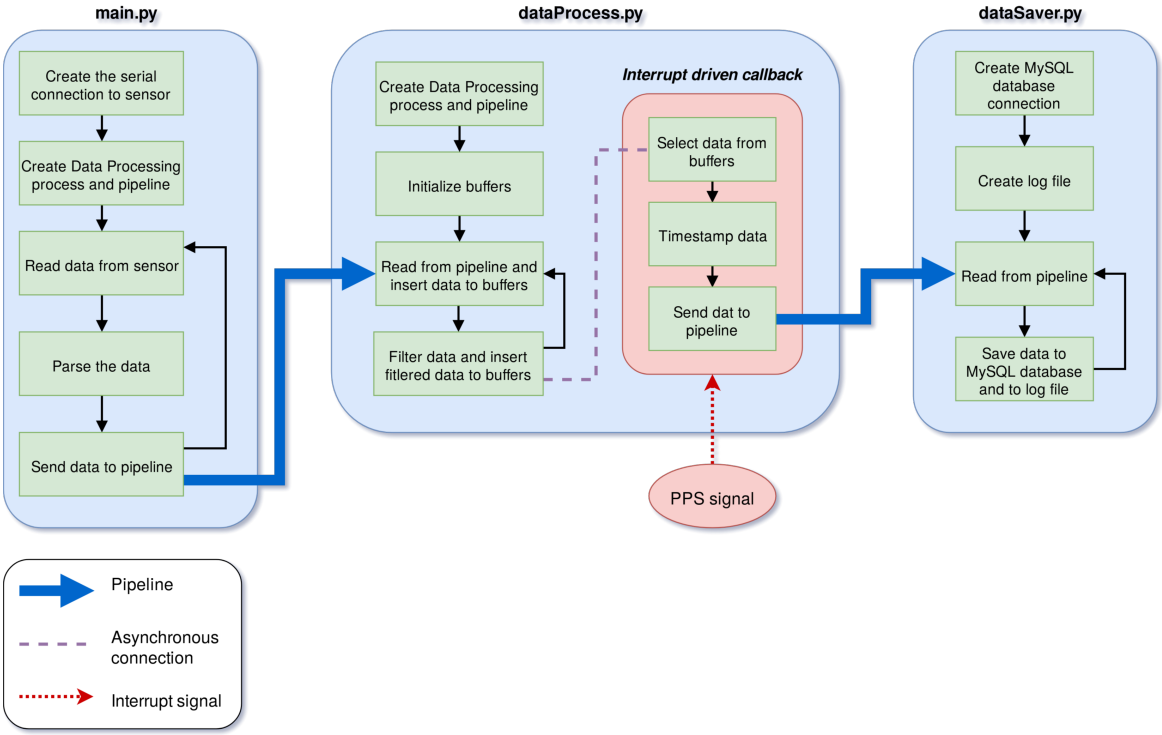

Figure 3.9: Datalogger application structure.

The three processes of datalogger application are: *Main* process, *Data Processing* process and *Data Saving* process. Processes are connected by two pipelines.
Application is started with launch of the main process *main.py* - other processes and pipelines are then created automatically.

As is shown in the Figure 3.9: *Main* process reads and parses data from the sensor. Parsed data are then sent by pipeline to *dataProcess.py* to be filtered and stored into buffers.

The *dataProcess.py* implements an asynchronous callback function triggered by PPS signal, that is selecting and timestamping data from buffers to create data with required sample rate. These data are then send by second pipeline to *dataSaver.py* to be saved to the database and also to the plain text file.

3.2.3 Connection and Synchronization of Processes

All 3 processes of the datalogger application are run as separate processes which leads to necessity of the process connection and synchronization.

The datalogger application starts with the launch of *main.py* process. This process launches *dataProcess.py* and creates simplex pipeline between *main.py* and *dataProcess.py*.
The process *dataProcess.py* launches *dataSaver.py* process and creates simplex pipeline between these two processes.

Processes are connected with simplex pipelines instead of duplex pipelines to make the connection faster.

3.2.3.1 Process Synchronization during Closing

While the process synchronization is not needed during the program execution it is necessary during the closing of the application. To close all connections (database, log file, pipelines) properly, processes are synchronized and if *main.py* receives the closing signal (signal SIGUSR1) it will send signals to other processes which will lead to safe close of the whole application.

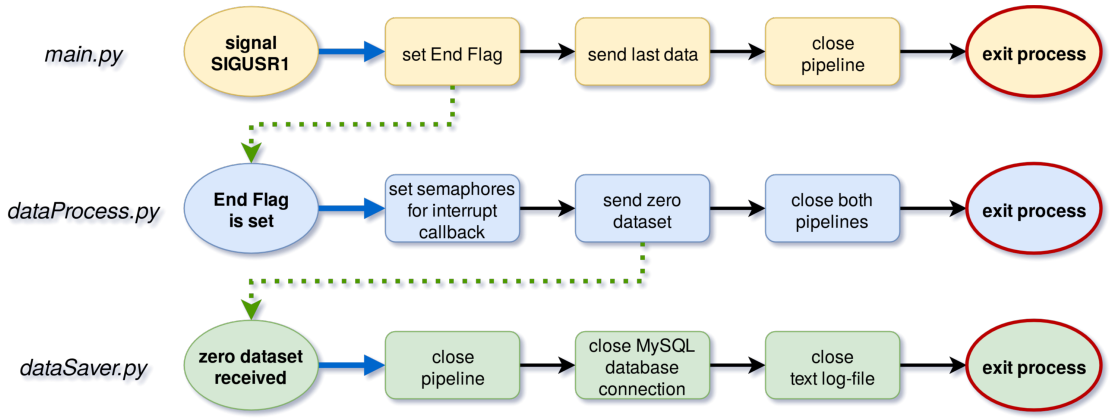

Figure 3.10: Diagram of proces synchronization during closing.

The process *main.py* implements a signal handler and detects signal SIGUSR1. If this signal is received, the method *doCleaning* is called. Cleaning method sets the end flag for *dataProcess.py* (flag is implemented as *manager.Event()* from Multiprocessing library [20]).
Then last two datasets are send to pipeline and pipeline is closed. Then the main process is terminated.

The data processing process checks end flag periodically during the execution of its main loop. If the flag is set, *doCleanup* method is called. This method will send last dataset to *dataSaver.py* - but this dataset contains only zeros. Then the receiving end of pipeline between *main.py* and *dataProcess.py* is closed. The sending end of pipeline between *dataProcess.py* and *dataSaver.py* is closed as well. The process is then terminated.

When the *dataSaver.py* process receives zero dataset (these values were chosen because it is impossible to receive zero dataset during regular run) the *cleanup* method is called. This method closes log files, database connection and pipeline. Process *dataSaver.py* is then terminated.

3.2.4 Receiving Data from Sensor

The purpose of the *main.py* process is to read geomagnetic data from sensor via serial connection, parse it and to send it through pipeline to *dataProcess.py*.

The process contains one main loop in which data are read from serial connection and then parsing method is called. Data from the sensor are in string format and need to be converted to integer format to be filtered.

During the implementation of the process the parsing error occurred from time to time. Received data from the magnetometer were missing some bytes and it resulted into receiving two lines merged together. This merge was causing that the false data were received and the whole measure was influenced.

After thorough debugging it came to be clear this error was caused by slow method for reading a line from serial connection. Originally the method *readline* from PySerial library [21] was used but it was not fast enough.
To resolve this problem the wrapper to a PySerial connection was used. The solution was implemented as python class and was taken from GitHub forum post by user *skoehler* [23]. Implemented class uses basic PySerial connection but implements readline function in a faster way.

Class implementation has two functions - constructor *__init__* and function *readline*:

```
class ReadLine:
 def __init__(self, s):
   self.buf = bytearray()
     self.s = s

 def readline(self):
   i = self.buf.find(b"\n")
   if(i >= 0):
     r = self.buf[:i+1]
     self.buf = self.buf[i+1:]
     return r

   while(1):
     i = max(1, min(2048, self.s.in_waiting))
     data = self.s.read(i)
     i = data.find(b"\n")
     if(i >= 0):
       r = self.buf + data[:i+1]
       self.buf[0:] = data[i+1:]
       return r
```

```
        else:
            self.buf.extend(data)
```

During the serial connection initialization, the serial port is opened at first and then the wrapper object is constructed. To read new line from connection class function *readline* is called (it is a different method than the PySerial *readline* method):

```
portRXTX = serial.Serial("/dev/ttyAMA0", baudrate=115200,...
    ... timeout=1, exclusive=True)
rl = ReadLine(portRXTX)
rcv = rl.readline()
```

After implementing the wrapper class the parsing problem was resolved and did not occur any more.

3.2.5 Data Processing

The process dedicated to filtering and timestamping received geomagnetic data is called *dataProcess.py*. This process consist of initialization part, main loop (data are periodically read from the pipeline, added to buffers and then filtered) and callback function for selecting and timestamping filtered data to achieve required new sample rate.

The callback function is interrupt driven and is called asynchronously - after its execution process returns to the last point before the interrupt and continues its run.

3.2.5.1 Initialization

During the initialization part of the process the filter and filtration parameters (filter coefficients, filter delay, original sample rate, new sample rate, etc.) are loaded from json file. The parameters of implemented filter are described in Chapter 2.3.5.1 in Table 2.4.

Buffers for received and filtered data are also created. There is one buffer for each type of value (such as X component of geomagnetic field, temperature value, etc.). Buffers are initially filled with zeros so it is necessary to wait with the time stamping and sending of data until buffers are filled with real values.
Buffers are implemented as circular: the newest value is always at the zero index and the oldest value is always at the last index.

Initialization part of code also implements GPIO library set-up. GPIO interface of Raspberry Pi is used for triggering interrupt driven callback.

3.2.5.2 Main Loop and Filtering

In the main loop of process geomagnetic data are read from pipeline (data are already parsed by *main.py* process) and added to buffers. Method *doFiltering* is then called to filter data in buffers.

Implementation of data read and buffer filling:

```
dataSample = pipelineFromMain.recv()
completeRead = 0     # set semaphore

#adding data to rolling FIFO buffer
#shift by one to the right (from index 0 to index 1)
buffX = np.concatenate(([dataSample[0]],buffX[:-1]))
buffY = np.concatenate(([dataSample[1]],buffY[:-1]))
buffZ = np.concatenate(([dataSample[2]],buffZ[:-1]))
buffTemp = np.concatenate(([dataSample[3]],buffTemp[:-1]))
doFiltering()

completeRead = 1     #reset semaphore
```

Filtering method implements FIR filter by its definition (equation 2.3) and corrects multiplication error of the filter. Each type of value is filtered separately and then added to dedicated buffer:

```
#filter values
xfilt = np.sum(b*buffX[0:countOfCoefs])/multipError
yfilt = np.sum(b*buffY[0:countOfCoefs])/multipError
zfilt = np.sum(b*buffZ[0:countOfCoefs])/multipError
tempfilt = np.sum(b*buffTemp[0:countOfCoefs])/multipError
previousValues=[buffXfilt[dataPoint],buffYfilt[dataPoint],...
    ... buffZfilt[dataPoint], buffTempfilt[dataPoint]]

#insert filtered data to circular buffer
buffXfilt = np.concatenate(([xfilt],buffXfilt[:-1]))
buffYfilt = np.concatenate(([yfilt],buffYfilt[:-1]))
buffZfilt = np.concatenate(([zfilt],buffZfilt[:-1]))
buffTempfilt = np.concatenate(([tempfilt],buffTempfilt[:-1]))
```

During the implementation of *dataProcess.py* the speed problem occurred - roll of buffer (using NumpPy [22] function *roll*) was too slow. This implementation of circular buffer also caused problems when interrupt handler was called during the roll - if the interrupt was triggered at the moment right after the roll and before the new value was inserted to the buffer, there was the oldest value at the zero index - instead of the required newest value.

After debugging of the process and identifying the cause of problems the *roll* method was replaced with more efficient method *concatenate* (also from NumPy library).

The semaphores *readyBuff* and *completeRead* indicating that new data are being added were also implemented. As is discussed in following chapter 3.2.5.3 these semaphores are used to control data selection during execution of the interrupt callback function.

3.2.5.3 Interrupt Driven Callback

As is shown in the Figure 3.9 the interrupt driven callback is used to select, timestamp and send filtered data to pipeline. The method *interruptCallback* is triggered by PPS signal from GNSS receiver (connected to GPIO pin - see Chapter 3.1.3).
To set up the callback the *add_event_detect* function from GPIO library is used. Implementation of this function is done in similar way as is described in the example code in [3](in the Chapter 9.12 Programming with Interrupts).

