
Universitat de les Illes Balears

Escola Politècnica Superior

**Memòria del Treball de Fi de Grau**

# Device for Movement Analysis Using Inertial Sensors

Bc. Bedřich Said

Grau de Enginyeria Informàtica

2017-18

DNI de l'alumne: 201613227

Treball tutelat per Dr. Antoni Jaume i Capó
Department of Mathematics and Computer Science
Supervisor: RNDr. Zdeněk Matěj, Ph.D. – Masaryk university, Faculty of Informatics

| S'autoritza la Universitat a incloure aquest treball en el Repositori Institucional per a la seva consulta en accés obert i difusió en línia, amb finalitats exclusivament acadèmiques i d'investigació | Autor | | Tutor | |
|---|---|---|---|---|
| | Sí | No | Sí | No |
| | | X | | X |

Paraules clau del treball:
Electronic Device, Printed Circuit Board, Inertial Sensors, Inertial Measurement Unit, Internet of Things, Movement Analysis, Horse

# CONTENTS

# ACRONYMS

**AHRS** Attitude and Heading Reference System

**API** Application Programming Interface

**ARM** Advanced RISC Machine

**BOM** Bill of Materials

**CPU** Central Processing Unit

**CSV** Comma-Separated Values

**DSP** Digital Signal Processing

**EAGLE** Easy Applicable Graphical Layout Editor

**ESP-IDF** Espressif IoT Development Framework

**GPIO** General-Purpose Input/Output

**IMU** Inertial Measurement Unit

**IoT** Internet of Things

**JTAG** Joint Test Action Group

**LED** Light-Emitting Diode

**Li-poly** Lithium polymer battery

**MEMS** Micro Electro Mechanical Systems

**Mo-Cap** Motion Capture

**PCB** Printed Circuit Board

**POSIX** Portable Operating System Interface

**RLC** Resistor (R), Inductor (L), Capacitor (C)

**SMD** Surface Mounted Device

**SMT** Surface Mount Technology

**SWD** Serial Wire Debug

**TCP** Transmission Control Protocol

**TDOA** Time Difference of Arrival

**UART** Universal Asynchronous Receiver-Transmitter

**USB** Universal Serial Bus

# ABSTRACT

Currently, the inertial sensors are more widely used, and their price is decreasing. The lower price allows creating new solutions with lower costs. Some examples can be found in wearable devices, mobile phones, navigation or control systems. There are several devices for various use cases on the market.

These devices usually cover their use cases and do not have any additional features like power independence, enough logging memory or openness for user code. The remaining hardware that fulfills all the conditions above is usually expensive.

I have developed a new wearable independent device for capturing and processing the measured data. The independence means no external wires and no external power supply here. The device is able to work outdoors, to log the measured data and to provide a direct output based on internal computations. The user can choose between completely wireless communication or wired connection to other electronics. The sensors measure inertial, attitude, position and atmospheric values.

For outdoor testing of the device I have selected the task about movement analysis of a horse. I placed the devices on the horse's body as wearable devices and I was developing algorithms for determination of the basic types of its movement – stand, walk, trot, canter or gallop.

In general, the developed device can be used for capturing data from sensors, onboard data processing, navigation or control of moving mechanics. The electronics work independently, so it is easy to install it on the measured or controlled objects.

CHAPTER 1

# INTRODUCTION

Currently, the inertial sensors are more widely used, and their price is decreasing. The decreasing price allows creating new solutions with lower costs. Some examples can be found in wearable devices, mobile phones, navigation or control systems. There are several devices for various use cases on the market.

These devices usually cover their use cases and do not have any additional versatility. For example, some wearable devices require permanent connection and do not have enough memory to log the captured data for several minutes. Another hardware needs an external power supply and cannot work independently. Some next boards are closed and do not allow to run user code. The remaining electronics that fulfills all the conditions above is very expensive.

The goal of this thesis is to develop a new independent device with several sensors and a user programmable controller. So, why we do not use a mobile phone? Well, the device should be able to run a real-time user program and to read the sensors at a precisely specified frequency. The size and weight of the hardware are also critical. With the decreasing price of MEMS (Micro Electro Mechanical Systems) sensors and controllers, we can build cheaper and lighter device than a smartphone.

In this thesis, I have utilized a microcontroller, which is new on the market. This microcontroller was released on September 2016, and it is designed for various real-time embedded tasks, and it is manufactured with on-chip WiFi and Bluetooth.

This thesis goes through the development process of the new wearable hardware including manufacturing, testing, API implementation, firmware for logging data to SD card and the first use of the prototype outside the laboratory. Each part is discussed in the separated chapter.

The main motivation was to create an independent wearable device with various sensors and enough memory for storing the measured data. The capacity of the battery and power consumption is also significant. During the design process, I have added some additional requirements that do not increase the cost very much and give us higher versatility and possibility to use the new hardware in more solutions.

When I finished the development process of the hardware, I had to prove the announced functionality of the device. I have selected one task about movement analysis of a horse using inertial sensors. The analysis of human movement is a widely studied problem. No other animal is so widely used in sports activities. For example, horses are the only animals participated in Olympics Games. When the device is mounted on a horse's body, the real-time analysis is able to detect the type of movement of this horse. With this task, I can compare the functionality of

my device to the other solutions available on the market. The movement analysis task is based on outdoor measurements so we can see the usability of my hardware and software outside of the laboratory conditions. These successful tests are mentioned in another part of this thesis.

When I was looking for a suitable algorithm for determination of the movement based on data from inertial sensors, I wrote my own algorithm with low requirements on the CPU power. The algorithm runs real-time with latency around one step of the horse. Lower latency cannot be achieved because one step is the basic element of each movement. During the outdoor tests, we could see the horse with the SensorBoard mounted on its back and the real-time results of the analysis on the screen of a connected tablet.

The algorithm first determines fundamental types of movements and then it is looking for more advanced ones based on previous results. The types of movement are defined as statements, so the new types can be detected without modification of the computation code. These tests achieved the last milestone of this thesis.

This thesis was created for a czech commercial partner Monet+ a.s., which is a provider of authentication services on the market.

CHAPTER 2

# HARDWARE

The hardware design is split into several steps in this thesis. First, I had to decide if it is really necessary to build a new device or if I can find a suitable solution on the market. When we want to select a suitable device for our task, we have to know the specification of the task itself. The task is specified in section 2.1. Based on the task we can specify all requirements that the selected device should meet (section 2.2). Now, we can start looking for a device that fulfills all the requirements (section 2.3).

When we do not find any suitable device on the market, we can start to think about creating a new one. It is not recommended to start any development before doing the previous steps.

When we already start development of new hardware we can think about adding some additional requirements for increasing the versatility of the final hardware. Of course, the new requirements must not rapidly increase the final prize. But if the hardware will be used only in the specific task, it is usually unnecessary to add any other functionality. In this case, I am adding some additional requirements in section 2.2.3, because I think that in this case, I can add very high versatility with very low additional cost. The analysis of the final expenses is in section 2.7.4.

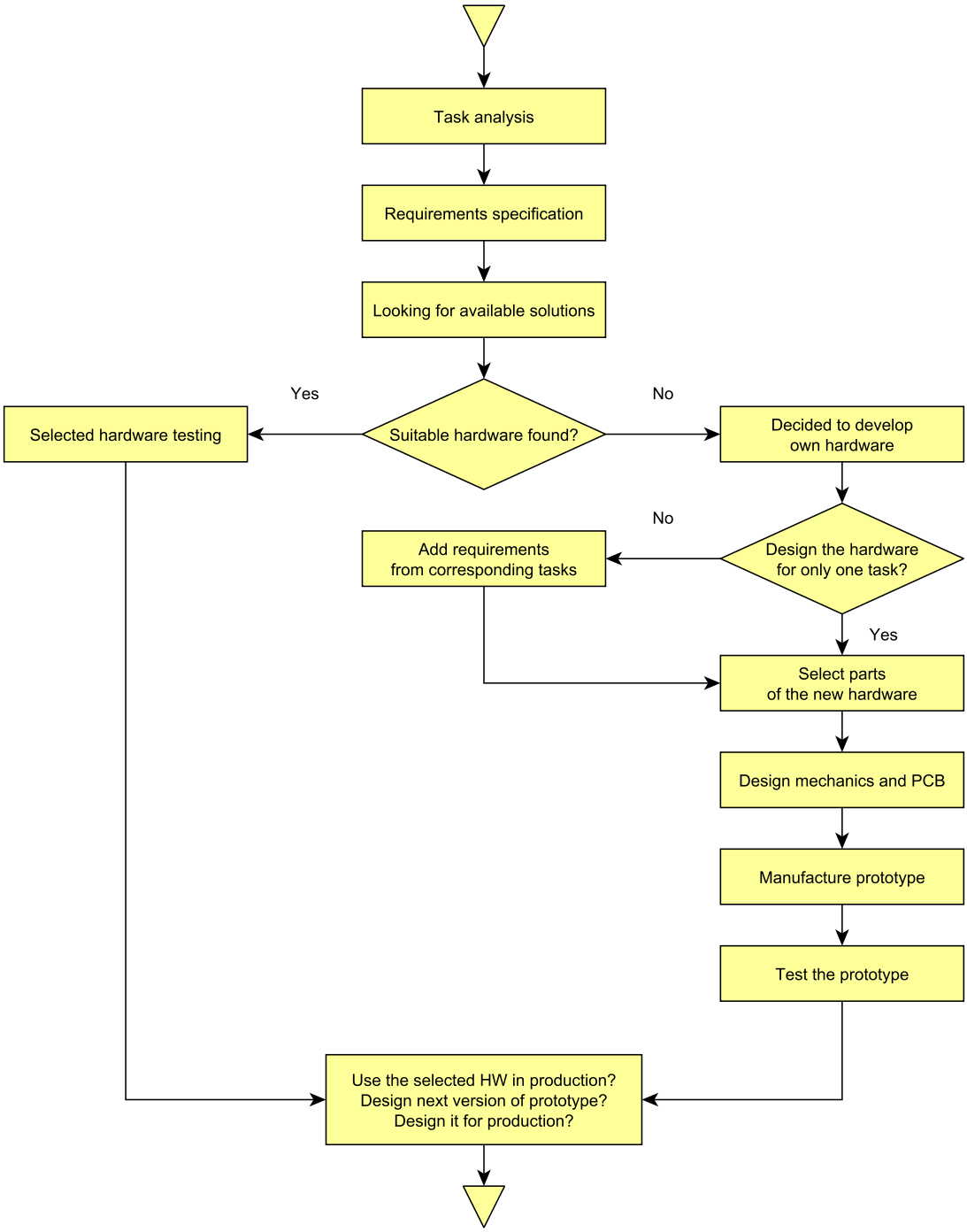
When I am decided to develop new hardware and I have the specification of all the requirements, I can start the development process. The selection of chips and other parts is in section 2.4 and the Printed Circuit Board (PCB) design is discussed in section 2.5.

Finally, the manufacturing of the first prototype is described in section 2.6. With the first prototype, it is time for testing. The tests should find as many errors as possible and they should proof us if the requirements were already met (section 2.7). The whole process is shown in figure 2.1.

## 2.1 Task Analysis

When we define our complete task in the first step, we can derive all requirements for our solution. Finally, we can check if our project was successful (based on the previously defined task and requirements).

Figure 2.1: Flowchart of the hardware design process



**Movement analysis task description:**

1. Measuring of the movement of a subject
2. Logging the measured data and sending them to post-processing
3. Analysing the movement and naming the specific categories of movements

The subject is an animal – horse in this thesis – or human being or a moving machine. The post-processing can be done real-time during the measurement process, but this is not mandatory. The analysis is primarily focused to recognize known movements in the logged data. For example, the categories of movement of a horse are a stand, walk, trot, canter, gallop, etc.

## 2.2 Requirements

The development of new hardware or software is usually driven by specified requirements. In this thesis, I have specified two groups of requirements. The first group is based on the selected task to solve. The second group has lower priority and specifies all the requirements that I found during other similar tasks with similar hardware. The second group of requirements is adding higher versatility to the final electronic device with very low additional cost. These two groups of requirements are later merged and used as a source for next development of the hardware.

The requirements below are based on several interviews with the commercial partner Monet+ a.s. company and with my tutor.

### 2.2.1 High level requirements

#### Measuring of the movement of a subject

The solution should work outdoor (in any weather conditions). It means that we cannot rely on any studio or laboratory-based technologies like Motion Capture (Mo-Cap). On the other hand, we can replace the passive or active markers with the whole sensors and remove the necessary cameras. The sensor-based electronics is easier to install and can be used in large and complex areas. Based on this outdoor requirement I will consider only wearable sensor systems.

#### Logging the measured data and sending them to post-processing

There are no wires acceptable for outdoor use, so we can log the data to internal memory or transmit them via any wireless technology. The wireless systems may not be fully reliable in complex areas with many objects. When we want to transmit the data directly, it is still better to log them internally for later downloads.

#### Analysis of the movement and naming the specific categories of movements

This is a software requirement, so it is not very important during the hardware design, but this requirement defines the sort of the data, which we need to measure. This defines what sensors we need, and this is a hardware requirement.

For the movement reconstruction, we primarily need the data about position and attitude of every sensor. Some methods for movement classification do not need to know the exact location (for example accelerometer-based step counter). This task is focused on developing a new analysis of the movement, so now I cannot exactly say which sensors will be needed in the future. I can make a prediction that the Inertial Measurement Unit (IMU) and some location

sensors will be needed. But there are some other sensors that produce interesting data according to the movement analysis, for example, a heart rate sensor.

Finally, I would like to add as many interesting sensors as possible, because it will give us more data sources. With more data sources we can get more detailed results. The other advantage is multifunctionality of the hardware. On the other hand, these additional sensors should be added only to additional requirements with lower importance. If they were added with high importance, the selection of appropriate hardware – based on the requirements – would be manipulated by some probably unnecessary sensors.

### 2.2.2 Low level requirements

Finally, I have chosen a wearable sensors technology. The next requirements are specific to this technology and focused on the selection of the devices. Let's call a used wearable device or devices as SensorBoard. The tables 2.1 and 2.2 shows the list of requirements for this SensorBoard.

### 2.2.3 Additional requirements

The additional requirements are not directly derived from the task, but they can increase the versatility of the final product. These requirements are shown in table 2.3.

## 2.3 Available solutions

The table 2.4 shows the overview of existing devices I have found.

## 2.4 Selection of parts for the new device

In this phase I have finally decided to develop new hardware. Now I am looking for suitable parts for this new electronic device. The table 2.5 and 2.6 lists all the components I took into account during the hardware development process.

## 2.5 PCB design

The PCB design should meet electrical and mechanical requirements, too. It is usually easier to design a large board in the first iteration, which is used only in the laboratory for software development and testing. Then the second iteration brings the first practically usable device and probably the third iteration is the first one dedicated to production use.

Here in this process, I will merge the first and the second version together. I will create a larger device which can be still used as a wearable device. This implies that I can do the software development, laboratory tests and the first outdoor tests with the same board designed during the first iteration. I am going to decrease the dimensions of the first prototype by splitting the PCB to separate sandwich modules. The testing process should give us advantages and disadvantages of this mechanical solution.

I have used CadSoft EAGLE (Easy Applicable Graphical Layout Editor) [34] during this process. I have finally chosen this editor because it has the availability of the libraries for the devices I wanted to use. I would probably use KiCad [35] if there were the same availability of the libraries.

### 2.5.1 Schematics and board layout

The schematics and the board layout are presented in appendix 5.

Table 2.1: SensorBoard low level requirements 1

| **Importance legend** | |
|---|---|
| Low | Nice to have |
| Medium | Very useful, reduce time or human effort |
| High | Mandatory, impossible without this functionality |

| **Low level requirements 1** | | |
|---|---|---|
| Category | Requirement | Importance |
| Power Supply | Accumulator | High |
| | Battery percentage indicator | High |
| | Charging when external power applied | High |
| | Voltage and current sensor | Medium |
| | Power LED | High |
| | Power switch | High |
| | Automatic selection between external and battery power | High |
| | Standardized charging connector | Medium |
| Sensors | Accelerometer | High |
| | Dynamic Gyroscope | High |
| | Magnetometer | High |
| | Indoor position | Medium |
| | Outdoor position | Low |
| | Barometer | Low |
| | Board temperature | Low |
| Communication | Allow user programming | High |
| | Wireless programming | Low |
| | Wired programming | High |
| | Wireless | High |
| | Standardized protocol | High |
| | Wired access | High |
| | Connector for external sensors | Low |

Table 2.2: SensorBoard low level requirements 2

| **Low level requirements 2** | | |
|---|---|---|
| Category | Requirement | Importance |
| Functions | Logging all data for several hours | High |
| | Sensor fusion coprocessor | Low |
| | LED indicators | High |
| Mechanical | Wearable design | High |
| | Dimensions under 6 cm | High |
| | Dimensions under 3 cm | Low |
| | Weight under 50 g | High |
| | Weight under 20 g | Low |
| Software | Control multiple devices simultaneously | Medium |
| | Download logged data | High |
| | Streaming data during measurement | Medium |
| | Start logging on multiple devices by clicking one button | Medium |
| | Time synchronization of multiple devices | High |
| | Possibility to upload data to a server | High |

Table 2.3: Additional requirements for the SensorBoard

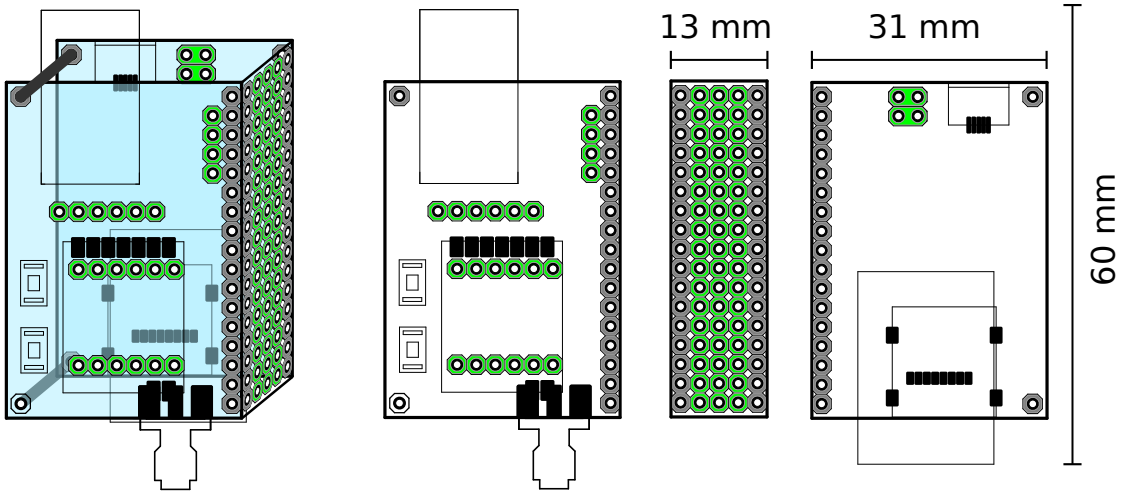
| **Importance legend** | |
|---|---|
| Low | Nice to have |
| Medium | Useful, can significantly improve functionality |
| ~~High~~ | Not applicable |

| **Low level additional requirements** | | |
|---|---|---|
| Category | Requirement | Importance |
| Functions | Coprocessor for periodic computations like sensor fusion | Low |
| | Piezo buzzer | Low |
| | Buttons | Medium |
| Sensors | Ambient light | Low |
| | Humidity | Low |
| | Sound (microphone) | Low |
| | Ambient temperature | Low |
| Communication | UART, I2C, SPI connector | Low |
| | PWM outputs | Low |
| | Analog inputs | Low |
| Software | NMEA input | Low |

Table 2.4: Available solutions

| Available solutions | | | |
|---|---|---|---|
| Device | Link | Price | Main disadvantages |
| MetaWear | [1] | $ 80 | Low memory, impossible longer logging. Loosing packets during streaming. |
| ArduPilotMega | [2] | $ 250 | Dependent on external power supply, higher dimensions. |
| pixHawk | [3] | $ 260 | Dependent on external power supply, higher dimensions. |
| Xsens MVN | [4] | $ 12000 | Good solution with very high price. |
| X-IMU | [5] | $ 400 | Price. |
| Smart Phone | N/A | $ 100 | Impossible logging at higher frequencies, inaccurate logging frequency. |
| Fitness sensors | N/A | $ 100 | Mostly closed commercial projects. |

Figure 2.2: SensorBoard dimensions




### 2.5.2 Mechanical layout and connectors

The mechanical layout is shown in figure 2.2.

### 2.5.3 Highlights of the board layout

The selected parts are mainly placed and connected on the PCB according to the recommendations in the datasheets. There are some differences in the SensorBoard layout that should be highlighted in this section.

#### Power LEDs

When we have more power supplies on the prototype, it is a good way to add a power Light-Emitting Diode (LED) to all the supplies circuits. For example the SensorBoard has 5 V supply,

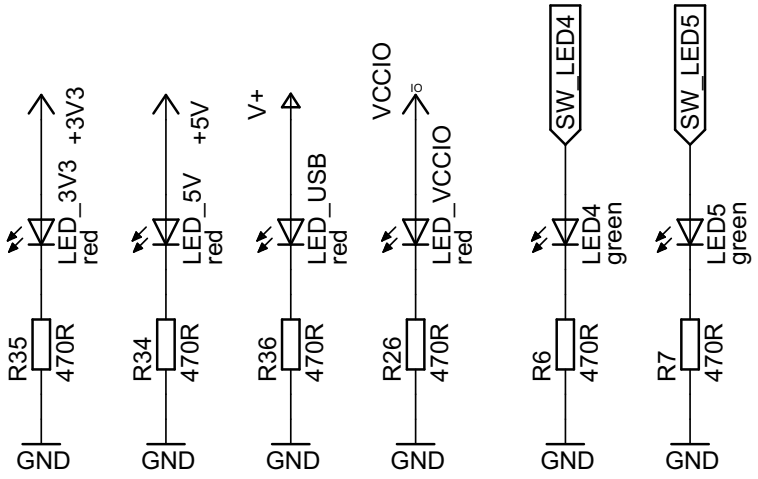
Table 2.5: Selection of parts for the new electronic device 1

| Available solutions | | | |
|---|---|---|---|
| Part ID | Description | Datasheet | Selected |
| TACTM-35N-F | Button | [6] | Yes |
| ADP5062 | Power management and battery charger | [7] | Yes |
| ADP5063 | Power management and battery charger | [8] | No |
| SI7006 | Humidity and temperature sensor | [9] | Yes |
| HDC2010YPAR | Humidity and temperature sensor | [10] | No |
| SHTC1 | Humidity and temperature sensor | [11] | No |
| DWM1000 | TDOA location sensor (with antenna) | [12] | Yes |
| DW1000 | TDOA location sensor (only sensor) | [13] | No |
| BMP280 | Barometer | [14] | Yes |
| BMP388 | Barometer | [15] | No |
| FT232R | UART to USB converter | [16] | Yes |
| CP2102 | UART to USB converter | [17] | No |
| CH340E | UART to USB converter | [18] | No |
| MPU-9250 | Accelerometer, dynamic gyroscope, magnetometer | [19] | Yes |
| BMI160 | Accelerometer, dynamic gyroscope | [20] | Yes |
| BMF055 | Accelerometer, dynamic gyroscope, magnetometer, ARM microcontroller | [21] | Yes |
| GY953 | Backup accelerometer, dynamic gyroscope and megnetometer with embedded pitch, roll and yaw angle estimation (sensor fusion) | [22] | Yes |
| HM-TRP | Long range 433 MHz radio | [23] | Yes |
| MOLEX-47219 | Micro SD card holder | [24] | Yes |
| LTR-329ALS | Digital ambient light sensor | [25] | Yes |
| EAALSTIC1708A0 | Analog ambient light sensor | [26] | No |
| NOA1212 | Analog ambient light sensor | [27] | No |
| ESP-WROOM-32 | Dual core controller with certified WiFi and Bluetooth | [28] | Yes |
| STM32 | ARM microcontroller | [29] | No |
| UM18533 | Linear stabilizer 3.3 V | [30] | Yes |

Table 2.6: Selection of parts for the new electronic device 2

| Available solutions | | | |
|---|---|---|---|
| Part ID | Description | Datasheet | Selected |
| LF33 | Linear stabilizer 3.3 V | [31] | No |
| USB-MICRO | Micro USB connector | [32] | Yes |
| Pin header | Servo connector 2.54 mm | [33] | Yes |

Figure 2.3: Schematics of the power LEDs




3.3 V supply, USB supply and digital power supply. The four LEDs give us the information that all the circuits are operating properly. Only one power circuit usually stops operation when it is overloaded and the stabilizer cannot provide enough power. The schematic of the signalization LEDs is shown in figure 2.3.

#### Automatic bootloader

When we connect the SensorBoard to a computer using the Universal Serial Bus (USB) cable, we would like to reset the ESP32 controller distantly from the computer and upload a new program by one click. This is allowed by two transistors shown in figure 2.4. The same solution is used in PCB layout of ESP32-DevKitC. [36] The transistors control the `ENABLE` and `RESET` pin on the ESP32 based on the `RTS` and `DTR` value on the Universal Asynchronous Receiver-Transmitter (UART) comming from USB via FTDI chip. [16].

#### ESP32 pins selection

The ESP-WROOM-32 [28] has only 26 usable General-Purpose Input/Output (GPIO)s. So, we cannot connect so many peripherals to the ESP32 controller. The final pin distribution is shown in figure 2.5.

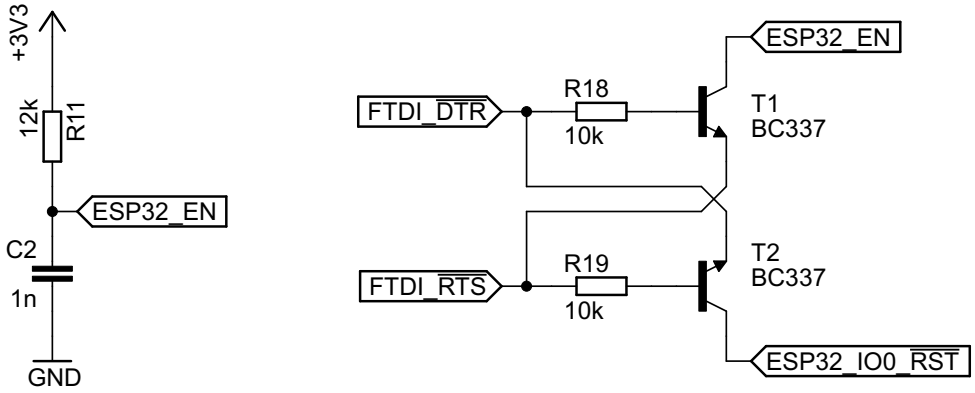
#### Layout of the buttons

When a button connected to a controller is pressed, the controller usually detects more pushes. This behavior is caused by the internal mechanics of the button and can be solved by adding a capacitor. The figure 2.6 shows the added capacitors to the buttons on the SensorBoard.

