Multi-sensor wireless sensor network nodes for alpine environment monitoring

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Multi-sensor wireless sensor network nodes for alpine environment monitoring

Semester Thesis

Tobias Gatschet
tobiasga@student.ethz.ch

Computer Engineering and Networks Laboratory
Department of Information Technology and Electrical Engineering
ETH Zürich

Supervisors:
Andreas Biri
Dr. Jan Beutel
Prof. Dr. Lothar Thiele

October 28, 2019

Acknowledgements

A special thanks to my supervisor Andreas Biri for his instructions, feedbacks and advices during my thesis. It was very pleasant to work with him on this project as Andreas was very motivated, helpful and professional. Thanks to him I learned a lot.

Another big thank you goes to my second supervisor Dr. Jan Beutel for his inputs, discussions and the opportunity to write this thesis.

I am very grateful to Professor Dr. Lothar Thiele to be able to write this interesting work in his group.

I would like to thank all members of the Geophone Project for their help and explanations.

Abstract

The Pizol Glacier is the first glacier in Switzerland to be declared dead in 2019 and it will not be the last if global warming continues to this extent [1]. Because glaciers and permafrost are melting, our mountains are becoming more and more unstable. In the coming years it is expected that dangerous rockfalls will increase in size and quantity and that they will also occur in previously safe areas [2]. At the same time, leisure activities such as hiking and mountain biking are becoming popular and more and more people are living in endangered places [3].

In order to better anticipate the effects of these changes and to be able to warn people, it has become very important to understand the movements and activities of our mountains. Despite recent efforts in deploying low-cost geophone sensors to accurately track these changes the sensor capabilities are still quite limited.

To resolve these limitations, we have introduced more geophones and sensors in the latest version. These new sensors allow us to capture more and new data and enable new capabilities such as monitoring of slow changes or determining the exact position. However, the platform is in the middle of development as it still lacks essential features such as reliable error handling and robust scheduling. To achieve this, we perform a thorough analysis on the entire code base and extend the new scheduler, debugging and logging features. To place the platform even in hard-to-reach locations we have to make sure the communication works in both directions so that we can remotely change the configurations and the sensor data are stored in the backend. Through detailed power measurements and intensive testing of the complete system and all subsystems we verify the correctness. This way we make the system more robust and fail-safe.

We demonstrate that if the task scheduler is improved as well as extended and the new sensor data is handled correctly, the new platform will be deployment-ready and ready for real-world testing in the coming months.

Contents

Chapter 1

Introduction

This chapter gives a brief overview of why it is important to measure in our mountains for our understanding of the environment. In a second part we present the goals of this thesis which cover program flow, debugging and communication improvements.

1.1 Motivation

In recent years there have been numerous rockfalls and mudslides in Switzerland. Some of them did not cause much damage, but some of them caused deaths or high material damage. The incidents in Bondo 2017 are a sad example [4]. The landslide killed eighth people and had an enormous impact on the economy and traffic. Despite the increasing dangers, more and more people are hiking, biking or taking up residence in these regions. For this reason, it is important to foresee dangerous movements and activities in the mountains to warn the population. Furthermore, it is also an important component to understand the impact of global warming and melting permafrost.

To gain access to this data, a **wireless sensor node network** was installed and deployed. Such a system offers a low-cost solution that does not relay on a expensive external infrastructure and can be deployed for months without human supervision. Thanks to this, the sensor density can be increased with minimal costs. The wireless sensor node network is the left part in Figure 1.1.

To get the collected data from the Sensor Network, we need **base stations** as a link between them and the backend. These stations are responsible for time synchronization between the sensors and transmit the data over TCP/IP to the backend.

The **backend** then processes and secures the data. It is also the place from where all other system parts (base stations and sensors) are controlled and monitored.

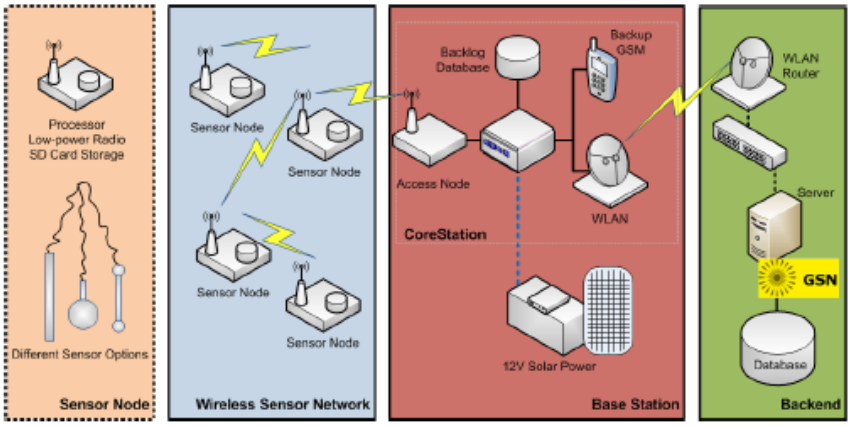

Figure 1.1: On the left side are the sensors which are connected to the base stations and on the right side is the backend. The base stations are linked with the backend, where all data is sent to be analyzed and stored. [[5], Figure 2.1]

Based on the feedback of the first Geophone Platform revision, the development of the 3 Axes Geophone Platform [6] was started. As seismic events occur not only in one dimension but on all three axes, the latest revision has more geophones to measure on all axes. To compensate a disadvantage of the geophones, which cannot measure slow incidents and movements, an additional inclinometer and a GNSS module have been installed.

With this second revision of Geophone Platform, we can measure seismic events more accurately. Due to the two new sensor types and the additional data gained, we can measure not only fast changes but also slow movements in the mountains. With this additional information, we hope to gain a better insight and understanding of our mountains in order to better understand the consequences and impacts of climate change.

1.2 Goal

The main goal of this work is to complete the second version of the 3 Axes Geophone Platform on the software-side so that the platform can be used in the real world.

The worst-case scenario that could happen is if the platform gets stuck and a human intervention would be necessary. For this purpose, the software should be made more robust and fail-safe so that if any error occurs it would be detected and handled by the platform itself.

In a second step, the correct operation should be ensured by various tests. Furthermore, in this context of testing, it should be ensured that only the correct data is logged and sent to the backend.

The second task is to complete the communication between backend and the platform. Since the new platform has more sensors and a new scheduler onboard we need to implement new commands so that the platform can be fully controlled from the backend. Secondly all the new data should also be sent over the network to the backend, where it will be analyzed and studied. In both cases, the corresponding messages and structured data must be defined, implemented and tested.

Last but not least, in addition to some visual tests, a comprehensive power measurement should be made. This is used to determine the power consumption of each individual task. With this data we can verify that the platform is running correctly and as desired. Another advantage is that we can use these measurements to estimate how long the installed battery will last with different configurations.

After this work, the platform should be deployment-ready on the software side and be ready for further real-world tests. To achieve this, we need to take a closer look at how the tasks are executed and the data is stored. For both we have to check if some improvements and extensions are necessary and check then for correctness.

Chapter 2

Background

This chapter presents a previous work on environmental sensing in mountainous regions. First we introduce the PermaSense Consortium before sketching the historical development of the hardware which applies to this thesis.

2.1 PermaSense

After the big rockfall at the Matterhorn in 2003, the PermaSense Consortium was founded in 2006 to predict such incidents with research measures and direct measurements in the mountains. Over the years, several solutions have been developed, different types of sensors have been installed and more than 115 million data points have already been recorded [7].

An overview where these were installed can be seen in Figure 2.1. The large base stations are labeled. These are responsible for transmitting the collected data from the sensors to the backend or for forwarding the commands to the nodes which is done by a multi-hop protocol. These stations are placed at a central point, where the sensor nodes can reach them and are solar powered. The other marked points are all the different sensors which include for example high-resolution camera, crackmeters, acoustic emissions, temperature, etc.

2.2 Geophone Platform

2.2.1 Geophone Platform, first Revision

A platform in use is the Geophone Platform which was developed in 2018 by Akos Pasztor[1] as a master thesis at ETH Zurich [5] in the TEC group (Computer Engineering Group). This platform leverages a geophone to record seismic activities and has an integrated rechargeable battery. It is wirelessly connected

[1] https://akospasztor.com/

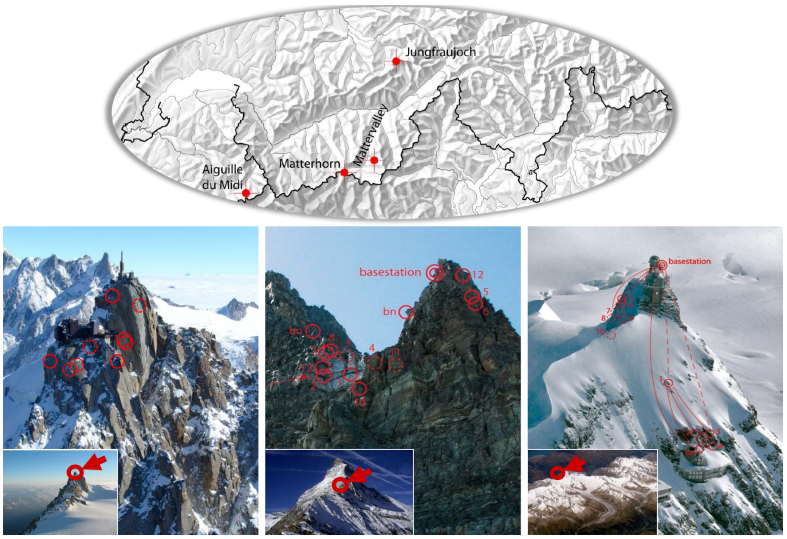

Figure 2.1: Overview of the placed sensors.
https://www.permasense.ch/en/fieldsites.html (12.10.2019)

to the base station using a IEEE 802.15.4 radio with eLWB [8] (Event-based Low-power Wireless Bus). All this allows a remote usability of the platform even in hard-to-reach places. All the electronics are mounted in a weatherproof case to withstand the harsh weather conditions in the mountains. A picture of the platform can be seen in Figure 2.2 on the left side.