During the development of the datalogger application the goal was to implement precise sample rate. To achieve it the GNSS receiver with PPS signal is connected to the Raspberry Pi. The PPS signal is used as the trigger for data selection and timestamping. Precise frequency of signal ensures that the time delta between samples is constant - under the condition the data selection and timestamping is done immediately after signal is received.

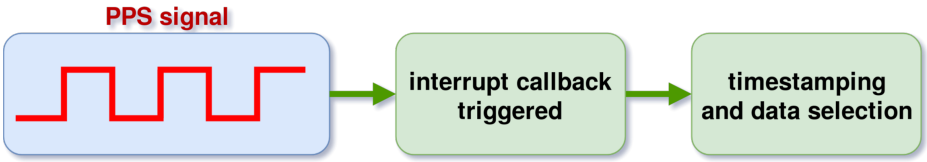

Figure 3.11: Visualization of interrupt driven callback with PPS signal.

The interrupt driven callback was chosen as the best solution for minimizing latency between receiving of the PPS signal (time pulse) and the execution of the timestamping method. The interrupt callback is called asynchronously and executed immediately after the signal is received.

Because the callback is an asynchronous function it can be called in any moment of process execution. If the callback is called during the adding the new values to the buffer or filtering, selected data can contain mixture of the new and older data values. To prevent that the semaphores are implemented.
The *completeRead* semaphore is set before data are added to buffers and filtered and unset after completion of this part of program.
If the callback is triggered and *completeRead* semaphore is set, instead of newest data values (which are in that moment not fully updated) the latest complete set of data values is selected.

This solution was selected because it does not delay timestamping and the maximal time difference between the timestamp of the PPS pulse the timestamp of the selected dataset is equal to the period of original geomagnetic signal.

Another solution was also tested during the development: in case of triggering the callback while *completeRead* semaphore is set the selection and timestamping was skipped and resumed after semaphore were unset. This solution was not used because it was causing the delay of timestamping.

The second semaphore *readyBuff* is implemented to prevent selecting and timestamping values before buffers are filled and filter stabilized.

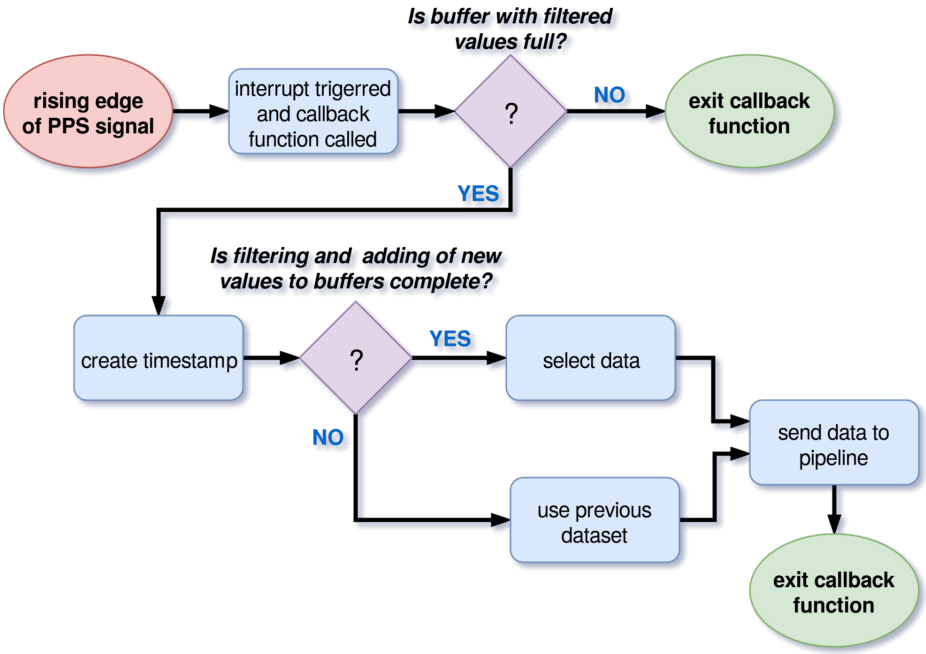

Figure 3.12: Process diagram of interrupt callback.

3.2.5.4 Timestamping Principle

The PPS signal triggering the callback function has the same frequency as is the requested frequency of filtered and decimated output data (10 Hz). Each time the callback function is triggered, filtered data are selected and timestamped. Precise PPS signal and immediate execution of the callback function means that the sample rate of output signal is constant.

The downsampling is implemented as simple select of the right value from the buffer with filtered data. The right value has to have the same timestamp as the time pulse (PPS signal).

Because of the delay of the used FIR filter the newest value is not the right one - the value that belongs to the current time pulse is not in the buffer yet. To solve this problem each time the callback is triggered, the function is not selecting data values belonging to the current time pulse - it is selecting data values belonging to the previous time pulse that already has belonging data in buffers.

Implemented timestamping method is shown on the Figure 3.13. Shown example has the filter delay 7 samples, original sample rate 1 samples per second and time pulse period 5 seconds. Desired new sample rate is 0.2 sample per second (one sample each 5 seconds).

These values were chosen instead of the real filter parameters to make the example more accessible and straightforward. the proper timestaming implementation and

real filter and timestamping parameters are described in the following chapter - Chapter 3.2.5.5.

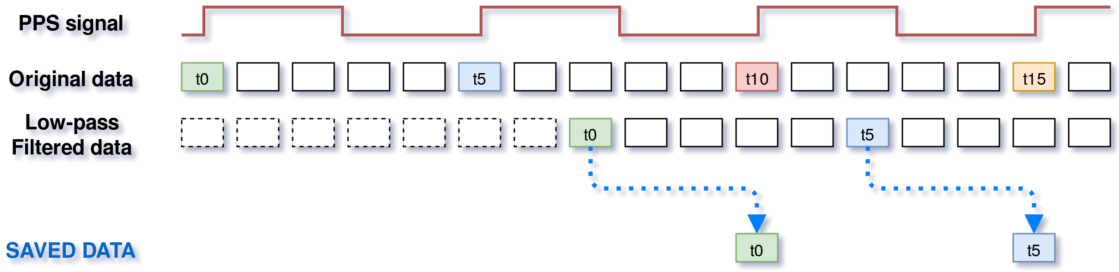

Figure 3.13: Timestamping and data selection visualization.

To timestamp values correctly, constants *delayInPulses* and *dataPoint* needs to be determined:

$$delayInPulses = \text{ceil}\left(\frac{delay}{sampesPerPulse}\right) \tag{3.1}$$

$$delayInPulses = \text{ceil}\left(\frac{7}{5}\right) = 2$$

The constant *delayInPulses* describes how many time pulses needs to pass after receiving the value, before the value is ready in the buffer with filtered data. We will use this constant to determine the time pulse in the past for which we are selecting data.

$$= \text{round}(delayInPulses \cdot samplesPerPulse - delay) \tag{3.2}$$

$$dataPoint = \text{round}(2 \cdot 5 - 7) = 3$$

The constant *dataPoint* describes the position of the required value in the buffer with filtered data (constant is the index in the array of buffer).

In the shown example there are two time pulses before the buffer is full: time pulse at time t_0 and t_5. During these pulses no timestamping is done.

When time pulse at time t_{10} comes, the first timestamp will be made: we will select data value with timestamp $T = t_0$. Value of T is determined by following equation:

$$T = t_{current} - delayInPulses \cdot samplesPerPulse \tag{3.3}$$

$$T = t_{10} - 2 \cdot 5 = t_0$$

To determine where in the buffer with filtered data is the value belonging to the time t_0 we will use constant *dataPoint*. Because $dataPoint = 3$ we will select the value 3 samples older than the newest data value in the buffer.

To sum up the whole timestamping and selection process from example: when time pulse in t_{10} comes we will select the value belonging to time t_0 with timestamp t_0 and send it to pipeline.

3.2.5.5 Timestamping Implementation

Similarly to the timestamping method example in previous chapter we will use equations 3.1, 3.2 and 3.3 to determine timestamping constants. To determine the filter delay we will use Equation 2.4.

Timestamping constants are computed with following part of *dataProcess.py* code:

```
delaySamples = (countOfCoefs-1)/2
pulsePeriod = 1/sampleRateNew
samplesPP = sampleRate/sampleRateNew
delayInPulses = int(np.ceil(delaySamples/samplesPP))
dataPoint=int(np.round(delayInPulses*samplesPP-delaySamples))
```

Timestamping itself is part of *interruptCallback* method in *dataProcess.py* and is executed every time the callback function is triggered by the PPS signal:

```
pulseTime = time.time()
#data timestamp = pulseTimestamp-delayOfSystem (in unix time)
realTimestampx10 = int(round(pulseTime -...
    ... (pulsePeriod*delayInPulses),1)*10)
# realTimestamp = timestampRound(realTimestamp)

#if callback is not iterrupting filling buffers with new data
if(completeRead == 1):
  Xx100 = int(round(buffXfilt[dataPoint],2)*100)
  Yx100 = int(round(buffYfilt[dataPoint],2)*100)
  Zx100 = int(round(buffZfilt[dataPoint],2)*100)
  tempx100 = int(round(buffTempfilt[dataPoint],2)*100)

#else use previous data (from last filtering)
else:
  Xx100 = int(round(previousValues[0],2)*100)
  Yx100 = int(round(previousValues[1],2)*100)
  Zx100 = int(round(previousValues[2],2)*100)
  tempx100 = int(round(previousValues[3],2)*100)
  completeRead = 1

dataToSend = [realTimestampx10, Xx100, Yx100, Zx100, tempx100]
sendPipeEnd.send(dataToSend)
```

Value of the timestamp is in Unix time and it is multiplied by 10 and then rounded and converted to Integer datatype. Data values (X,Y,Z and temperature) are multiplied by 100 and then rounded and converted to Integer as well.
Reason for multiplication and conversion into Integer data type is to reduce size of record in the MySQL database (and to preserve requested decimal place values).

The PPS signal is synchronized to the precise clock of the GNSS receiver. It means the 10 Hz time pulse comes every 0.1 seconds of UTC time - for example in 12:03:05.1, 12:03:05.2, 12:03:05.3, etc.

But the previously described timestamp of selected data is created from the system time. System time of Raspberry Pi is synchronized when it is connected to the internet (the device is connected to the internet permanently) but it is not as precise as the GNSS time.

Because the relatively small difference between system time and "real" time of PPS signal (the time of the GNSS receiver), it was possible to implement a simple solution to synchronize timestamps. At the moment of timestamping the system time is read but it is rounded to 1 digit to the right of the decimal point. This correction ensures that timestamps has the precise 0.1 second interval synchronized with GNSS clock.

Implemented synchronization is part of conversion of Unix timestamp to integer:

```
realtime = pulseTime - (pulsePeriod*delayInPulses)
realTimestampx10 = int(round(realtime,1)*10)
```

The system time is synchronized with the use of the native *systemsuncd* deamon service that implements NTP (network time protocol) client. The status of the NTP synchronization is periodically checked by the System and Battery Monitor script and logged to the *DeviceStatus* table in the MySQL database (implementation is described in the Chapter 3.2.8.2).

3.2.6 Data Saving

The purpose of the third process called *dataSaver.py* is to save filtered and decimated data to the MySQL database and to the plain text logfile.

As is described in the Chapter 1.1.1 the 5 Hz data are stored in the MySQL database and the 10 Hz data are stored in the txt logfiles.

Filtered data has 10 Hz sample rate (same as the frequency of the PPS signal). Process *dataSaver.py* saves every record into the log-file and every even record into the MySQL database (with this simple decimation the 5 Hz sample rate is achieved).

The database is the main source for further processing and analysis of recent data. Plain text logfiles serves as a data backup in the case of the database malfunction or in the when the analysis of archive data is requested.

The client programs can download data both from MySQL database and logfiles.

3.2.6.1 Process Description

Before the main loop of process is entered, the database connection is created. To connect to the MySQL database the *mysql.connector* python library is used. After the database connector initialization, the txt logfile is created.

In the main loop, processed datasets are received via pipeline and after the value check (in the case of zero dataset the closing method is called as is described in the Chapter 3.2.3) saved to the MySQL database and to the logfile.

The main loop is also periodically creating new logfiles. To limit the size of each logfile, new file is created every 12 hours.