#### Optional 10k pull-up on RESET pin

When I was connecting the 10 kΩ pull-up resistor to RESET pin on the controller, I have added a soldering pad. The pad allows disconnecting the pull-up resistor when the RESET pin is controlled from another source.

Figure 2.4: Schematics of the boot transistors




## 2.6 Manufacturing

The manufacturing process of the prototype consists of these steps:

1. Collecting source data – schematics and board layout (in this thesis in Eagle)
2. Manufacturing of the Printed Circuit Board (PCB)
   a) Panelization of the board layout
   b) Adding calibration markers
   c) Export gerber files
   d) Send exported files to manufacturing company
3. Machine surface mount technology (Surface Mount Technology (SMT))
   a) Export data for the template for applying solder paste
   b) Export partlist – the list of all devices with their coordinates
   c) Export bill of materials (Bill of Materials (BOM))
   d) Manufacturing of the template for applying solder paste
   e) Order all devices according to BOM
   f) Sending template for applying solder paste, packages with devices and part list to the manufacturer
4. Finalization of the prototype
   a) Hand soldering of some remaining devices like connectors or wires
   b) Completing the final prototype from possible parts and packaging
   c) Applying power and first testing

Figure 2.5: Schematics of the ESP32 pin distribution

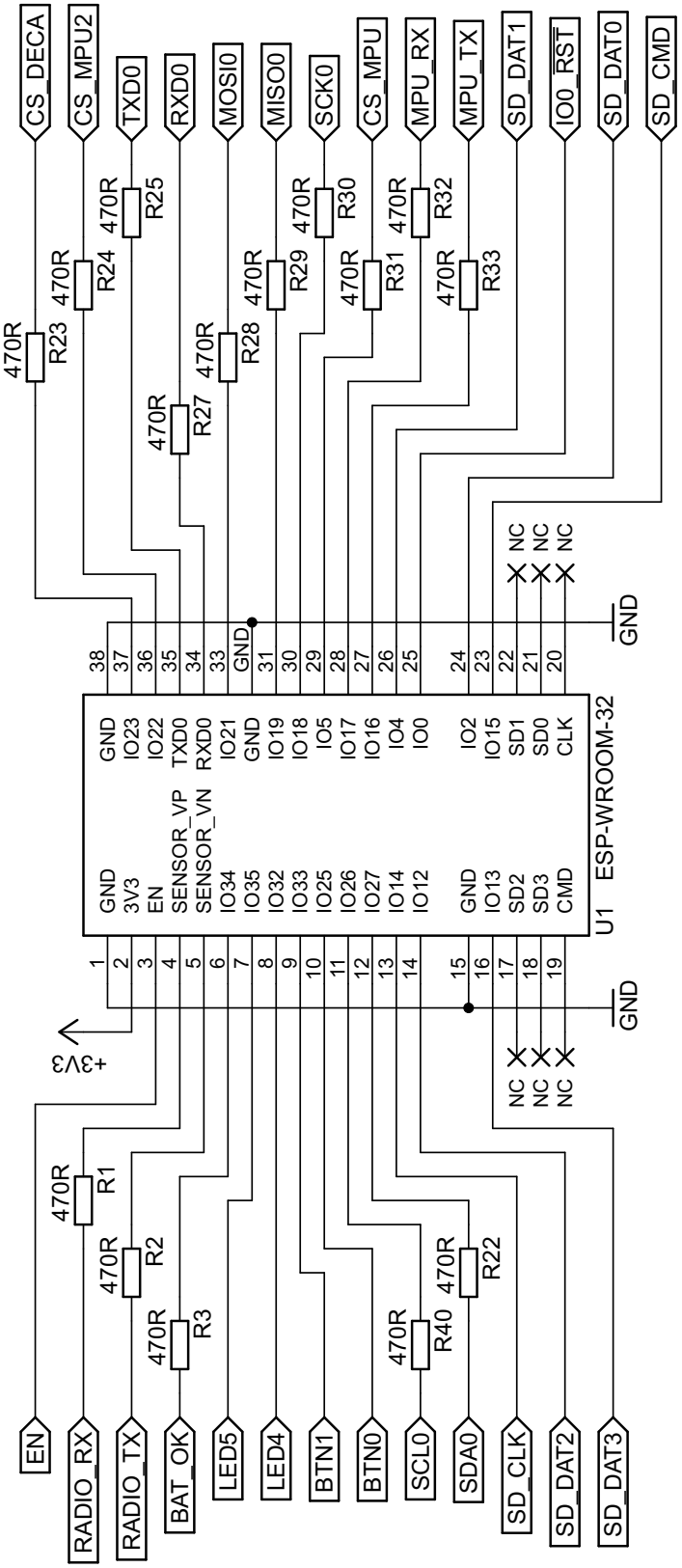
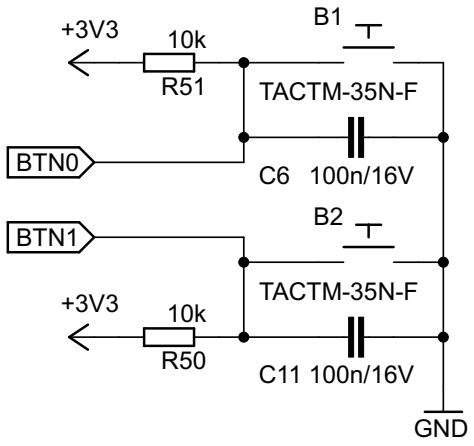


Figure 2.6: Schematics of the buttons on the SensorBoard




### 2.6.1 Collecting source data

The source data may be in various formats according to the used tools. I used CadSoft Easy Applicable Graphical Layout Editor (EAGLE) [34] during the development, but there are many other tools available on the market. It depends on the manufacturer company if they accept the source data in my format. They usually accept the source data, but they usually apply a special fee. All PCB editors should be able to export the manufacturing data in standardized formats. (Sometimes it is very hard or impossible to edit the exported data.)

### 2.6.2 Manufacturing of the Printed Circuit Board

When we have more than one board, we should assemble all boards onto one or more panels. If we plan to solder the devices using the SMT, we should add a calibration markers on the final panel. I have followed the instructions from SMTplus company [37].

First, I have panelized the separated boards and I have added the calibration markers according to the rules of SMTplus company [38]. Finally, I have exported the gerber files using a CAM job [39] in EAGLE. I have followed the design rules for class 3 by Gatema a.s. company [40].

### 2.6.3 Machine surface mount technology

We need the PCB, template for soldering paste and all devices in this step. Some PCB manufacturers offer to create the template as well, so this is the way I used. The thickness of the template indicates the volume of the paste on each pad. For the small Surface Mounted Device (SMD) parts I used 100 µm, but I recommend to discuss the parameters with the manufacturer. The used template is shown in appendix 5.

The BOM and part list export depends on the used design tools. When we connect used devices in the sources directly with their part numbers on selected shops, we can simply export the part list and then import it on the seller's website. I recommend buying some spare devices to minimize the risk of their damage.

### 2.6.4 Finalization of the prototype

We can discuss the finalization work with the manufacturer, but when we create a prototype, I would like to recommend to do this job by yourself. It allows to perform some tests before packaging and we do not have to spend time and money with preparation of the documentation about how to assemble and package our prototype. Before assembling, the different software developers and testers can receive the different parts of the prototype for their next work.

## 2.7 Testing

The testing process has been split into two parts: the laboratory testing during the software development and the outdoor testing. I have found some errors in the version 1.0 during the laboratory testing. These errors are described in Errata list located in section 2.7.1. I did some other tests against the specification of used devices. The results show us what components should be used in the next version of the hardware and what devices are redundant. The results are shown in section 2.7.2. The outdoor tests did not find any significant errors, but there were found several recommendations for the future versions of the SensorBoard. These recommendations are mentioned in section 2.7.3.

### 2.7.1 Errata

The tables 2.7 and 2.8 show all errors I have found during software development and laboratory testing of the SensorBoard prototype.

### 2.7.2 Device testing

The device testing helped me to select the parts recommended for the future versions of the SensorBoard. It also helped me to mark all spare parts in the prototype. The table 2.9 shows the results.

### 2.7.3 Recommendations for the next version

Recommendations for the future versions of the SensorBoard based on the outdoor testing:

- Replace software LEDs by one or more RGB LEDs. For example WS2812 RGB LED [41].
- Move the sensor fusion and computations to Atmel SAM D21 [42] processor inside the BMF055 [21] chip. Keep the ESP32 [28] processor only for communication and user computations.
- Decrease board dimensions using 4 layer PCB. The increased cost for four layer board may be saved on a smaller surface.
- Sometimes it is better to have all SMD parts on one side of the board. The manufacturing is cheaper and the space on the second side can be used, for example, for battery mounting or connectors.
- Split the board area to an area for sensors, an area for power electronics and an area for computing. The power distribution is easier in this situation and the sensors are less influenced by other electronics.
- Use another driver with more capabilities for USB connection. In this version, the USB on board supports only UART communication on the virtual COM port. The File Transfer

Table 2.7: Errata of the SensorBoard 1

| Severity legend | |
|---|---|
| Low | No significant affects |
| Spare | Only spare devices may not work |
| Medium | Some part of the prototype may not work |
| High | The prototype has to be remade |

| Errata 1 of 2 | | | |
|---|---|---|---|
| Number | Description | Error created during | Severity |
| 1 | Bridge between boards under micro USB connector<br><br>What happens:<br>The micro USB connector may be placed with lower accuracy | panelization | Low |
| 2 | Swapped MISO and MOSI on pins IO19 and IO21 on the ESP-WROOM-32 chip according to ESP32 Arduino standard<br><br>What happens:<br>The SPI interface is not working in Arduino compatible mode without modification of the Arduino SPI library | schematics design | Medium |
| 3 | LED5 connected to IO35 on ESP-WROOM-32 is not working<br><br>What happens:<br>The pins IO34 – IO39 are input only, so they cannot drive a LED | schematics design | Spare |
| 4 | Missing pull-up resistors on SD card<br><br>What happens:<br>The SDIO interface needs external pull-up resistors to work properly. I have added these resistors later by hand. | schematics design | Medium |

Table 2.8: Errata of the SensorBoard 2

| **Errata 2 of 2** | | | |
|---|---|---|---|
| Number | Description | Error created during | Severity |
| 5 | The temperature measurement is placed very close to the main processor<br><br>What happens:<br>The processor heating affects the measured temperature | board layout | Spare |
| 6 | Low current voltage regulator placed on power supply<br><br>What happens:<br>When we use WiFi and SD card simultaneously, some brownouts can be detected. I have added a bigger voltage regulator and a capacitor later by hand. | schematics design | Medium |
| 7 | The light sensor is placed on inner side of the board<br><br>What happens:<br>The measured values are affected by shadow of the board. | board layout | Spare |
| 8 | Missing safety resistors on pins with buttons. These pins are directly connected to processor.<br><br>What happens:<br>When the pins are defined in software to be used in different way, incorrect connection can burn the processor. | schematics design | Medium |

Table 2.9: Parts of the SensorBoard recommended for the future versions

| Decision legend | |
|---|---|
| Selected | Recommended for use in future versions |
| Possible | Not necessary, but has use-cases, may be replaced with another part |
| Removed | Not recommended or will be spare |

| Parts of the SensorBoard recommended for the future versions | | | |
|---|---|---|---|
| Part ID | Description | Datasheet | Selected |
| TACTM-35N-F | Button | [6] | Selected |
| ADP5062 | Power management and battery charger | [7] | Selected |
| SI7006 | Humidity and temperature sensor | [9] | Selected |
| DWM1000 | TDOA location sensor (with antenna) | [12] | Selected |
| BMP280 | Barometer | [14] | Selected |
| FT232R | UART to USB converter | [16] | Selected |
| MPU-9250 | Accelerometer, dynamic gyroscope, magnetometer | [19] | Removed |
| BMI160 | Accelerometer, dynamic gyroscope | [20] | Removed |
| BMF055 | Accelerometer, dynamic gyroscope, magnetometer, ARM microcontroller | [21] | Selected |
| GY953 | Backup accelerometer, dynamic gyroscope and megnetometer with embedded pitch, roll and yaw angle estimation (sensor fusion) | [22] | Removed |
| HM-TRP | Long range 433 MHz radio | [23] | Possible |
| MOLEX-47219-2001 | Micro SD card holder | [24] | Selected |
| LTR-329ALS | Digital ambient light sensor | [25] | Selected |
| ESP-WROOM-32 | Dual core controller with WiFi and Bluetooth | [28] | Selected |
| UM18533 | Linear stabilizer 3.3 V<br>Note: Insufficient current | [30] | Removed |
| LF33 | Linear stabilizer 3.3 V | [31] | Selected |
| USB-MICRO | Micro USB connector | [32] | Selected |
| Pin header 2.54 mm | Servo connector | [33] | Possible |

protocol for the SD card should be supported, too. (If it is needed we can add full USB host support or USB-C compatibility.)

- Add specialized pads for an oscilloscope. The pads are connected to the signal and to the ground, so the oscilloscope measurements are more accurate.
- The main processor should be able to switch on/off the power supply of the other parts (sensors). The sensors support sleep modes, but we cannot completely switch them off to save more energy from the battery.
- If we recognize that the ESP-WROOM-32 controller is not powerful enough, we can use a fully compatible alternative with larger memory ALB32-WROVER [43]. It should be possible to compile a terminal based Linux distribution for this controller.

### 2.7.4 Analysis of additional costs

Here, the analysis of additional costs means counting the prices of all parts that were finally spare or were not used. The table 2.10 counts all parts of the SensorBoard and their prices.

The column "Used" tells us if the item was used for any use case or if it was spare at all times. If any item is still unused, it does not mean that it cannot be used in some future use case. The column "Is necessary" points the basic items, the SensorBoard cannot work or cannot be built without them.

It means that the green cost "Total used and necessary" is the lowest cost of the prototype without any additional functionality. The yellow cost "Total used and unnecessary" is the cost of all items that had significant value during development, but were not mandatory for the main task. These items also made the development process easier and faster, so I do not count this cost as additional. The last red cost "Total unused" covers all items that were not used. These items were a part of the design because they lowered some risks in functionality. For example, if I was not sure that an accelerometer would work properly, I added another spare accelerometer, which should cover this risk. I could get a working prototype in the first iteration using this strategy. I finally count the red cost as a cost of decreased risks and as a cost of higher versatility.

We have to take into account that all the mentioned costs are costs of the prototype. The same items would have significantly lower costs during the production of more devices.

### 2.7.5 Comparison of the used IMUs

There are three different accelerometers, dynamic gyroscopes and two magnetometers on the prototype. Here is the right time to select the most appropriate IMU. Of course, we can use all the sensors and get the data with higher precision. The three different sensors on the prototype allow testing many more functions and experiments.

- **BMI160:** ($ 4.3) [20]
  - \+ The most accurate gyroscope and accelerometer
  - − Without magnetometer
- **BMF055:** ($ 11) [21]
  - \+ Good accuracy
  - \+ Accelerometer, dynamic gyroscope, magnetometer and Advanced RISC Machine (ARM) controller in one package
  - \+ Heated to constant temperature if the controller is in constant load

Table 2.10: Additional costs calculation

| Additional cost calculation | | | |
|---|---|---|---|
| Part ID | Price | Used | Is necessary |
| PCB | $ 23 | Yes | Yes |
| SMT job | $ 65 | Yes | Yes |
| TACTM-35N-F | $ 0.4 | Yes | No |
| ADP5062 | $ 3.8 | Yes | Yes |
| SI7006 | $ 2 | No | No |
| DWM1000 | $ 22 | Yes | No |
| BMP280 | $ 4 | Yes | No |
| FT232R | $ 5 | Yes | Yes |
| MPU-9250 | $ 10 | Yes | No |
| BMI160 | $ 4.3 | No | No |
| BMF055 | $ 11 | Yes | Yes |
| GY953 | $ 11 | No | No |
| HM-TRP | $ 13 | Yes | No |
| MOLEX-47219-2001 | $ 1.2 | Yes | Yes |
| LTR-329ALS | $ 0.7 | No | No |
| ESP-WROOM-32 | $ 12 | Yes | Yes |
| **Total used and necessary** | **$ 121** | Yes | Yes |
| **Total used and unnecessary** | **$ 49.4** | Yes | No |
| **Total unused** | **$ 18** | No | No |

  - Variable temperature of the chips when the controller is not under constant load

- **MPU9250:** ($ 10) [19]
  - \+ Highest community support
  - – Maybe in the end of its lifecycle
  - – Less accuracy than the other presented sensors

I would like to recommend to use the BMF055 chip for the next versions of the SensorBoard, but it does not mean that I recommend this chip to any use case. It heavily depends on the selected task and requirements.

# CHAPTER 3

# SOFTWARE

The SensorBoard hardware has two programmable CPUs and several configurable chips. The main ESP32 CPU (two low-power Xtensa 32-bit LX6 microprocessors) [28] is programmable via USB or Bluetooth or Joint Test Action Group (JTAG). The JTAG connector is not present on the SensorBoard. The second microcontroller is a part of BMF055 [21] multifunctional chip. It is Atmel SAMD20 [42] with ARM Cortex-M0+ CPU programmable via Serial Wire Debug (SWD) interface [44].

**Microcontrollers:**

1. Espressif ESP-WROOM-32 [28]:
   - dual core, 240 MHz, 448 kB ROM, 520 kB SRAM, 4 MB SPI flash memory
   - Designated for main program handling all communication and interaction with user or other devices.
2. Atmel SAMD20 [42]:
   - ARM Cortex-M0+ CPU, 48 MHz, 32 kB SRAM, 256 kB flash memory
   - Designated for processing inertial data (for example computing sensor fusion), it can be used for example as an emulator of other sensors or as a simple flight controller.

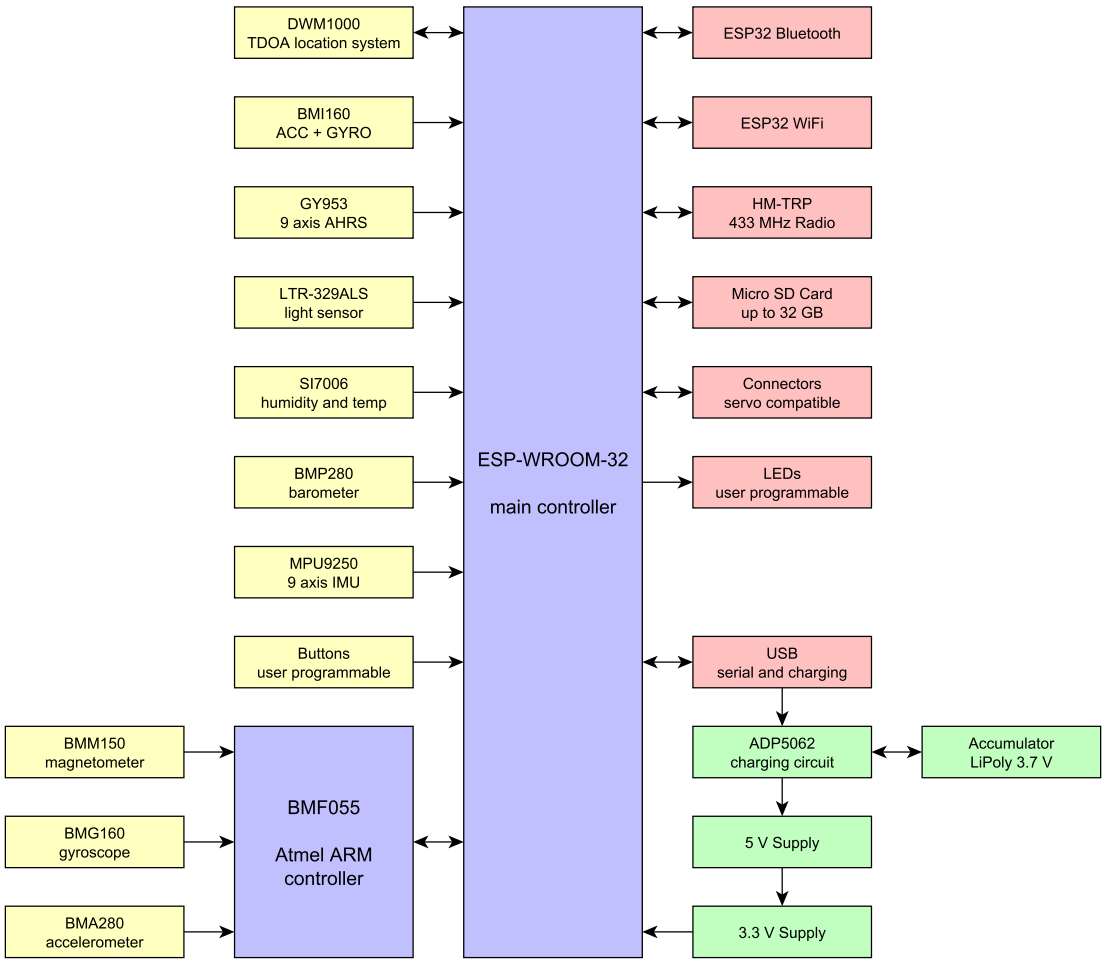
## 3.1 Programming the Board

Each processor on the SensorBoard has to be programmed separately via its interface. Only ESP32 [28] (main processor) can be programmed over the air via Bluetooth. This feature is disabled by default. The pinout and other information about the SensorBoard and BMF055 board are described in appendix 5.

### 3.1.1 Programming ESP32

There are several ways how to program and use the ESP32 [28] controller. The official framework is Espressif IoT Development Framework (ESP-IDF) [45] and supports all the chip functionality. The ESP-IDF framework is Portable Operating System Interface (POSIX) compatible.

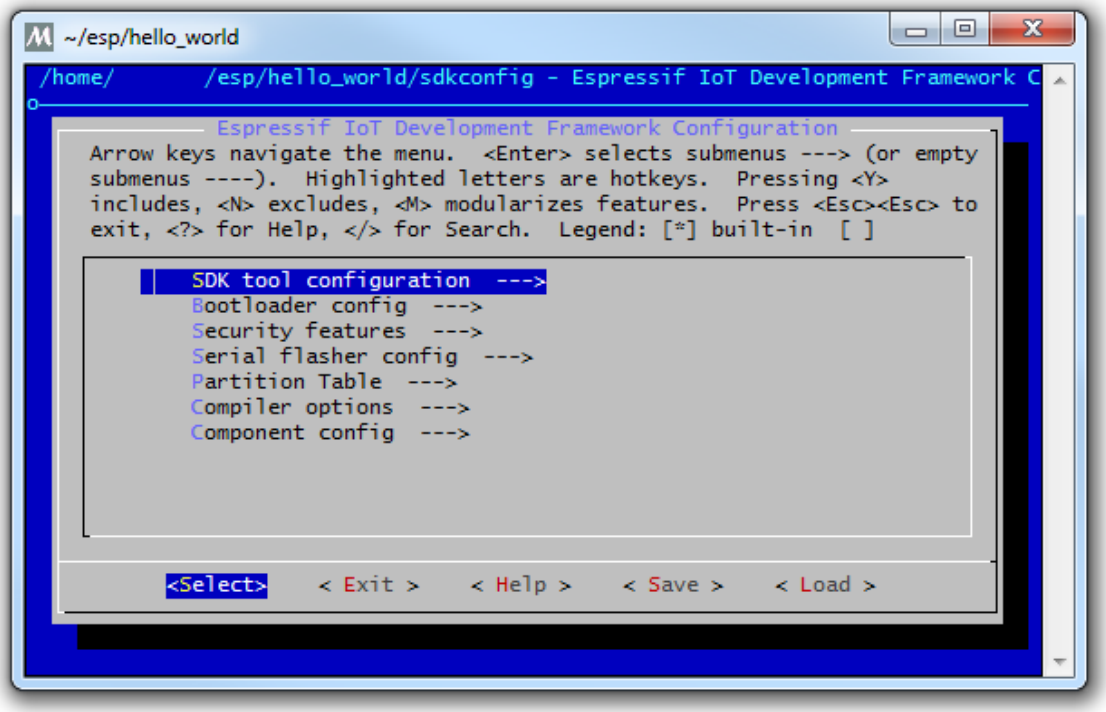
Figure 3.1: Schema of available modules inside the SensorBoard hardware




Legend:

| | |
|---|---|
| Yellow | Sensors and other input components |
| Blue | User programmable controllers |
| Green | Power supply circuits |
| Red | Wired and wireless communication and data storage |

Figure 3.2: Configuration of the ESP-IDF program in terminal before the first compilation




The chip can be programmed using Arduino compatible framework [46] which creates an easy way for prototyping and learning, but does not offer all the functionality of the chip. The Arduino framework uses the ESP-IDF framework, so we can create "hybrid" programs that use both frameworks.

The last mentioned programming method is scripting in Python. We can upload the MicroPython [47] firmware directly to the ESP32 controller. The Python libraries, scripts and other files are stored on the SD card. The MicroPython firmware supports most of the hardware functionality, but there are still some restrictions.

Although, there are some other ways how to create a program for this hardware, many of them use one of the mentioned frameworks. For example, we can create a program in Simulink and then export the code to the ESP32 controller. [48]

#### ESP-IDF Framework

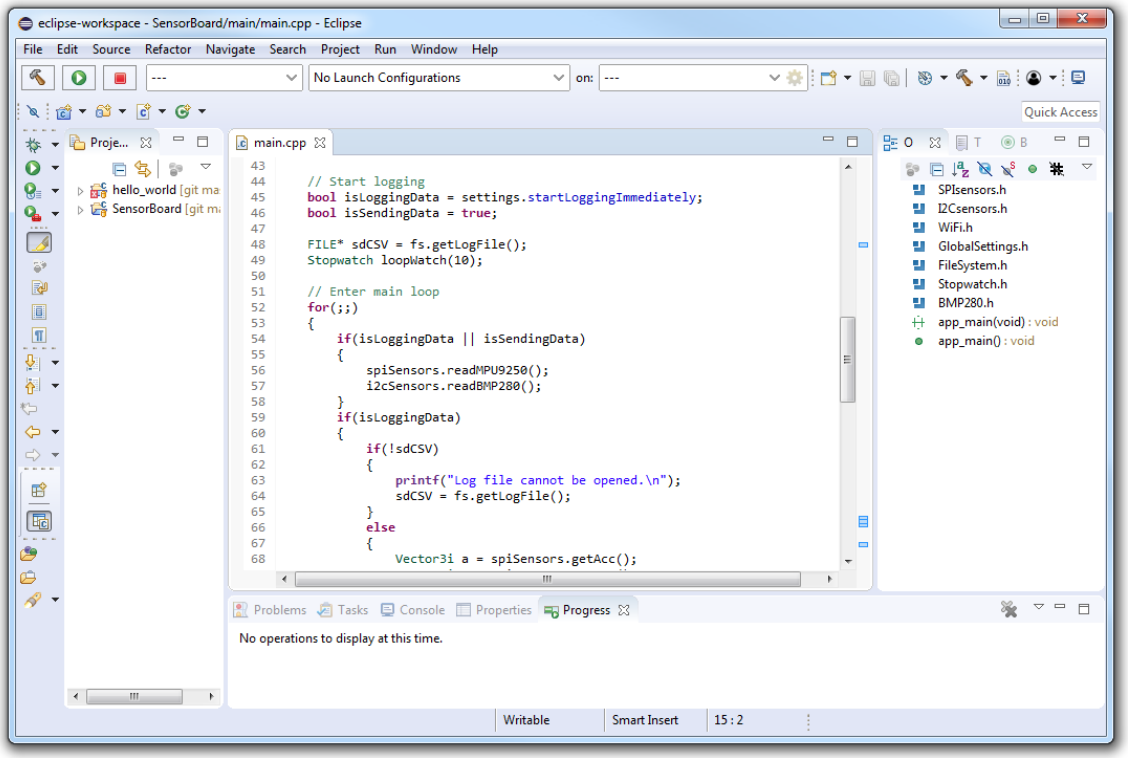
The ESP-IDF framework is almost POSIX compatible. [49] It supports many POSIX compatible functions, but it still does not support all of them. It means that we can compile many POSIX compatible programs for ESP32, but not all of them.