2.2.2 3 Axes Geophone Platform

Hardware

Based on the feedback of the first revision 2.2.1, the development of a second revision was started. After analyzing the data from the first one, it was found out that it would be interesting to measure seismic activities not only on one axis but on all three axes. Therefore four geophones are now integrated: one for each axis and one for the trigger. When the trigger geophone registers some activities, this sends an interrupt to the microcontroller to wake it up. This initializes the three other geophones and starts the measurements.

Geophones have a great disadvantage. They can only measure fast changing movements. Slow changes like rotating or detecting slow shifts in the mountains are not possible. Since we also want to measure these motions, a GNSS module and an inclinometer are now on-board as can be seen in Figure 2.2. With these two sensors we hope to be able to measure these movements and rotations accurately, so that it will be possible to measure both: fast seismic events and slow shifts. With this new amount of data, it should be easier to understand

and predict the dynamics of the Alps.

Bolt

Both platform revisions use BOLT [9] a stateful processor interconnect, which was developed at ETH Zurich. BOLT decouples the application from the communication layer, uses ultra-low power and has an asynchronous message passing protocol. Thanks to this interconnection, the application and communication processor can work independently from each other.

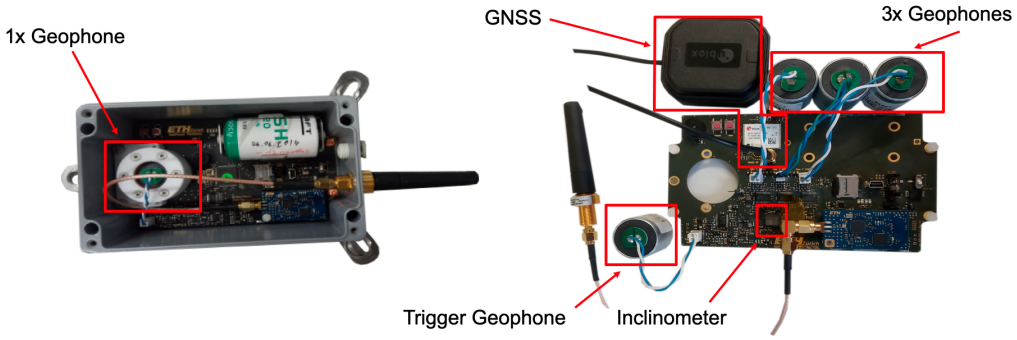

Figure 2.2: The first revision on the left has only one geophone. The second hardware revision includes now the following sensors: - Trigger Geophone to recognize seismic activities. - 3 Geophones, which are used to measure seismic events on all three axes. - Inclinometer to measure angle and magnetization. - GNSS Module to determine precisely the location.

Software

The first platform was only in operation when seismic activities were measured or during the periodic health task, which sent a status report to the backend every second minute by default. With the new sensors the old settings are no longer sufficient and a schedule has been implemented [6]. The use of the new scheduler is based on the fact that we want to choose between which data we want to read from which sensor. Compared to the two previous modes where we could either measure data from seismic events or continuously, the scheduler offers the big advantage of being able to acquire targeted data and save memory space and energy. We can specify an operation mode as shown in Table 2.1.

This system permitted the specification of the period or time at which measurements should be performed. In the SCHEDULE.txt file we can specify the scheduler entries. An example of such an entry can be seen in Listening 2.1. The first argument shows the starting time of the measurement, followed by its period and length. However, it was not possible to change the operation mode in the

Mode	Bitmask	Comment
NORMAL	00000000	No bit set, normal (triggered) mode
NOLEDS	00000001	Do not turn on LEDs
NOTRG	00000010	No measurement on trigger
CONT	00000100	Measure always
SCHED	00001000	Load and execute measurements from SDCard
X-Axis	00010000	Measure X-axis geophone
Y-Axis	00100000	Measure Y-axis geophone
Z-Axis	01000000	Measure Z-axis geophone
GPS	10000000	Log GPS on scheduled measurement

Table 2.1: Operation Modes.

scheduler. This must be done via a network command, which then changes the operation mode for the whole system. This system is not practical and desirable for us. On the one hand, someone would have to send commands all the time on all platforms which is exhausting and error-prone. On the other hand, we change the current status of the platform every time we change the operation mode, which can also cause problems, because we change the internal state and could miss therefor some events or the system could hang if too many changes take place. We would like to avoid this problem in the future by modifying the scheduler so that no system mode change has to be done. For that we should be able to specify in the scheduler itself what we want to do. It is also not possible to activate the scheduler and the trigger at the same time. Another problem is that the inclinometer is completely missing in this list.

```
1 0           120  3       // every two minutes from now on execute
                           the defined operation mode for 3 seconds
2 1570507256  0    5       // on the 08.10.2019 at 4:00 am execute
                           the operation mode for 5 seconds
```

Listing 2.1: An example of entries in the scheduler.

The scheduler is programmed with a fixed period to check every few seconds if any entries are due. We will see that the first software version improves on this concept as it permits both time as well as event triggers.

As we have seen, the current version can still be improved. We need to support a dynamic change of operation mode in the scheduler. It should be possible to access all sensors through it and triggering and scheduling should be able to run side by side. Logging and sending the new sensor data still contains some inconsistencies, which should be also eliminated. We succeed when we have solved the problems with the scheduler and all data is stored correctly in the backend.

Chapter 3

Implementation

In this chapter we will have a look at different software aspects of the 3 Axes Geophone Platform. Starting with the improvements of the scheduler we switch to an overview of the running system. Then we want to discuss the different tasks, how they can be accessed and with which sample rates data can be generated. Finally, we look at what changes we applied to the communication end.

3.1 Scheduler

As already briefly mentioned in Section 2.2.2, the scheduler has some aspects which we would like to improve. The main disadvantage so far has been that it has been impossible to define a specific task for an entry in the scheduler. Up to now, everything has been controlled by the operation mode, as shown in Table 2.1. To tell the system which data we would like to acquire we had to change its operation mode. This process is not desirable as it has some drawbacks as already mentioned in Section 2.2.2.

To solve this problem, we extend the existing format by one argument. The purpose of this extra input is that we can use it to specify which task we want to have performed at that entry. You can see the new structure of such an entry in Table 3.1.

first time	period	duration	task
32 bit	32 bit	32 bit	8 bit
seconds	seconds	seconds	number of task (bitwise)

Table 3.1: Structure of a scheduler entry.

The first argument, **first time**, specifies when the first time measurements with this task should be carried out. We have two different options. We can

either specify an exact time using the UNIX timestamp or simply set a zero as the input. With a zero we indicate to the scheduler that we want this entry executed as soon as possible (ASAP scheduling).
With the **period** we can choose the interval between two executions of this entry. When the period argument is set to zero, the entry is removed after the first execution.
The **duration** is only required for the acquisition task, this argument is for all other tasks irrelevant. With the duration we can control how long the acquisition task should record data from the geophones.
The fourth input describes which **task** we would like to have executed at this time. In contrast to the previous three arguments, which have seconds as input, this one has the enum taskExe shown in Table 3.3 as argument. taskExe is a new enum which we introduce to specify which tasks should be executed. We implemented this new enum because with the previous operation modes, as shown in Table 2.1, the problem was that it did not contain all possible tasks and mixed different things. We see that at first different system modes are defined like triggering, scheduling or continuous mode and in the second part some tasks. To simplify this we have separated these things from each other. We took all tasks out of the operation modes and defined them in the task execution. The simplified operation modes can be seen in Table 3.2.

Mode	Bitmask	Comment
SYS_OPMODE_NORMAL	0001	Normal (TRG) mode (triggering)
SYS_OPMODE_SCHED	0010	Scheduled mode, Scheduler is active
SYS_OPMODE_CONT	0100	Always-on (CONT) mode
SYS_OPMODE_NOLEDS	1000	Disable LEDs

Table 3.2: Simplified Operation Modes.