3.2.6.2 MySQL Database Connection

The *mysql.connector* python library is used to connect to the MySQL database:

```
#connect to mysql database
mydb = mysql.connector.connect(host="localhost", ...
    ... user="maglab",passwd="pass",database="datalogger")
mycursor = mydb.cursor()

#upload data to MySQL database every 0.2 second
if(round(data[0] % 2) == 0.0):
  try:
    sql = "insert into Data(TimeStampx10,Xx100,Yx100,Zx100,...
    ...Tempx100,Location,SensorType,ProcessingMethod)...
    ...values (%s,%s,%s,%s,%s,%s,%s,%s)"
    val = (data[0],data[1],data[2],data[3],data[4],1,1,1)
    mycursor.execute(sql,val)
    mydb.commit()

  except Exception as e:
    print("dataSaver: unable to insert data")
    print(e)
```

Alongside with the processed dataset of geomagnetic data other parameters are saved to the database as well. Parameters of location, type of sensor and used processing method are inserted to the database to provide detailed information about measurement. These parameters are stored in the saving process as constants.

Structure of inserted data is dependent on the database structure (Chapter 3.2.1).

3.2.6.3 Log-files Format

Filtered data are in *dataSaver.py* process saved both to the MySQL database and to the plain text logfile. The logfile is located on RAM disk (RAM disk setup is described in the Chapter 3.2.7.1) and stores 12 hours data. After creating a new logfile the old one is compressed and moved to the USB flash disk (directory */media/txt/*).

Name of the logfile has always the same format: logFile_YYYY-MM-DD_hh-mm.txt .

Each dataset is saved to the logfile as one line in the following format:

```
Unix timestamp x10 [sec];X value x100 [EU];Y value x100 [EU];Z
value x100 [EU];temperature x100 [EU];
```

Saved data has the same format as the data in the MySQL database.

Example of records in log-file:

15774396920;-549535605;-137551439;211047558;375636609;
15774396921;-549322076;-137499583;210965397;375490644;
15774396922;-549322068;-137499424;210965331;375490370;

Implementation of log-file set-up and data saving process:

```
#create first log file
filePath = '/home/pi/ramDisk/'
filename = filePath+'logFile_'+str(datetime.datetime.now( ...
    ... ))[:16].replace(' ','_').replace(':','-')+'.txt'

fp = open(filename,'w')
logCreate = time.time()

#write data to log-file
line = str(data[0])+';'+str(data[1])+';'+str(data[2])+';'...
   ...+str(data[3])+';'+str(data[4])+";\n"

fp.write(line)

#create new log-file
currTime = time.time()
if(currTime - logCreate > logPeriod):
  filename = filePath+'logFile_'+str(datetime.datetime.now(...
    ... ))[:16].replace(' ','_').replace(':','-')+'.txt'
  fp = open(filename,'w')
  logCreate = currTime
```

3.2.7 Backup Systems

The developed datalogger uses several backup systems to ensures data are safely saved in the case of application, computing or power malfunction.

To understand the data backup procedures we need to explain how data are stored.

3.2.7.1 Data Storing

The Raspberry Pi is using SD card as the storage medium. Use of SD card comes with the risk of the wear-out and data lost. To minimize the unnecessary wear-out of SD card caused by frequent write operations, the MySQL database is moved to the external USB flash drive (description in the Chapter 3.2.1.2) and the logfile that is written to is located in the RAM disk and then moved to another external USB flash drive.

The following shell commands were used to create a 200 MB RAM disk in directory */temp/log* accessible via symbolic link from directory */home/pi/ramDisk*.

```
#create a backup copy of fstab file
sudo cp -p /etc/fstab  /etc/fstab.save
#create a directory
sudo mkdir /tmp/log
#edit fstab as superuser
sudo nano /etc/fstab
```

To the *fstab* file was added the following line:

```
tmpfs /tmp/log tmpfs nodev,nosuid,size=200M 0 0
```

And finally the disk was mounted and the symbolic link was created:

```
sudo mount /tmp/log
ln -s /tmp/log/ /home/pi/ramDisk
```

The RAM disk is used as a temporary storage for the logfile before its completion and closing. Logfile is closed after the given period of time (12 hours) and the new logfile is immediately created. After the new logfile is created the old one is compressed and moved to the external USB flash drive that is auto-mounted to directory */media/txt/*. Compression and moving of the logfile is done by the bash script *logFileMove.sh*. This script is called by *cron* scheduler every 12 hours.

Implementation of *logFileMove.sh* script:

```
#!/bin/bash
RAMDISKDIR="/home/pi/ramDisk/"
NEWDIR="/media/txt/data"

oldFiles=(`find $RAMDISKDIR -type f -name "*.txt" -mtime +1`)
gzip ${oldFiles[*]}
mv $RAMDISKDIR*.gz $NEWDIR
```

3.2.7.2 Data Backup

To ensure the memory of Raspberry Pi in will not become full and to prevent data loss in case of device malfunction, the backup of stored data is implemented as described previously. The backup medium is a USB flash drive.

Data from the MySQL database are backed up every 30 days in the form of mysqldump file. This file is created by script *dbBackup.sh* and stored on the external USB flash drive alongside with compressed txt logfiles (directory */media/txt/*).
Name of the sql file is in format: mysqlBackup_YYYY-MM-DD.sql.

Implementation of the MySQL database backup:

```
#!/bin/bash

TARGETDIR="/media/txt/dbBackup/"
DATE=`date +%Y-%m-%d`

mysqldump -hlocalhost -umaglab -ppassw ...
   ... --single-transaction --routines --triggers ...
   ... --all-databases > "$TARGETDIR"mysqlBackup_"$DATE".sql
```

3.2.8 System and Battery Monitor

Apart of the hardware brownout circuit that ensures that datalogger turns off in the case of low battery voltage and turns on the device in the case of restoring the power supply, developed device has also a software battery monitoring and system control.

Battery monitoring feature is part of System and Battery Monitor program. This program is periodically called by*cron* scheduler (every 5 minutes).

3.2.8.1 Battery Monitoring

The monitoring program measures the battery voltage with AD converter connected to Raspberry Pi's SPI interface. Value of the voltage is stored into MySQL database and it is also used for determination of the next actions.
Implementation of voltage reading with use of Adafruit MCP3xxx library:

```
import adafruit_mcp3xxx.mcp3008 as MCP
from adafruit_mcp3xxx.analog_in import AnalogIn

#INITIATE ADC
# create the spi bus
spi=busio.SPI(clock=board.SCK,MISO=board.MISO,MOSI=board.MOSI)
# create the cs (chip select)
cs = digitalio.DigitalInOut(board.D5)
# create the mcp object
mcp = MCP.MCP3008(spi, cs)
# create an analog input channel on pin 0
chan = AnalogIn(mcp, MCP.P0)
```

```
#voltage divider value
divider=0.136
measure = []
for i in range(5):
    measure.append(chan.voltage/divider)
voltage = round(sum(measure)/len(measure),3)   #[V]
```

3.2.8.2 System Status

Depending on the battery voltage level the device status is determined. In the case of low voltage (less than 11.65 V) the datalogger application is terminated (closing signal SIGUSR1 is sent by *stop.sh* script), log-files and database are backed up and the device is set to shutdown.

```
#baterry voltage levels
lowVoltage = 11.65
highVoltage = 12.8

devStatus = 0;
appStatus = 0;
#power OK
if(voltage > lowVoltage):
 #if power is ok and logger app is not running - start again
 if(count < 1):
   subprocess.check_output("/home/pi/logger/run.sh",shell=1)
   appStatus = 1
 else:
   #check NTP synchronization
   ntpStatus=subprocess.check_output("timedatectl",shell=True)
   ntpStatus = ntpStatus.decode(encoding='utf-8')
   ntpStatus = ntpStatus.split('\n')
   ntp = ntpStatus[5][18:]
   if(ntp == 'no'):
     appStatus = 3
   else:
     appStatus = 0

 #battery powered
 if(voltage < highVoltage):
   devStatus = 1

 #power from external power source
 else:
   devStatus = 0
#LOW power
else:
 devStatus = 2
 appStatus = 2
 #stop logger app
 subprocess.check_output("/home/pi/logger/stop.sh", shell=1)
```

```
#create mysql database backup
subprocess.check_output("/home/pi/logger/dbBackup.sh",shell=1)
#compress and move all txt logfiles
subprocess.check_output("gzip /home/pi/ramDisk/*.txt",shell=1)
subprocess.check_output("mv /home/pi/ramDisk/*.gz ...
   ... /media/txt/data", shell=True)
```

The monitoring script uploads following data to the database: timestamp (in unix time), device status (tiny integer), battery voltage in millivolts (integer) and status of logger application (tiny integer).

The script also checks whether the NTP (network time protocol) is synchronized. The NTP synchronization is needed to maintain precise timestamping.
If the NTP is not synchronized the script changes the datalogger app status.

The device status and the datalogger app status can have following values:

Value	Device Status	Logger App Status
0	powered from external power source	running properly
1	battery powered	app restarted
2	battery voltage critically low	app closed
3	—	running but no NTP synchronization

Table 3.1: Values and meaning of System Status and Logger App Status.

In the case of a long-term power failure the battery is eventually disconnected by the brownout circuit. After the power supply is restored the datalogger application is launched during next *cron* execution and data logging continues.

To determine whether the datalogger application is already running or not, the following code is used:

```
runningApps = subprocess.check_output('ps aux | grep ...
   ... "main.py"', shell=True)
apps = output.decode(encoding='utf-8')
apps = out.split('\n')[:-1]
count = 0
for i in apps:
    if(i.find('python3 ./main.py') > 0):
        count += 1
```

During the regular datalogger app execution 3 instances of *main.py* are running (*main.py* and 2 sub-processes *dataProcess.py* and *dataSaver.py*).

3.2.8.3 Database Cleanup

Apart of the saving device status to the MySQL database, the System and Battery monitoring program also deletes old data from the database.

In the table *Data* any record older than 90 days is deleted, in the *DeviceStatus* table any record older than 180 days is deleted:

```
#timestamp 90 days before
dataThreshold = int((timestamp-(90*24*60*60))*10)
#timestamp 365 days before
statusThreshold = int(timestamp-(180*24*60*60))

#delete geomagnetic data older than 90 days from DB
sql = "delete from Data where TimeStampx10 < %s"
val = (dataThreshold)
mycursor.execute(sql, val)
mydb.commit()
#delete gsystem status info older than 180 days from DB
sql = "delete from DeviceStatus where TimeStamp < %s"
val = (statusThreshold)
mycursor.execute(sql, val)
mydb.commit()
```

3.3 Client Software

As described in previous chapters filtered and decimated geomagnetic data are stored in the database without any data correction. The correction (such as orthogonalization, temperature compensation, etc.) is done by the client program that is designed for the data visualization and analysis.

The reason for implementing the data correction and compensation in the client program instead of as a part of the datalogger, was to enable more flexible data analysis. Methods used for correction can change and improve during the time and without original raw data it would be impossible to re-calculate data in a new way.

Two client programs were created: one implemented in MATLAB and the second one in LabView. The MATLAB client is primarily designed for processing and analyzing of archive data.
The LabView client is designed to serve as a "dashboard" - to visualize pseudo-live (with delay) data from geomagnetic observatory.

The LabView was selected as the platform for client program because of its easy accessibility for first-time users. The goal was to make client program as easy understandable as possible, so even user without the knowledge of MATLAB or any other data analysis tool can use it.

The MATLAB client was originally intended only as a testing platform for correction and compensation methods but during the development it has proved to be an useful tool for more detailed data analysis. It was transformed into proper client program with GUI. Because of its flexible structure additional functions and analysis method can be easily added.

Both client programs with configuration files and example logfiles are attached on the CD with software (structure of the CD is described in the Chapter 9).

3.3.1 MATLAB Client

The MATLAB datalogger client program implements MySQL database connection, data correction and data visualization. Client program also enables to analyze txt logfiles with historical data.