We can follow the official ESP32 programming guide [50] to set up the environment and program the board. The user can do everything from the terminal (command line). The figure 3.2 shows a configuration tool used for setup the program before the first compilation and upload to the ESP32 microcontroller.

If we prefer some GUI for development of our software we have several options. I will mention two of them:

1. **Eclipse IDE** can be setup using the guide in ESP-IDF Programming Guide [51]. In my opinion, the compilation of our programs is very slow when we use this option. The figure 3.3 shows the SensorBoard project opened in Eclipse IDE.

Figure 3.3: The SensorBoard project opened in Eclipse IDE




2. **PlatformIO IDE** is an open source ecosystem for IoT development. [52] There is integrated ESP-IDF and Arduino framework for ESP32 microcontrollers. This option was more familiar to me with much faster compilation process. I have used the PlatformIO ecosystem inside Atom [53] advanced text editor. The figure 3.4 shows the SensorBoard project opened in Atom editor with PlatformIO plugin.

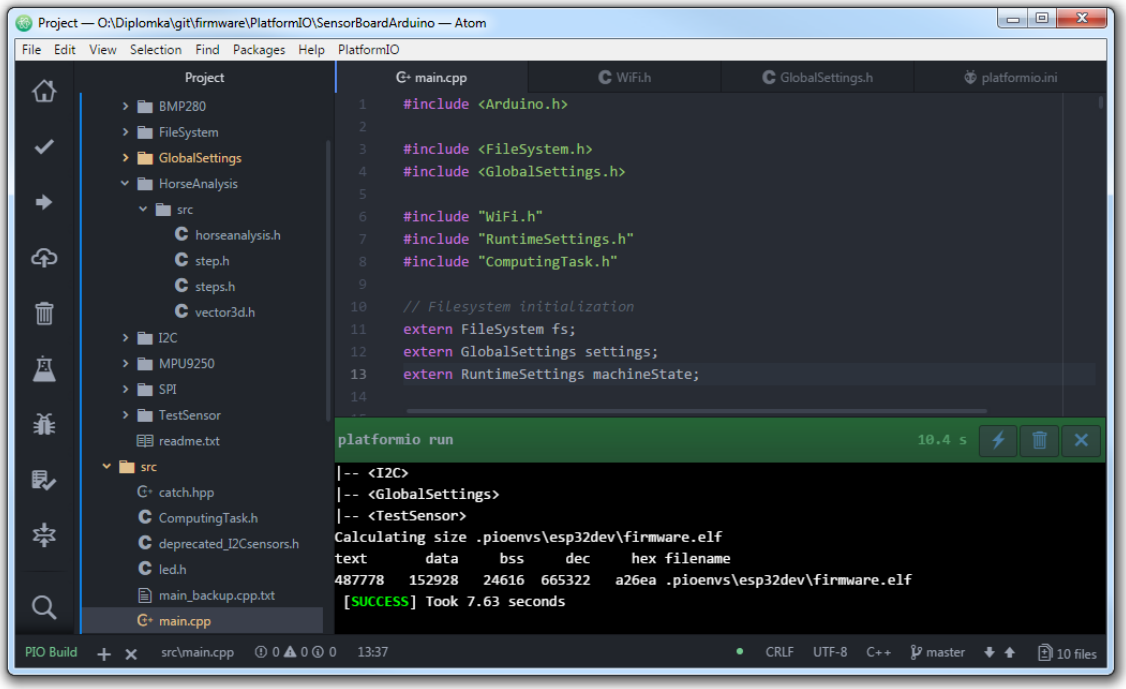
### Arduino Compatibility

The Arduino compatible framework for ESP32 [46] is dependent on ESP-IDF framework [45]. It means that we can use both – the ESP-IDF functions and the Arduino functions – in our programs. The Arduino framework is primarily targeted for beginners and for fast prototyping. I do not recommend to use this framework for advanced applications, because it does not cover the whole ESP32 functionality. We can develop our Arduino compatible programs in Arduino IDE, but we can use the PlatformIO ecosystem, too. It means that we can use the same environment like in the figure 3.4. All libraries are downloaded and installed automatically inside the PlatformIO. We have to set only the `platformio.ini` file in the root directory of our project. We can add a line `framework = arduino` for Arduino compatible programs and a line `framework = espidf` for programs using the ESP-IDF framework.

### MicroPython Compatibility

The built MicroPython binary for ESP32 can be directly downloaded from MicroPython website. [47] We can directly flash this binary to our ESP32 controller (to the SensorBoard) and then directly run our Python scripts. Of course, it is possible to download the MicroPython source code from the same website. The MicroPython allows to control the SensorBoard via Python serial terminal, so we can send a separate command via this serial terminal or run the whole

Figure 3.4: The SensorBoard project opened in Atom text editor with PlatformIO plugin




Python scripts stored as files in the SD card. The SD card slot is a part of the SensorBoard. One of the Python scripts on the SD card can be configured to be executed automatically after powering on the SensorBoard. The running Python serial terminal on the SensorBoard is shown in figure 3.5.

### 3.1.2 Programming BMF055

The BMF055 [21] is a custom programmable 9-axis motion sensor. It is a single chip triaxial accelerometer, dynamic gyroscope, magnetometer and ARM controller. The BMF055 chip is mounted on a separated board, which can be optionally mounted to the SensorBoard or it can be used independently. The figure 3.6 shows the separated BMF055 board and the same board mounted to the SensorBoard.

The BMF055 board has several usages because it contains a user programmable controller. It can be used for example as a simple UAV controller or autopilot. The project BMF055-flight-controller [54] implemented an autopilot solution for multicopters on this controller.

The Atmel SAM D20 controler [42] can be programmed in Atmel Studio [55] and the program can be flashed to the chip via SWD interface [44]. We have to use Atmel ICE programmer [56] or similar hardware. For details about configuration and programming of the BMF055 chip we can follow the BMF055 datasheet [21] or the Atmel SAM D20 datasheet [42]. The pinout of the BMF055 board and other hardware details are described in Appendix 5. The figure 3.7 shows an opened BMF055 project in Atmel Studio 7.

## 3.2 Application Programming Interface (API)

The programming interface for the SensorBoard can be split into several categories. There is a native support from the manufacturer of the microcontrollers. This API is written for any electronic device with the targeted microcontroller and this API is mentioned in section 3.2.2.

Figure 3.5: The running Python serial terminal on the SensorBoard

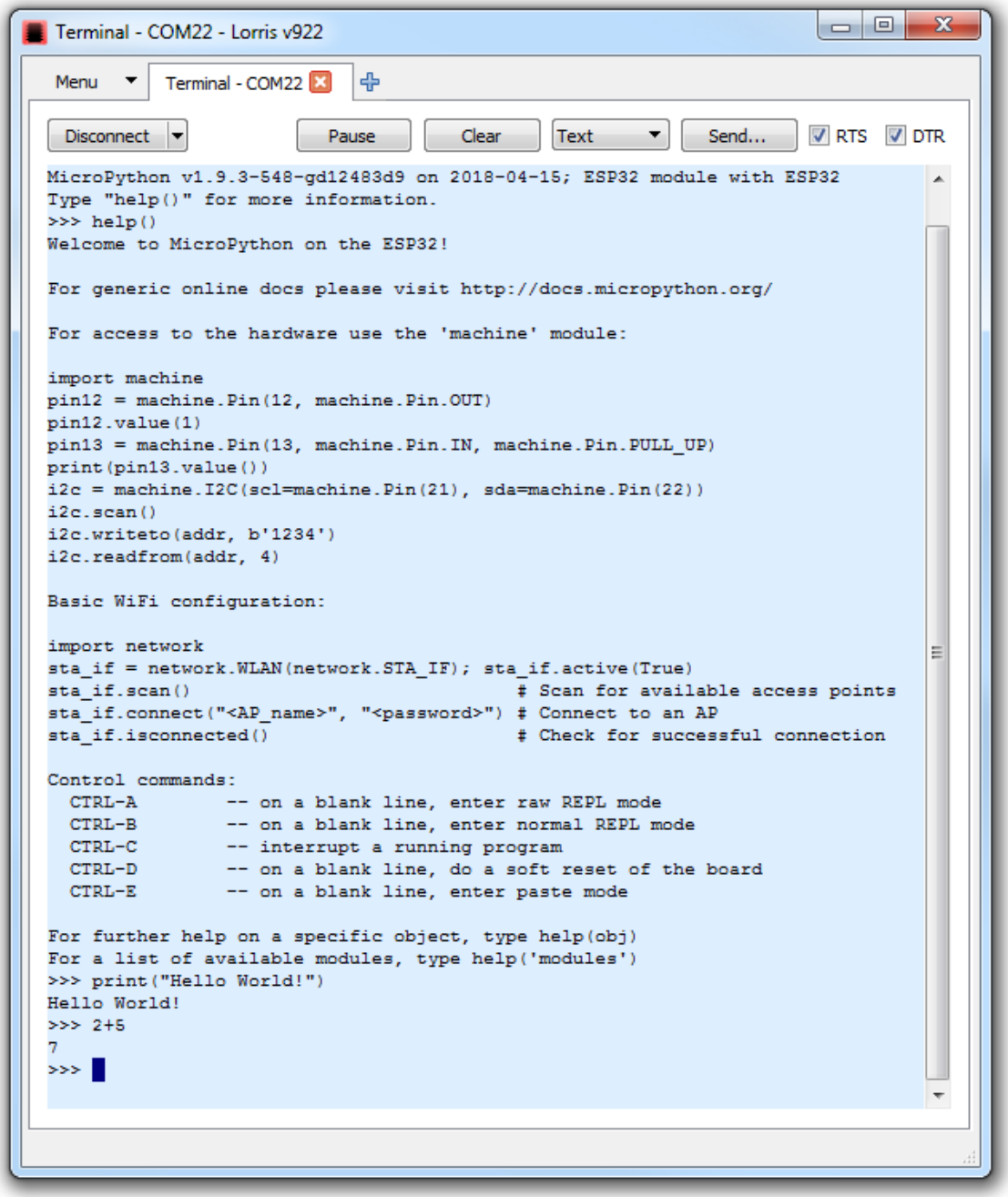


Figure 3.6: The separate BMF055 board on the left and the same board mounted to the Sensor-Board on the right

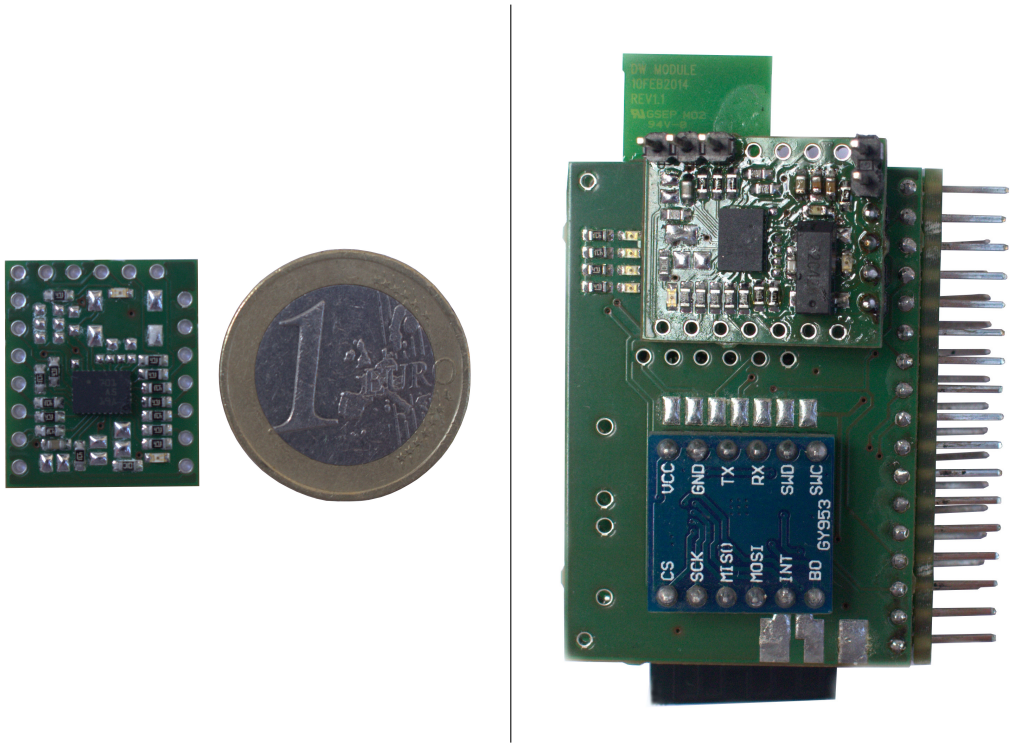
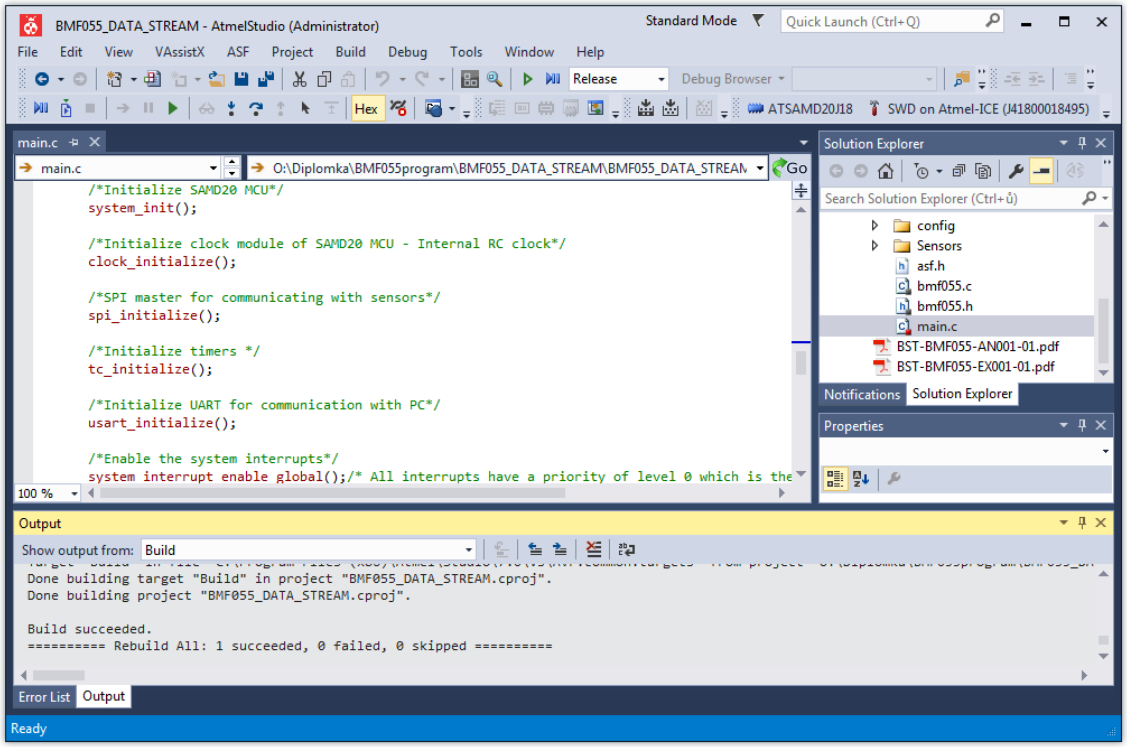



Figure 3.7: An opened BMF055 project in Atmel Studio 7



For easier working with the sensors on the SensorBoard I have implemented a new library. This library is described in section 3.2.1.

### 3.2.1 SensorBoard API

The ESP32 and Atmel SAM controller have defined API by the manufacturer, but these APIs do not implement a communication with specific sensors and peripherals connected to the controllers. For easier work with all the functionality of the SensorBoard, I have implemented specific API for the ESP32 controller.

The API is built on ESP-IDF and it should be used together with the ESP-IDF API from the manufacturer. For example, it allows usage of a FreeRTOS, because the FreeRTOS is a part of ESP-IDF framework [28]. The SensorBoard API can be used together with Arduino API for ESP32, too.

**The SensorBoard API implements**

- Peripherials:
  - **MPU9250** Accelerometer, dynamic gyroscope and magnetometer
  - **BMP280** Barometer (Altitude meter)
  - **Programmable LEDs** Two green LEDs
  - **PWM Servo output** Controlling connected servos or regulators
  - **File System** SD card and SPI file system
  - **ADP5062** Battery charger and power supply configuration
  - **Buttons** Read button state as input device
  - **WiFi** Access Point, FTP server, remote control and communication protocol
- Internal functionality:
  - **Global Settings** Configuration file stored on the SD card
  - **Stopwatch** Checking the timing of tasks
  - **Test Sensor** Virtual sensor used for testing other functionalities
- Partially implemented and actually unused
  - **BMI160** Acclerometer and dynamic gyroscope
  - **LTR-329ALS** Light sensor
  - **SI7006** Humidity and teperature sensor

The API is documented in the source code and in the separate generated file.

### 3.2.2 General API for the controllers

Both controllers used on the SensorBoard have an API defined and implemented by the manufacturer.

**Atmel SAM D21:** [57]

- AC – Analog Comparator (Callback APIs)
- ADC – Analog-to-Digital Converter (Polled APIs)
- T30TSE75X Temperature Sensor
- AT45DBX DataFlash
- AVR2025 – IEEE 802.15.4 MAC Stack v3.1.1
- AVR2025 – TAL
- AVR2025 – TFA
- AVR2025-MAC Serial Interface Module
- AVR2130 – LW MESH v1.2.1
- BOD – Brown Out Detector
- CRC-32 calculation
- CRC32 – 32-bit cyclic redundancy check
- DAC – Digital-to-Analog Converter (Callback APIs)
- Debug Print (FreeRTOS)
- Delay routines
- EEPROM Emulator Service
- Ethernet Physical Transceiver (ksz8851snl)
- EVSYS – Event System with interupt hooks support
- EXTINT – External Interrupt (Polled APIs)
- FatFS file system
- Generic board support
- GFX Monochrome – Menu System
- GFX Monochrome – Monochrome Graphic Library
- GFX Monochrome – Spinner/Spin control widget
- GFX Monochrome – System Font
- Interrupt management – SAM implementation
- IOPORT – General purpose I/O service
- Memory Control Access Interface
- NVM – Non-Volatile Memory
- PAC – Peripheral Access Controller
- Performance Analyzer Application
- PORT – GPIO Pin Control
- QTouch Library for SAMD20/D21
- RTC – Real Time Counter in Calendar Mode (Callback APIs)
- RTC – Real Time Counter in Count Mode (Callback APIs)
- SAM D20/D21 implementation of AT25DFx SerialFlash with vectored master SPI
- SD/MMC stack on SPI interface
- SERCOM I2C – Slave Mode I2C (Polled APIs)
- SERCOM SPI – Serial Peripheral Interface (Callback APIs)
- SERCOM SPI – Serial Peripheral Interface (Master Mode, Vectored I/O)
- SERCOM USART – Serial Communications (Polled APIs)
- Serial I/O – Host using UART
- Serial I/O – NCP Using UART
- Sleep manager – SAMD implementation
- Smart Card
- SSD1306 OLED controller
- Standard serial I/O (stdio)
- SYSTEM – Clock Management for SAMD20
- SYSTEM – I/O Pin Multiplexer
- TC – Timer Counter (Callback APIs)
- Unit test framework – SAM0 implementation
- USART – Serial interface – SAM implementation for devices with only USART
- WDT – Watchdog Timer (Polled APIs)

**Espressif ESP-WROOM-32:** [58]

- Wi-Fi
  - Wi-Fi
  - Smart Config
  - ESPNOW
- Mesh
  - ESP Mesh
- Bluetooth
  - Bluetooth Controller && VHCI
  - Bluetooth Common
  - Bluetooth LE
  - Bluetooth Classic
- Ethernet
  - Ethernet
- Peripherals
  - ADC
  - DAC
  - GPIO (including RTC low power I/O)
  - I2C
  - I2S
  - LED Control
  - MCPWM
  - Pulse Counter
  - Remote Control
  - SDMMC Host
  - SD SPI Host
  - Sigma-delta Modulation
  - SPI Master
  - SPI Slave
  - Timer
  - Touch Sensor
  - UART
- Protocols
  - mDNS
  - ESP-TLS
- Storage
  - SPI Flash and Partition APIs
  - SD/SDIO/MMC Driver
  - Non-Volatile Storage
  - Virtual Filesystem
  - FAT Filesystem
  - Wear Levelling
  - SPIFFS Filesystem
- System
  - FreeRTOS
  - FreeRTOS Hooks
  - Heap Memory Allocation
  - Heap Memory Debugging
  - Interrupt Allocation
  - Watchdogs
  - Inter-Processor Call
  - High Resolution Timer
  - Logging
  - Application Level Tracing
  - Power Management
  - Sleep Modes
  - Base MAC address
  - Over The Air Updates (OTA)
  - ESP pthread
- Configuration Options
  - Kconfig

## 3.3 Usage Examples

The SensorBoard is a multifunctional user programmable hardware that can be used in several ways. There are some examples presented in this section.

Figure 3.8: Simple logging application with SensorBoard

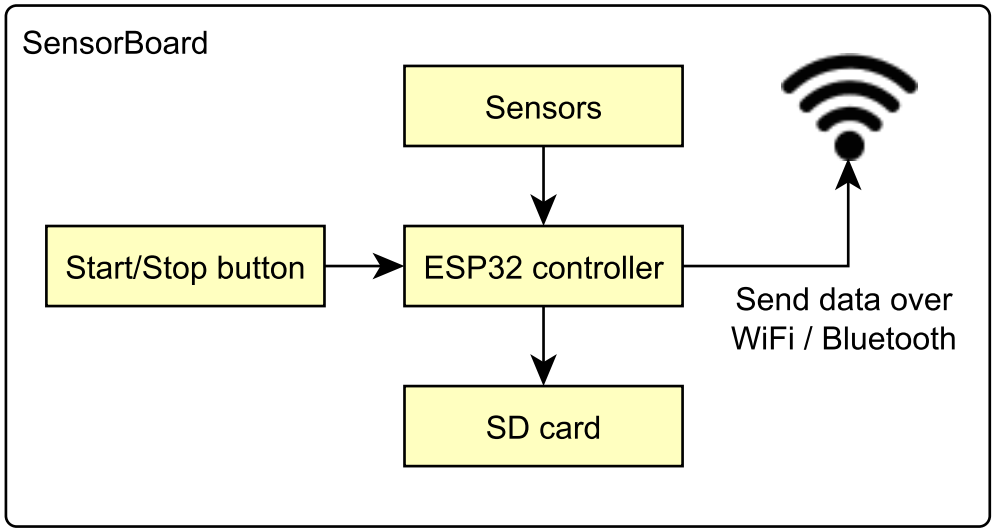
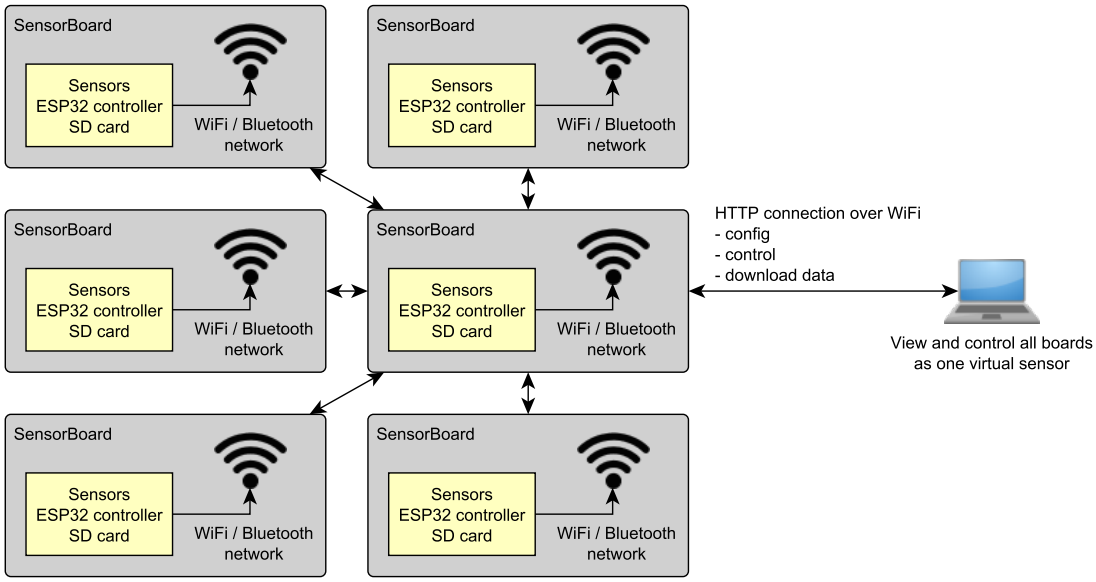



Figure 3.9: Advanced logging application with SensorBoard



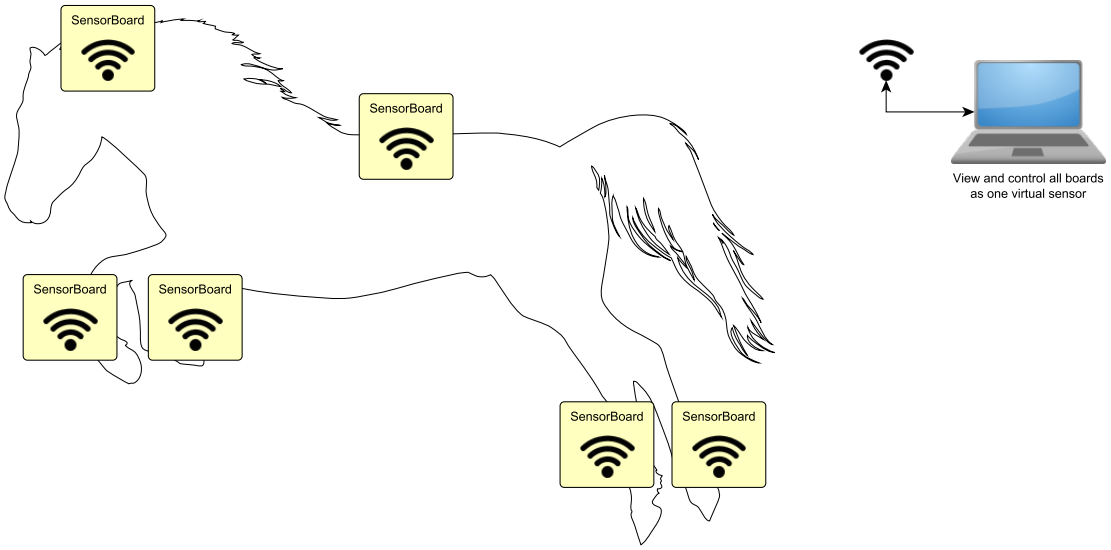

### 3.3.1 Sensors data logger

Sensor logger can be understood as very simple logging device or as a very advanced application with an integrated web server and with the ability to connect more boards into one synchronized grid with centralized control. The simple version is shown in figure 3.8.