Task	Bitmask	Comment
TASK_HEALTH	00001	Health Task
TASK_ACQ	00010	Acquisition Task
TASK_GNSS	00100	GPS Task (raw data)
TASK_INCLI	01000	Inclinometer Task
TASK_TSYNC	10000	Time Syncronisation Task

Table 3.3: Task Execution Modes. The IMU is executed by the acquisition task and can therefore not be called by an individual task.

Below we will explain some examples of possible scheduler entries.

Example 1: In Listing 3.1 as soon as possible, the health task should be executed every two minutes.

Example 2: Listing 3.2 has an additional GNSS measurement on 16.10.2019 at 2:52pm (UTC) compared to Example 1. This entry will be executed once since

the period is set to zero.

Example 3: In Listing 3.3, the inclinometer and acquisition task will be triggered every hour. The acquisition task will measure over three seconds as configured in the duration argument.

```
0           120   1 1       // periodic execution of health task
```

Listing 3.1: Example 1: every two minutes the health task executes.

```
0           120   1  1      // periodic execution of health task
1571237579 0     1  4      // one GNSS measurement at given time
```

Listing 3.2: Example 2: Periodic entry and an entry executed only once.

```
0            3600   3 10    // periodically execute acquisition and
                               inclinometer task
```

Listing 3.3: Example 3: The inclinometer and acquisition task will start every hour.

In the following we want to have a closer look at the functionality of the scheduler. For this we look at Figure 3.1 and explain the most important elements.

At the top we see the two methods how we can actively change the schedule entries. This is either at **startup** using the SCHEDULE.txt file with the given structure as listed above in Table 3.1 or over the **network** using two commands which we will explain in Section 3.4. The scheduler is implemented as a linked list. The list is sorted by execution time so that the head will be the next executed entry. The insertion of individual entries can be seen in the middle of the figure. We iterate through the list until we have found the appropriate location and insert it there.
When the scheduler wakes up, which is determined with the wake-up period, it checks whether the head of the list needs to be executed or not. When the head is due, it is pulled out of the list and added to a task list. If the entry has a period greater than zero, it is added back to the list according to the above procedure. This process is repeated until the execution time of the head is more than one wake-up period apart and therefore will not be executed in this iteration. From the task list, the maximum duration for the acquisition task will be aggregated. We iterate through all pulled entries where the acquisition task should be executed and take the longest duration length. After that, each task in this list will be executed exactly once according to its priorities. The priorities are shown in Section 3.3. For example, if there are two tasks sets which execute simultaneously the GNSS task, we will start the GNSS task only once in this time period and discard the second execution. This prevents redundant measurements and saves execution time and energy.

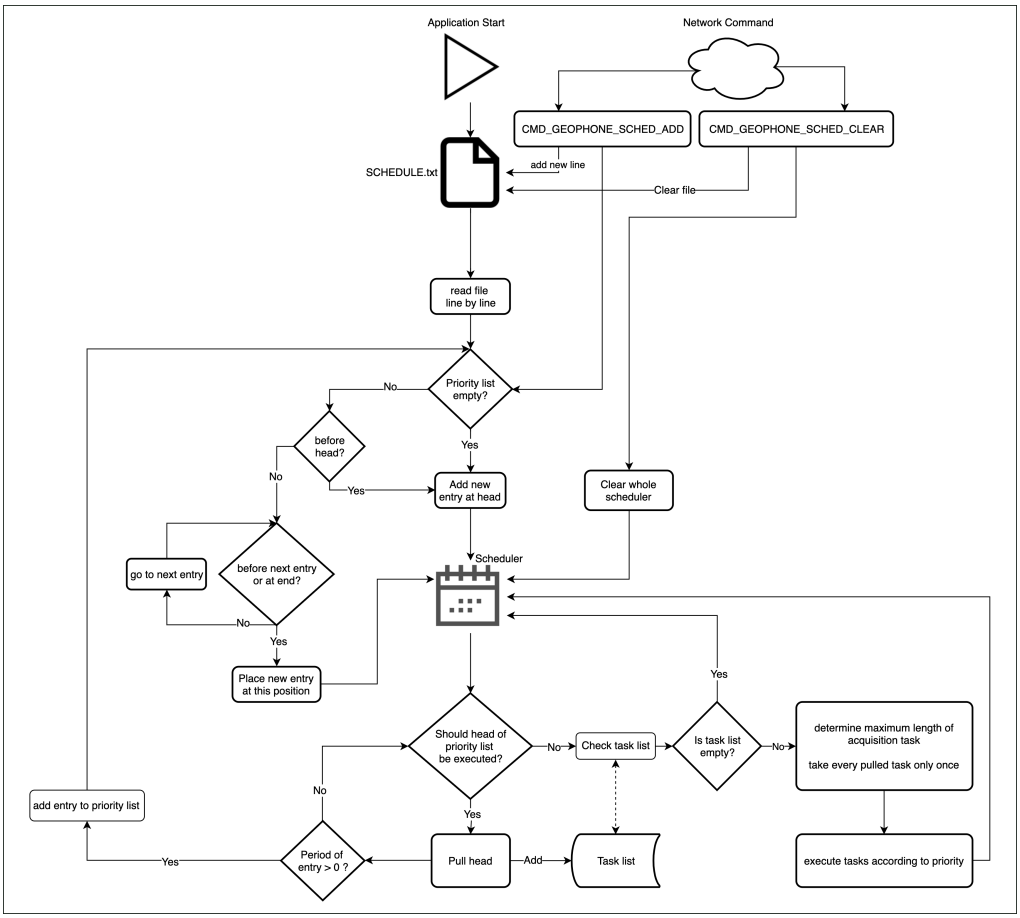

Figure 3.1: Overview of how the scheduler adds entries to its priority list. In the lower part the pulling of tasks are shown.

3.2 Software Overview

In this Section we want to explain how the software flow looks like. The whole flow is shown in Figure 3.2. We can imagine the flow going around in a circle. We start in the Stop Mode and we want to end in this mode again after the execution of the current task. The Stop Mode consumes the least amount of energy as all unused components are either disconnected from the power supply or are in their Stop Mode. This mode drawn around 0.17mA of current, more details can be see in Section 4.2.2. The longer we are in this mode, the longer the platform survives without an active power supply.

There are two event which change this state. The first is that the trigger geophone measures activities and triggers an interrupt to wake up the microcontroller. Thus the acquisition task starts and seismic activities are measured for at least one second. The duration is extended if a new event is registered during

the measurement. It is now measured form this event on for another second. The second option is that the wake-up period expires which also generates an interrupt. Why we need this periodic wake-up is explained in Section 3.2.1. In this routine, the scheduler is called to check according to Section 3.1 whether any tasks should be executed. If so, the tasks will be executed according to their priorities as can be see in Figure 3.3. After each task, the collected data is logged on the SD card and sent to the backend via the network. When all tasks have been executed, the controller returns to Stop Mode and the circle closes.

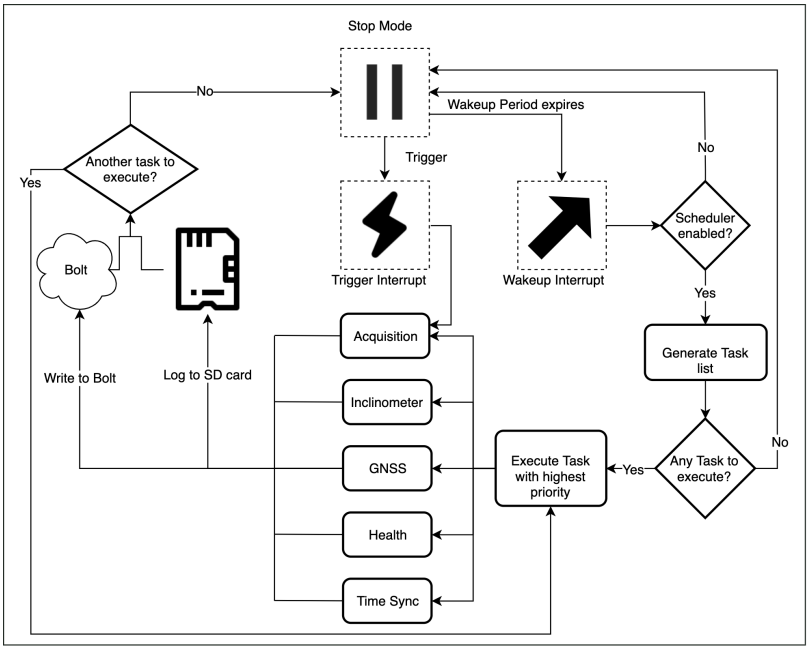

Figure 3.2: This figure gives a rough overview of how the software flow looks like.

In Listing 3.1, we have seen how we can add a periodic health task to the scheduler. Since this is now possible with the improved scheduler, we have decided to remove the periodic health task from the previous versions and access all tasks only via the scheduler. This gives us more flexibility and simplifies the system as there is no third way to the two presented. This makes the system clearer and more robust than before, as there are fewer interrupts and issues affecting it.