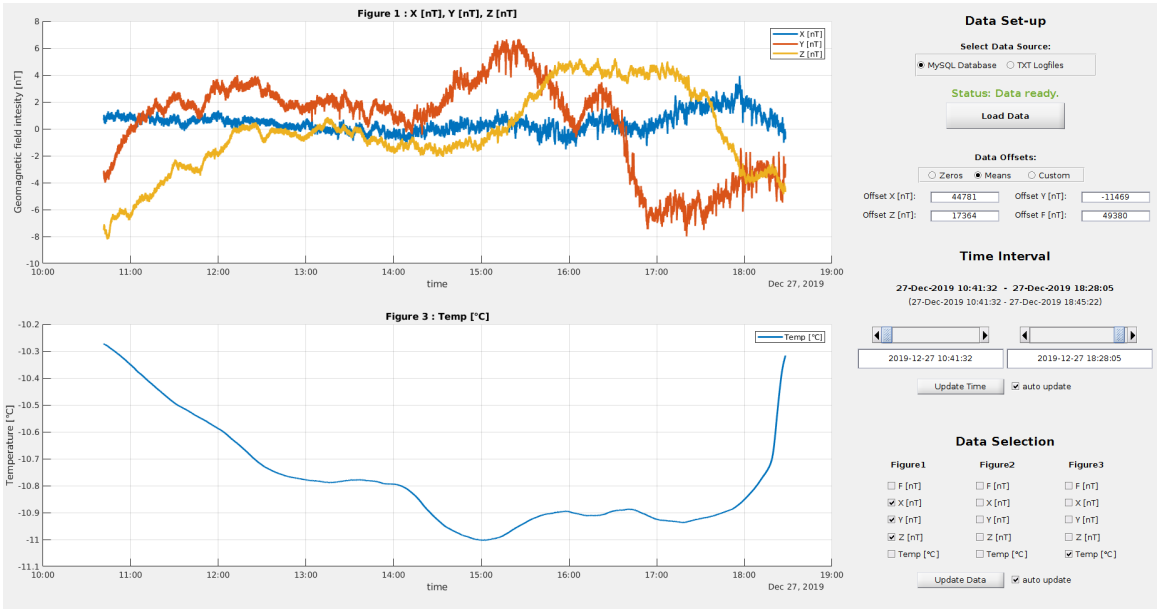

Figure 3.14: MATLAB client program for data analysis.

3.3.1.1 Program Structure

The MATLAB client program is started by running the script *main.m*. This script contains initialization function and callback functions of GUI. Data correction and compensation is implemented as a separate function *dataCorrection.m* to make its editing (and possible replacing with another correction method) easier and more straightforward.

Data correction method can be edited or replaced, but it needs to implement following interface:

```
function [B,X,Y,Z,temp]=dataCorrAlt(dataX,dataY,dataZ,dataTemp)
%DATA CORRECTION AND COMPENSATION of raw filtered data
%    Input:
%        dataX ... X axis raw data [EU]
%        dataY ... Y axis raw data [EU]
%        dataZ ... Z axis raw data [EU]
%        dataTemp ... temperature [EU]
%    Output:
%        F ... total field [nT]
%        X ... X axis field intensity [nT]
%        Y ... Y axis field [nT]
%        Z ... Z axis field [nT]
%        temp ... temperature [Celsius degree]
end
```

The implementation of the default correction method is described in the Chapter 3.3.1.5.

The structure of the main script of client program is shown on Figure 3.3.1.1. During program initialization the GUI is created. After selecting the data source (MySQL database or txt logfiles), data are loaded and corrected. Then it is possible to set the default (maximal) time interval of datasets. This interval is then used during plotting of default dataset (total value of field **F**).
It is possible to set offset for each dataset. Default value of offset is 0, client app also implements function that automatically sets the offsets equal to mean values of datasets.

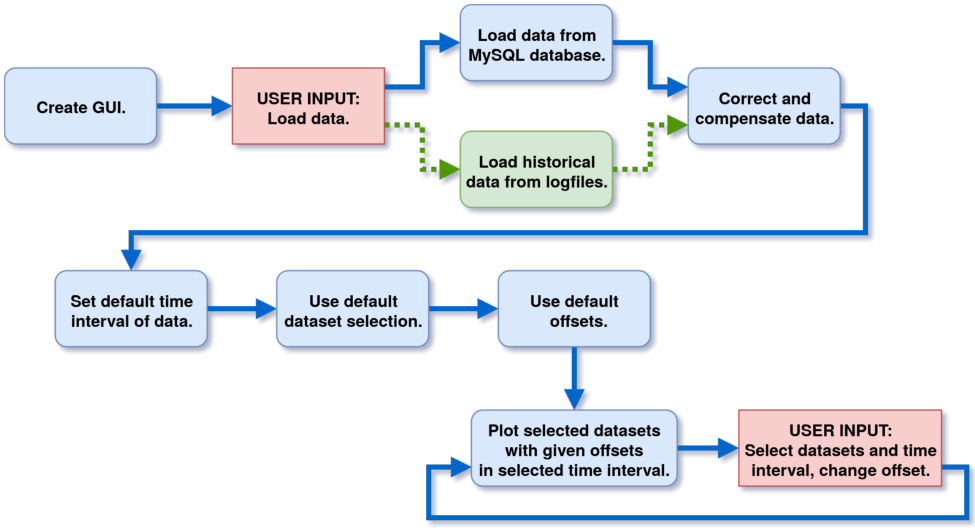

Figure 3.15: Matlab client program structure - main script.

3.3.1.2 Graphical User Interface

The GUI of MATLAB client consist of 4 main parts - figures and 3 control panels: data loading and offsets control panel, time interval selection panel and dataset visualization control panel.

Data loading control panel (Figure 3.3.1.2) enables to select the source of data (txt files or MySQL database), to load data and to set the offset for each dataset.

When logfiles are selected as the datasource dialog windows will appear to guide the user through the loading mode selection (log single logfile or all logfiles in folder) and the directory selection.

User can also choose between 3 options of offset: Zeros, Means and Custom.
Zeros and Means options sets the values of offsets to zeros or mean values of each dataset. Custom option enables to change offsets manually.

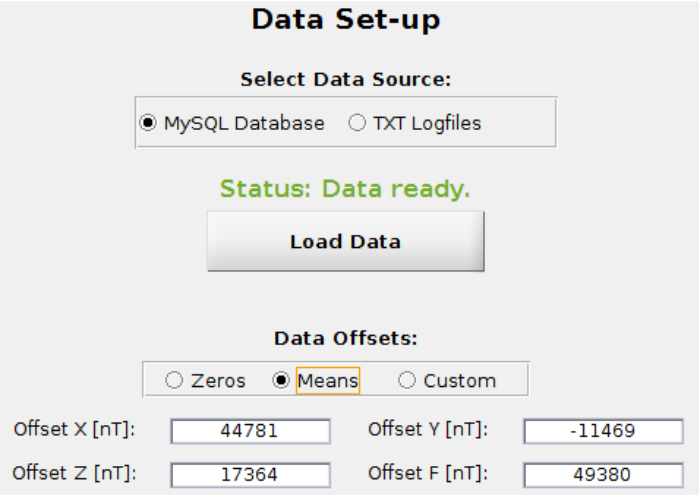

Figure 3.16: Matlab client GUI - data source selection and offset control.

Time interval selection panel (Figure 3.3.1.2) enables user to select desired time interval for displayed data. This part of GUI embody text label with currently selected time interval (bold text), static text label with default/maximal time interval and controls to set the new interval.

There are two possible ways to set the time - with slider (the left slider is controlling beginning of interval and the right slider is controlling the end of interval) or with text fields (there is currently set time in this fields so it is easy to edit it).

Data input from UI controls is checked and corrected so it is impossible to set time interval outside the default interval or e.g. set the beginning later in the time than end of interval. This check is done by function *updateTime*.

Unless the checkbox *auto update* is selected it is necessary to press the *Update Time* button to update time interval.

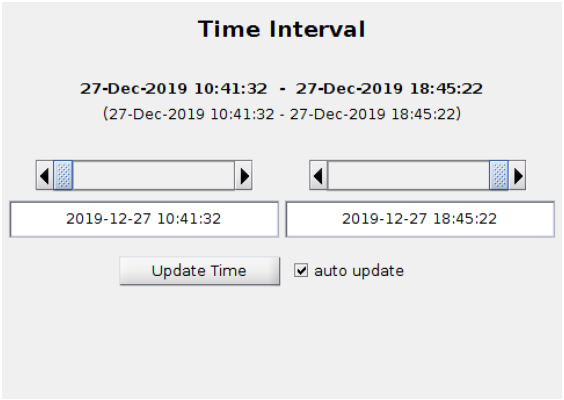

Figure 3.17: Matlab client GUI - time interval selection.

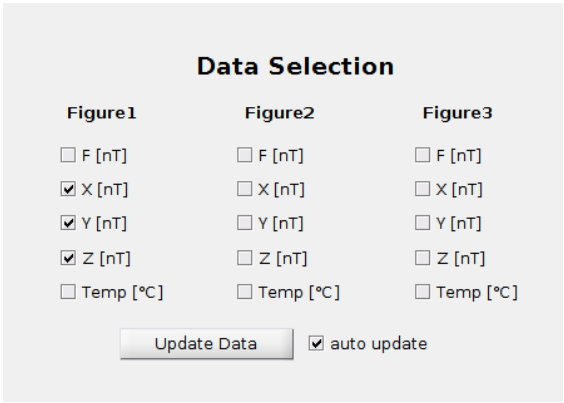

Figure 3.18: Matlab client GUI - time data selection.

Dataset selection panel (Figure 3.3.1.2) consist of 3 columns of check boxes. Each col-

umn controls one figure. It is possible to display any combination of datasets in any figure. If there is no dateset selected in figure column, this figure will not be displayed. Based on this selection one, two or three figures will be created in figure area.

Similarly to time interval selection the button *Update data* has to be pressed to update datasets in figures - unless the checkbox *auto update* is selected.

Selection of dataset is processed in callback function *selectData* and the data visualization is done by function *plotData*.

3.3.1.3 Database Connection

To establish connection to the MySQL database the MATLAB Database Toolbox and the *mysql-connector-java-5.1.48* connector are used.

Database connection is implemented in dedicated function:

```
function connect2DB(~,~)
 statusLabel = findobj('Tag', 'statusLabel');
 statusLabel.String = 'Status: Connecting to the DB.';
 conn = database('datalogger','maglab','password',...
     'com.mysql.jdbc.Driver',...
     'jdbc:mysql://10.42.0.11:3306/datalogger');

 loadedDB = 1;
 fig = findobj('Tag', 'client');
 setappdata(fig,'loadedDB',loadedDB);
 setappdata(fig,'dbConn',conn);
end
```

This function is called during the data loading process (unless the database connection has been already created):

```
try
  if(loadedDB)
    conn = getappdata(fig,'dbConn');
  else
    statusLabel.String = 'Status: Connecting to the DB.';
    statusLabel.ForegroundColor = [0.446 0.674 0.188];
    connect2DB()
    conn = getappdata(fig,'dbConn');
  end
  data = select(conn,'select * from Data');
  % parse data
  statusLabel.String = 'Status: Parsing data.';
  timeDecim = datetime(double(table2array(data(2:end,...
    ...1)))./10,'ConvertFrom','posixtime','TimeZone',...
    ...'Europe/Zurich','Format','yyyy-MM-dd HH:mm:ss.SSSSSS');
  dataX = double(table2array(data(2:end,2)))./100;
  dataY = double(table2array(data(2:end,3)))./100;
  dataZ = double(table2array(data(2:end,4)))./100;
```

```
  dataTemp = double(table2array(data(2:end,5)))./100;
  connStatus = 1;
catch error
  disp(error)
  statusLabel.String='ERROR: Unable to connect to database.';
  statusLabel.ForegroundColor = [0.635 0.078 0.184];
end
```

3.3.1.4 Data from Logfiles

MATLAB client also enables to load data from *.txt* logfiles instead of database. Because of this function client can be used also for analysis of historical data or data from another measurement stations.

If the logfiles are selected as the datasource the user has to decide whether to load single logfile or all logfiles in the folder (there has to be only logfile in selected folder).

```
try
  statusLabel.String = 'Status: Loading data from logfile.';
  statusLabel.ForegroundColor = [0.446 0.674 0.188];

  loadType=questdlg('Load single TXT logfile or all logfiles...
    ...in the folder?','Load logfiles','Single file',...
    ...'Folder','Single file');

  switch loadType
    case 'Single file'
    [file,path] = uigetfile('*.txt');

    fp = fopen([path,file],'r');
    formatSpec = '%f;%f;%f;%f;%f;';
    data = fscanf(fp,formatSpec,[5 Inf]);
    data = data';

  case 'Folder'
    path = uigetdir;
    [~,files] = system(['ls ',path]);
    files = files(1:end-1);
    files = strsplit(files);
    fNames = sort(files);

    formatSpec = '%f;%f;%f;%f;%f;';
    data = [];
    for i=1:length(fNames)
      fp = fopen([path,filesep,fNames{i}],'r');
      partData = fscanf(fp,formatSpec,[5 Inf]);
      partData = partData';
          data = vertcat(data,partData);
    end
```

```
  otherwise
    return;
  end
```

After loading the data are edited (datastamps are converted from the Unix time to datetime format and data are divided by 100 to get actual values in double data type) and saved.