When we need to measure some inertial data, we sometimes need more sensors placed in different places of the analyzed object. Tt is a big advantage when we can start and stop all the sensors synchronously and control all of them by one button. This solution is shown in figure 3.9.

I am using this advanced configuration during the analysis of a movement of a horse. This solution is explained in detail in section 3.4. The goal of this task is how to recognize the type

Figure 3.10: Configuration of the SensorBoard on a horse




of the movement of a horse (stand, walk, trot, canter or gallop). We can get some results from only one sensor, but we get more accurate results when we can measure each leg and the other places on the horse's body separately. How the SensorBoards are used in this task is shown in figure 3.10.

### 3.3.2 Sensor fusion and AHRS

Sensor fusion is a method that combines data from more sensors and calculates a new information. The new information is read from the sensor fusion algorithm like from a virtual sensor. [59]

This example shows how to use a sensor fusion algorithm for computing AHRS data on the SensorBoard. AHRS is Attitude and Heading Reference System and computes pitch, roll and yaw values from the triaxial accelerometer, dynamic gyroscope and magnetometer data. We can use for example Madgwick's algorithm [60] in the way shown in figure 3.11. This application is the base step for example 3.3.3.

We can use an advantage of two controllers at one board and run the sensor fusion algorithm on the BMF055 chip and the ESP32 is still free for some user program like is shown in figure 3.12. When an error occurs in the user program, the sensor fusion at BMF055 chip continues to run without any problem. This feature is a big advantage for Flight Controllers where we want to allow the user to run for example his navigation code.

### 3.3.3 UAV Flight controller with non-critical user programmable processor

The s must be very reliable devices. They usually consist of two parts. The first part controls the pitch, roll, yaw, velocity, the angle of attack etc. The second part is a navigator that processes user commands and makes high-level flight planning. The pilot's input can be processed on both parts. We usually want to create only the basic control algorithm, which is safety critical, and develop all the remaining parts as a non-critical code.

Here we can use dual controller advantage on the SensorBoard and run the basic critical code at the first controller and the remaining non-critical code on the second controller. The non-critical program on the second controller is usually a navigator or user program. The

Figure 3.11: Sensor fusion example on custom AHRS

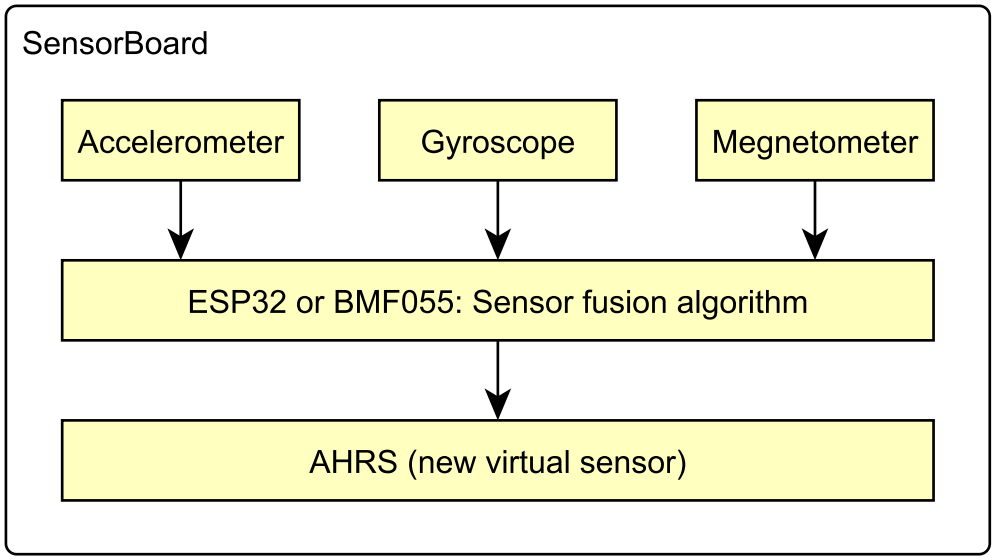



Figure 3.12: Sensor fusion AHRS example with two controllers

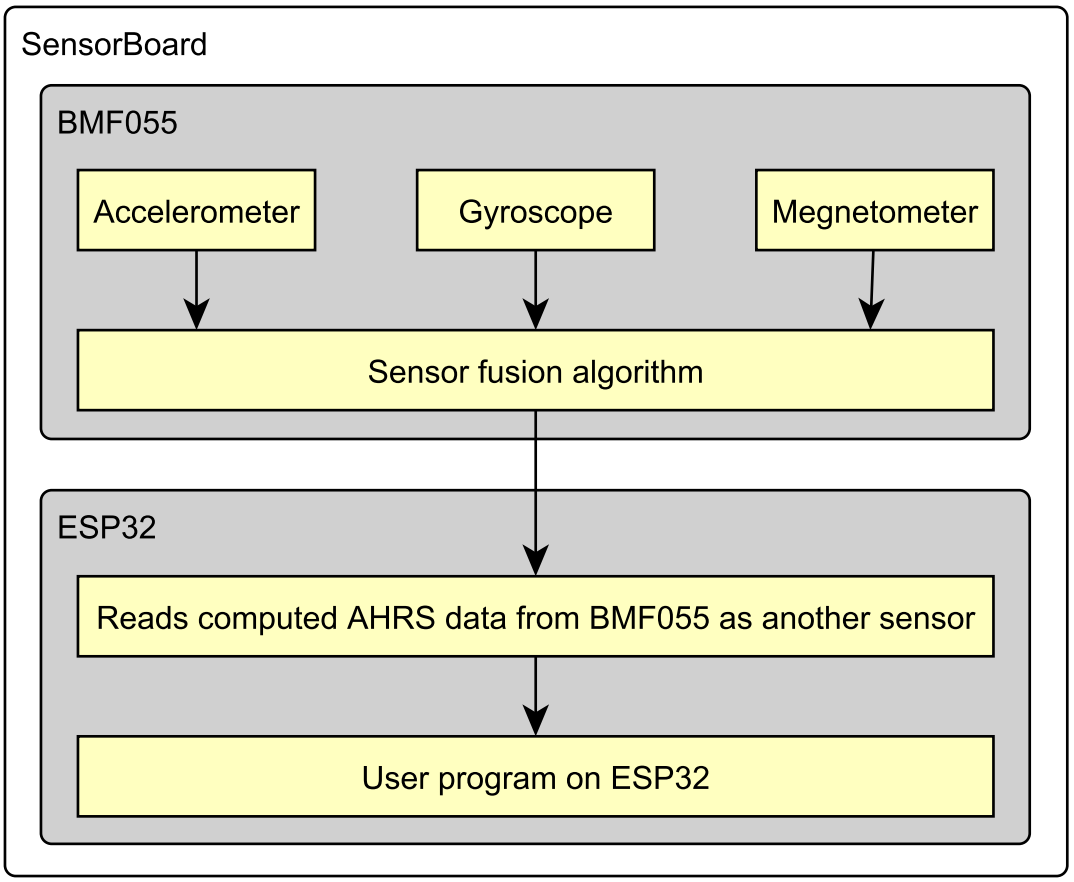


Figure 3.13: SensorBoard used as a quadcopter flight controller

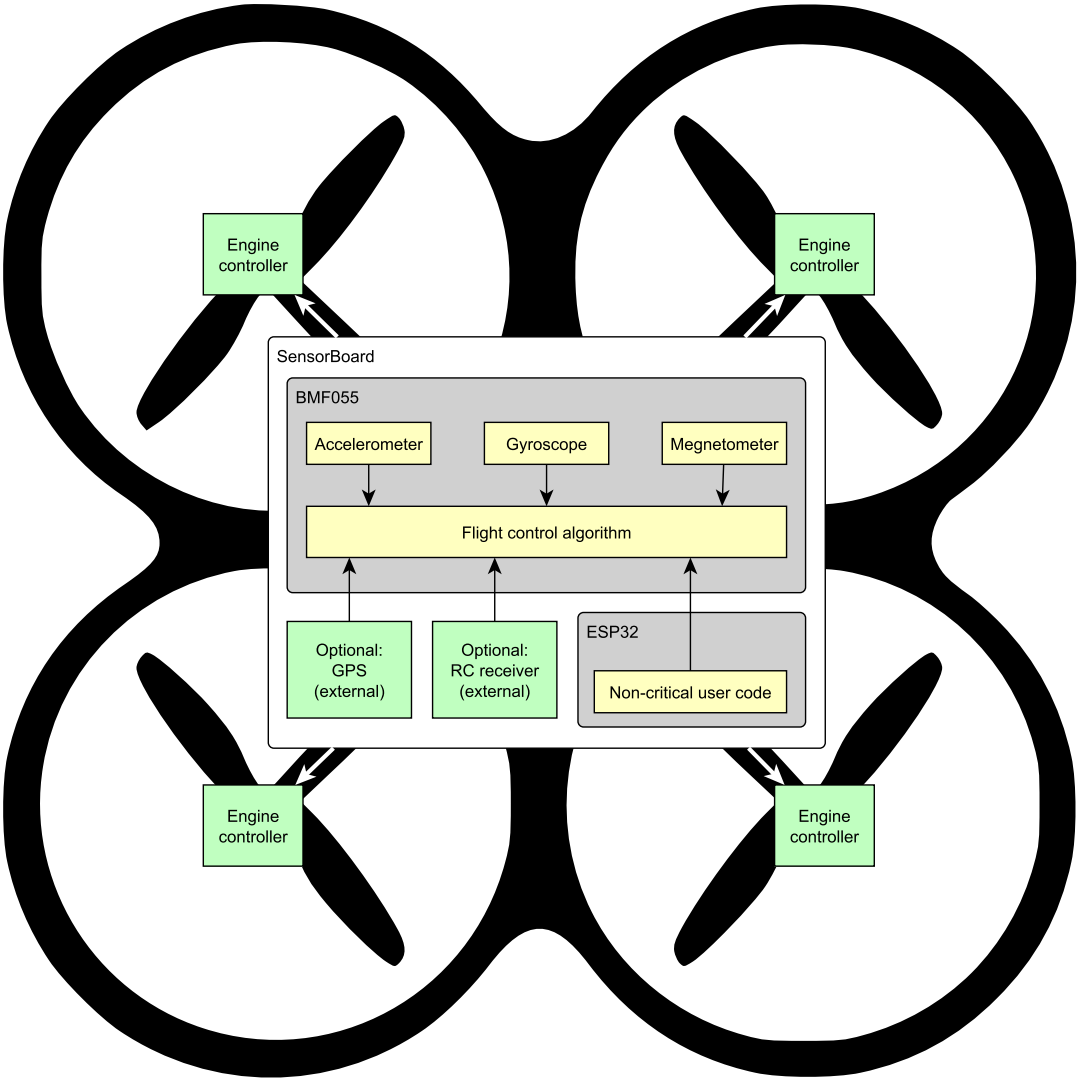


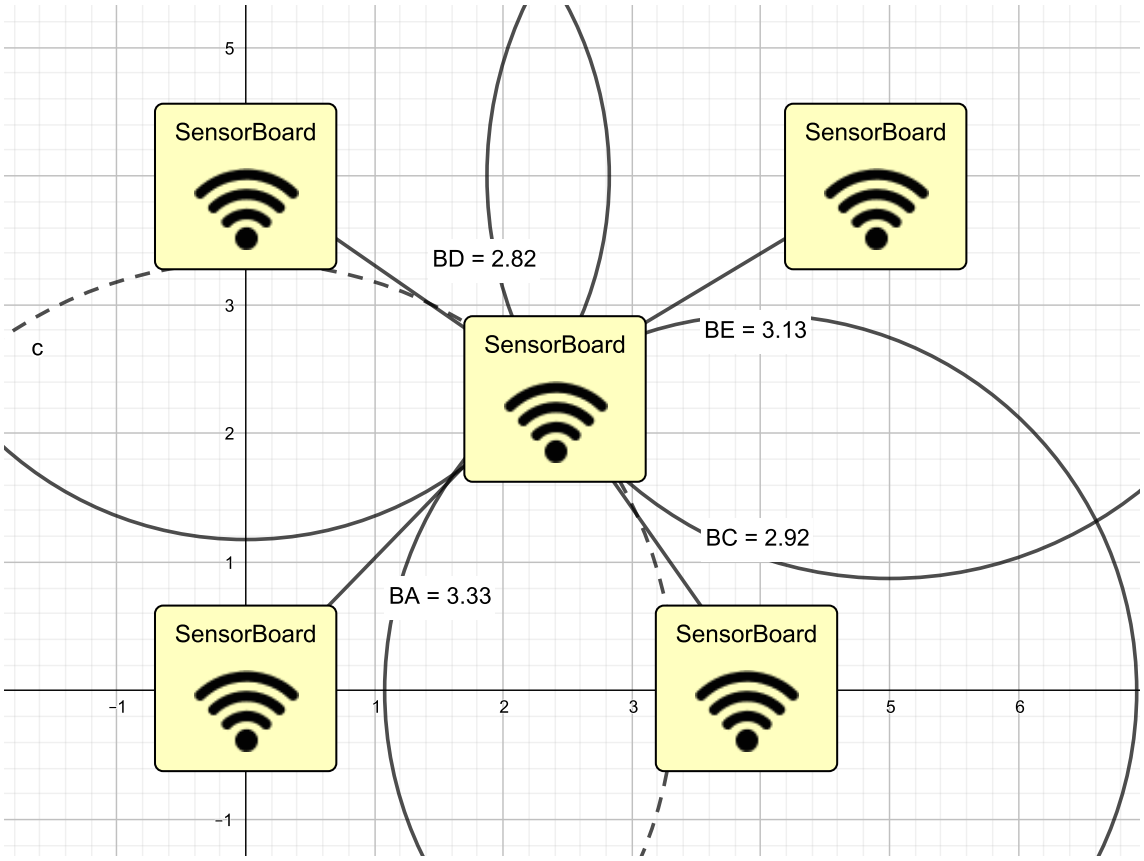

figure 3.13 shows the possible configuration of the flight controller on the SensorBoard. This configuration uses BMF055 Flight Controller [54] by Lukas Blocher. His code can be compiled directly for the SensorBoard without any modification.

### 3.3.4 Indoor navigation using TDOA

Indoor navigation requires high accuracy positioning. We usually need a higher accuracy than given by GPS. I know many projects approaching the indoor positioning. For example, there was an Indoor Positioning and Indoor Navigation conference in September 2017 in Japan.

This example uses Time Difference of Arrival (TDOA) method to compute its position in space with the precision of 10 cm. The device measures time differences in received messages from other devices. When the device knows the time differences and the speed of light, it can compute its relative position to the other boards. There is a DWM1000 chip that is able to measure and compute these data. [12]. The figure 3.14 shows how the relative positioning works on the SensorBoard.

Figure 3.14: TDOA relative location service on the SensorBoard




The DWM1000 [12] module is present on the SensorBoard and when it connects to at least three other boards, it computes its relative position with the mentioned precision. When the board knows an absolute position of the other boards, it can compute its absolute position, too. The absolute location can be obtained for example from GPS receiver or from static transmitters.

When we use the AHRS mentioned in section 3.3.2, we have all data about current position and attitude of the SensorBoard. We can use some other sensors on the SensorBoard like BMP280 barometer [14] to improve the current position and attitude data.

### 3.3.5 RTOS education board

Real-time systems are different from common operating systems in personal computers and they cannot be emulated as a standard application for a computer with the common operating system. The common operating system does not give a guarantee that a job completes on time, so any application running on this operating system cannot give this guarantee, too.

The ESP32 controller on the SensorBoard uses FreeRTOS and supports all its functionality. [61] We can run a hard real-time system on the SensorBoard or on the other hand, we can use the SensorBoard for education about real-time systems. Simple FreeRTOS application uses only the ESP32 controller like is shown in figure 3.15 and optionally we can use the sensors as data sources and as resources.

### 3.3.6 MicroPython Robot Controller

The SensorBoard supports MicroPython firmware [47] on the ESP32 controller. When we create a Python library that implements simple API for controlling some electromechanical device. We

Figure 3.15: FreeRTOS application diagram on ESP32 on the SensorBoard

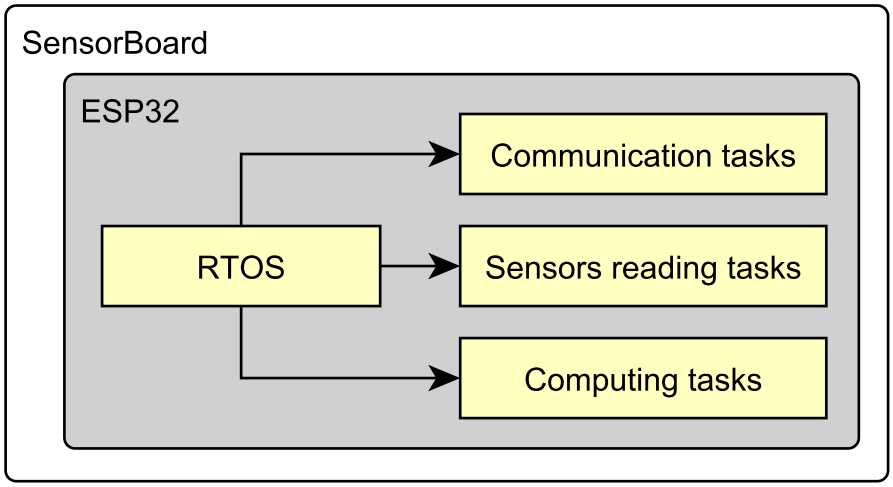
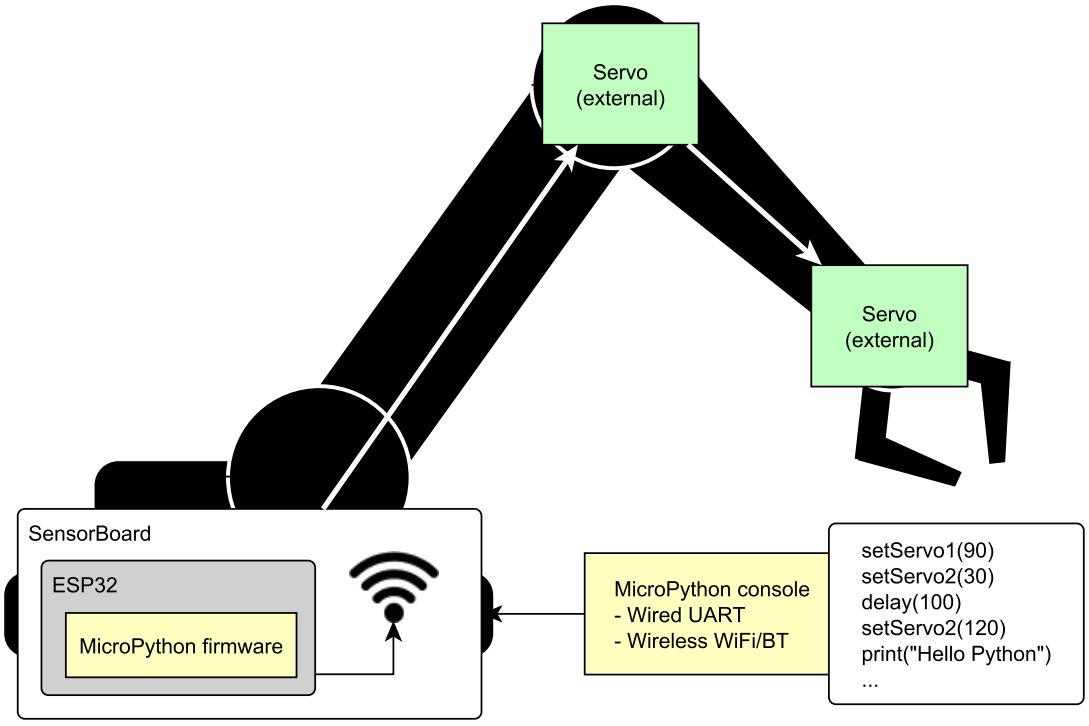


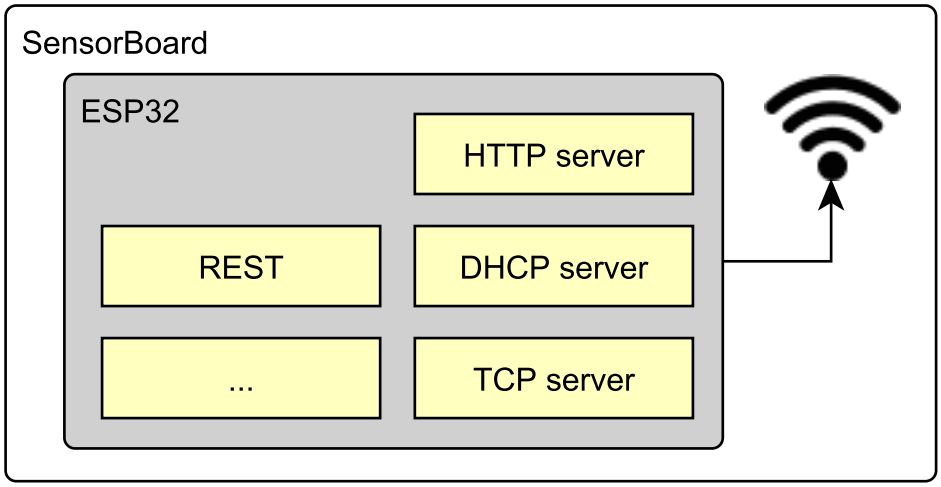

Figure 3.16: MicroPython robot controller example on the SensorBoard




can use the SensorBoard as a controller of this electromechanical hardware. Then we simply send Python commands via UART or WiFi to the target device.

In my opinion, this setup is an easy way how to teach students the basics of programming (in Python) with some dynamic hardware. This way of teaching is probably easier for the students and it probably creates more interactivity between students and the device. The figure 3.16 shows an example of using MicroPython inside the SensorBoard.

Figure 3.17: WiFi based services example on the SensorBoard




### 3.3.7 WiFi AccessPoint with server services

The ESP32 controller has a built-in Bluetooth and WiFi module. So, we can create any WiFi based application on the SensorBoard. The only limitation is memory (SD card up to 32 GB) and CPU power (dual-core 240 MHz). We can use the SensorBoard for example as a simple HTTP server [62] or FTP server [63] or implement any other WiFi-based service like is shown in figure 3.17.

### 3.3.8 Grid computing education board

The biggest advantages of the SensorBoard are hidden in its real-time system, wireless technology and independence. These advantages can create a grid of multiple items. We can use the whole grid as one virtual unit. When we solve the problems with failures and accessibility, we have a grid with swappable items. An example of a grid with two failures is shown in figure 3.18.

### 3.3.9 IoT wireless sensors

The Internet of Things (IoT) is a commercially used word for all devices connected to the Internet with some sensors onboard. The SensorBoard fulfills both conditions and it can be used directly this way. The figure 3.19 shows an example of using the SensorBoard as a device reading sensors data and streaming them via GSM network or via WiFi connection.

## 3.4 Movement Analysis Firmware

The firmware is a sensor data logging application similar to the example in section 3.3.1. The program is able to read values from sensors in a defined frequency and store the captured data to SD card and/or send them directly via WiFi. The application is controlled by start/stop button or via TCP connection.

When the SensorBoard is switched on, it initializes WiFi access point and waits 5 seconds before the reading and logging procedure starts. After these 5 seconds, the board enters a loop and starts logging data to SD card. The logging loop can be stopped via TCP connection, by pressing the start/stop button or by switching the board off.