3.2.1 Watchdog

We use a watchdog timer to reset the application if the system gets stuck somewhere. For this reason we have to reset the watchdog timer from time to time. That is the reason why we have the periodic wake-up interrupt. The built-in

watchdog timer of the microcontroller has a range between 125 microseconds and 32.8 seconds [10]. This means that the wake-up period must have a value between these two values, otherwise the watchdog would perform an application reset. So we have to make sure that if a measurement is longer than the wake-up period, we have to reset the watchdog at some point during recording.
Currently, we reset the timer on the microcontroller wake up and interrupts from the trigger geophone or GNSS module. There is one important point we need to keep in mind. The watchdog timer depends on the wake-up period. If we would increase the wake-up period when the platform is deployed, we should perform an application restart as otherwise the watchdog timer is smaller than the wake-up time and therefore the watchdog would restart the system. That is because once the watchdog is initialized, it can no longer be stopped or changed.

3.3 Tasks

Figure 3.3 presents an overview of all available tasks and their associated priorities. The higher the number, the more important the task is. Therefore, the acquisition task is the most important. As soon as it is called up, it is executed no matter what other task currently being executed. This is possible because we use a preemptive scheduling where the controller can interrupt tasks in the middle of their execution. If two tasks of the same priority are started, they are called one after the other. The second task has to wait until the first started interrupts its measurement or is finished. The running task can however be interrupted at any time by a task with higher priority.
As we see, the bolt and log tasks are arranged lower than the sensor tasks. This is because all data is written into a buffer which is then read as soon as we have finished all measurements. In this way we do not falsify or interrupt the recordings. These two tasks cannot be called directly by the scheduler or trigger compared to all the others. These are called by the others once they have written data into the corresponding buffer. The IMU task is managed by the acquisition task and cannot be specified via the scheduler either.

Let us have a brief look at the most important tasks and how many data packets have been generated.

3.3.1 Acquisition task

The acquisition task is responsible for the geophones synchronization and reads the data from them. This task also calls the IMU task and handles its data. A new file is generated on the SD card for each new event, one for the IMU data and one for each axis. Every second the sampled data are saved in the corresponding files. Depending on the duration we give this task or if the event still continues and constantly triggers it can be executed for a few seconds. When

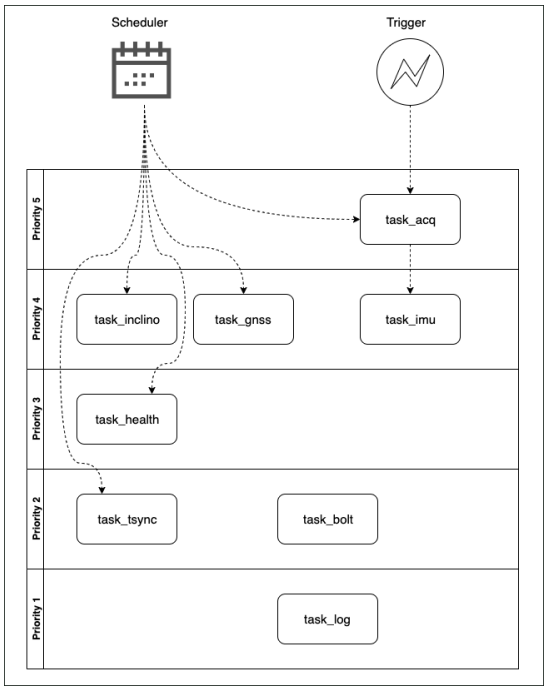

Figure 3.3: The individual tasks are listed according to their priorities. Compared to all other tasks, the bolt, log and IMU task are not directly callable by the scheduler or trigger.

the acquisition task is finished, it will send meta data of the event via the network to the backend and make sure that it also logged on the SD card. The geophones of the three axes are set up at start and are disconnected as soon as they are not longer needed.

3.3.2 Inclinometer task

The inclinometer is normally in a Stop Mode where it needs almost no energy. Before we can read the data, we have to wake it up and initialize it first. Then we can read the angle and acceleration data. We measure only once per call, accordingly we produce one data set which is transferred to the log task and to the backend. After the measurement the inclinometer is set back in Stop Mode.

3.3.3 GNSS task

When the GNSS task is executed, it will enable the GNSS module and synchronize the module with all necessary values. Then we can start sampling. The GNSS module sends a new sample back to the microcontroller via an interrupt every second. However, depending on the connection to the satellites, some sam-

ples will be needed until valid data arrives. In the worst case, we do not get a signal and we cannot measure any data. But as soon as the first valid data set arrives, we abort the measurement and log the data.
Since in theory samples from more than 180 satellites could be collected, we installed a limitation so that we only read the amount of data that is of interest for us. These data are then stored in a raw form on the SD card as well as in the backend. Each measurement from a satellite produces its own packet. In the current state this would be up to five packets per measurement that would be sent. Each packet contains 58 bytes, 16 bytes are for the header and 42 bytes are data. In total will be 290 bytes sent over the network and 210 bytes will be logged on the SD card.

Due to the long waiting time for valid data and the high power consumption of about 125mA, which is twice as high compared to the acquisition. This task should only be used twice a day at least.

3.3.4 Health task

The health task does not require any additional components to power up. Like the inclinometer task, it is a single shot task. After only one measurement, the task is completed. It sends a status report about the platform to the SD Card and backend which uses one packet for communication.

3.3.5 Time synchronisation task

As the platform clock drifts apart on different devices, it has to be synchronized from time to time. This is done in this task. Therefore, the task sends a time request to the network and gets a response back with the current time. The platform is then synchronized to this time.

In the old version this task was dependent on the health task because the time sync also had to be executed regularly. This is no longer necessary in the new version as the health task is now integrated in the scheduler, this task is now implemented as an independent task.
Since it is extremely important and should not be forgotten, this task is inserted in the scheduler as default and remains there even if the scheduler would be cleared.

3.4 Communication

As already indicated, with the new features and sensors we need new commands and data structures for the communication between the platform and the backend. On the one hand, we enabled the possibility to add and remove entries to

the scheduler from the backend as you can see in Figure 3.1. In addition we add commands to adapt the system settings and obtain a fine-grained control of the IMU sensor. From the eight implemented commands which are listed in Table 3.5, the CMD_GEOPHONE_SCHED_ADD command is the most complex. It consists of two values that need to be transferred. The first is a UNIX timestamp for the first time input. In the second argument the fields period, duration and task are passed according to the defined mask which can be seen in Table 3.4.

period	duration	task
20 bits [12...31]	4 bits [8...11]	8 bits [0...7]

Table 3.4: The structure of the second argument of the CMD_GEOPHONE_SCHED_ADD command.

Except for the CMD_GEOPHONE_SCHED_CLEAR command, which expects nothing, all others use at least one 32-bit value.

command	description
CMD_GEOPHONE_SCHED_ADD	add new entry to scheduler
CMD_GEOPHONE_SCHED_CLEAR	clear scheduler
CMD_GEOPHONE_TSYNC_PERIOD	set tsync period
CMD_GEOPHONE_SYS_WAKEUP_PERIOD	set Wake-up period
CMD_GEOPHONE_IMU_TRG_LVL	set IMU Trigger Level
CMD_GEOPHONE_IMU_OPMODE	set IMU Operation Mode
CMD_GEOPHONE_IMU_FREQ_AA	set IMU Frequency Anti-Aliasing
CMD_GEOPHONE_IMU_FREQ_LP	set IMU Frequency Low Power

Table 3.5: All new commands.

In order to transfer the new sensor data, first the corresponding data structure has to defined. At the inclinometer data structure the missing temperature field is added. To reduce the messaging overhead, we remove legacy fields which are of no use in the current implementation. At the GNSS and inclinometer structure the sample and time field are redundant and therefore not included, as these are already set in the header of the corresponding packet. The amount of packets and sent data are listed in Table 3.6. With the GNSS_MAX_MEASUREMENTS_LOG variable we limit the amount of GNSS packets to five in the current implementation. Theoretically the maximum amount of GNSS packet could be 184.
These structures are then implemented in the backend and added to the database. It is now possible to send all data from the platform via the network to the backend where they can be stored.

task	max amount of packets	packet + header size [bytes]
Acquisition	1 packet	65 + 16 = 81
Inclinometer	1 packet	14 + 16 = 30
GNSS	5 packets	5 * (42 + 16) = 290
IMU	1 packet	12 + 16 = 28
Health	1 packet	22 + 16 = 38

Table 3.6: Packet size of the different tasks. To each packet the 16 byte header is added.

Chapter 4

Testing

In this chapter we look at how we have tested the correctness and robustness of the platform. We carried out several power measurements, which we will discuss and briefly highlight the most interesting parts. Finally, we take a look at an example of a possible scenario and how long the platform should be run according to our simulations.

4.1 Verification & Robustness

To ensure that the platform works as intended, some tests have to be carried out. We go through different test phases. In each of these, we test different elements of the software and fix any errors we notice. In the following we want to go through the main steps of the various phases.