3.3.1.5 Data Correction

MATLAB client implements offset adjustment, sensitivity correction, orthogonalization, field compensation and thermal drift compensation.

Correction parameters are load from text file. By default, the file *corrDetails.txt* is placed in the same directory as *dataCorrection.m* function and is used. If this file is not found, a dialog box is displayed and user can manually select the file with correction parameters.

```
try
    info = importdata('corrDetails.txt');
catch
    [file,path] = uigetfile({'*.txt'},'Select File with ...
      ... Correction Parameters');
    info = importdata([path,file]);
end

corrParams = info.data;

%offsets [EU]
O = corrParams(1:3);
%sensitivity [T/EU]
S = [corrParams(4),0,0; 0,corrParams(5),0; 0,0,corrParams(6)];
%orthogonalization matrix
P = [corrParams(7:9)'; corrParams(10:12)'; corrParams(13:15)'];
%compensation field [T]
C = corrParams(16:18);

tempOff = corrParams(19);          %temperature offset [C]
tempGain = corrParams(20);     %temperature gain [C/EU]
refTemp = corrParams(21);              %rederence temp [C]
tempDrifts = corrParams(22:24); %temp drift constants [nT]
```

The file with correction parameters contains following values, each on separate line:

```
#offX[EU];offY[EU];offZ[EU];
#gainX[T/EU];gainY[T/EU];gainZ[T/EU];
#orthoA11;orthoA12...orthoA32,orthoA33;
#Cx[T];Cy[T];Cz[T];
#tempOff[C];tempGain[C/EU];refTemp[C];
#tempDriftX[nT];tempDriftY[nT];tempDriftZ[nT];
```

Data correction is implemented as is described in Chapter 2.4.1: with use of equation 2.7, orthogonalization matrix 2.9 and correction parameters 2.6:

```
%% offsets, sensitivity and orthogonalization

%offsets [EU]
O = [3.434531398e+03; -85.12483;2.585290932e+03];

%sensitivity [T/EU]
S = [-8.143760169520512e-12,0,0; 0,8.223217597027799e-12,0; ...
    ... 0,0,8.177029416045621e-12];

%orthogonalization matrix
P = [1,0,0; -0.002625330292398,1.000003446173634,0; ...
    ... 0.004190527854485,0.007839927070293,1.000039596867103];

%compensation field
C = [0;0;0];

ortoX = zeros(length(dataX),1);
ortoY = zeros(length(dataY),1);
ortoZ = zeros(length(dataZ),1);

for i=1:length(dataX)
    F = [dataX(i);dataY(i);dataZ(i)];
    B = P*S*(F-O);

    ortoX(i) = B(1)+C(1);
    ortoY(i) = B(2)+C(2);
    ortoZ(i) = B(3)+C(3);
end
```

Corrected geomagnetic data are then compensated for thermal drift. Compensation is implemented according to theory in Chapter 2.4.2:

```
%% temperature drift compensation

%transforamtion from EU to Celsius degree
temperature = dataTemp*6.648e-5 - 2.599e2;

%temperature of calibration
refTemp=4.7;

%temperature delta
tempDelta = temperature-refTemp;

%temperature drift constants [nT/K]
offDrifts = [1.2243;-3.2482;1.8257]*1e-9;

X = ortoX - offDrifts(1).*tempDelta;      % [T]
```

```
Y = ortoY - offDrifts(2).*tempDelta;      % [T]
Z = ortoZ - offDrifts(3).*tempDelta;      % [T]

X = X*1e9;  %conversion from [T] to [nT]
Y = Y*1e9;  %conversion from [T] to [nT]
Z = Z*1e9;  %conversion from [T] to [nT]

B = sqrt(X.^2+Y.^2+Z.^2);     % [nT]
```

Whole correction and compensation process is implemented as function *dataCorrection.m* that is called by client program.

3.3.2 LabView Client

The LabView Client program was developed as an more accessible and portable (in an executable form) client for data visualization and analysis. It enables to to analyze current geomagnetic data in pseudo real-time (new data are periodically loaded from database) and it is intended to be run 24/7 as a dashboard software to display the variometer output and health status.

It was implemented in the LabView 8.6 to be compatible with all Windows OS (this version is compatible with the old devices with the Windows XP and also with the new devices with the Windows 10).

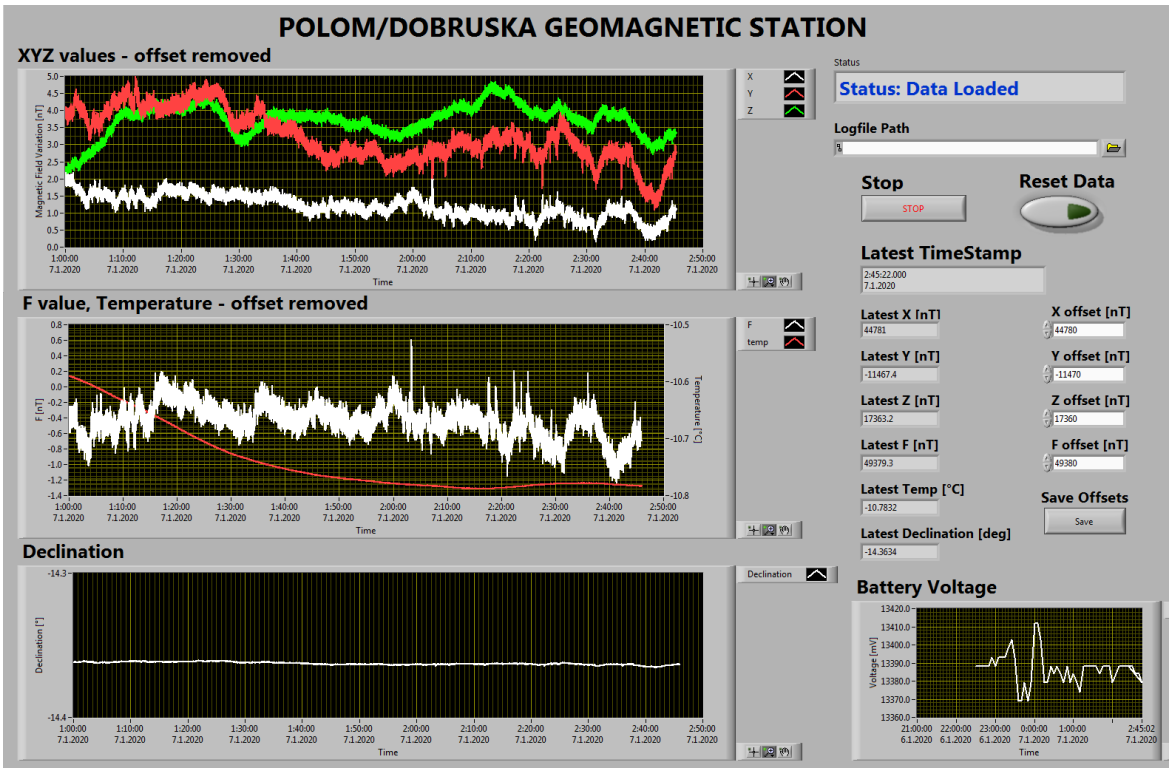

Figure 3.19: labView Client for data analysis.

3.3.2.1 Program Structure

The client program can operate in two modes depending on its data source: real-time data (datas ource is the MySQL database) or archive data (data source is the txt logfile). To swith between modes it is necessary to tun program again.

As is shown in the Figure 3.3.2.1, user chooses data source at the beginning of the program execution, right after offsets are loaded from file.

If the archive data mode is selected, user selects directory of txt logfile, raw data are loaded, corrected and visualized.

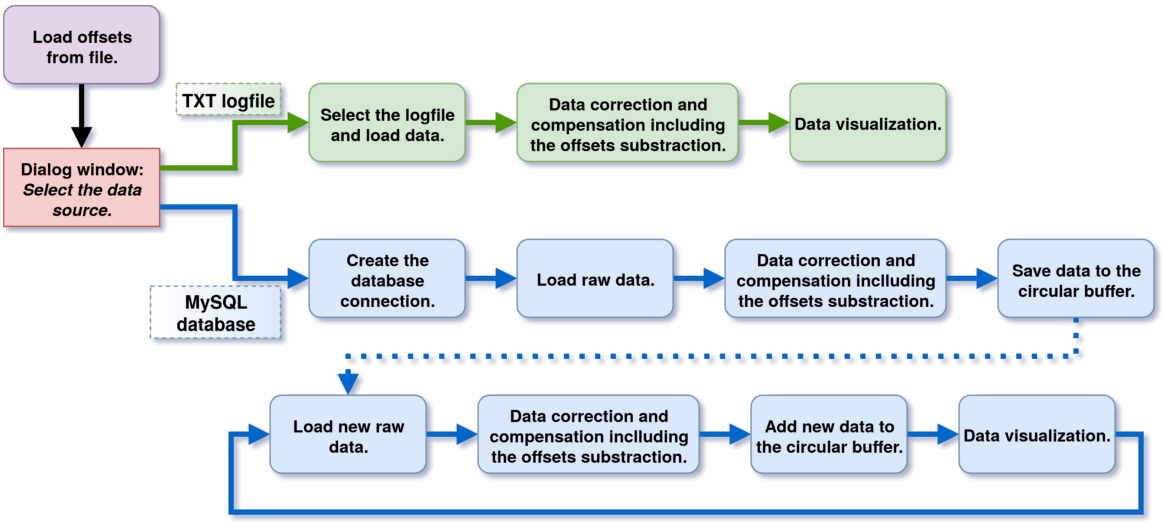

Figure 3.20: Structure of the LabView Client.

In the case of real-time data mode the database connection is created, limited amount of archive data (last 5 hours) is loaded from database, corrected and visualized. Then new data are downloaded periodically, corrected and added to the circular buffer with archive data.

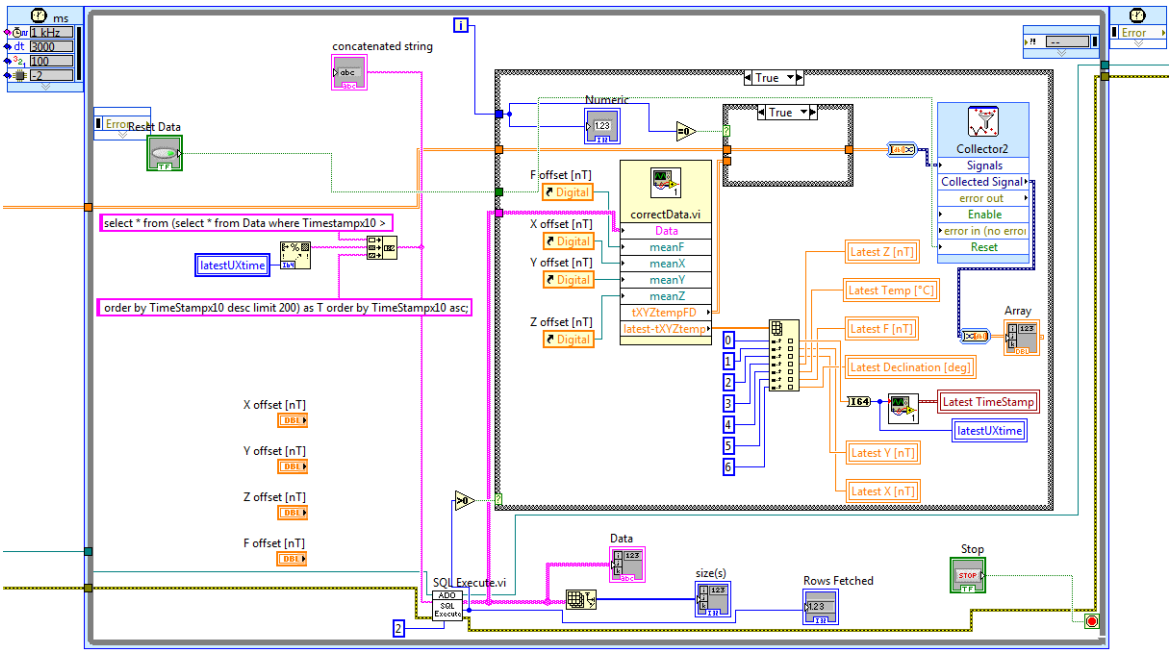

Figure 3.21: LV Client - main loop for acquisition of new data.