Figure 3.18: Example of a grid with two failures created from multiple SensorBoards

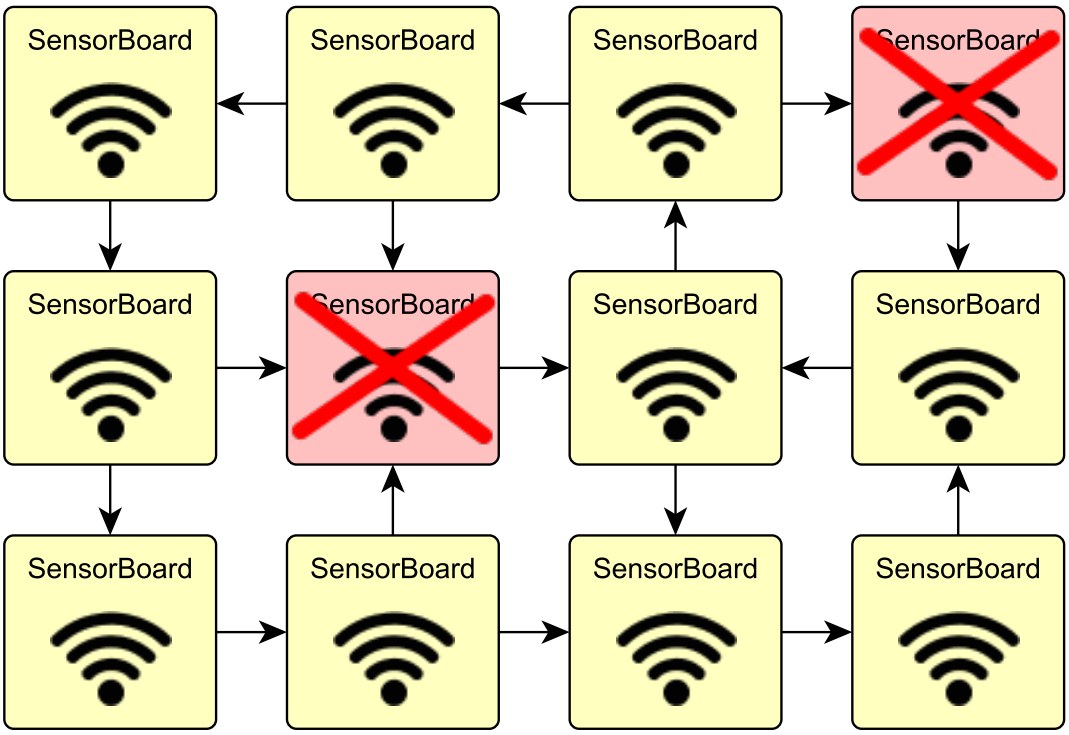
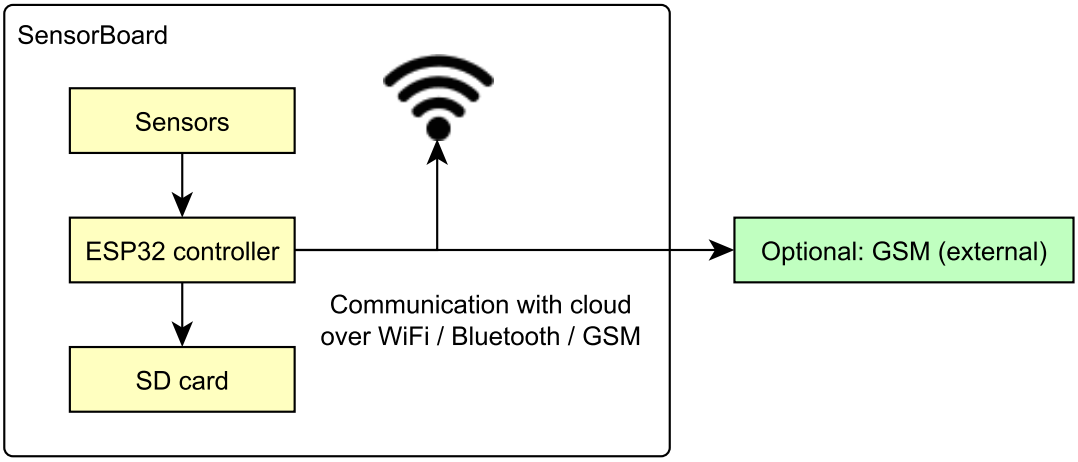



Figure 3.19: Example of using SensorBoard as an IoT device streaming data from sensors via WiFi or GSM network



When we want to control the SensorBoard via TCP client, we have to connect our device to SensorBoard's WiFi access point. Then we can connect to the TCP server and send the control commands.

### 3.4.1 Before first use

All the configuration is saved in `config.ini` file on the SD card (root directory). The configuration is read whenever the SensorBoard is powered on. The configuration file is never modified by the SensorBoard. An example of the configuration file is in figure 3.20. If no configuration file is present on the SD card, default values are used.

### 3.4.2 Files on the SD card

All the files are stored and read from the root directory of the SD card. The files are:

- **config.ini**: (read-only) Configuration file explained in section 3.4.1.
- **counter.txt**: (read/write) This text file contains only one integer on the first line, other data are ignored. The SensorBoard has no real-time clock reference, so it cannot add a timestamp to each log file. The board increments this number after each startup. This number is used as session number of each log file.
- **X_Y_log.csv**: (write only) The main log file in CSV format. The name consists of the session number `X`, log number `Y` and the configured filename. The format is `X_Y_NAME`, for example `12_34_log.csv`. It means that the file was created after 12th switching on, it is 34th log during the session and the configured file name is `log.csv`.
- **format.txt** (write only) Format of the CSV main log file. This file contains names of all columns in order in the main log file. Example of this file is shown in figure 3.21.
- **X_Y_horse.txt**: (write only) The movement analysis log file in plain text or CSV format. The `X`, `Y` numbers correspond to the main log file numbers. The files with the same numbers were captured simultaneously during the same measurement.

### 3.4.3 Using the firmware

The application is usually used only during the measurement process. The behavior of the program can be described using the state machine presented in figure 3.22.

The meaning of each state of the machine:

- **Startup** Powering on the board
- **WiFi initialization** Initialization of the WiFi adapter according to the configuration file
- **Waiting N seconds** Waiting a few (milli)seconds according to the configuration file. We usually configure this waiting when we want to prevent the creation of very short logs.
- **System idle** The system is waiting for control commands. If the board is configured to start logging immediately, the system automatically switches to running state.
- **Downloading data** The data can be downloaded directly from the SD card or to a connected device.

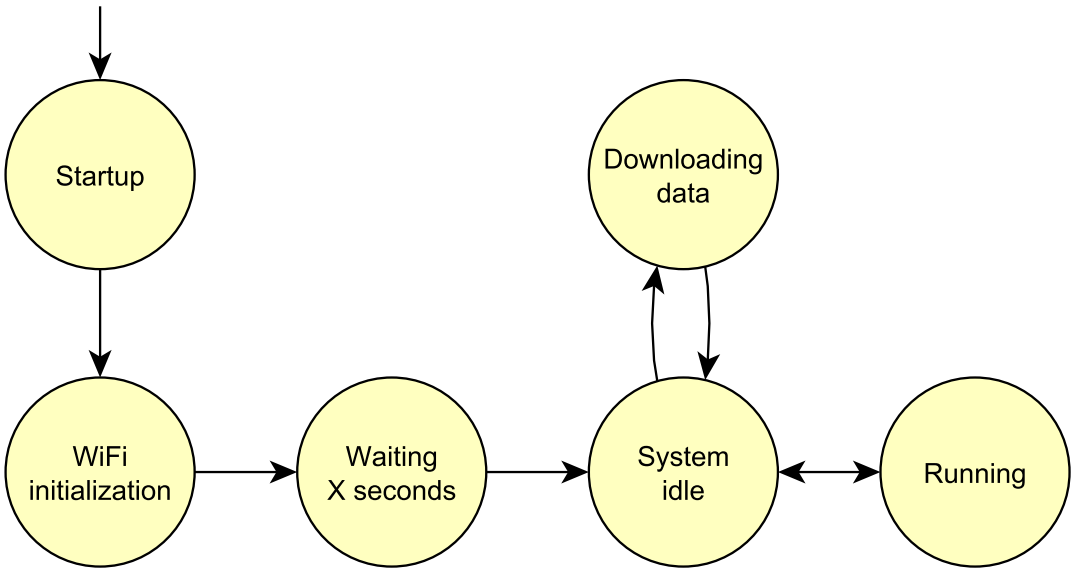
Figure 3.20: Example of the configuration file on SD card

```
% Enable or disable WiFi adapter
isWiFienabled = true

% Initializes WiFi as an access point when true,
% otherwise as a station
isWiFiAccessPoint = true

% Network SSID
% When initialized as AP: creates the new network
WiFiSSID = SensorBoard

% WiFi network password
WiFiPassword = 1234567890

% IP address of the device,
myIP = 192.168.1.1

% TCP server port
TCPseverPort = 3000

% If enabled, starts logging automatically 'startUpWaitTime' seconds
% after powered on
startLoggingImmediately = true

% Waits N milliseconds before entering the logging loop
startUpWaitTime = 5000

% If enabled, the horse movement analysis starts automatically
isHorseAnalysis = false

% If enabled, the data streaming starts automatically
isStreaming = false

% If enabled, the logging can be controlled by the start/stop button
isStartButtonUsed = false

% Log file name for sensor data,
% a number prefix is automatically added
logFileName = log.csv

% Log file name for movement analysis,
% a number prefix is automatically added
horseAnalysisFileName = horse.txt
```

Figure 3.21: Example of the 'format.txt' file on SD card.

```
'S'; #; milliseconds; baroAltitude QFE [m]; baroTemp [degree C];  \
ACC_X [g]; ACC_Y [g]; ACC_Z [g];                                  \
GYR_X [rad/s]; GYR_Y [rad/s]; GYR_Z [rad/s];                      \
MAG_X [degree];  MAG_Y [degree];  MAG_Z [degree];                 \
...
```

Figure 3.22: State machine of the firmware




- **Running** The Running state starts the measuring loop and allows to log, stream or analyze the measured data. It is not possible to download data from SD card in this state. This state is activated when the user starts logging, streaming or movement analysis process from the Idle state. The Running state can be activated automatically according to the configuration file.

### 3.4.4 Horse movement analysis feature

The application has built-in movement analysis tool. When the SensorBoard is placed on a horse's body, the tool can determine when the horse is moving and what kind of movement it does. The figure 3.23 shows a screenshot from the tablet used for controlling the SensorBoard during measurement process on a horse's body.

This tool demonstrates that the SensorBoard is able to run some user code in real-time with real-time results.

### 3.4.5 Controlling via TCP

The application can be controlled via TCP connection using single letter commands. I recommend implementing a web-page based user interface in the future versions of the application. One letter commands are used because it is the easiest way how to control the board from a

Figure 3.23: Screenshot from the tablet used for controlling the SensorBoard during measurement process


tablet with on-screen keyboard only. I would recommend using the more advanced protocol in the future versions. All the communication is possible via TCP connection or UART through USB connection. The control commands are:

**r** Reset the board and restart the application. We get the same result when we switch the board off and on.

**p** Ping command. When received, the text "Ping received." is printed.

**h** Start/stop the horse movement analysis. When the analysis is running the text output is sent every second.

**s** Start/stop streaming the raw data from the sensors.

**l** Start/stop logging the raw data from the sensors to the SD card. When the logging is in progress, the green software LED is ON on the SensorBoard.

### 3.4.6 Using multiple SensorBoards

When we have more SensorBoards, we can connect them like in figure 3.9. Then all the hardware connected together behaves like one SensorBoard with so many sensors in different places. All the SensorBoards should connect to one WiFi network created by one board or by an external WiFi access point (e.g., mobile phone hotspot). All the SensorBoards except one (configured on SD card) behaves as TCP clients and connects to TCP server (the last SensorBoard). The TCP server resends all the commands to the other boards, so all the SensorBoards behaves synchronously. The log files are stored on the SD card inside each device and the names of these files are the same on all connected boards. So, the TCP server dictates the names of all log files. One synchronous session creates multiple files on multiple SD cards, but with the same filenames on all devices. The whole system of connected boards can be used like in figure 3.10.

CHAPTER 4

# ANALYSIS OF A HORSE MOVEMENT

The movement analysis is a widely studied problem in many fields. We can measure the movement inside a laboratory using a group of (high-speed) cameras or we can use some kind of wearable sensors. The wearable sensors do not require an installation of the cameras so we can achieve higher mobility. A possible motion tracking solution using wearable sensors is provided by Xsens company [4]. Other possible solutions are mentioned in chapter 2.3.

I have selected the movement analysis task for proving the functionality of the . It means that I will use the hardware with an existing or a new algorithm for analyzing the movement. I do not have the other hardware solutions available, so I do not do any measurements for comparison of the results.

The analysis algorithm will be designed to determine the kind of a movement of a horse – stand, walk, trot or gallop.

## 4.1 Definitions of types of the movement

Before we start any analysis, we have to define all the movement types taken into account. First, I will define a gait in general, and then each type of gait taken into account.

**Definition 4.1.1. Gait** *Gait is the pattern of movement of the limbs of animals, including humans, during locomotion over a solid substrate. Most animals use a variety of gaits, selecting gait based on speed, terrain, the need to maneuver, and energetic efficiency.* [64]

I will define each gait as a sequence of the following symbols. All the definitions I created in this section are based on external sources. [64] [65] [66]

**List of used symbols in the definitions:**

| | |
|---|---|
| $LF$ | Left front leg. |
| $LH$ | Left hind leg. |
| $RF$ | Right front leg. |
| $RH$ | Right hind leg. |
| $\wedge$ | Any leg touched the ground, but we do not know which one. The symbol replaces $\downarrow LF$ or $\downarrow LH$ or $\downarrow RF$ or $\downarrow RH$ symbol. |
| $\uparrow$ | The leg left the ground and cannot generate acceleration. |
| $\downarrow$ | The leg touches the ground and generates acceleration (generates a touch in the signal based on definition 4.4.1). |
| $\ldots$ | The whole movement is repeating forever from this point. |
| $X, Y$ | $X$ and $Y$ occurs at the same time or the time difference is lower than $\epsilon$. |
| $X \cdot Y$ | The time between $X$ and $Y$ is significantly higher than $2\epsilon$, but lower than $\delta$. |
| $X \sim Y$ | The time between $X$ and $Y$ is significantly higher than $2\delta$, but lower than $\gamma$. |
| $\gamma$ | It is the longest time when the horse is able to be in the air by his own force. ($\gamma \in \mathbb{R}^+$) |
| $\delta\epsilon$ | $\exists \delta, \epsilon : 0 \leq \epsilon < 2\epsilon < \delta < 2\delta < \gamma$ |

### 4.1.1 Stand

**Definition 4.1.2. Stand** A horse is standing when all its four hoofs are touching the ground. [66]

1. The complete definition using the symbols:

$$\downarrow RH, \downarrow LH, \downarrow RF, \downarrow LF, \ldots$$

2. With only the detection of touching the ground:

$$\downarrow RH, \downarrow LH, \downarrow RF, \downarrow LF, \ldots$$

3. Without knowledge about what leg touched the ground:

$$\wedge, \wedge, \wedge, \wedge, \ldots$$

4. Short form:

$$\wedge, \ldots$$

### 4.1.2 Walk

**Definition 4.1.3. Walk** *The walk is a four-beat gait that averages about 4 miles per hour (*6.4 km/h*). When walking, a horse's legs follow this sequence: left hind leg, left front leg, right hind leg, right front leg, in a regular 1-2-3-4 beat. At the walk, the horse will alternate between having three or two feet on the ground. A horse moves its head and neck in a slight up and down motion that helps maintain balance.* [66]

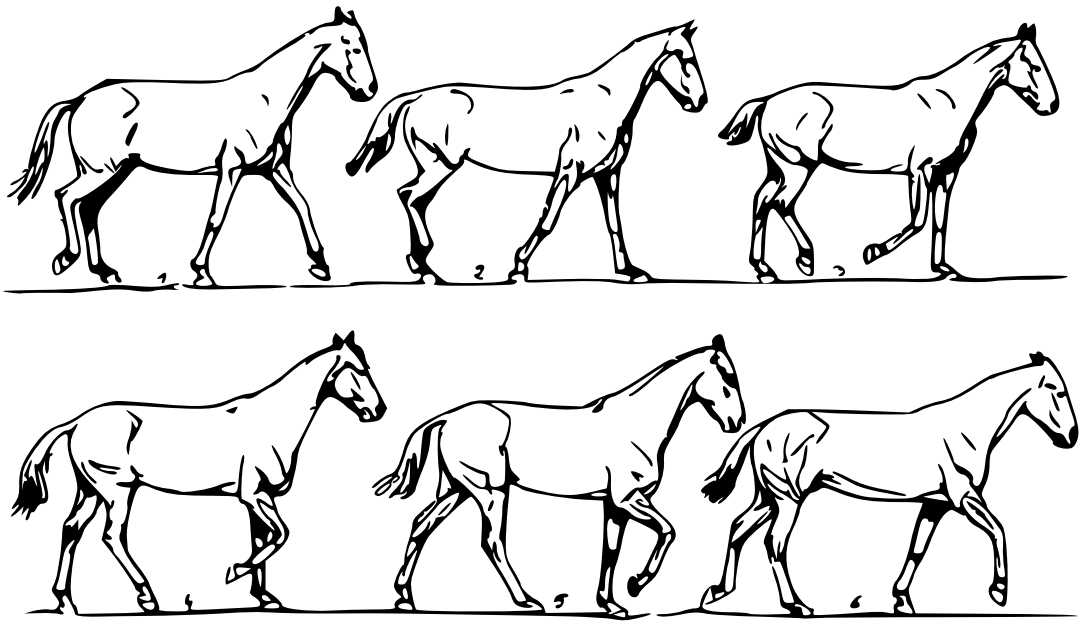
1. The complete definition using the symbols:

$$\uparrow RH \sim \downarrow LF \sim \uparrow RF \sim \downarrow RH \sim \uparrow LH \sim \downarrow RF \sim \uparrow LF \sim \downarrow LH \sim \ldots$$

2. With only the detection of touching the ground:

$$\downarrow LF \sim \downarrow RH \sim \downarrow RF \sim \downarrow LH \sim \ldots$$

Figure 4.1: A horse during walk. [65]



3. Without knowledge about what leg touched the ground:

$$\wedge \sim \wedge \sim \wedge \sim \wedge \sim \ldots$$

4. Short form:

$$\wedge \sim \ldots$$

*Remark.* An amble

$$\downarrow RH, \downarrow RF, \downarrow LH, \downarrow LF, \ldots$$

is not a walk according to the definition 4.1.3, but the parts (3) and (4) count an amble as a walk.

The figure 4.1 shows the horse during walk.

### 4.1.3 Trot

**Definition 4.1.4. Trot** *The trot is a two-beat gait that has a wide variation in possible speeds, but averages about 8 miles per hour (*13 km/h*). A very slow trot is sometimes referred to as a jog. An extremely fast trot has no special name, but in harness racing, the trot of a Standardbred is faster than the gallop of the average non-racehorse.* [66]

1. The complete definition using the symbols:

$$\uparrow RH, \uparrow LF, \downarrow LH, \downarrow RF \sim \downarrow RH, \downarrow LF, \uparrow LH, \uparrow RF \ldots$$

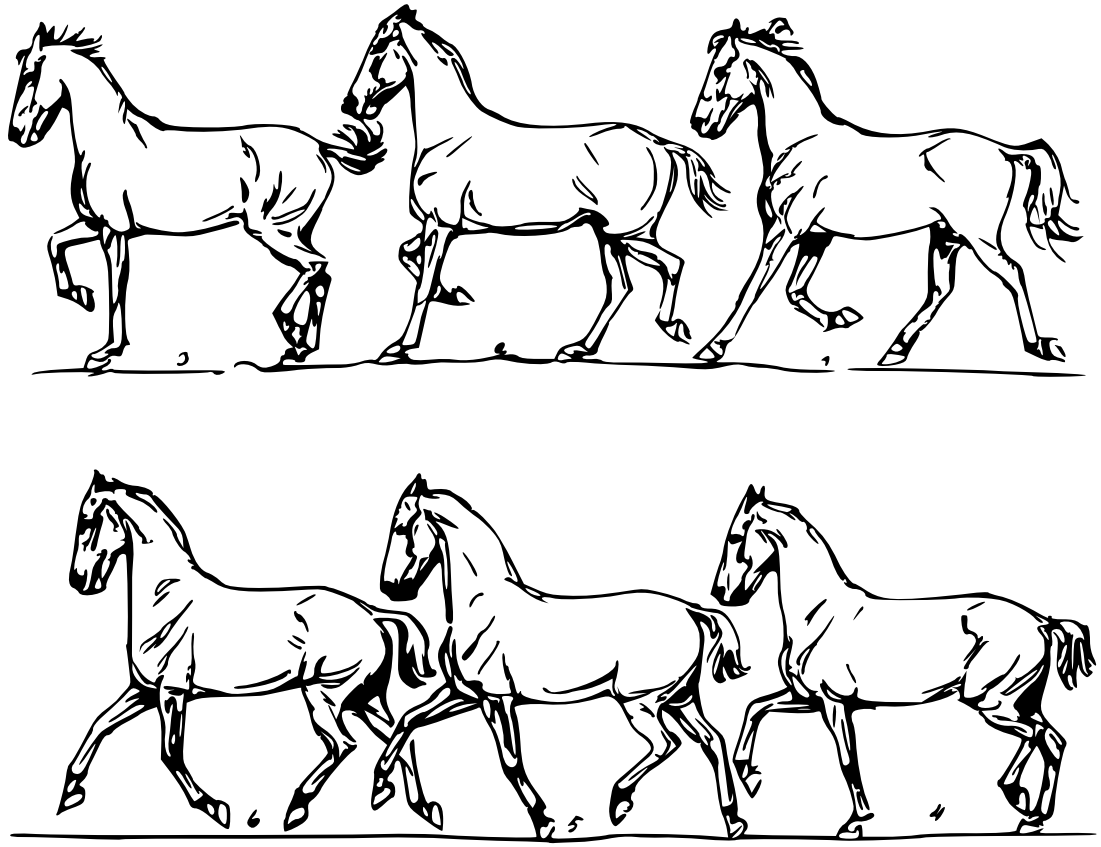
2. With only the detection of touching the ground:

$$\downarrow LH, \downarrow RF \sim \downarrow RH, \downarrow LF \ldots$$

3. Without knowledge about what leg touched the ground:

$$\wedge, \wedge \sim \wedge, \wedge \sim \ldots$$

Figure 4.2: A horse during trot. [65]



4. Short form:

$$\wedge, \wedge \sim \ldots$$

*Remark.* The pace

$$\uparrow RH, \uparrow RF, \downarrow LH, \downarrow LF \sim \downarrow RH, \downarrow RF, \uparrow LH, \uparrow LF \ldots$$

is not a trot according to this definition, but (3) and (4) count a pace as trot.

The figure 4.2 shows the horse during trot.

### 4.1.4 Canter

**Definition 4.1.5. Canter** *The canter is a controlled, three-beat gait that usually is a bit faster than the average trot, but slower than the gallop. The average speed of a canter is* 16 km/h *to* 27 km/h *(*10 mph *to* 17 mph*), depending on the length of the stride of the horse. Listening to a horse canter, one can usually hear the three beats as though a drum had been struck three times in succession. Then there is a rest, and immediately afterward the three-beat occurs again. The faster the horse is moving, the longer the suspension time between the three beats.*

*In the canter, one of the horse's rear legs – the right rear leg, for example – propels the horse forward. During this beat, the horse is supported only on that single leg while the remaining three legs are moving forward. On the next beat the horse catches itself on the left rear and right front legs while the other hind leg is still momentarily on the ground. On the third beat, the horse catches itself on the left front leg while the diagonal pair is momentarily still in contact with the ground.* [66]

1. Using the symbols (driven by right hand):

$$\downarrow RH \cdot \downarrow LH, \downarrow RF \cdot \uparrow RH \cdot \downarrow LF \cdot \uparrow LH \cdot \uparrow RF \cdot \uparrow LF \sim \ldots$$

2. With only the detection of touching the ground (driven by right hand):

$$\downarrow RH \cdot \downarrow LH, \downarrow RF \cdot \downarrow LF \sim \ldots$$

3. Without knowledge about what leg touched the ground (driven by right hand):

$$\wedge \cdot \wedge, \wedge \cdot \wedge \sim \ldots$$

4. Short form (driven by right hand):

$$\wedge \cdot \wedge, \wedge \cdot \wedge \sim \ldots$$

5. Using the symbols (driven by left hand):

$$\downarrow LH \cdot \downarrow RH, \downarrow LF \cdot \uparrow LH \cdot \downarrow RF \cdot \uparrow RH \cdot \uparrow LF \cdot \uparrow RF \sim \ldots$$

6. With only the detection of touching the ground (driven by left hand):

$$\downarrow LH \cdot \downarrow RH, \downarrow LF \cdot \downarrow RF \sim \ldots$$

7. Without knowledge about what leg touched the ground (driven by left hand):

$$\wedge \cdot \wedge, \wedge \cdot \wedge \sim \ldots$$

8. Short form (driven by left hand):

$$\wedge \cdot \wedge, \wedge \cdot \wedge \sim \ldots$$

The figure 4.3 shows the horse during canter.

### 4.1.5 Gallop

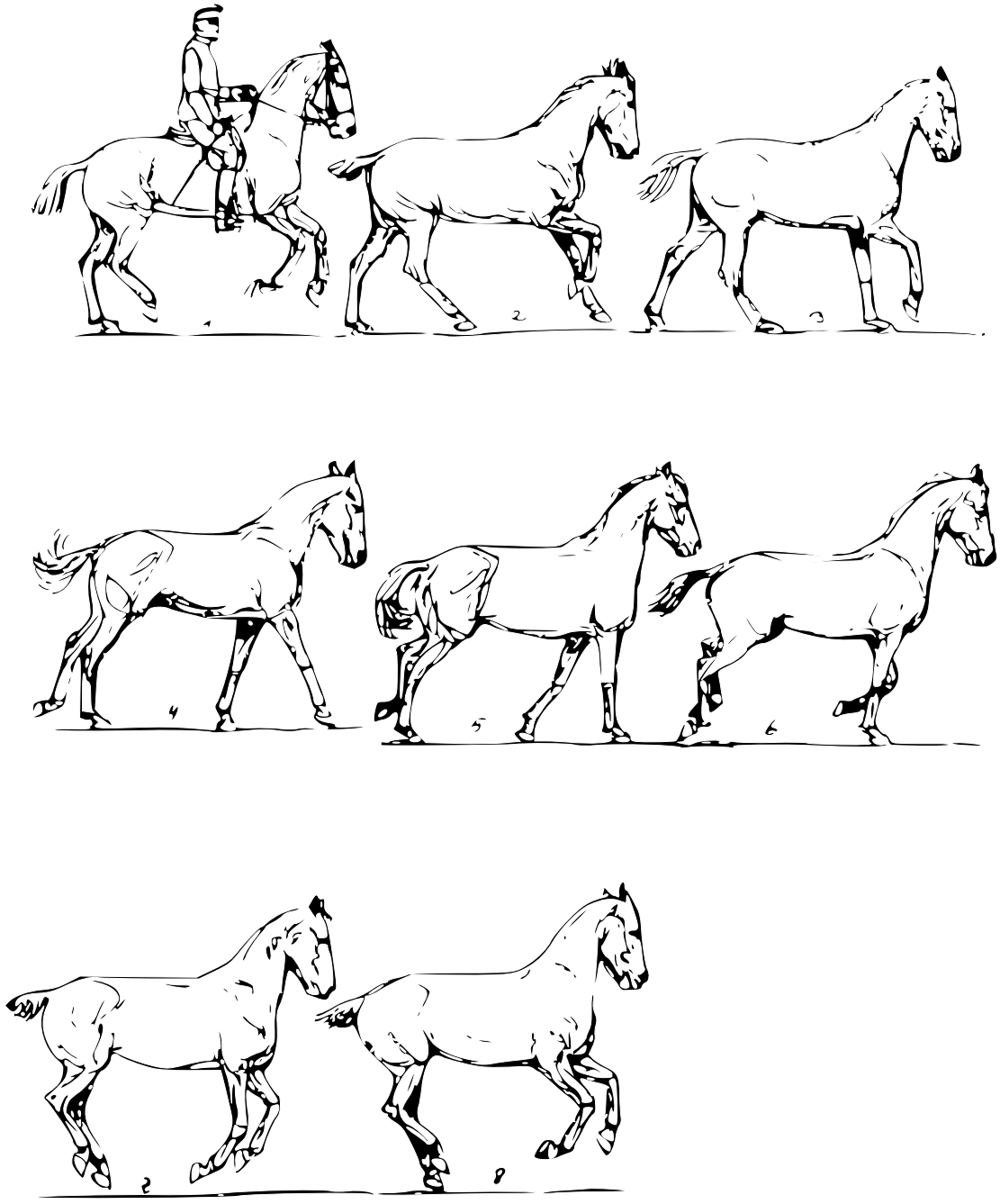
**Definition 4.1.6. Gallop** *The gallop is very much like the canter, except that it is faster, more ground-covering, and the three-beat canter changes to a four-beat gait. It is the fastest gait of the horse, averaging about 25 to 30 miles per hour (*40 km/h *to* 48 km/h*), and in the wild is used when the animal needs to flee from predators or simply cover short distances quickly. Horses seldom will gallop more than 1 or 2 miles (*1.6 km *or* 3.2 km*) before they need to rest, though horses can sustain a moderately paced gallop for longer distances before they become winded and have to slow down.*

*Like a canter, the horse will strike off with its non-leading hind foot; but the second stage of the canter becomes, in the gallop, the second and third stages because the inside hind foot hits the ground a split second before the outside front foot. Then both gaits end with the striking off of the leading leg, followed by a moment of suspension when all four feet are off the ground. A careful listener or observer can tell an extended canter from a gallop by the presence of the fourth beat.* [66]

1. The complete definition using the symbols:

$$\downarrow RH \cdot \downarrow LH \cdot \downarrow RF \cdot \uparrow RH \cdot \downarrow LF \cdot \uparrow LH \cdot \uparrow RF \cdot \uparrow LF \sim \ldots$$

Figure 4.3: A horse during canter. [65]



2. With only the detection of touching the ground:

$$\downarrow RH \cdot \downarrow LH \cdot \downarrow RF \cdot \downarrow LF \sim \ldots$$

3. Without knowledge about what leg touched the ground:

$$\wedge \cdot \wedge \cdot \wedge \cdot \wedge \sim \ldots$$

4. Short form:

$$\wedge \cdot \wedge \cdot \wedge \cdot \wedge \sim \ldots$$

*Remark.* The canter and gallop have a very similar definition. We can see that the definitions (2), (3) and (4) are the same for both gaits. So, when we do not measure the complete information, we are not able to distinguish these two types of movement.