Debugger

During the development we test a lot with the debug possibilities provided by the stm32CubeIDE[1] and the Segger J-Link Debugger[2]. For example, we might be able to tap through the code step by step and verify the code flow. In case of errors, we can find out relatively quickly where something was wrong. Breakpoints can also be set to read out variable values at certain points.

In addition to these methods, we implement our own logging method to print strings to the terminal. An advantage of our implementation is that we can specify different modes which are shown in Table 4.1. As an example, we can determine whether we only want to output information or only error strings.

These strings can be easily removed from the code when we set the DEBUG_OUTPUT_ENABLED variable in the config.h file to zero. This is useful

[1] https://www.st.com/en/development-tools/stm32cubeide.html (18.10.2019)
[2] https://www.segger.com/products/debug-probes/j-link/ (18.10.2019)

Output Mode	Bitmask	Comment
MOD_ERROR	0001	Enable Error Outputs
MOD_WARNING	0010	Enable Warning Outputs
MOD_INFO	0100	Enable Info Outputs
MOD_DEBUG	1000	Enable Debug Outputs

Table 4.1: Output modes.

when the development of the platform should be completed. Because we can remove them, it means that the controller has less context changes. Therefore we can save execution time, memory space and eliminate a source of error.

After we have finished debugging, we remove the debugger and test the platform using different tools. Each of the following tests is intended to test some isolated elements and track down any errors. At the end we put everything together and want to do a long-term test with a realistic configuration to test the whole system.

LED

We use the four LED's on board to indicate which tasks are activated by the scheduler. Through this test we can verify that the scheduler works without debugger. We can also show that if, for example, the GNSS task is running, indicated by the red LED and the trigger geophone detects any event, indicated by flashing the blue or green LED, the GNSS task pauses and the acquisition task starts. As expected, the orange LED intended for logging goes on after one second after the acquisition has started and logs the collected samples from the geophones. After the acquisition is finished, the green and blue LED are off, the GNSS resumes and after completion, the orange LED indicates that the GNSS data is now logged.

In a second phase we checked the correctness of the watchdog. For this we manually set the wake-up period above the value of the watchdog timer or intentionally implemented an infinity loop. By flashing all four LED's we indicate an application reset. In a first implementation, we found out, that the watchdog is triggered correctly when the wake-up period is longer than the watchdog timer. But if an infinity loop occurs in a task the watchdog never resets the application. To find out where the error is, we reconnect the debugger and tried to find the mistake using the output strings. The step-by-step method or breakpoints do not help in this situation because the watchdog is suspended in debug mode. We found out that the watchdog timer is reset at every wake-up interrupt. Since the interrupt has a higher priority than all other tasks and is therefore always executed, the timer is reset even if a task gets stuck somewhere.

We expect the controller to go into power down mode as soon as all tasks have been completed correctly. We solve this problem with the watchdog timer by resetting it when the controller wakes up out of Stop Mode. This ensures that an infinity loop is detected as the controller goes no longer in power down mode. However, this solution has its pitfalls with tasks that last longer than the wake-up period. These are the GNSS task and the event-driven acquisition task. For these tasks to be executed correctly, the watchdog timer must be reset somewhere during its execution. In the acquisition task we solve this by resetting the timer at each interrupt coming from the trigger geophone. This way the measurement is not interrupted should an event last longer than the wake-up period.
With the current configuration, the GNSS module produces an interrupt every second when a new sample is ready. We reset the timer in this interrupt which protects the platform from a reset.

After these corrections we did the test again and found out that our changes had the desired effect. The watchdog resets the platform should a task stuck in an infinite loop unless an event occurs or the GNSS task takes longer.

Now that we have verified these functionalities, we go further and take a closer look at the logging of the data.

SD card

All measured data is stored on the SD card. Therefore this is a good place to first check that only valid data sets are stored. At the same time we can check if the logging of the data works correctly.

We determine that the inclinometer and GNSS still store data sets that should not be logged because they either contain zero or incorrect data values. In the case of the inclinometer, we solved the problem by first checking whether the RS bits from the received packet are set to 01. Then taken from the data sheet, the inclinometer only sends correct data if these bits are set to this value. If this is the case, we can be sure to have received a valid dataset and log this.

With the GNSS we do the similar as with the inclinometer. However, instead of checking the RS bits, we check if the sample received contains any measured data. This is done via the "numMeas" argument of the received packet. Only if this is not equal to zero correct data can be arrived. Then we check whether the given version number corresponds to the known value and if this is the case we can log the data.

With these two modifications only valid data sets are now stored and after another test we can verify that no longer zero or invalid data is stored. We also check that all data is written in the correct folder. For each day a new folder is created and the data should be stored in the current folder. We do this after the long-term test where we can see that the measurements are written in the

correct location.

Serial Outputs

After all other tests were successful, we check through a serial output to ensure that the communication between the microcontroller and the communication board works. This is necessary as there are some new events, data and commands in the new version. This leads us to the error that we do not get a time sync and only send or receive irregularly packets.
After some research it turned out that everything works correctly except that we do not have that good reception at our working environment. This problem is solved by increasing the transmission power or by positioning the platform closer to the base station.

Backend

After everything has been tested locally, we also have to verify that the sent data also arrives at the backend and is stored correctly in the database. In addition, we want to test that the newly introduces commands arrive at the platform and have been executed correctly.

We execute the individual tasks and send the measured data to the backend. There we check the database to see whether the data has arrived and been entered correctly.

In the beginning we had the problem that the new sensor packets had zero length and accordingly no data was read. That is because, we forgot to assign the correct packet length from the platform to the new sensor data packets. We fixed this and now the packets arrive correctly and are stored in the database. In Figure 4.1 we see an example of the stored data of a GNSS measurement.

Then we test all new commands by sending them to the platform, using the debugger and then verifying with the LED's if they have arrived and been executed correctly. Thanks to these test we are now sure that the platform is now remotely configurable.

Long-term test

After all the other tests have been performed and it has been verified that the platform works as it should, we want do some long-term tests.

After we did some shorter tests where we let the platform run from a few minutes to a few hours, we switched to a long-term test that lasted several days. For this test we configure the platform similar to how we think it will be used outdoors later. We disable all debug outputs and add some entries to

generation_time	device_id	generation_time_microsec	timestamp	target_id	message_type	seqnr	payload_length	rcvtow	week	leaps	nummeas	recstat
2019-10-24T16:00:38.000+0200	21054	1571925638000976	1571925654587	16	34	11833	42	396038	2076	18	5	2
2019-10-24T16:00:38.000+0200	21054	1571925638000854	1571925640602	16	34	11832	42	396038	2076	18	5	2
2019-10-24T16:00:38.000+0200	21054	1571925638000854	1571925640365	16	34	11831	42	396038	2076	18	5	2
2019-10-24T16:00:38.000+0200	21054	1571925638000732	1571925640161	16	34	11830	42	396038	2076	18	5	2
2019-10-24T16:00:38.000+0200	21054	1571925638000732	1571925639924	16	34	11829	42	396038	2076	18	5	2

prmes	cpmes	domes	gnssid	svid	cno	locktime	prstdev	cpstdev	dostdev	trkstat
23175191.071225565	121786450.43670718	-2799.179931640625	0	23	36	1000	5	3	7	15
22482552.10466959	118146608.66730095	-2173.443359375	0	6	28	1000	8	7	8	7
21644970.638255868	113745087.70201243	141.34765625	0	4	28	1000	6	6	8	15
25254315.108691473	132712320.9949971	-3965.625	0	3	29	1000	6	6	8	15
21539998.832711972	113193456.58296773	-1412.54638671875	0	9	35	1000	5	3	8	15

Figure 4.1: Database entries of a GNSS measurement.

the scheduler which are listed in Listing 4.1. We execute the health task every second minute as it was already the case in the first version. Then every fifth minute the inclinometer task is executed, every tenth minute the acquisition task for three seconds and every hour the GNSS task. In this test, we execute the GNSS task more often than it should be. It is more important for us to test the functionality and communication of the platform and not to be as energy efficient as possible.

After seven days we finish the test and check the data in the backend and on the SD card whether all tasks have been executed at the right time and that only valid data is stored in the right place. The result reflects our expectations, all data was stored in the right place at the right time. The number of restarts is stored on the SD card and when we checked this number we see that the platform did no restart during the test. This meets our expectations as we are sure that in all previous tests the platform runs stable under normal circumstances.

```
1 0 120  1 1
2 0 300  1 4
3 0 600  3 2
4 0 3600 1 8
```

Listing 4.1: Test configuration for the long-term test.

4.2 Power Measurements

In order to be able to determine the energy consumption of the platform, we carry out extensive power measurements. For this we use the RocketLogger with which we can perform very accurate measurements. We expect current flows in the range of 0.1 - 250mA and so the range of the RocketLogger of 4nA up to

500mA is sufficient for us. [11]
The RocketLogger can be operated via a web server from where the measured data can be downloaded. Then we analyse the data through a provided python script and determine the consumption of the different tasks.
In the following we want to deal with the different cases in more detail.