3.3.2.2 Database Connection

To connect to the MySQL database the collection of LabView VIs called LabSQL [27] is used. This VIs enable to connect to the database and to execute SQL query with the use of Windows ODBC API. That means that PC that is running the LV Client need to have installed proper MySQL drivers and the DSN (data source name) needs to be set in the ODBC Data Source Administrator.

To set up the ODBC source the user have to open ODBC manager and add a new User DSN. After selecting the driver (in our case the MySQL ODBC Unicode Driver), the MySQL Connector/ODBC window appears and it is necessary to fill in all connection parameters.
After the setup, the name of the DSN is used as a parameter for LabSQL Conncection Create VI.

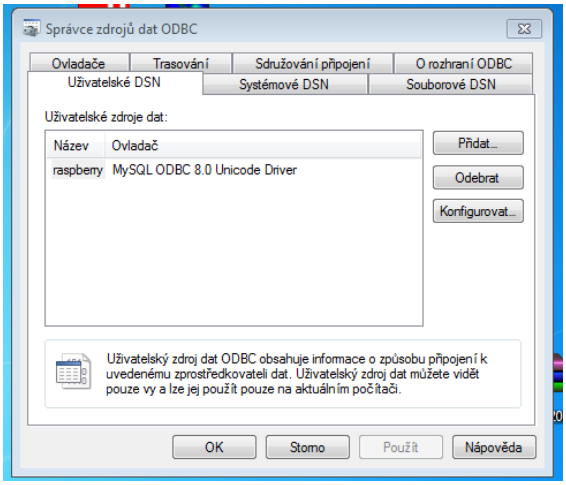

Figure 3.22: Setup of the ODBC.

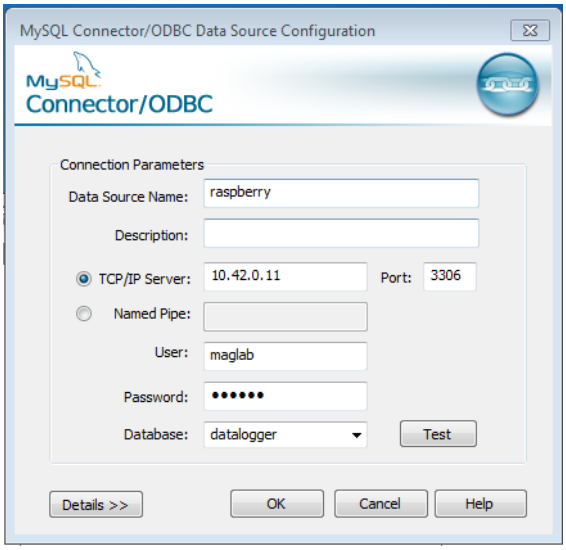

Figure 3.23: ODBC - connection details.

Database connection is created at the beginning of program execution. Archive data from database (limited amount of data up to the current time) are loaded in one SQL query and then the update loop is entered. In this loop new data are periodically downloaded from database and analyzed.

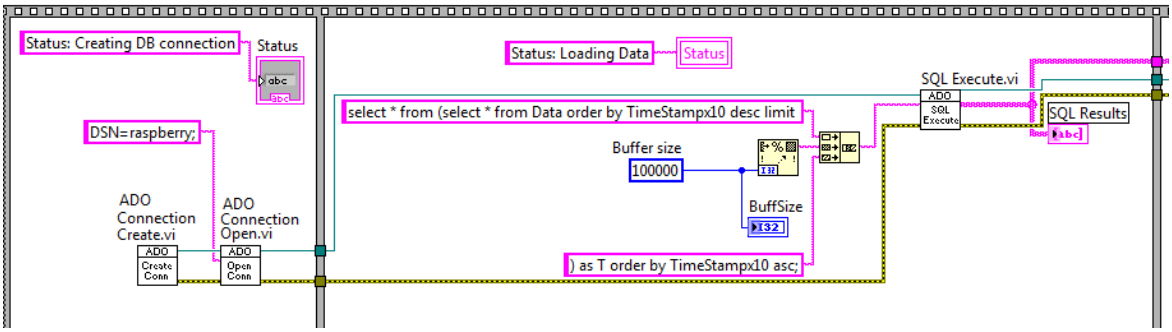

Figure 3.24: MySQL database connection and SQL query execution.

3.3.2.3 Data Correction

Similarly to the MATLAB client the LabView Client implements correction and compensation of data according to the Chapter 2.4.
Data correction procedures are implemented in a separate VI called *correctData.vi*

Correction parameters are load from the *corrDetails.txt*. It is a plain file only with correction parameters - each parameter on one line. Parameters are in the file in following order:

```
offX [EU]; offY [EU]; offZ [EU];
gainX [T/EU]; gainY [T/EU]; gainZ [T/EU];
orthoA11; orthoA12 ... orthoA32, orthoA33;
Cx [T]; Cy [T]; Cz [T];
tempOff [C]; tempGain [C/EU]; refTemp [C];
tempDriftX [nT]; tempDriftY [nT]; tempDriftZ [nT];
```

The output of data correction and compensation method contains array of processed datasets: X,Y and Z values without offset (in nT), temperature (in Celsius degrees), total value of geomagnetic field FF and computed value of declination D.

3.3.2.4 Graphical User Interface

The GUI of LV client program consists of 4 charts (Figure 3.3.2.4 + chart with the voltage of the battery) and control panel (Figure 3.3.2.4).

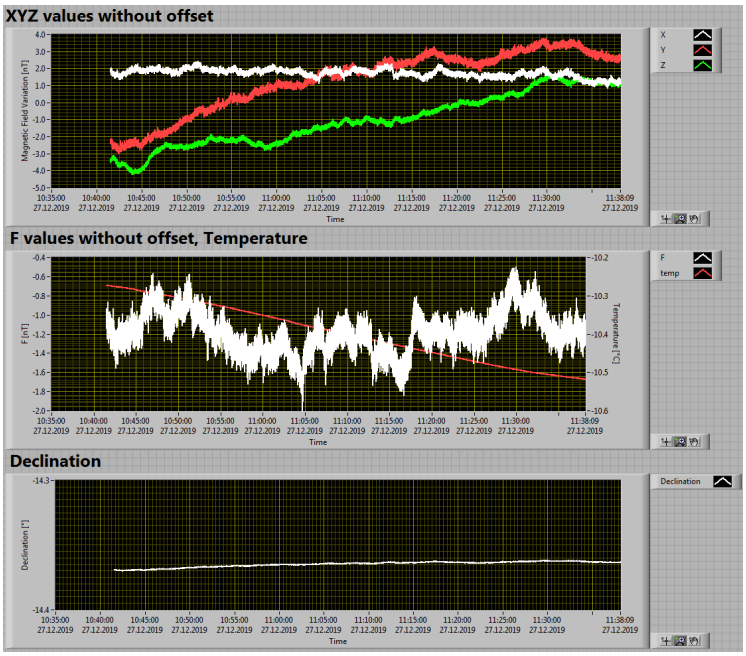

Figure 3.25: LabView Client - charts.

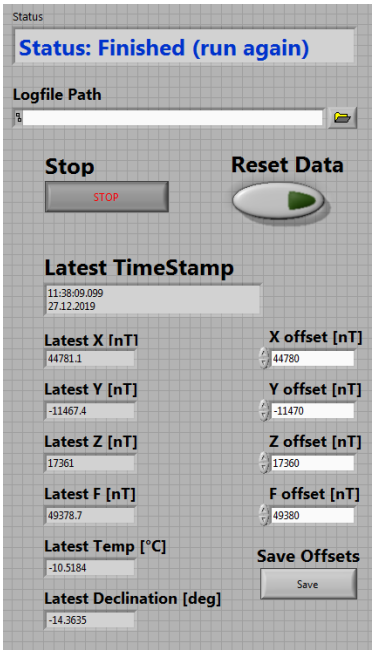

Figure 3.26: Control panel.

The first chart displays X, Y and Z component of geomagnetic field without offsets (which can be customized in control panel) in nT. The second chart displays total value of geomagnetic filed F and temperature (this chart has two scales - on the left

side there is a scale for F in nT, on the right side there is a scale for temperature in Celsius degree). The third chart displays value of declination D that is computed from obtained data.

The control panel consist of status label (indicating the status of data loading and processing), editable field for directory of logfile (if this field is empty a dialog window will appear), control buttons (Stop button to stop whole program and Reset Data button to clear all data in charts), newest/latest values and editable fields with currently used offset values.

Latest values of time, geomagnetic field, temperature and declination are the newest data in the MySQL database and are updated every loop execution.

User can edit the offset of each field component (X, Y and Z) and the offset total value of geomagnetic field F. The offsets are implemented to make data more readable and clear.
The default values of offsets are loaded at the start of the program from file *dataOffsets.txt*. Edited offset values can be saved to the default file with press of *Save Offsets* button and will be used in the next run of the program.

Chapter 4

Device Testing

During the development of the datalogger it was necessary to test it. In the early stages of development the simulated data were used (both artificial and historical data). To simulate data stream with parameters similar to the real sensor the Sensor Simulator was developed.

Later in the development - after solving all errors and bugs that occurred, real geomagnetic sensor was used for testing as well.

4.1 Datalogger with Simulated Data

To test the datalogger with various datasets with various parameters the sensor simulator was developed.

The simulator consist of Raspberry Pi 3 computer, serial connection interface (using MAX 3232 Maxim similarly as datalogger) and a Python script.

The simulator is connected to the datalogger's serial port and act in the same way as the real magnetometer. This setup enables to test the performance of the datalogger in controlled situations (with various types of signal, various signal frequencies, etc.).

Used hardware setup is shown on Figure 4.1. During the device testing the Arbitrary Waveform/Function Generator (Keithley 3390) was used for PPS signal simulation.

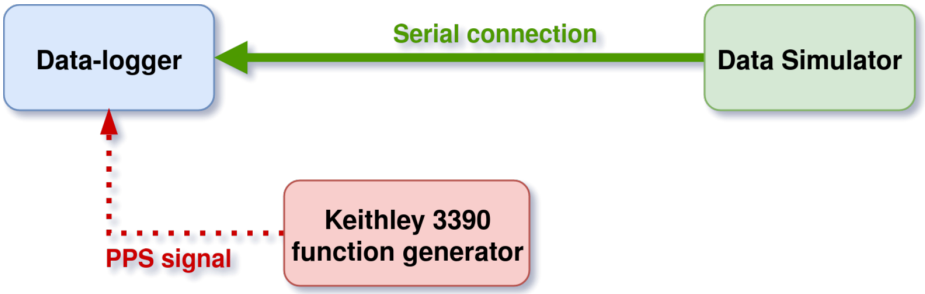

Figure 4.1: Datalogger in setup with data simulator and function generator.

The simulator enables to generate both artificial and historical data. Artificial data such as step and ramp signals were used to debug datalogger software.

Historical data (from long-term measurement at Polom/Dobruska station) were used to test device performance.

4.1.1 Sensor Simulator

As described, the sensor simulator is the Raspberry Pi 3 computer with a hardware serial interface. From the datalogger point of view, the simulator appears to be the same as the real magnetometer.

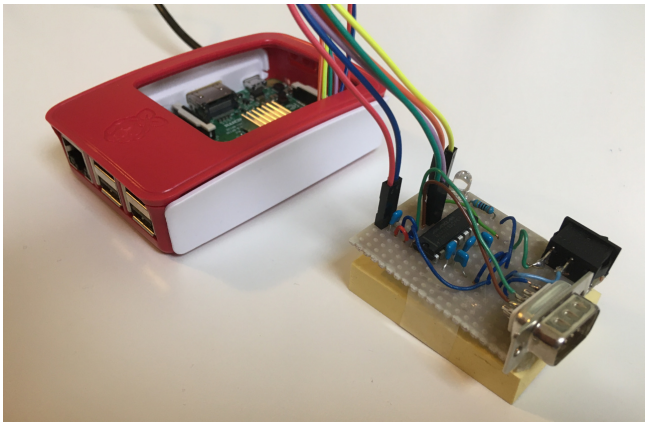

Figure 4.2: Sensor simulator with serial connection interface.