The figure 4.4 shows the horse during gallop.

## 4.2 The input data

The movement analysis is based on the data from inertial sensors in this task. The input is a stream of values from the triaxial accelerometer, Micro Electro Mechanical Systems (MEMS) gyroscope and magnetometer at a constant frequency. If we want to run the algorithm on very cheap hardware, we would like to use only values from the accelerometer, gyroscope, or both. The hardware is able to measure some additional values like relative position to other sensors, sound etc. These values are important, but they are not used in the basic version of the algorithm.

All the data in this chapter were captured during March 2018. The SensorBoard was mounted on the thoroughbread race horse, 14 years old and without movement restrictions. The horse was trained for a recereational riding. The rider has the Basic rider training license obtained from the Czech Riding Federation.

- **Accelerometer:** Values for each axis at frequency 100 Hz with maximal range of ±16 G.
- **Gyroscope:** Values for each axis at frequency 100 Hz with maximal range of $2000°s^{-1}$.
- **Magnetometer:** Values for each axis at frequency 8 Hz.

*Remark.* When we convert the measured signals to sound, we get interesting results. We have to speed up the signal because we need to move the sound to hearable frequencies. When we listen to the signal, we can hear the differences and later we can learn to distinguish the kind of movement. The figure 4.5 visualize the conversion process mentioned in this remark.

Figure 4.5: Conversion of the accelerometer data to hearable sound.

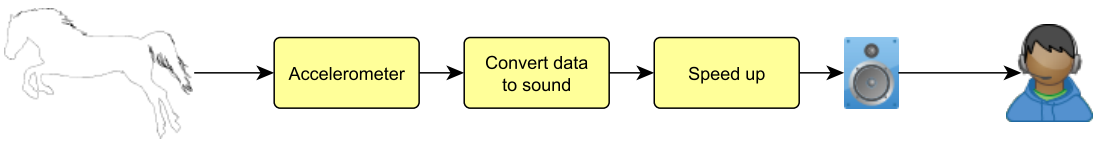
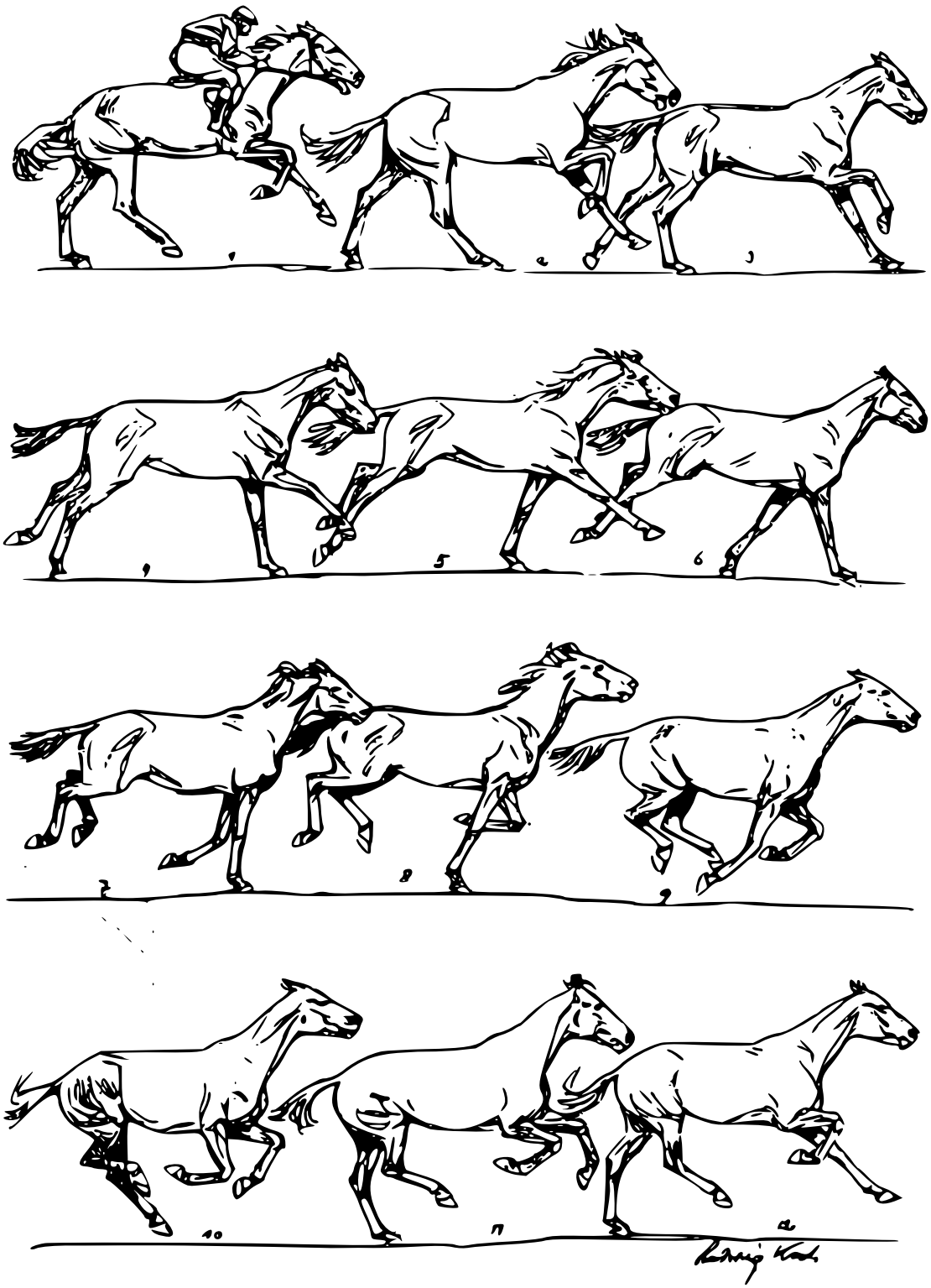


Figure 4.4: A horse during gallop. [65]



### 4.2.1 Positions of the sensors

The position of the sensors on the body is also very important. For example, when the sensor is mounted on the cannon, on the head or on the saddle, the measured data are very different. The figure 4.6 shows the differences between the measured data. The visualized values $s$ are computed as sizes of the measured acceleration vectors $\vec{v} = (x, y, z)$:

$$s = \sqrt{x^2 + y^2 + z^2}$$

When we have more sensors mounted at different places, we can determine the type of movement more accurately. There is a question if we can determine the type of movement 100 % accurately with using only the inertial sensors. This problem is analyzed later in this chapter.

### 4.2.2 The measured data as digital signal

A digital signal is a signal represented by a sequence of discrete values. [67] The input data fulfill the definition of the digital signal. It means that the algorithm used for the movement analysis is a Digital Signal Processing (DSP) algorithm.

The horse gait is a periodical movement. The period is changing with time, so we cannot specify a constant period for more than one cycle. For example, when the horse is going around, the inner legs do shorter steps than the outer legs. The graph of the measured data in figure 4.7 shows the different period of each step.

### 4.2.3 Input data requirements

The digital signals have to be captured in constant frequency. [68] The frequency must be at least two times higher than the highest frequency in the signal (Nyquist rate). [69] When the logging frequency is variable, the data gives wrong information about changing the speed of the horse and its steps.

The range of the sensor must be higher than the data range. When this condition is not fulfilled and the algorithm receives saturated values, the output can be negatively affected.

The requirements:

- Constant frequency
- Not exceeded Nyquist rate
- No saturated values, all measured data are in a given range

## 4.3 Tools for the analysis

Before choosing the algorithm for the movement determination, we will discuss some options in this section.

### 4.3.1 Step counter

A simple step counter is not a good solution for determination of the movement. For example, a slow canter has fewer steps per second than a fast trot. So, the number of steps per second is not an accurate pointer of the type of movement.

The amplitude of each step can help with the movement determination, but it is still not accurate. The canter and trot can have the same amplitudes of their steps.

For an accurate determination, we need to analyze the definition of each kind of movement in detail. The types of gaits are described in section 4.1. From the definition, the main difference

Figure 4.6: Different position of the inertial sensors gives different measurements. The graphs show size of the acceleration vector.

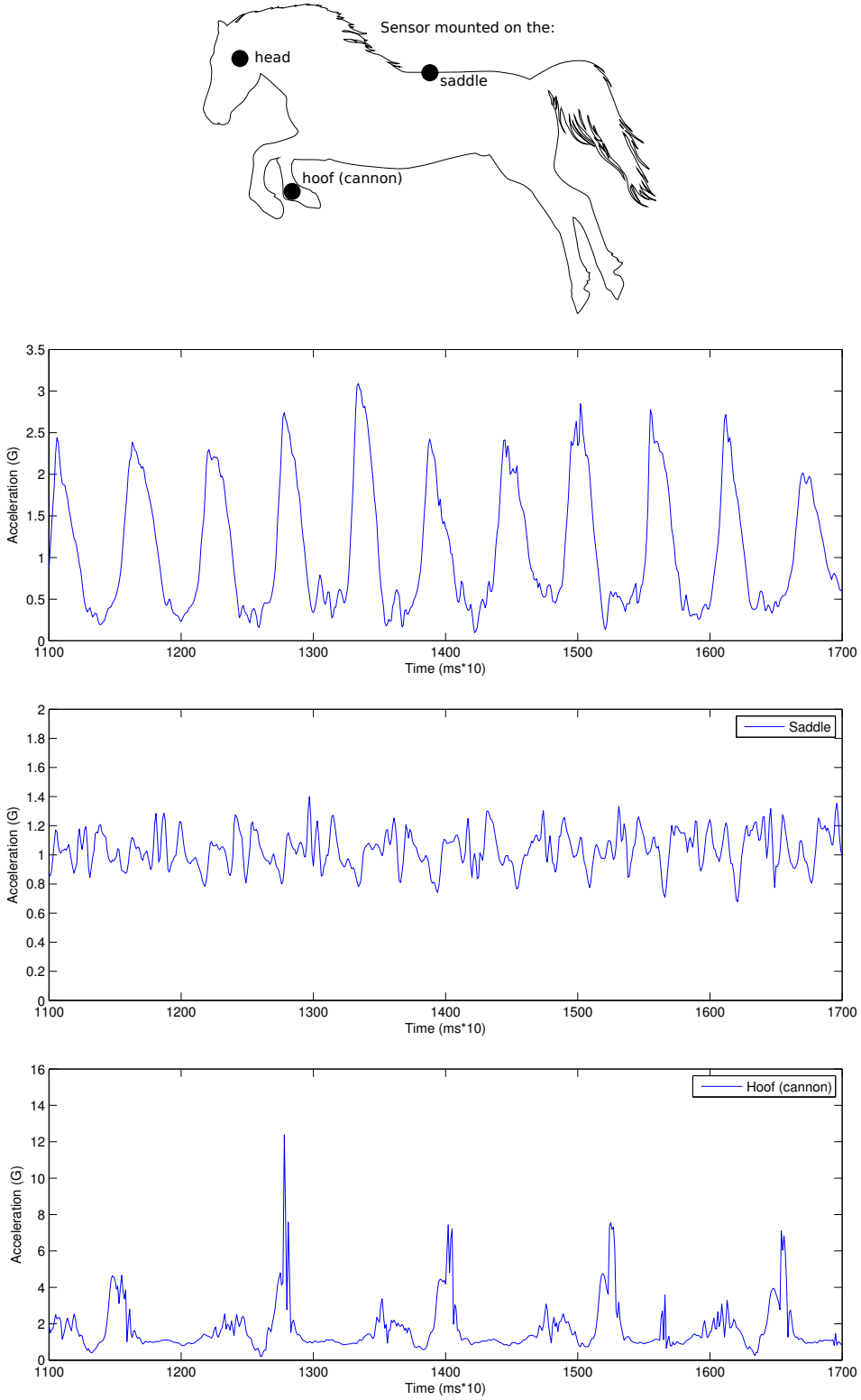
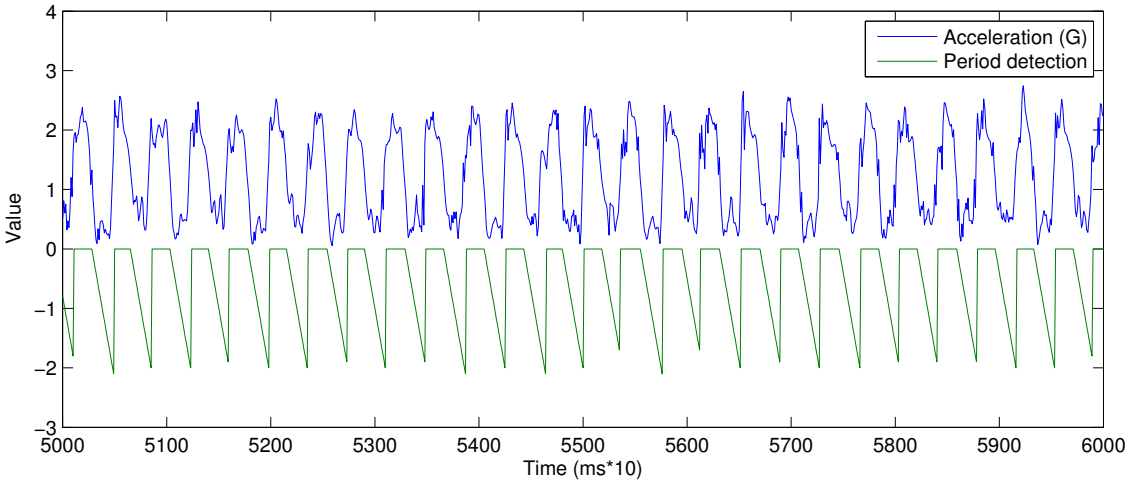


Figure 4.7: Detection of the period of the walk. The narrow green line shows the new detected step. The descending green line shows the time when the next step is awaited. The peroid is different in every cycle.




between each gait is in order of hoofs touching the ground. All the other parameters (acceleration in peaks, timing, ...) can be the same for more than one type of movement. Of course, we probably can find a correlation between the other parameters and the type of movement, but we cannot guarantee the accuracy without any deeper analysis.

When we have a sensor on each cannon with synchronized start and stop, we can distinguish each leg (hoof touching the ground) and get accurate results. When we have only one sensor, for example on the saddle, we still know that any hoof touched the ground, but the sensor measures the values with higher noise and we can capture some false positives. These false positives can negatively affect the final analysis. This method is discussed in detail in section 4.3.4. The differences between the measured values on different positions of the sensor are shown in figure 4.6.

### 4.3.2 Spectral analysis

The spectral analysis of the measured data can help to find some correlation between the gait type and the frequencies occurred in the digital signal. There is no guarantee that we will find a suitable correlation, but the probability is not low. The visualization of the spectral analysis of the data captured on the saddle is shown in figure 4.8.

The figure is generated using the `spectrogram` function in Matlab with 2 s Hamming window. The longest step should not be longer than 1 s. We can see some sections in the figure that correspond to different gaits.

Now, we can distinguish the sections by some markers. The marker can be, for example, the strongest frequency, average amplitude, occurrence of higher frequencies, etc. Here in this example have marked the highest frequency in figure 4.9.

Sometimes, the highest frequency is affected by other factors because the sensor is mounted on the saddle – far away from the hoofs. I will filter single values that point to the other type of movement than its neighbors. The result is shown in figure 4.10.

Now, we can compare this example analysis to the reality. The figure 4.11 shows the regions marked by colors. Each region with the same color represents the same type of movement.

The analysis gave us suitable results, but we cannot guarantee that the results will be accurate at all times. Different horses will probably need different calibration of the used constants. Of

Figure 4.8: Spetral analysis of the variable movement.

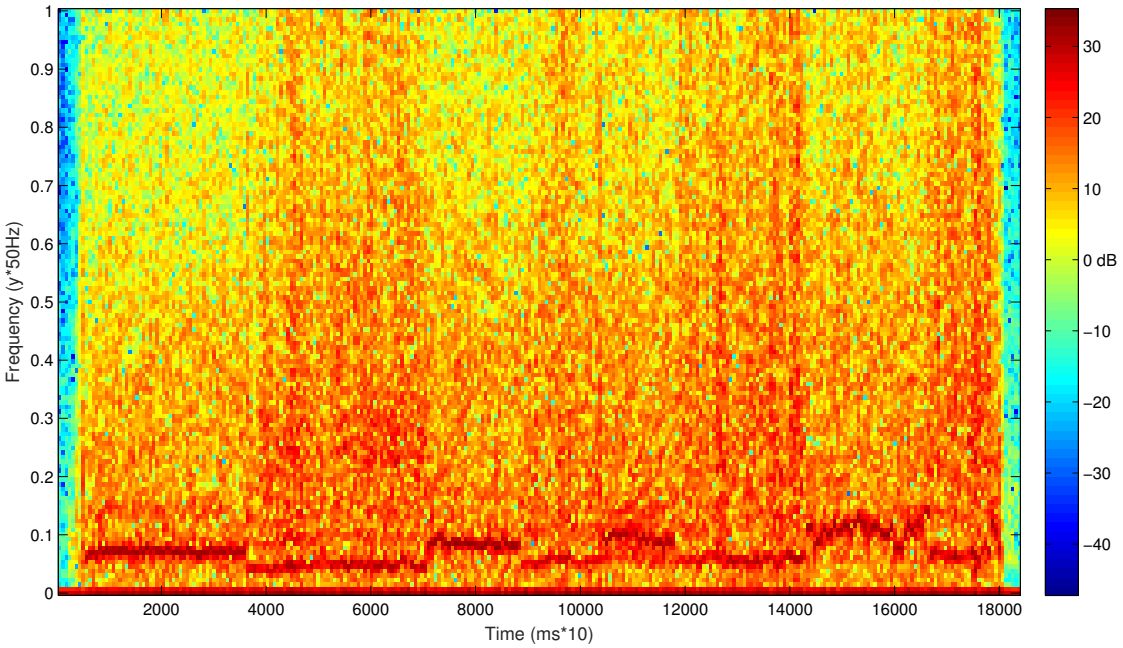



Figure 4.9: Selected strongest frequency from the spectrogram.

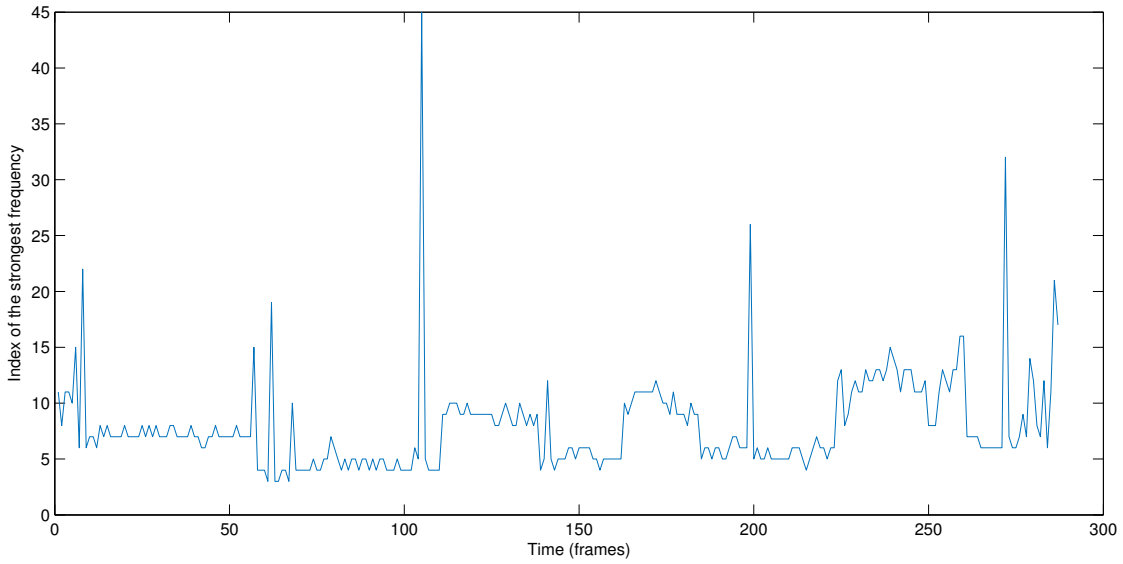


Figure 4.10: Strongest frequency from the spectrogram filtered by its neighbors.

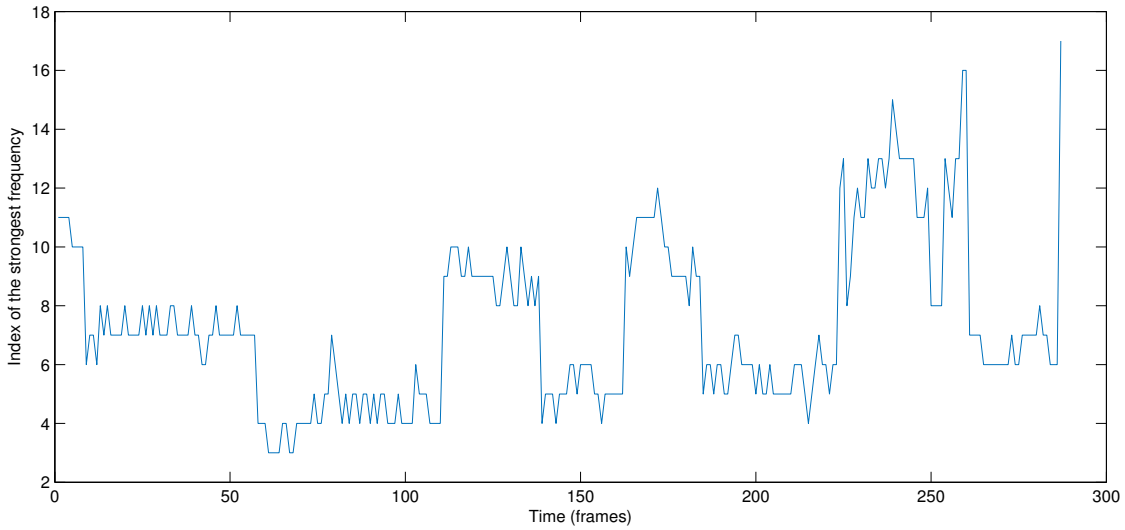
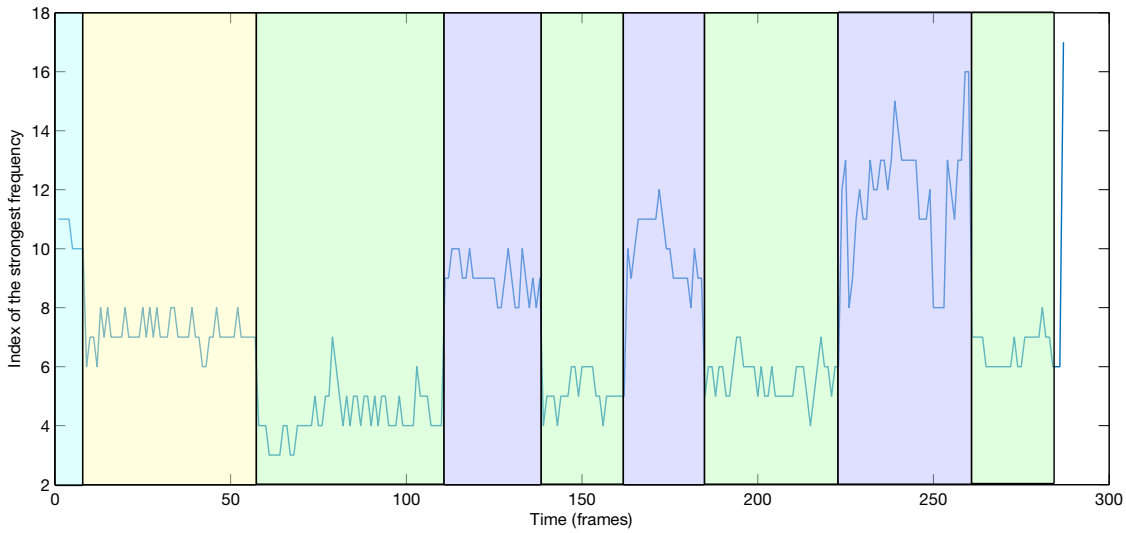



Figure 4.11: Strongest frequency from the spectrogram compared to the different types of movement. The types are: blue – stand; yellow – walk; green – trot; violet – canter



course, we can analyze more factors and then select the most probable movement. We can get almost accurate results with adding more other factors, but we can never guarantee a 100 % correctness of the algorithm. This type of analysis used only the size of the acceleration vector in each time. With other values, we can get more accurate results as well.