4.2.1 Power Down Mode

Figure 4.2 shows the power down mode when no tasks are executed. We can see that the platform has a low energy consumption. This comes from the fact that all components are either disconnected from the power supply or in a Stop Mode. Only the periodic interrupt and the short wake-up of the communication board are visible.

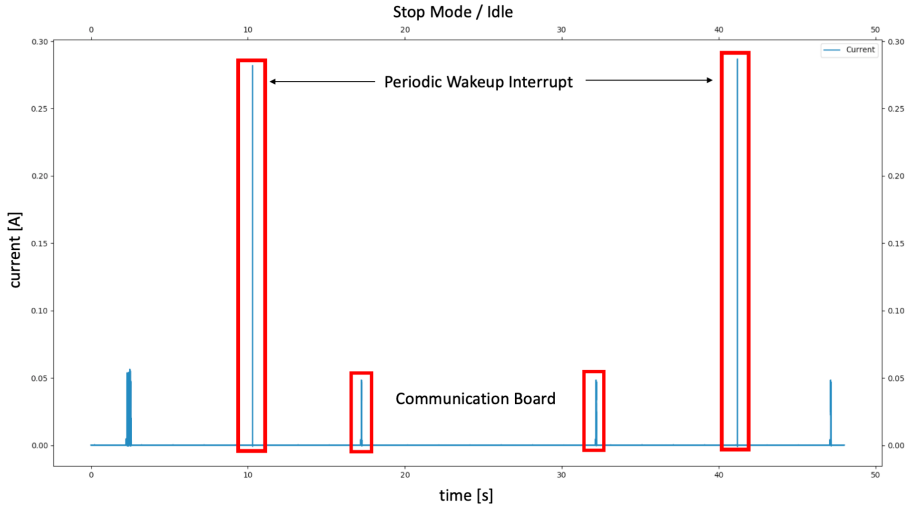

Figure 4.2: Power down mode energy consumption.

The Stop Mode consumes current of about 0.17mA without peaks. Compared to the previous version of Hanna Müller [6] where the power down mode required about 2mA of current, this 0.17mA is a big improvement. This is due to the fact that the inclinometer now switches on and off correctly and other improvements ensure that all components that are not needed are removed from the power supply. Most of the time the platform will be in this mode and therefore it is important to use as less power as possible. If we include the peaks we get about 0.2mA which is around three to four times higher than the first revision. We expect this result because the new platform has more geophones and sensors on board which leads to a higher consumption.

The communication board needs about 20mA over a duration of 40ms if no messages are sent or read. The consumption when messages have to be read is

listed in Section 4.2.7.
The periodic interrupt lasts only about 2ms but requires a very high current of over 250mA. This enormous peak comes from switching on of the ADC2. On the ADC2 the three new geophones are connected which convert their analog signals into digital ones. The peak should be improved and investigated why it needs so much energy in later phases.

4.2.2 Health

The energy consumption of the health task is shown in Figure 4.3. The whole task needs about 210ms and 50mA on average. Only the first 50ms are necessary to initialize the microcontroller. The health task takes only a short time between the initializing and writing to the SD card. Writing to the SD takes a lot of energy and has a big fluctuation. This is not very desirable but can be expected from SD cards. They also have a big differences between each other, with another SD this could be much lower or higher. For this reason we should test different SD cards and choose the most economic ones for operation.

Every time we wake up the microcontroller we wait at least 200ms until we send it back to the power down mode. This is because if anything should be done during this time, for example a new trigger event, we do not have to wake up the controller right it has just been set to power down. However, to save some energy, we could reduce this time to 100ms. So we could eliminate the whole end part of 100ms at 43mA.

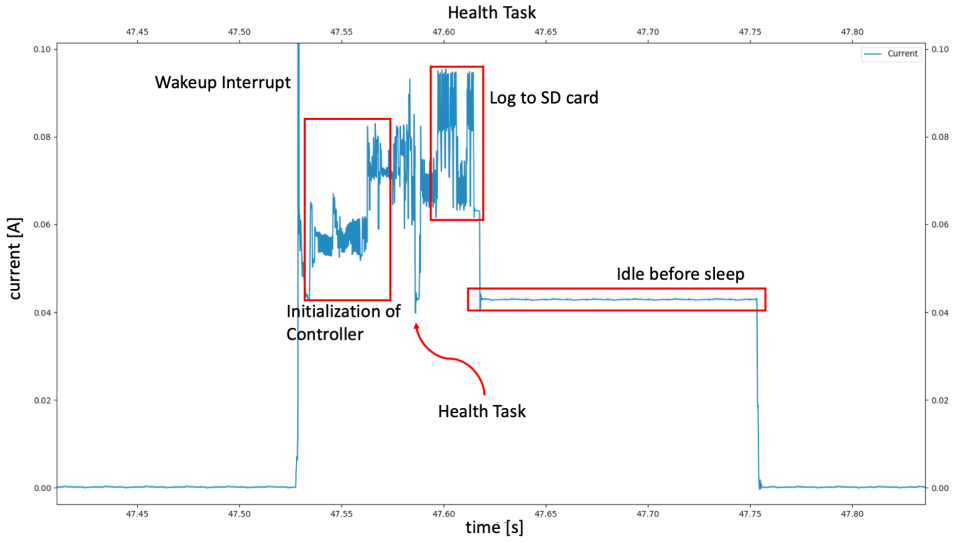

Figure 4.3: Power consumption of a health task.

4.2.3 Acquisition

In Figure 4.4 we see the diagram of an acquisition and IMU task measured for three seconds. In the first second we are at a level of around 60mA. After this second we start to log and for this we have to mount the SD card. This mounting causes the jump to about 100mA. And as we have seen at the health task, the SD card has a very high fluctuation. At the various peaks of up to 125mA, the data has been written to the SD card. After three seconds the measurement is finished but the remaining data must be logged and so the consumption drops to only about 80mA.
In total the task takes four seconds, three to measure and one to log. Thereby we have an average current consumption of 87mA.
If we would like to measure a longer time, the middle part between the mounting and the last logging becomes even longer. This middle part needs around 105mA of current.

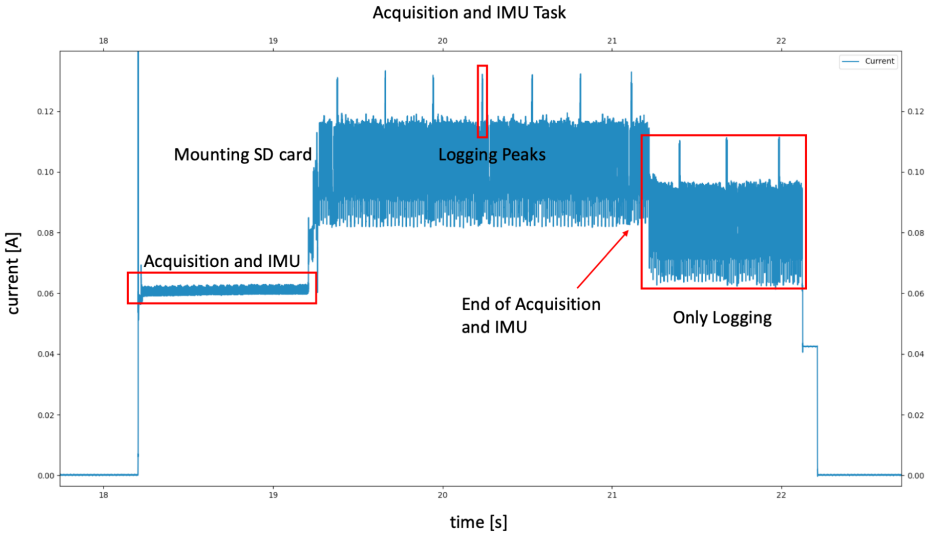

Figure 4.4: Power consumption of a 3 seconds acquisition measurement.

An improvement could be to log everything at once after three seconds when the task is finished. We sample with 1kSps, each sample is 3 bytes large accordingly a second needs 3kB. For all three axes we use 9kB per second of buffer size. When we measure for three seconds we would need 27kB in buffer, which is no problem because currently every geophone ring buffer could contain data in the size of 30kB.
If we would write all data only at the end, we could stay at 60mA for the first three seconds. During this time we could save about 40% of energy. We would mount the SD card at the very last end and then go up to 80mA for half a second. This improvement would need an average 65mA and compared to the

current 87mA would mean a saving of about 25%.

4.2.4 GNSS

The GNSS task is the task with the highest energy consumption as we can see in Figure 4.5. The first two seconds are needed to set up the GNSS module. The current varies between 80mA and 100mA. On average we are around 90mA. From the moment we sample the power consumption increases to 125mA. We see that every second we get a new sample as expect. If we sample ten samples, we are at 125mA for ten seconds. In this example we do not receive any valid data and therefore we do not log anything.