The wiring of serial interface circuit is similar to datalogger interface (see Chapter 3.1.2). It implements MAX 3232 Maxim line drive/receiver connected to the Raspberry Pi GPIO. Simulator also implements few minor additional hardware functions: the simulation is controlled with mechanical switch and the running simulation is signalized with LED.

The simulator script is implemented in Python and uses Python's *time* library to achieve desired sample rate (206.5 Hz).

In its initialization phase it sets GPIO pins and loads names of all data files in dedicated folder *./data/*. Then the first data file is loaded and simulator is ready to transmit data.

```
import serial
import time
import datetime
import subprocess
import RPi.GPIO as GPIO

#initiate GPIO pins
try:
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(18,GPIO.OUT)
    GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
    GPIO.output(18,GPIO.LOW)
except Exception as e:
```

```
    print("unable to initiate GPIO pins")
    print(e)

#load all data file names
ls=subprocess.check_output(["ls", ...
   ... "./data/"]).decode("utf-8")[:-1]
fileNames = ls.split("\n")
fileIndex = 0
maxFileIndex = len(fileNames)
print(maxFileIndex)

#open first data file
fp = open('./data/'+fileNames[fileIndex],'r')

#open serial port
try:
    portRXTX = serial.Serial("/dev/ttyS0",baudrate=115200, ...
        ... timeout=0.002)
    print("simulator ready")
except Exception as e:
    print("unable to open serial port")
    print(e)
```

During the execution of the main loop of the program the status of the switch is checked and if the switch is on, the simulator is activated. To achieve precise sample rate program sends data and then waits in short sleeps until the next transmission. This solution deals with possibility of uneven transmission times. Interval between transmissions is 0.00484 sec which means the sample rate is 206.5 Hz.

```
#simlulator loop-simulator will start after turning switch ON
while(1):
   #set timestamp
   tic = time.time()
   #check switch status
   input_state = GPIO.input(17)

   #if switch is ON -> run simulator
   if(input_state == False):
      if(led == 0):
         led = 1
         GPIO.output(18,GPIO.HIGH)
         print("running")

      line = fp.readline()
      # line example: "-5493236;-1374885;+2110060;+3755107;"

      #if its end of the file
      if(line == ''):
         fileIndex += 1
         if(fileIndex == maxFileIndex):
```

```
            fileIndex = 0

        fp = open('./data/'+fileNames[fileIndex],'r')
        line = fp.readline()

    data = line[:-1]    #remove "\n"

    #send data
    print("sending data")
    try:
        portRXTX.write(bytes(str(data)+'\r\n','utf-8'))
    except Exception as e:
        print("unable to send data - waiting 5 secons")
        print(e)
        led = 0
        GPIO.output(18,GPIO.LOW)
        time.sleep(5)

    #variable wait to make frequency precise
    while(time.time()-tic < 0.00482):
        time.sleep(0.00002)

#if switch is OFF
else:
    if(led == 1):
        led = 0
        GPIO.output(18,GPIO.LOW)
        print("stopped")
    time.sleep(0.5)
```

4.1.2 Test Results

Sensor simulator was crucial in the debugging process during the datalogger application development. It helped to solve problems with data filtering and application speed.

4.2 Laboratory Tests

Later in the development of the datalogger the simulator was replaced with proper fluxgate variometer and the function generator was replaced with GNSS receiver.

The purpose of laboratory measurements was to test device in its final setup. The goal was to not only test datalogger application and its performance but also to test device's hardware and overall performance of the datalogger.

4.2.1 Measurement Setup

During the laboratory measurements the developed device was debugged and tested in its final setup: datalogger connected to the fluxgate variometer and the GNSS receiver (with PPS signal).

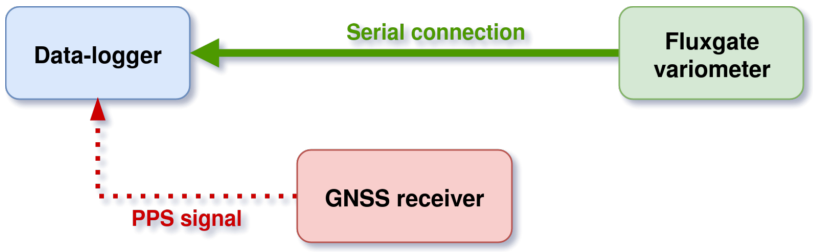

Figure 4.3: Datalogger in setup with fulxgate variometer and GNSS receiver.

Variometer and GNSS receiver were tested separately at first (with use of function generator and simulated data) to resolve eventual problems. After separate testing the device was tested in final setup.

4.2.2 Long-term Measurement Test

To thoroughly test performance of datalogger app, MySQL database and also client programs long-term measurement tests were conducted.

During these tests both real sensor data and simulated historical data were used to detect and eliminate bottlenecks and bugs in applications.

After several code modifications (such as change of read method for serial connection described in Chapter 3.2.4) programs were optimized and stable performance of datalogger was achieved.

An example of long-term data is shown in the Figure 4.2.2. Offsets of shown data were following:

$$X_{off} = 44\,781\,\text{nT} \quad Y_{off} = -11\,489\,\text{nT} \quad Z_{off} = 17\,339\,\text{nT} \quad F_{off} = 49\,376\,\text{nT}$$

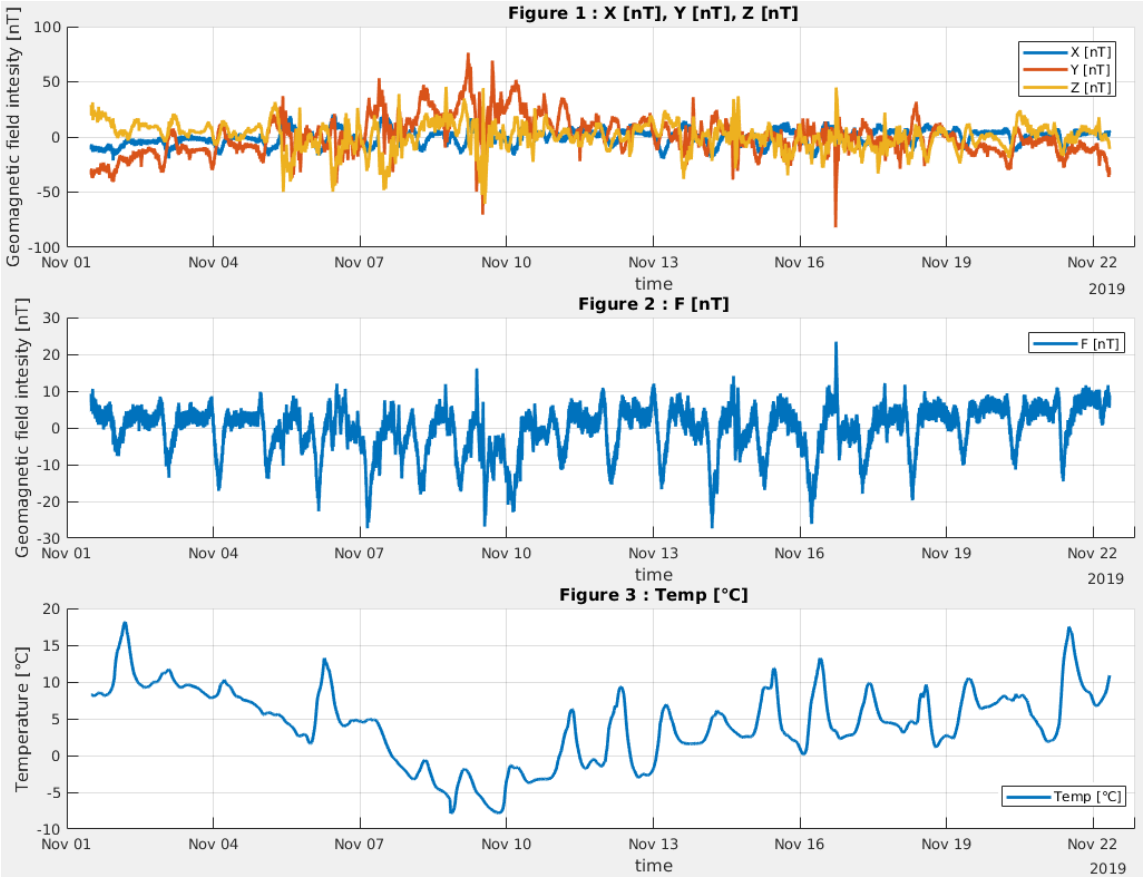

Figure 4.4: Example of long-term data in MATLAB client program.

4.2.3 Device Performance Test

Performance tests of the device were focused on the usage of the CPU, the power consumption and the battery performance during failure of main power source.

As is shown in Table 4.2.3 datalogger application uses 71% of device's CPU and circa 5.5% of its memory.
The highest usage of CPU has the *main.py* process (reading raw data from sensor) followed by the *dataSaver.py* process (saving filtered data to txt log-files and to the MySQL database). The *dataProcess.py* process (filtration and processing of raw data) has the lowest CPU consumption.

Process Name	CPU Usage	Memory Usage
main.py	38%	2.1%
dataProcess	8%	1.6%
dataSaver.py	25%	1.7%
Sum	*71%*	*5.4%*

Table 4.1: Datalogger application - CPU and memory usage.

To measure device's power consumption an ammeter placed between power source and DC/DC converter was used. From obtained values of current the power consumption was computed.

Device Stata	Current [mA]	Power [W]
Idle state	240	1.2
Active logging	260	1.3

Table 4.2: Power consumption of datalogger device.

To measure the performance of backup battery a main source power failure was simulated. The device continued in data logging until the level of battery voltage dropped bellow set threshold (11.65 V for device monitor program and 11.6 V for analog brownout circuit). Then the datalogger application was stopped, data were saved and device was turned off (software shutdown is followed by the battery cut-off by the brownout circuit).
Power source was restored shortly after shutdown of the system.

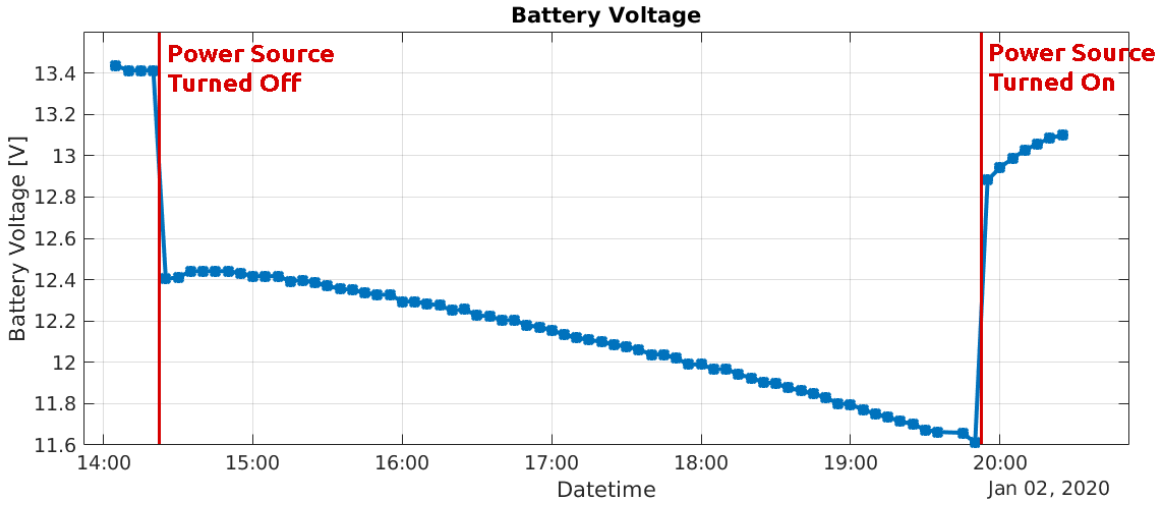

Figure 4.5: Battery voltage level during the failure of the main power source.

As is shown in Figure 4.2.3 the datalogger battery can safely operate as a backup power source for 5.5 hours (then the voltage drops bellow 11.65 V and device is turned off).
The average battery voltage decrease during measurement was 150 mV per hour.

Chapter 5

Conclusions

The goal of the thesis was to develop a time-synchronous datalogger with GUI for geomagnetic observatories/repeat stations. Device is supposed to be based on the Raspberry Pi platform and implements a MySQL database as a storage for filtered and timestamped data. Data from the MySQL database are accessed and processed with developed client program. The device should also supposed to implement backup solutions.
These goals was completed.

The required hardware of datalogger was created. The developed device consist of Raspberry Pi 2B, power circuit with backup battery and multiple communication interfaces (serial connection, connection to GNSS receiver).