I finally did not select this type of analysis because of multiple factors:

- The analysis needs higher computation power in comparison to other methods.
- The method does not analyze the movement itself, but only its corresponding factors. The method tries to guess the type of the movement based on these factors.
- It is hard to add a new type of the movement. The types of gaits are defined inside the algorithm, and there is no possibility to add changes easily.

### 4.3.3 Machine learning

When we have a "hand" created results and their source data we can try to use some machine learning methods. The machine learning methods can be combined with other methods mentioned in this chapter. For example, the spectral analysis from section 4.3.2 can be followed by a neural network, which gives the final results. I was not working with any machine learning algorithm in this thesis and I only mention this type of analysis as a possibility. This analysis again needs a high computation power and there is no guarantee of the accuracy.

### 4.3.4 Looking for structures

The non-trivial digital signal contains some important points, for example, zeros, maximal or minimal values (peaks), inflection points or points that form an obtuse angle with their neighbors in any scale of their plot.

**Definition 4.3.1. Peak** A peak $p_t$ of a signal $P = \{p_0, p_1, \ldots, p_i, \ldots, p_n\}$ is a measured value at time $i$ that fullfills one of these conditions:

$$p_{i+1} > p_i \wedge p_{i-1} > p_i \tag{4.1}$$

$$p_{i+1} < p_i \wedge p_{i-1} < p_i \tag{4.2}$$

The peak that fulfills the first condition is called minimal peak and the peak that fulfills the second condition is called maximal peak.

**Definition 4.3.2. Heel** A heel $p_t$ of a signal $P = p_0, p_1, \ldots, p_i, \ldots, p_n$ is a measured value at time $i$ that fullfills one of these conditions:

$$(p_{i+1} > p_i \wedge p_{i-1} = p_i) \wedge (\exists n : p_n > p_i \wedge \forall j \in \langle n, i \rangle : p_j = p_i) \tag{4.3}$$

$$(p_{i+1} < p_i \wedge p_{i-1} = p_i) \wedge (\exists n : p_n < p_i \wedge \forall j \in \langle n, i \rangle : p_j = p_i) \tag{4.4}$$

A heel that fulfills the first condition is a minimal heel and a heel that fulfills the second condition is a maximal heel.

**Definition 4.3.3. Hill** A hill $H$ is an interval $I_H = \langle i, j \rangle$ in signal $S$ where:

1. The points $s_i$ and $s_j$ are minimal peaks or heels.
2. $\forall$ peaks $p \in I : p > s_i \wedge p > s_j$.

*Remark.* Size of hill $H$, written $|H|$, is the length of interval $I_H = \langle i, j \rangle$. ($|H| = j - i$)

Figure 4.12: Example of the peak, heel and hill.

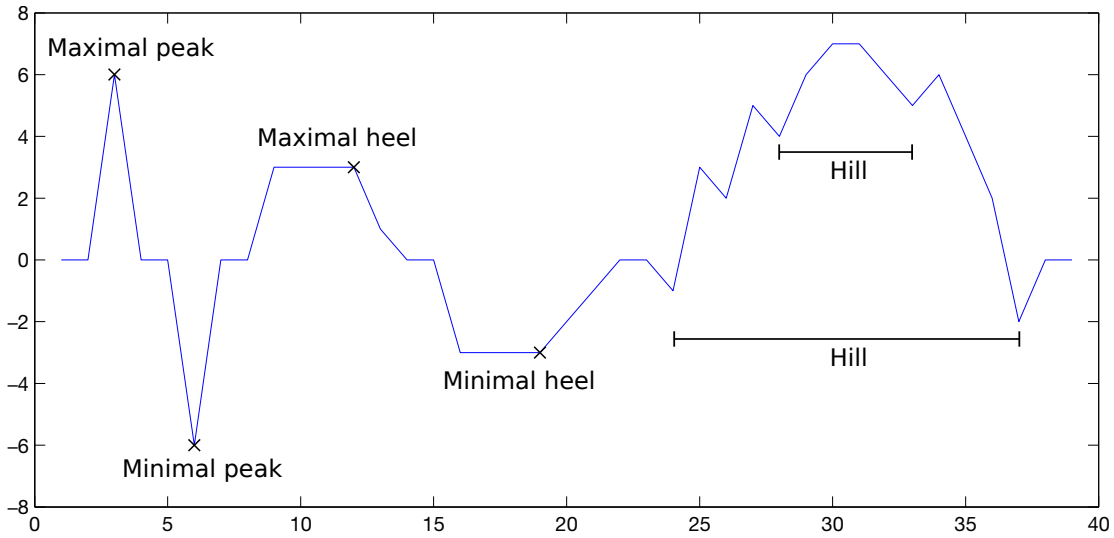
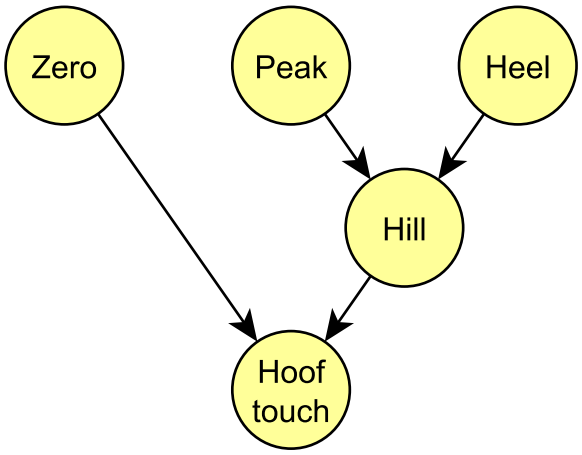



Figure 4.13: Example of the dependencies among definitions displayed as a tree.




*Remark.* A hill $H$ is a part of another hill $G$, written $G \in H$, when the interval $I_G \in I_H$.

We can look for non-trivial structures like hills, peaks occurred at the same time in different signals or repetition of important points with the same properties. For example, we can count only hills $H$ that are not a part of another hill $F : H \in F$ and do not contain another hill $G$, where $|H| > c \cdot |G|$. The $c \in \mathbb{R}$ is an arbitrary constant that means that the hill $H$ is $c$ times longer than hill $G$.

The figure 4.12 shows examples of the minimal and maximal peak, heel and hill.

We can define more structures and use already defined structures in the new definitions. It means that we get a tree of new structures, where the edges are dependencies inside the definitions. The example of this tree is shown in figure 4.13.

We can combine the definitions with outputs of other types of analysis. The other types can be added as new inputs (sensors). When we add an inaccurate (not 100 % accurate) analysis, we have to take into account that the final result can be inaccurate, too.

The "Looking for structures" analysis is in basics similar to the step counter in section 4.3.1,

but it can be extended to a more powerful tool by the changes mentioned in this section. This is the reason why I finally selected this analysis for implementation into the hardware. Another reason is based on low Central Processing Unit (CPU) power requirements and the possibility to get accurate results.

The results are as accurate as the used sensors, definitions and methods. The 100 % accuracy of the algorithm can be achieved by computing the results based on their definitions. Of course, it may be very hard to create an accurate definition of each movement, but when we have a definition, it is easier to test the algorithm against the definition and get indisputable results. The implementation is discussed in section 4.4.

## 4.4 Implemented algorithm

I have selected the method described in section 4.3.4. The algorithm is implemented as a C++ library, so it can be a part of the firmware of the , or it can be compiled as a separate application.

**Example of the code calling the implemented algorithm:**

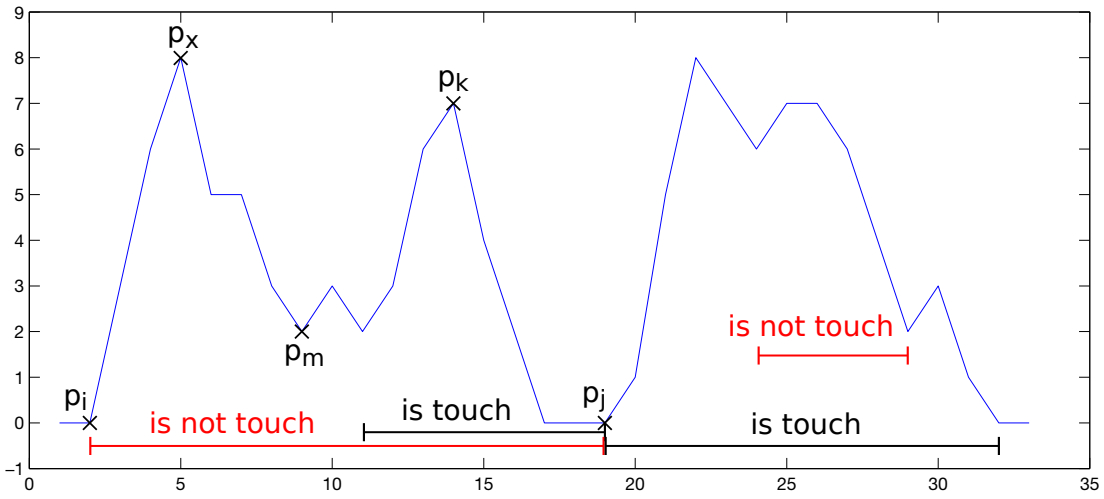
```
1  const int frequency = 100; // Hz
2  HorseAnalysis horse(frequency);
3
4  for(;;)
5  {
6      // Load data from sensors to AccX, AccY, AccZ, GyrX, GyrY, GyrZ
7      horse.addData(AccX, AccY, AccZ, GyrX, GyrY, GyrZ);
8
9      // get elapsed time in milliseconds
10     int elapsedTime = horse.elapsedTime();
11     // true if horse is moving
12     bool isMoving = horse.isMoving();
13     // get number of hoof contacts with ground
14     int numberOfSteps = horse.numSteps();
15     // get type of movement as int representing enum
16     int movementType = horse.detectAndNameMovement();
17 }
```

### 4.4.1 Workflow of the algorithm

The algorithm stores a history of the measured values (5 seconds by default). The history is used for dynamic calibration and as the source for the analysis. The algorithm can run real-time with an approximate latency of one horse's step. So, it cannot detect an unfinished step. The algorithm uses this workflow:

1. Read the new input data.
2. Calibrate the vertical direction and morph the values into the vertical plane.
3. Mark all the peaks and other interesting points based on their definitions.
4. Mark more complex structures based on the previously marked points.

Figure 4.14: Visualization of the used symbols in definition 4.4.1.




5. Detect a new step. If the new step is detected, continue.
   Otherwise, move to step (1).

6. Evaluate definitions of the gaits.

7. Select the most appropriate definition and return the result.

### 4.4.2 Detection of touching the ground

**Definition 4.4.1. Touch** In this definition, by peak is meant peak or heel. A hill $P$ (based on definition 4.3.3) is a touch, when the hill $P$ does not contain a minimal peak $p_m$ that fulfills:

1. Let $p_i$ and $p_j$ are border peaks of the hill $P$. Let peak $p_x$ is the highest maximal peak $\in P$. Then:
$$\exists p_k : p_k > \frac{p_x}{2} \wedge p_m < \frac{p_x}{2} \wedge (i < x < m < k < j \vee i < k < m < x < j)$$

2. The hill $P \notin H$, where the hill $H$ is a touch, too.

*Remark.* The symbols used in the definition 4.4.1 are visualized in figure 4.14.

Touching the ground by hoof is detected when the algorithm detects a touch (based on definition 4.4.1).

All the hills $H$ are counted as touching the ground by hoof. We know the

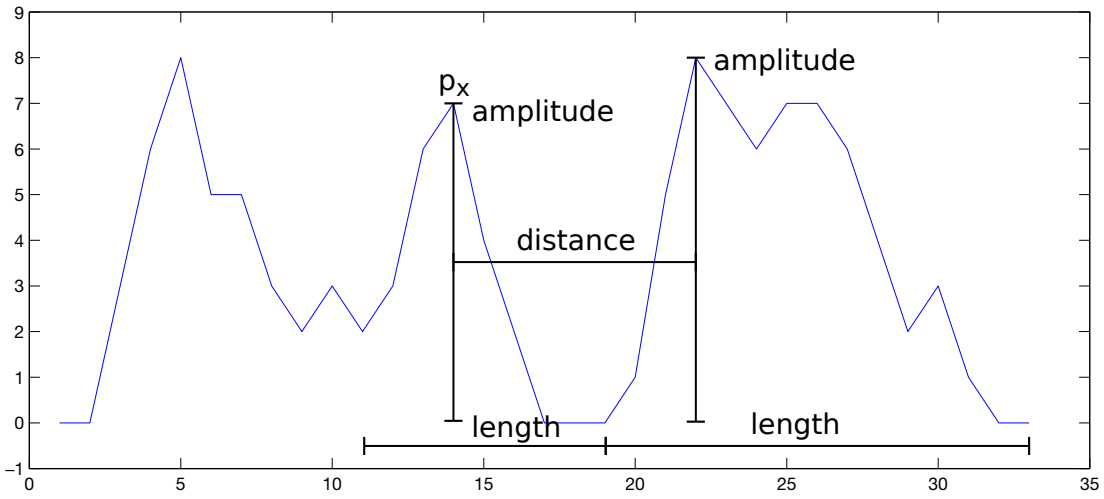
1. amplitude – corresponds to $p_x$,

2. length – corresponds to $|H|$,

3. distance between two hills $H$ and $G$ – corresponds to $|h_g - h_x|$

The amplitude, length and distance is visualized in figure 4.15.

Figure 4.15: Visualization of the amplitude, length and distance of the touch (or hill).




### 4.4.3 Definitions of the gaits

From this point, the algorithm is different when we have the sensor or sensors mounted on the cannon or on the saddle. When the sensor is mounted on the saddle, it can detect touches by all four hoofs, but there may be errors caused by accelerations from different sources. The sensor on the saddle can detect that a hoof touched the ground, but it cannot detect which one.

When we have four sensors – one on each cannon, the algorithm can detect the movement as accurately as accurate is our definition of the selected movement. The definition defines the step as a structure inside the digital signal, so the definition does not have to exactly correspond to the real steps made by a horse. For example, we can simulate the steps by moving the sensor inside the laboratory without any horse.

Now, I will start with the situation when we have only one sensor. We can use the definitions 4.1.2, 4.1.3, 4.1.4, 4.1.5 and 4.1.6, but only their parts (3) and (4). When the algorithm detects all the touches based on definition 4.4.1, it can measure the amplitudes, lengths of the touches and distances between them. Then the algorithm is able to select the definition, which fits the best to the detected movement. The example of the definitions presented in the source code of the algorithm is shown below.

**An example of the definitions used for the analysis:**

```
1  int detectMovement()
2  {
3      if(!isMoving())
4          return 0;
5
6      // (...)
7
8      // First definition
9      // -------------------- event ---------------------------
10     // if the amplitude of last step is more than 5.0 G,
11     // -------------------- action --------------------------
12     // increase the probability of gallop (cval = 3)
```

```
13      if(m_lastStep.amplitude > 5.0 /*G*/)
14          prob[3]++;
15
16      // Second definition
17      // -------------------- event --------------------------
18      // if the time from last step is more than 400 ms,
19      // and the amplitude of the last step was less than 2.0 G
20      // -------------------- action -------------------------
21      // increase the probability of walk (krok = 1)
22      if(
23          m_time - m_lastStep.time > 400 /*ms*/ &&
24          m_lastStep.amplitude < 2.0 /*G*/
25      )
26          prob[1]++;
27
28      // other definitions
29      // (...)
30
31      return getIndexOfMaxValue(prob, 5 /*size of array prob*/);
32  }
```

When the horse is standing, we cannot detect any touches with significant amplitude. So, all the touches with the amplitude smaller than 1.5 G are not counted as touches, and the movement is evaluated as standing.

When we have a sensor on each cannon (or hoof), the algorithm is able to detect which hoof touched the ground. The signal measured on the cannon has much lower noise level than the signal measured on the saddle. When we have multiple sensors, we have to extend the definitions and take into account the part (1) of each definition. Then we get more accurate results with the same algorithm, but we need at least four sensors and the installation time is longer.

### 4.4.4 Possibilities of extension of the algorithm

The hardware contains some other useful sensors that can help with the movement analysis. These sensors can be used in similar way like the accelerometer. Using these sensors is not a part of this thesis.

- **Gyroscope or megnetometer:** can be used as input for sensor fusion and emulation of AHRS.
- **DWM1000:** Relative location to other sensors with 300 m range and precision 10 cm.
- **GPS:** Externally connected GPS with NMEA 2000 output on 3.3 V UART.
- **Heart rate:** Externally connected heart rate sensor.

### 4.4.5 Results

The results of the analysis is a text file with information about the actual movement type in every second and a Comma-Separated Values (CSV) file with all computed data during the analysis. The example of the text file is shown in figure 4.16.

### 4.4.6 Comparison of the SensorBoard to MetaWear

During the work on this thesis, I was measuring the data with two different devices. In the beginning I used the MetaWear [1] sensor and later the new developed . Here I will mention some differences I recognized during each measurement process.

- The MetaWear sensor was not able to log more than 30 s of data at 100 Hz frequency. The logged more than 10 hours and the time depends only on the capacity of the SD card.

- The MetaWear sensor was not able to switch off itself. So, the sensor operated only 3 hours after removing from the charger. The can be switched on or off manually and it operates about 10 hours until the battery is discharged.

- The MetaWear sensor has a Bluetooth connection that can transmit approximately at 2 m range. The transmits approximately in 25 m range.

- The MetaWear sensor is smaller than . The dimensions of the MetaWear are approximately twice smaller than the SesnroBoard prototype.

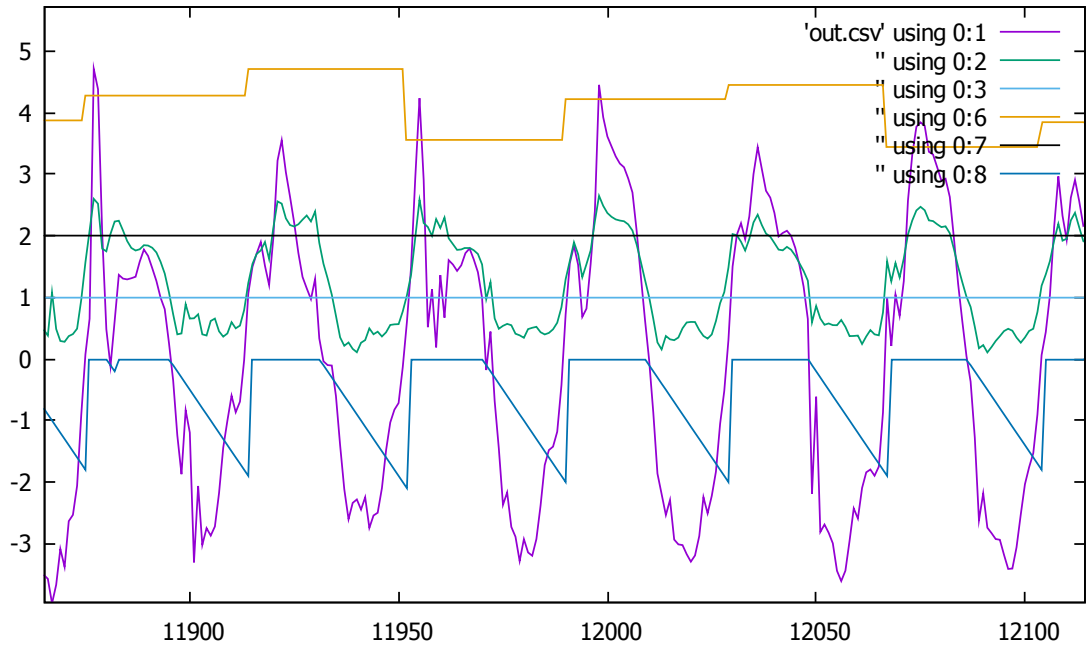
Figure 4.16: Example of the text output of the selected movement analysis.

```
(...)
Second   55: moving yes, steps   93, klus , ...
Second   56: moving yes, steps   95, klus , ...
Second   57: moving yes, steps   97, klus , ...
Second   58: moving yes, steps   99, krok , ...
Second   59: moving yes, steps  101, krok , ...
Second   60: moving yes, steps  102, krok , ...
Second   61: moving no , steps  103, krok , ...
Second   62: moving no , steps  103, stani, ...
Second   63: moving yes, steps  103, stani, ...
Second   64: moving no , steps  104, stani, ...
(...)
```

The names of the movement are in Czech language for easier testing in the Czech environment. The translation is:
"stani" – stand; "krok" – walk; "klus" – trot, "cval" – canter.

The CSV output can be visualized into a graph for easier orientation in the values. The columns represent selected variables inside the analysis. An example of this graph is shown in figure 4.17.

Figure 4.17: An example of the graph generated from the CSV file given as output of the analysis.




| **Data line** | **Explanation** |
|---|---|
| using 0:1 | Measured data projected into the vertical plane. |
| using 0:2 | Size of the measured acceleration vector. |
| using 0:3 | Horse is moving (value 1) or horse is standing (value 0). |
| using 0:6 | Amplitude of the last finished step (not the actual unfinished one). |
| using 0:7 | The detected type of movement (value 2 is trot). |
| using 0:8 | When the value is 0, the step is in progress, when the value is descending, the new step is awaiting. When the new step is comming very soon (the blue line is not under the treshold), the new step is false positive and ignored. |

CHAPTER 5

# CONCLUSION

The thesis was finally split into four milestones. The first milestone was achieved when I have decided to develop a new data logging hardware because the existing solutions did not fulfill all the requirements. I have successfully developed the SensorBoard hardware in the second step, and then the board was manufactured in six copies. I have achieved the second milestone with the manufactured hardware. The third phase was about testing the device and about implementing libraries for communication between the controller and other parts, mainly sensors. I have found several mistakes in the hardware design during programming, but all of them were corrected. The third milestone was achieved when I had working hardware with C++ libraries for easier communication with sensors and other parts of the device. These libraries are a part of the API. During the last phase, I have implemented the firmware for real-time movement analysis. When the SensorBoard is placed on a horse under the saddle, it is able to recognize the kind of horse movement. This firmware demonstrates the functionality of the designed hardware in real use. I have achieved the last milestone during the first successful test of the platform on a horse's body.

There are many applications and use cases of the SensorBoard presented as examples in the thesis. I have selected the movement analysis case because this task started the whole project in the beginning. The movement analysis firmware shows the advantages of running some user code on the hardware. We can see the results of the analysis in real-time. The implemented movement analysis algorithm on the SensorBoard was working on any tested horse. The types of movement were distinguished with more than 99 % accuracy with the sensor located under the saddle. More than 1 % error was present only in determination between trot and canter.

If there is an interest in future work with this hardware, I would like to recommend to create the second version of the SensorBoard with attention to the list of hardware errors presented in this thesis. The actual version was designed as a prototype and it is not recommended for production. The prototype of the SensorBoard was used, for example, as a simple autopilot of a small quadcopter, I believe it will be used in several applications in the near future. I hope that the firmware for the movement analysis of a horse will help to work with this kind of animals easier.