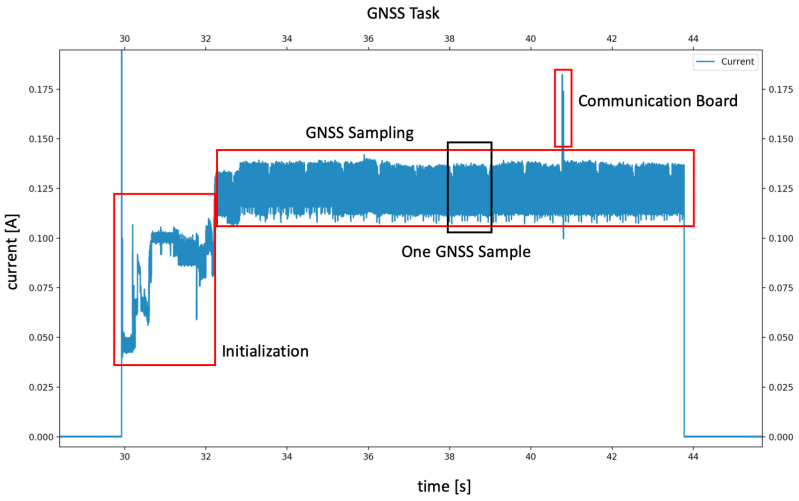

Figure 4.5: Power consumption of a GNSS measurement.

But when we get data, the consumption increases to 160mA on average because the SD card is mounted and data is logged. As we see in Figure 4.6 it even jumps up to 200mA. We log a maximum of five measurements. These five measurements can be easily distinguished in the figure. At the high peaks data is written to the SD card and in between the next data packet is prepared for logging.

When we change the number of samples we have to take into account that the first two seconds are fixed and after that we are at 125mA for the number of samples until we get a valid data set. Then we abort the measurement.

4.2.5 Inclinometer

The inclinometer task has approximately the same shape as the health task. This is because the inclinometer uses only 2.1mA in the power-up mode [12]. All other

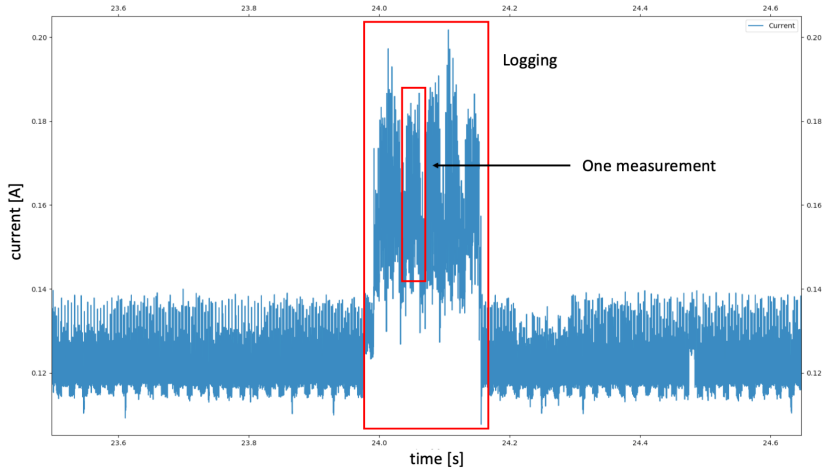

Figure 4.6: Power consumption of a GNSS measurement when data is logged.

components, like the microcontroller or SD card need much more current as we already saw in Section 4.2.2. That is why the inclinometer does not weighted as much and does not change the shape. The duration remains the same as the health task. The average current rises by 1mA to 51mA compared with the health task because of the inclinometer sensor.

4.2.6 Time Sync

The time sync task is extremely dependent on the time difference between the periodic wake-up interrupt and the wake-up of the communication board. In Figure 4.7 we can see that after have requested the time sync we have to wait about six seconds until we get a response. With the present configuration of 30 seconds on wake-up period and 15 seconds period for the communication board we have to wait up to 14 seconds in the worst case.

While waiting, the task consumes 40mA. We cannot enter the Stop Mode during this time as we have to know the time between sending and receiving the request. There is no simple improvement for this task. We would have to adjust the periodic wake-up interrupt to the wake-up of the communication board which is not that easy. Both wake-ups can vary slightly in time so an adjustment could be quite complex.
A possible solution would be if we could wake-up the communication board by an event. This would allow us to process the request as soon as possible.

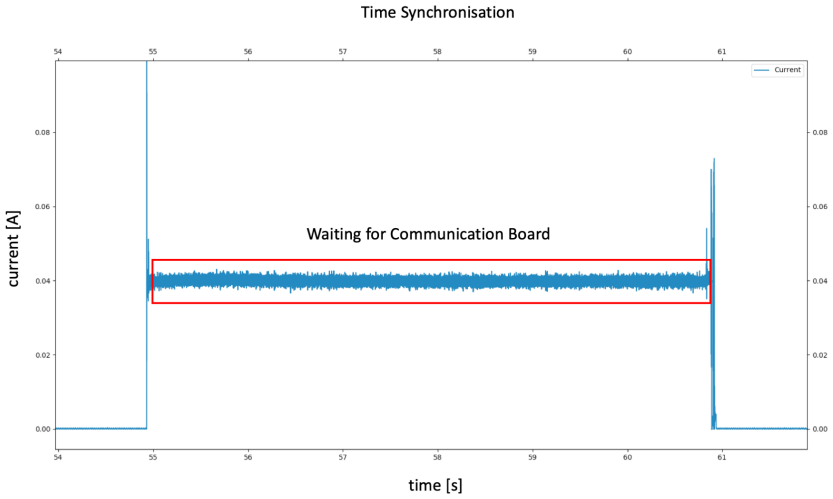

Figure 4.7: Power consumption of a time synchronisation task.

4.2.7 Communication

Receiving a command from the network can be divided into two parts. The first part is the receiving of the command which we see in the first part in Figure 4.8. During this time the current varies between 1 - 60mA. Then we can see that when the microcontroller wakes up, it reads and processes the received command. If necessary, values are also stored on the SD card.

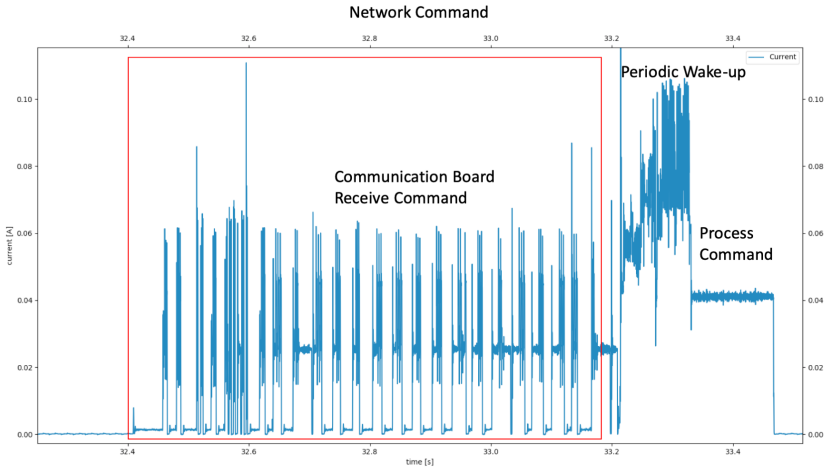

Figure 4.8: Power consumption when the platform receives a command.

In Figure 4.9 the sending of a health packet is displayed. We can see how the

packet is send byte by byte. The health packet is 38 bytes big, 22 bytes for the health data and 16 bytes for the packet header. If we take a closer look we see a total of 38 peaks. The average of all peaks is around 48mA. The whole sending takes 250ms.

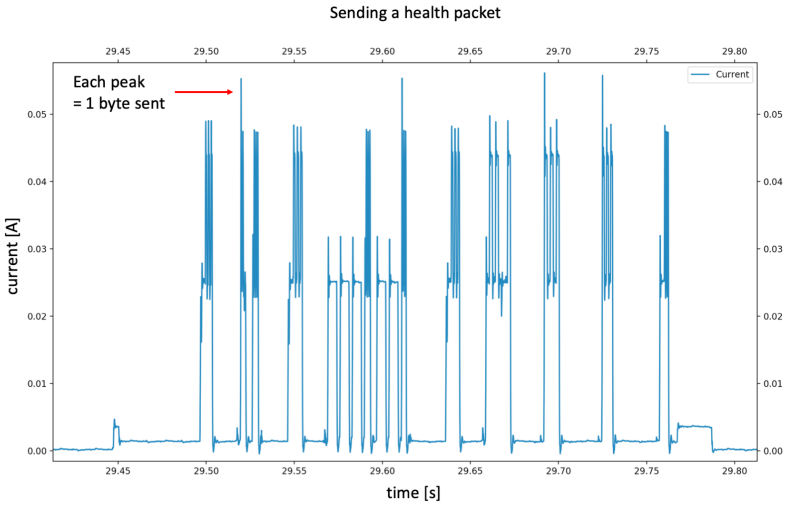

Figure 4.9: Power consumption when the health packet is sent.