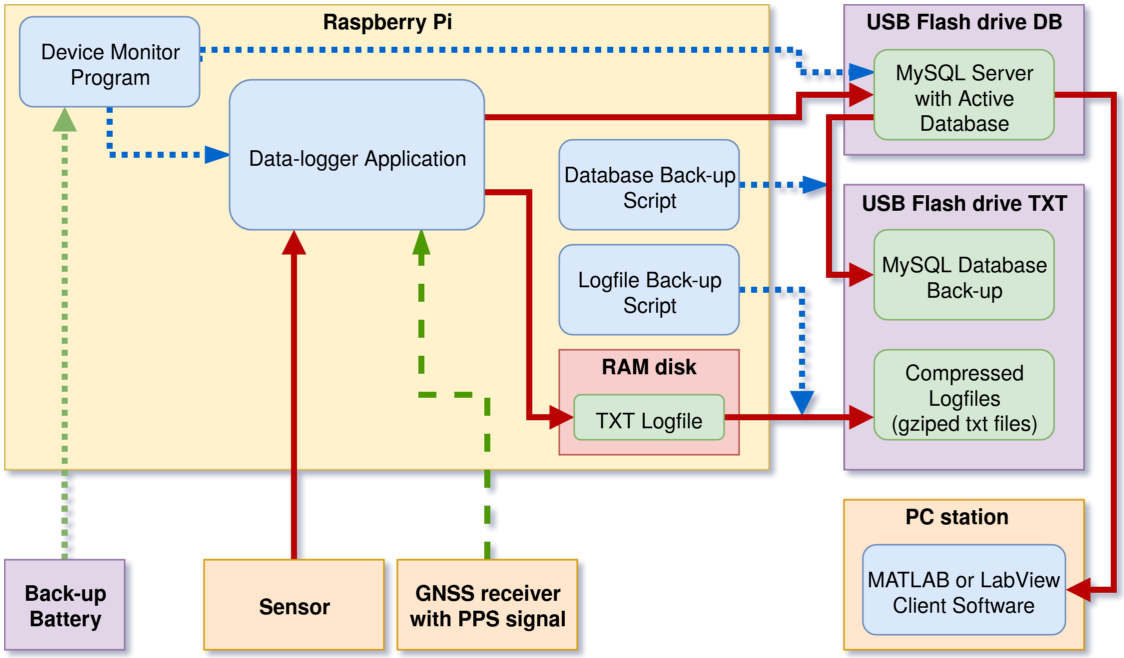

Figure 5.1: Logical scheme of the developed datalogger.

The datalogger application receives raw data from the magnetometer, process them (filtration and sample rate conversion) and saves filtered (but uncorrected and uncompensated) data to the MySQL database and also to plain text logfiles. Timestamping is implemented as an interrupt driven callback triggered by the PPS signal from the

GNSS receiver.

Both MySQL database and archive of compressed log-files are located on external USB flash rives (to reduce wear-out of the system SD card) and are periodically backed-up by scripts called by *cron* scheduler.

The power source failure prevention is implemented both on software and hardware level. The device monitor program monitors battery voltage, the status of the device and it also controls the performance of the datalogger.
The analog brownout circuit prevents critical discharge of the backup battery.

To achieve more flexibility in the data processing, the correction and compensation of filtered data is implemented not on the datalogger but as a separate Client program for PC stations.
Two client program were created - the MATLAB client and the LabView client. Both clients enables to review and analyze current data from MySQL database and historical data from logfiles. LabView client also implements "Live mode" enabling to review data in pseudo real-time (with delay).

All parts of the thesis assignment were achieved.The developed device is fully functional and ready for a deployment at the geomagnetic observatory Polom/Dobruska. The observatory is currently under the construction and the deployment of the device is planned after the stabilization of the sensor head is finished.

Although the datalogger is complete, there are still ways to further improve the device and its functions. For example, there is an opportunity to improve device performance by implementing critical parts of the code as an extension module in C programming language. Another possible improvement of the datalogger is to implement synchronization of the system time with the time of the GNSS receiver (it would be more flexible than the current NTP synchronization).

Chapter 6

Bibliography

[1] JANKOWSKI, Jerzy; SUCKSDORFF, Christian. *Guide for magnetic measurements and observatory practice.*, Boulder: International Association of Geomagnetism and Aeronomy, 1996, ISBN 0-9650686-2-5.

[2] OLSEN, Nils, et al. *In-flight calibration methods used for the Oersted mission.*, Technical University of Denmark, unpublished, 2001.
[seen 7.3.2019] Source: https://www.researchgate.net/profile/Nils_Olsen/publication/2596023_In-Flight_Calibration_Methods_Used_For_The_Orsted_Mission/links/00b7d529d81d0eba14000000/In-Flight-Calibration-Methods-Used-For-The-Orsted-Mission.pdf

[3] MONK, Simon. *Raspberry Pi Cookbook: Software and hardware problems and solutions.*, O'Reilly Media, Inc., 2013, ISBN 978-1-449-36522-6.

[4] ALBERSHTEYN, Andrey. *Data recorder for observatory magnetometer.*, Prague 2016, Bachelor's thesis, CTU in Prague, Faculty of Electrical Engineering, Department of Measurement.

[5] JANOŠEK, Michal, et al. *Improving Earth's Magnetic Field Measurements by Numerical Corrections of Thermal Drifts and Man-Made Disturbances.*, Hindawi Journal of Sensors, Volume 2018, Article ID 1804092, 10 pages, 2018.
[seen 7.3.2019] Source: https://doi.org/10.1155/2018/1804092

[6] MORSCHHAUSER, Achim, et al. A low-power data logger system for 1 second INTERMAGNET data. In: *CobsJournal 5 - Special Issue IAGA Workshop 2018.*, Zentralanstalt für Meteorologie und Geodynamik, 2019. Page 17.
[seen 7.3.2019] Source: http://www.conrad-observatory.at/zamg/index.php/downloads-en/category/5-cobsjournal

[7] MORSCHHAUSER, Achim, et al. A low-power data acquisition system for geomagnetic observatories and variometer stations. In: *Geoscientific Instrumentation, Methods and Data Systems.*, 2017, Issue 6, Page: 345-352.
[seen 1.12.2019] Source: https://www.geosci-instrum-method-data-syst.net/6/345/2017/gi-6-345-2017.pdf

[8] FÚRA, Viktor. *Datalogger for Vector Magnetometer.* Prague, 2014. Bachelor's thesis, CTU in Prague, Faculty of Electrical Engineering, Department of Measurement.

[9] ST-LOUIS, B. J., et al. *INTERMAGNET Technical Reference Manual, version 4.6.*, INTERMAGNET Operations Committee and Executive Council, 2012 [seen 7.3.2019] Source: http://www.intermagnet.org/publications/intermag_4-6.pdf

[10] BISHOP, Charles. *Effects of averaging to reject unwanted signals in digital sampling oscilloscopes.*, In: 2010 IEEE AUTOTESTCON. IEEE, 2010. p. 1-4. [seen 7.3.2019] Source: https://ieeexplore.ieee.org/abstract/document/5613545

[11] HRVOIC, Ivan; NEWITT, Lawrence R. *Instruments and methodologies for measurement of the Earth's magnetic field.*, In: Geomagnetic Observations and Models. Springer, Dordrecht, 2011. p. 105-126.

[12] Avisaro AG. *Avisaro Data Logger Box 2.0 - M21133: with RS232 interface and D-Sub-connector.*, Product Flyer, 2012. [seen 7.3.2019] Source: http://www.avisaro.com/files/Avisaro/20_Flyer/C23766_Flyer_ENG.pdf

[13] ELSASSER, Walter M. *Hydromagnetic dynamo theory.* Reviews of modern Physics, volume 28, number 2. 1956, p. 135-163.

[14] *Geomagnetic observatory Budkov.* Institute of Geophysics of the Czech Academy of Science. [seen 7.3.2019] Source: https://www.ig.cas.cz/en/observatories/geomagnetic-observatory-budkov/

[15] *Dobruska/Polom (DPC).* Institute of Geophysics of the Czech Academy of Science. [seen 7.3.2019] Source: https://www.ig.cas.cz/en/observatories/czech-regional-seismic-network/dobruska-polom-dpc/

[16] LAŽA, Libor. *Čtyři dekády na stanici Polom.* In: Vojenský geografický obzor, volume 2/2014, p. 4-32. [seen 1.12.2019] Source: http://www.vgo.army.cz/sites/vgo.army.cz/files/dokumenty/zakladni-stranka/vgo_2014_02.pdf

[17] LITWIN, Louis. *FIR and IIR digital filters.* IEEE potentials, 2000. Volume 19, Issue 4, p. 28-31.

[18] OKONIEWSKI, Piotr; PISKOROWSKI, Jacek. *An analytical approach to the group delay compensation of digital IIR filters.* 2012 17th International Conference on Methods & Models in Automation & Robotics (MMAR). IEEE, 2012. p. 75-78.

[19] JANOŠEK, M.; PETRUCHA, V.; VLK, M. *Low-noise magnetic observatory variometer with race-track sensors.* IOP Conference

[20] *Multiprocessing Python Library.* Multiprocessing Python Library. [seen 11.11.2019] Source: https://docs.python.org/3.7/library/multiprocessing.html

[21] *PySerial.* Python Serial Library. [seen 11.11.2019] Source: https://pyserial.readthedocs.io/en/latest/

[22] *NumPy.* Python package for scientific computing. [seen 11.11.2019] Source: https://numpy.org/

[23] User *skoehler* on Github forum. *Fast Readline method for PySerial.* Class wrapper to a pyserial object speed improvement for reading from serial connection. [seen 11.11.2019] Source: https://github.com/pyserial/pyserial/issues/216#issuecomment-369414522

[24] *GEM GSM-19 Overhauser Magnetometer .* Cost Effective and High Precision Overhauser Magnetometer. [seen 11.11.2019] Source: http://www.gemsys.ca/rugged-overhauser-magnetometer/

[25] BUTTA, Mattia. *Orthogonal fluxgates.* Principles and Applications, 2012, 19-44.

[26] JANOŠEK, M.; RIPKA, P; PLATIL, A. *Magnetic sensors & applications - Part 2.* MSZ Course Presentation, CTU in Prague, Faculty of Electrical Engineering, Department of Measurement, 2017, page 5.

[27] TRAVIS, J.; ROTH, C. *LabSQL - LabVIEW Open Source Tools.* Collection of LabView VIs enabling to connect to SQL databases. [seen 11.11.2019] Source: http://jeffreytravis.com/lost/labsql.html

Chapter 7

List of Figures

Chapter 8

List of Tables

Chapter 9

Content of the CD with Software

- /
  - thesis — folder with the thesis
    - thesis-final.pdf
  - code — folder with the developed software
    - clinetPrograms — developed client programs
      - labview — LabView Client Program
        - LabSQL ADO functions
        - ADO210.CHM
        - startLVclient.vi — main function of the client
        - unix2timestamp.vi
        - correctData.vi
        - correctBattData
        - corrDetails.txt — data correction parameters
        - dataOffsets.txt — offets of datasets
        - logFile_2019-12-27_10-41.txt
      - matlab — MATLAB Client Program
        - logfiles — folder with example logfiles for analysis
        - main.m — main function of the client
        - dataCorrection.m
        - corrDetails.txt — data correction parameters
    - dataloggerApp — datalogger software and control scripts
      - logger — datalogger application
        - run.sh — start scriptfor datalogging
        - stop.sh — stop script for datalloging
        - main.py
        - dataProcess.py
        - dataSaver.py
        - firFilter.json — file with filter constants
      - scripts — control and backup scripts
        - deviceMonitor.py — system and battery monitor program
        - logFileMove.sh
        - dbBackup.sh
    - simulator — sensor simulator
      - data — fodler with example data for sensor simulator
      - runSim.sh — start script for the simulator
      - stopSim.sh — stop script for the simulator
      - sim.py

Continuing from the previous page:

- /
  - code
    - sql ..... folder with sql scripts
      - restoreDB..... sql script for datalogger database setup
    - exampleData..... example of logfiles and mysql dump
      - logFile_2019-12-27_10-41.txt
      - logFile_2020-01-02_14-05.txt
      - mysqlBackup_2020-01-02.sql
    - others ..... additional programs
      - simpleDBclient.m .. script for DB connection and battery voltage read
      - simpleListener.py ..... script for reading from the serial connection
      - createInsertCSV.py .... script for generating table with historical data