4.2.8 Multiple Tasks

After having looked at individual tasks, we now want to look at a special case, namely when the two task that consume the most current are running at the same time. With this test we would to control if the built-in fuse provides enough power. The highest current level is expected when during a GNSS measurement the trigger registers an event and starts the acquisition task. Since this task has a higher priority, it is executed and the GNSS task is paused during this time but the GNSS module stays on.

As we can see in Figure 4.10 the average current increases to 180mA when both tasks are on. If something is written to the SD card the current even rises to 220mA and if the communication board is also on, it is almost at 250mA. This high current is to be expected based on the previous measurements. The problem is that a fuse is built in which limits the current to 250mA for the board. We almost reach this limit with this constellation. So we do not have any reserve. If for any reason the current should be a slightly higher, too little current is provided for the components and this might influence the measurement in an unpredictable fashion.
A simple solution would be to install a fuse with a higher current limit. This installation would give us more reserve without any big change to the platform.

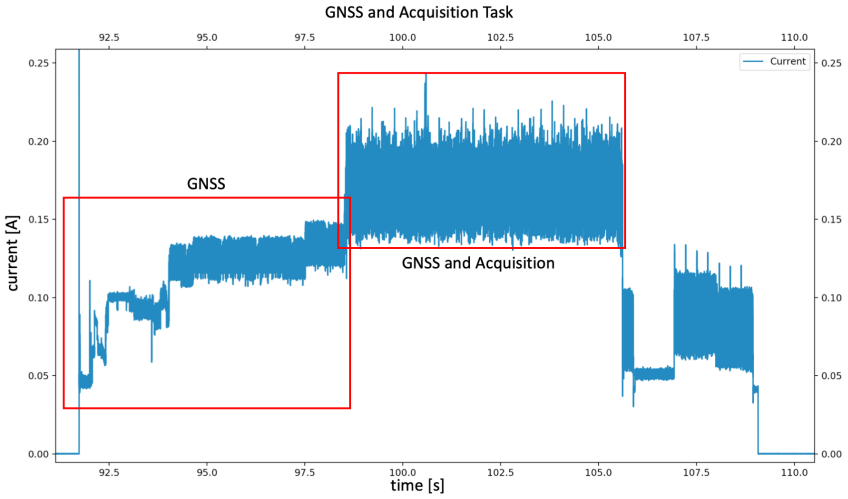

Figure 4.10: Power consumption when GNSS is active and a trigger event occurs.

4.2.9 Runtime estimation

In order to simulate the runtime with different scenarios, we wrote a script where we can specify which tasks should be executed. The script can be found in the Script folder of the project[3]. We will now go through some simulations.

When we do not execute any tasks and no event occurs, the platform is constantly drawn 0.2mA of current. The 13Ah battery will lasts then about 7.5 years. Compared to the first Geophone Platform this is much less as the estimation was 47 years because of idle current of 30.6μA. But we have much higher idle current due to the new and more components.
We now consider the following scenario where we execute:

- every second minute the health task
- every hour the inclinometer task and acquisition task for 3 seconds
- every day a GNSS execution
- every day the trigger detects an event and starts the acquisition task for 3 seconds
- every 20 minutes request a time synchronisation

Based on this test scenario the runtime estimation will drop to about 2 years.
If we would have a very busy underground and an event would be triggered every

[3]https://gitlab.ethz.ch/tec/research/dpp/software/dev$_f$$_o$$_r$$_k$$_s$/$_s$$_a$$_t$$_o$$_b$$_i$$_a$$_s$$_g$$_a$/$_g$$_e$$_o$$_p$$_h$$_o$$_n$$_e3_x$/$_t$$_r$$_e$$_e$/$_m$$_a$$_s$$_t$$_e$$_r$/$_S$$_c$$_r$$_i$$_p$$_t$$_s$

minute, we would have a total of 1440 executions of the acquisition task. The runtime would drop to about a quarter of a year.
Our goal is that the platform should perform for about three years. To reach this with the assumption that 24 events happen during a day we have various possibilities. One possibility would be to remove the time-triggered acquisition task and increase the period of the time synchronisation task up to 60 minute. With these changes we are about 2.6 years. When we also increase the period of the GNSS task to three days, then we would be about 2.8 years. To reach the three years the health task should only be executed every third minute.

Chapter 5

Conclusion

5.1 Conclusion

During this thesis we were able to realize the goals from Section 1.2. We extend the scheduler with a new argument we now can specify a task for each entry. Thanks to that we were able to extend the first version of the 3 Axes Geophone Platform with a time-driven part. We reduce the software flow to two elements: triggering and scheduling. Tasks are only started by these two elements, there is no third option anymore. We also simplify the old operation mode structure by splitting it into two different structures. Also thanks to other improvements we make the code base more robust and easier to understand.
Subsequently, we introduce new commands so that the platform can be operated and controlled remotely. We also ensure that only valid data is logged and sent to the backend. For the sending, the data structures for the new sensor data have been be defined and implemented.
In the end, the platform was tested intensively to verify the correctness of the software. Power measurements give us a deep insight of how the individual tasks are running, how high the consumption is and where there is still room for improvements.

Finally we can say that the initial goals have been achieved. The platform is now ready for further testing in a real-world environment.

5.2 Future Work

While we achieved our goal, we also identified potential future improvements.

- The **improvements** described in Chapter 4 can be implemented. Namely that we log data from the geophones only at the end of the task or when a threshold of nine seconds is reached. The waiting time after the health task could be reduced to 100ms instead of 200ms. Also the ADC2 power need to be further investigated and to be reduced during the wake-up.

- **Trigger on all three axes** and not only on one. With this we could remove the fourth geophone, could reduce the cost of the platform and would have more space available in the case. But for that, the trigger interface should be available on each geophone, which brings more complexity and requires more space on the board.

- **Testing** the device in the **real world** at different locations. Test with a battery to confirm the runtime estimations.

- Implement of a **fuse** with a higher current limit. And maybe there are some **SD cards** with a lower current consumption or with less fluctuations.

Bibliography

[1] T. Häusler, "Abschied vom Pizolgletscher," 2019, [accessed 10.10.2019]. [Online]. Available: https://www.srf.ch/kultur/wissen/zu-klein-fuer-die-vermessung-abschied-vom-pizolgletscher

[2] A. Müller, "Flutwellen und Eislawinen: Was die Gletscherschmelze für die Schweiz bedeutet," 2019, [accessed 10.10.2019]. [Online]. Available: https://www.watson.ch/schweiz/umwelt/486803824-klimawandel-das-bedeutet-die-gletscherschmelze-fuer-die-schweiz

[3] "In der Schweiz ist das Wander-Fieber ausgebrochen," 2019, [accessed: 10.10.2019]. [Online]. Available: https://www.travelnews.ch/destinationen/12920-in-der-schweiz-ist-das-wander-fieber-ausgebrochen.html

[4] "Bergsturz im Bergell - Das Wichtigste zum Unglück bei Bondo," 2017, [accessed 12.10.2019]. [Online]. Available: https://www.srf.ch/news/schweiz/das-wichtigste-zum-unglueck-bei-bondo

[5] A. Pasztor, "Event-based Geophone Platform with Co-detection," Master's thesis, ETH Zurich, https://pub.tik.ee.ethz.ch/students/2017-HS/MA-2017-25.pdf, 2018.

[6] H. Müller, "Detecting Ambient Vibration Patterns using Multiple Sensors." ETH Zurich, 2019, Semester's thesis.

[7] P. Rüegg, "Monitoring the Matterhorn with millions of data points," 2019, [accessed 15.10.2019]. [Online]. Available: https://ethz.ch/en/news-and-events/eth-news/news/2019/08/monitoring-the-matterhorn-with-millions-of-data-points.html

[8] F. Sutton, R. Da Forno, D. Gschwend, T. Gsell, R. Lim, J. Beutel, and L. Thiele, "The Design of a Responsive and Energy-efficient Event-triggered Wireless Sensing System." in *EWSN*, 2017, pp. 144–155.

[9] F. Sutton, M. Zimmerling, R. Da Forno, R. Lim, T. Gsell, G. Giannopoulou, F. Ferrari, J. Beutel, and L. Thiele, "Bolt: A stateful processor interconnect," in *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2015, pp. 267–280.

[10] *STM32L47xxx microcontroller Reference manual*, 2018, Rev. 6. [Online]. Available: https://www.st.com/content/ccc/resource/technical/document/reference_manual/02/35/09/0c/4f/f7/40/03/DM00083560.pdf/files/DM00083560.pdf/jcr:content/translations/en.DM00083560.pdf

[11] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele, "Rocketlogger: Mobile Power Logger for Prototyping IoT Devices: Demo Abstract," in *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems*. ACM, 2016, pp. 288–289.

[12] *SCL3300-D01 3-axis inclinometer with angle output and digital SPI interface*, muRata, 2018, Rev. 1. [Online]. Available: https://www.murata.com/-/media/webrenewal/products/sensor/pdf/datasheet/datasheet_scl3300-d01.ashx?la=en-sg