# BIBLIOGRAPHY


[1] Sensors for r&d. [Online]. Available: https://mbientlab.com/ (Accessed 2018-04-01). 2.4, 4.4.6

[2] Ardupilot mega. [Online]. Available: https://www.ardupilot.co.uk/ (Accessed 2018-04-01). 2.4

[3] Px4 autopilot. [Online]. Available: https://pixhawk.org/ (Accessed 2018-04-01). 2.4

[4] Xsens mvn: Consistent tracking of human motion using inertial sensing. [Online]. Available: https://cdn2.hubspot.net/hubfs/3446270/Downloads/Whitepapers/MVN_Whitepaper.pdf (Accessed 2018-04-01). 2.4, 4

[5] X-imu our original versatile imu board. [Online]. Available: http://x-io.co.uk/x-imu/ (Accessed 2018-04-01). 2.4

[6] *Product card TACTM-35N-F*, Ninigi, 01 2018. 2.5, 2.9, 5

[7] *Linear Li-Ion Battery Charger with Power Path and USB Compatibility in LFCSP*, Analog Devices, 2 2017, rev. B. 2.5, 2.9

[8] *Linear LiFePO4 Battery Charger with Power Path and USB Compatibility in LFCSP*, Analog Devices, 2 2017, rev. 0. 2.5

[9] *I2C HUMIDITY AND TEMPERATURE SENSOR*, Silicon Laboratories, 8 2016, revision 1.2. 2.5, 2.9

[10] *HDC2010 Low Power Humidity and Temperature Digital Sensors*, Texas Instruments, 3 2018, revision A. 2.5

[11] *Datasheet SHTC1 Humidity and Temperature Sensor IC*, Sensirion, 8 2015, version 4. 2.5

[12] *DWM1000 IEEE 802.15.4-2011 UWB Transceiver Module*, Decawave Ltd, 8 2016, v1.6. 2.5, 2.9, 3.3.4, 3.3.4, 5, 11, 12, 13, 14

[13] *DW1000 IEEE802.15.4-2011 UWB Transceiver*, Decawave Ltd, 2015, version 2.09. 2.5

[14] *BMP280 Digital Pressure Sensor*, Bosch Sensortec, 05 2015. 2.5, 2.9, 3.3.4

[15] *Digital pressure sensor*, Bosch Sensortec, 01 2018, revision 1.0. 2.5

[16] *FT232R USB UART IC*, Future Technology Devices International Ltd, 11 2015, v2.13. 2.5, 2.5.3, 2.9, 5, 1, 2, 3

[17] *Single-Chip USB-to-UART Bridge*, Silicon Labs, 01 2017, rev. 1.8. 2.5

[18] *USB to serial chip CH340*, wch.cn, 2017, version 1D. 2.5

[19] *MPU-9250 Product Specification*, InvenSense, 6 2016, revision 1.1. 2.5, 2.9, 2.7.5

[20] *Small, low power Inertial Measurement Unit*, Bosch Sensortec, 2 2015, revision 0.8. 2.5, 2.9, 2.7.5

[21] *BMF055 Custom programmable 9-axis motion sensor*, Bosch Sensortec, 11 2015. 2.5, 2.7.3, 2.9, 2.7.5, 3, 3.1.2, 3.1.2, 5, 5, 5

[22] H. Xiaoyan and K. S. C. Kuang, "Structural motion monitoring systems using 9-axis sensing modules," in *The 2016 Structures Congress*, 9 2016, pp. 4–6. 2.5, 2.9, 5

[23] *HM-TRP Series 100mW Transceiver modules V1.0*, HopeRF Electronic, 2006, v1.0. 2.5, 2.9, 5, 5, 5, 7

[24] *MicroSD Memory Card Connector*, Molex, 8 2016. 2.5, 2.9, 5

[25] *Digital Ambient Light Sensor*, LiteON Technology Corp., 2018, revision 1.1. 2.5, 2.9

[26] *Ambient Light Sensor – Surface Mount*, Everlight America, 12 2013, issue No: 1. 2.5

[27] *Ambient Light Sensor with Dark Current Compensation*, ON Semiconductor, 11 2017, revision 4. 2.5

[28] *ESP-WROOM-32 Datasheet*, Espressif Systems, 9 2017, v2.1. 2.5, 2.5.3, 2.7.3, 2.9, 3, 1, 3.1, 3.1.1, 3.2.1, 5

[29] *ARM-based 32-bit MCU*, STMicroelectronics, 8 2016, rev15. 2.5

[30] *300mA, Micropower, VLDO Linear Regulator*, Union Semiconductor, 7 2008, rev01. 2.5, 2.9

[31] *Very low drop voltage regulator with inhibit function*, STMicroelectronics, 5 2017, rev31. 2.6, 2.9

[32] *Micro-USB B Receptacle with Flange, Bottom Mount, Surface Mount, Lead-Free*, Molex, 3 2018. 2.6, 2.9

[33] *BERGSTIK Headers*, Amphenol FCI, 7 2016, rev:L. 2.6, 2.9

[34] Easy applicable graphical layout editor. [Online]. Available: https://www.autodesk.com/products/eagle/overview (Accessed 2018-04-01). 2.5, 2.6.1, 5

[35] Kicad eda: A cross platform and open source electronics design automation suite. [Online]. Available: http://kicad-pcb.org/ (Accessed 2017-08-01). 2.5

[36] E. Systems, "Esp32-devkitc getting started guide," https://dl.espressif.com/doc/esp-idf/latest/get-started/get-started-devkitc.html, 2018. 2.5.3

[37] Zadávání zakázek pro strojní osazování dps na osazovacím automatu samsung cp45v. [Online]. Available: http://smtplus.cz/pdf/oa.pdf (Accessed 2018-04-01). 2.6.2

[38] Podmínky zadávání zakázek. [Online]. Available: http://smtplus.cz/?page=podminky (Accessed 2018-04-01). 2.6.2

[39] Export gerber dat z eagle. [Online]. Available: https://www.gatema.cz/www/files/file/eagle-002.pdf (Accessed 2017-08-01). 2.6.2

[40] Technické informace. [Online]. Available: https://www.gatema.cz/plosne-spoje/technicke-informace (Accessed 2017-08-01). 2.6.2

[41] *Intelligent control LED integrated light source*, Worldsemi, 01 2017. 2.7.3

[42] *SMART ARM-Based Microcontrollers SAM D20E / SAM D20G / SAM D20J*, Atmel, 09 2016. 2.7.3, 3, 2, 3.1.2, 10

[43] Alb32-wrover – esp-wroom-32 module with 128mb flash and 32mb psram. [Online]. Available: https://www.analoglamb.com/product/alb32-wrover-esp32-module-with-64mb-flash-and-32mb-psram/ (Accessed 2018-04-01). 2.7.3

[44] *ARM Debug Interface Architecture Specification*, ARM Holdings, 08 2013, third issue, for ADIv5.0 to ADIv5.2. 3, 3.1.2

[45] E. Systems, "Espressif iot development framework," https://github.com/espressif/esp-idf, 2018. 3.1.1, 3.1.1

[46] E. Systems, "Arduino core for esp32 wifi chip," https://github.com/espressif/arduino-esp32, 2018. 3.1.1, 3.1.1

[47] Micropython. [Online]. Available: http://micropython.org/ (Accessed 2018-04-01). 3.1.1, 3.1.1, 3.3.6

[48] Arduino support from simulink. [Online]. Available: https://uk.mathworks.com/hardware-support/arduino-simulink.html (Accessed 2018-04-01). 3.1.1

[49] N. Kolban, *Kolban's Book on ESP32*. Leanpub.com, 5 2018, p. 1059. 3.1.1

[50] Esp-idf programming guide. [Online]. Available: https://esp-idf.readthedocs.io/en/v2.0/index.html (Accessed 2018-04-01). 3.1.1

[51] Build and flash with eclipse ide. [Online]. Available: https://esp-idf.readthedocs.io/en/v2.0/eclipse-setup.html (Accessed 2018-04-01). 1

[52] Platformio is an open source ecosystem for iot development. [Online]. Available: https://platformio.org/ (Accessed 2018-04-01). 2

[53] A hackable text editor for the 21st century. [Online]. Available: https://atom.io/ (Accessed 2018-04-01). 2

[54] L. Blocher, "Bmf055-flight-controller," https://github.com/NightHawk32/BMF055-flight-controller, 2016. 3.1.2, 3.3.3

[55] Atmel studio 7. [Online]. Available: http://www.microchip.com/avr-support/atmel-studio-7 (Accessed 2018-04-01). 3.1.2

[56] Atmel-ice. [Online]. Available: https://www.microchip.com/atatmel-ice (Accessed 2018-04-01). 3.1.2

[57] Microchip advanced software framework samd20 documentation. [Online]. Available: http://asf.atmel.com/docs/latest/samd20/html/index.html (Accessed 2018-04-01). 3.2.2

[58] Esp-idf programming guide api reference. [Online]. Available: http://esp-idf.readthedocs.io/en/latest/api-reference/index.html (Accessed 2018-04-01). 3.2.2

[59] W. Elmenreich, "Sensor fusion in time-triggered systems," Ph.D. dissertation, 2002-10-01. [Online]. Available: https://mobile.aau.at/~welmenre/papers/elmenreich_Dissertation_sensorFusionInTimeTriggeredSystems.pdf 3.3.2

[60] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm," in *2011 IEEE International Conference on Rehabilitation Robotics*, 6 2011, pp. 1–7. 3.3.2

[61] Esp-idf programming guide freertos. [Online]. Available: http://esp-idf.readthedocs.io/en/latest/api-reference/system/freertos.html (Accessed 2018-04-01). 3.3.5

[62] I. Grokhotkov, "Simple http server," https://github.com/igrr/esp32-http-server, 2018. 3.3.7

[63] C. McIntyre, "Ftp server for the esp-wroom-32," https://github.com/cgmcintyr/esp32-ftp, 2017. 3.3.7

[64] G. Higgins and S. Martin, *Koně a jejich pohyb*, 1st ed. Praha: Metafora, 2009. 4.1.1, 4.1

[65] M. Duruttya, *Velká etologie koní*, 2nd ed. Košice: HIPO-DUR, 2005. 4.1, 4.1, 4.2, 4.3, 4.4

[66] S. E. Harris, *Horse gaits balance and movement*. New York: Maxwell Macmillan International, 1993. 4.1, 4.1.2, 4.1.3, 4.1.4, 4.1.5, 4.1.6

[67] J. G. Proakis and D. K. Manolakis, *Digital Signal Processing (4th Edition)*. Pearson, 2006. [Online]. Available: https://www.amazon.com/Digital-Signal-Processing-John-Proakis/dp/0131873741?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0131873741 (Accessed 2018-04-01). 4.2.2

[68] S. W. Smith, *The Scientist & Engineer's Guide to Digital Signal Processing*. California Technical Pub, 1997. [Online]. Available: https://www.amazon.com/Scientist-Engineers-Digital-Signal-Processing/dp/0966017633?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0966017633 (Accessed 2018-04-01). 4.2.3

[69] Y. Geerts, M. Steyaert, and W. M. Sansen, *Design of Multi-Bit Delta-Sigma A/D Converters (The Springer International Series in Engineering and Computer Science)*. Springer, 2002. [Online]. Available: https://www.amazon.com/Multi-Bit-Delta-Sigma-Converters-International-Engineering/dp/1402070780?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=1402070780 (Accessed 2018-04-01). 4.2.3

[70] *SMA esnd launch receptacle reverse polarized*, Amphenol Connex, 11 2007. 5

# HARDWARE DOCUMENTATION

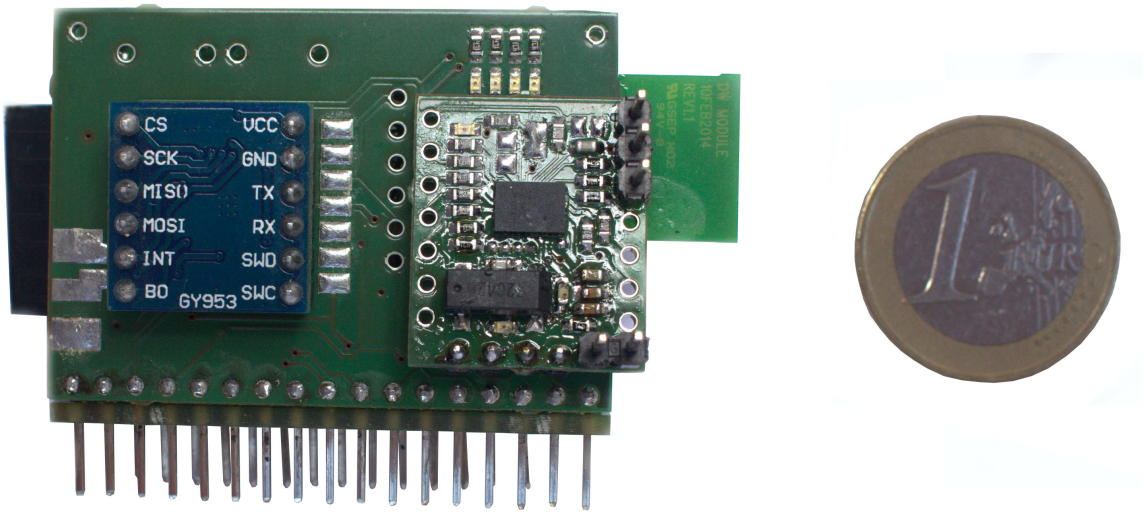



Figure 1: Assembled SensorBoard prototype

## Overview

SensorBoard is a prototype of autonomous hardware platform with microprocessor and various inertial, atmospheric and navigation sensors. The device can be used in many cases from logging data to movement control. The prototype works fully autonomously without connection to external power supply or other electronics, but allows wireless or wired connection to other electronic devices.

### Features

- No external power supply needed
- Internal rechargeable battery
- Charging from USB or any 5 V source
- Allowed simultaneous connection to USB and other power supply
- Three different triaxial acceleromemers, dynamic gyroscopes and two different triaxial magnetometers
- Barometer, light sensor, air humidity, temperature

- A/D converters for connecting external sensors
- TDOA location system with 10 cm precission in 300 m radius
- External UART connector for GPS or radio communication
- Hybrid WiFi and Bluetooth
- Servo outputs and connector with other peripheries
- User programmable dual core ESP32 processor
- Various sleep modes to save battery power
- More than 10 hours of operation from battery
- Buttons and LEDs for interaction with user
- Micro SD card up to 32 GB for logging data (accessible from user program)
- ARM Cortex-M0 co-processor for computing (sensor fusion, data analysis)

### Properties

| **SensorBoard properties** | | |
|---|---|---|
| Parameter | Value | Unit |
| Dimensions | $60 \cdot 31 \cdot 13$ | mm |
| Absolute minimum voltage | $-0.5$ | V |
| Minimum USB voltage | 4.35 | V |
| Minimum working voltage | 3.2 | V |
| Maximum voltage | 5.5 | V |
| Maximum current | 2.1 | A |
| Maximum battery life | 12 | hours |
| WiFi antenna range | 10 | m |
| Bluetooth antenna range | 10 | m |
| TDOA location precission | 10 | cm |
| TDOA location range | 300 | m |
| Average charging time | 60 | minutes |
| Min temperature | $-10$ | °C |
| Max temperature | 80 | °C |

Table 1: SensorBoard properties

### Power Supply

The SensorBoard can be powered from up to three independent power sources. The combination of more simultaneously connected supplies is allowed.

- **Internal battery:** The internal battery is a Lithium polymer battery (Li-poly) 3.7 V (1S) accumulator. When no other supply is applied, the accumulator is used. There is an undervoltage protection for the battery. The SensorBoard can be switched off by hardware switch. The switch disconnects the battery, external supply can be still applied.

- **USB power supply:** When the USB cable is connected, the Li-poly accumulator is charging until the power consumption of the elctronics is lower than the maximal USB supply. There is an overcharging protection of the battery.
- **External** 5 V **power:** The SensorBoard can be powered from an external 5 V supply, for example from the servo connector. The 5 V output on the board can be used as 5 V power input as well.

## Getting Started

### Assembling

The device is assembled from three parts, two sandwich boards and one connector board. Optional devices like BMF055 board, GY953, HM-TRP or GPS can be connected to prepared ports. The parts are shown in figure 2.

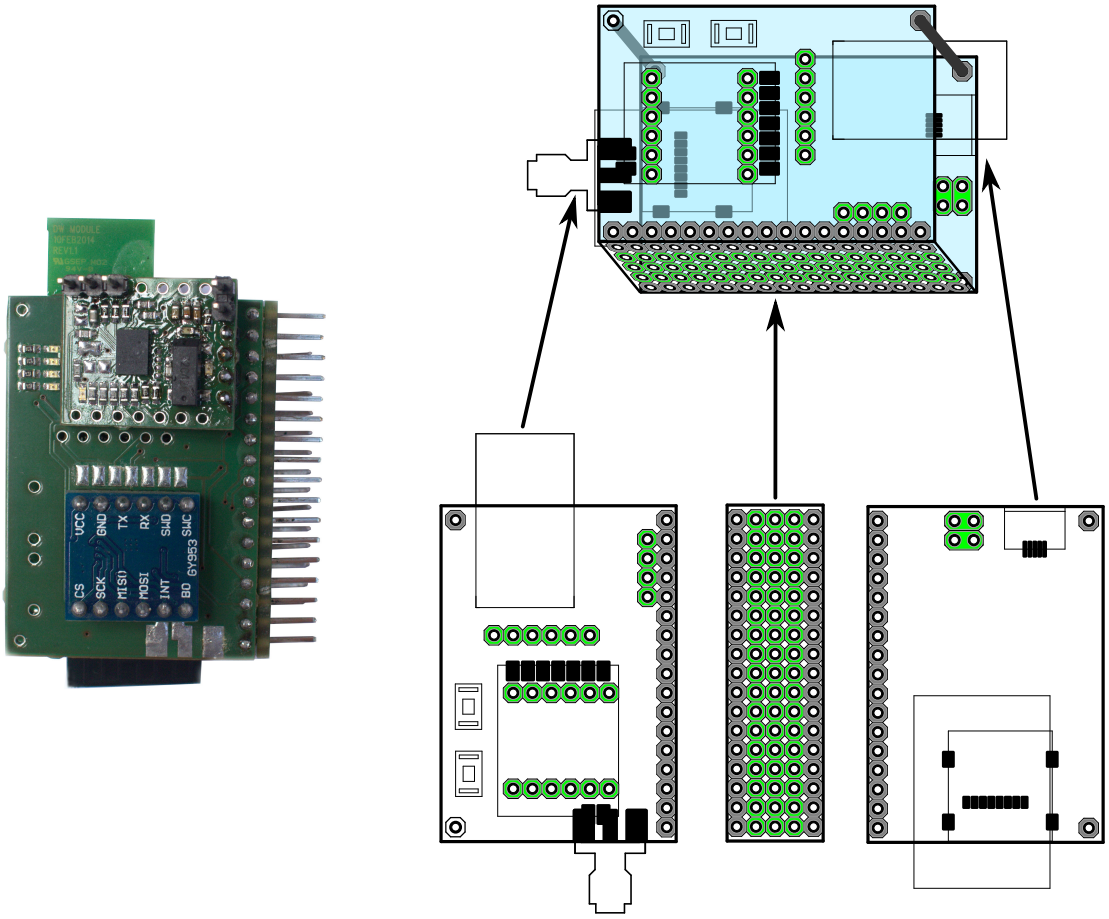



Figure 2: Assembling the SensorBoard prototype

### Components Description

The components are numbered in figure 3 and described in table 2.

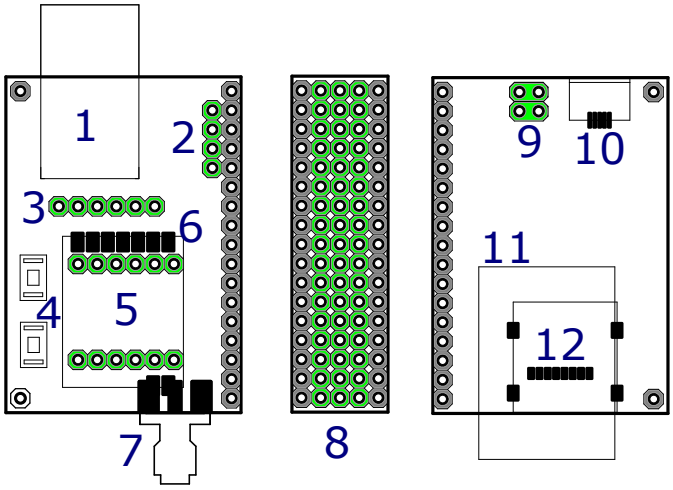



Figure 3: Components of the SensorBoard

| Description of components of the SensorBoard | | |
|---|---|---|
| Number | Description | Datasheet |
| 1 | DWM1000 location sensor | [12] |
| 2 | BMF055 board connector (UART) | [21] |
| 3 | HM-TRP 433/868 MHz radio connector | [23] |
| 4 | Software buttons | [6] |
| 5 | GY-953 connector | [22] |
| 6 | HM-TRP 433/868 MHz radio SMD pads | [23] |
| 7 | RPSMA antenna connector for HM-TRP radio | [70] |
| 8 | External pins | Section 5 |
| 9 | Battery connector | Section 5 |
| 10 | Micro USB connector with FTDI chip | [16] |
| 11 | ESP-WROOM-32 with WiFi/Bluetooth antenna | [28] |
| 12 | Micro SD card slot | [24] |

Table 2: Description of components of the SensorBoard

# Pin Connections

## Pin Numbering

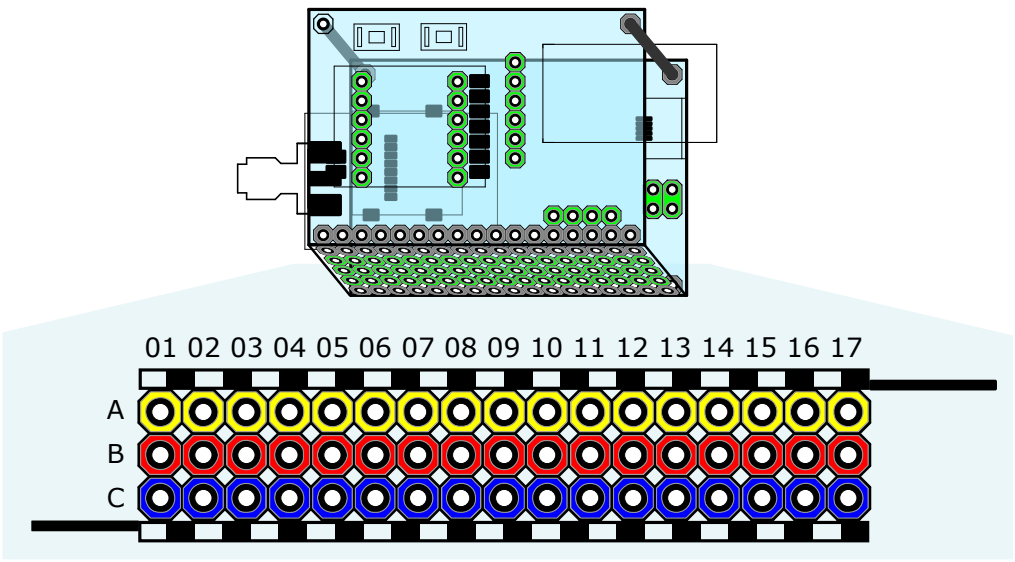



Figure 4: External pins numbering

| External pins 01 – 05 | | | | | |
|---|---|---|---|---|---|
| | 01 | 02 | 03 | 04 | 05 |
| A | GND | +3V3 | SENSOR_VP | SENSOR_VN | IO33 |
| B | Pins B01 – B17 internally connected, details in section 5 | | | | |
| C | GND | | | | |

| External pins 06 – 17 | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
| A | IO25 | IO32 | IO26 | IO18 | IO19 | IO21 | IO16 | IO17 | IO23 | IO22 | IO27 | +5V |
| B | Pins B01 – B17 internally connected, details in section 5 | | | | | | | | | | | |
| C | GND | | | | | | | | | | | |

Table 3: External pins mapping

| Legend | |
|---|---|
| Number | Number of the pin corrsponding to figure 4 and table 3 |
| Name | Name of the pin corresponding to table 3 |
| Safety resistor | Yes if the 220 Ω safety resistor is present |
| Pull-up | Yes if the 10 kΩ pull-up resistor is present |
| Part | The pin is internally connected to which controller |
| Pin | The name of the controller's pin according to the datasheet |

| External pins properties | | | | | |
|---|---|---|---|---|---|
| Number | Name | Safety resistor | Pull-up | Part | Pin |
| 01 | GND | N/A | N/A | N/A | N/A |
| 02 | +3V3 | N/A | N/A | N/A | N/A |
| 03 | RADIO_RX | Yes | No | ESP32 | SENSOR_VP |
| 04 | RADIO_TX | Yes | No | ESP32 | SENSOR_VN |
| 05 | BTN1 | No | Yes | ESP32 | IO33 |
| 06 | BTN0 | No | Yes | ESP32 | IO25 |
| 07 | LED4 | Yes | No | ESP32 | IO32 |
| 08 | SCL0 | Yes | Yes | ESP32 | IO26 |
| 09 | SCK0 | Yes | No | ESP32 | IO18 |
| 10 | MISO0 | Yes | No | ESP32 | IO19 |
| 11 | MOSI0 | Yes | No | ESP32 | IO21 |
| 12 | MPU_TX | Yes | No | ESP32 | IO16 |
| 13 | MPU_RX | Yes | No | ESP32 | IO17 |
| 14 | CS_DECA | Yes | No | ESP32 | IO23 |
| 15 | CS_MPU2 | Yes | No | ESP32 | IO22 |
| 16 | SDA0 | Yes | No | ESP32 | IO27 |
| 17 | +5V | N/A | N/A | N/A | N/A |

Table 4: External pins properties

*Remark.* Some external pins are not connected only to processor, but they are connected to other chips in parallel. The other chips are connected only via input only pins or via safety resistor. So, the possibility of damage by incorrect connection or by wrong software is minimized.

### Pin Description

- 01 – **GND**: Ground
- 02 – **+3V3**: Power Supply 3.3 V from linear stabilizer
- 03 – **RADIO_RX**: UART receive pin for external devices like radio communication or GPS
- 04 – **RADIO_TX**: UART transmit pin for external devices like radio communication or GPS
- 05 – **BTN1**: Usage defined by user software, internally connected to button 1
- 06 – **BTN0**: Usage defined by user software, internally connected to button 0

- 07 – **LED4**: Usage defined by user software, internally connected to LED4
- 08 – **SCL0**: $I^2C$ clock pin for external devices, internally connected to sensors
- 09 – **SCK0**: SPI clock pin for external devices, internally connected to sensors
- 10 – **MISO0**: SPI data output pin for external devices, internally connected to sensors
- 11 – **MOSI0**: SPI data input pin for external devices, internally connected to sensors
- 12 – **MPU_TX**: UART transmit pin for GY953 or external device
- 13 – **MPU_RX**: UART receive pin for GY953 or external device
- 14 – **CS_DECA**: Usage defined by user software, internally used as chip select pin for DWM1000 TDOA location sensor
- 15 – **CS_MPU2**: Usage defined by user software, internally connected to GY953 port
- 16 – **SDA0**: $I^2C$ data pin for external devices, internally connected to sensors
- 17 – **+5V**: 5 V power supply from battery or USB or external source, the battery voltage is converted to 5 V

**Note:** All pins except power supply pins can be redefined by software. Their new definition should not be in collision with other connected parts. For details see the section 5.

### Power Supply

The pins B01 – B17 are defined by jumpers. We usually need these pins connected to power supply. We can define the voltage by connecting arbitrary B pin to target power supply. The pins B01 – B17 are internally connected, but they are not connected to anything else. We can define pins B01 – 17 as:

- **GND** by connecting A01 and B01 by jumper
- **+3V3** by connecting A02 and B02 by jumper
- **+5V** by connecting A17 and B17 by jumper
- **anything** by connecting any B pin to our target

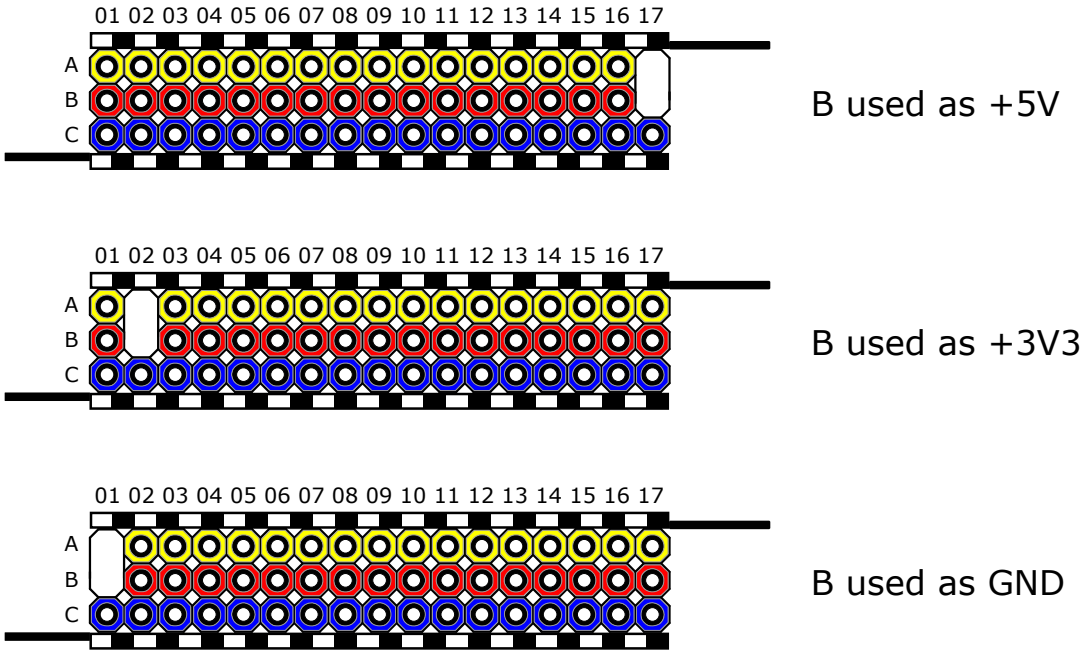



Figure 5: Power supply definition by jumpers

## LED Meanings

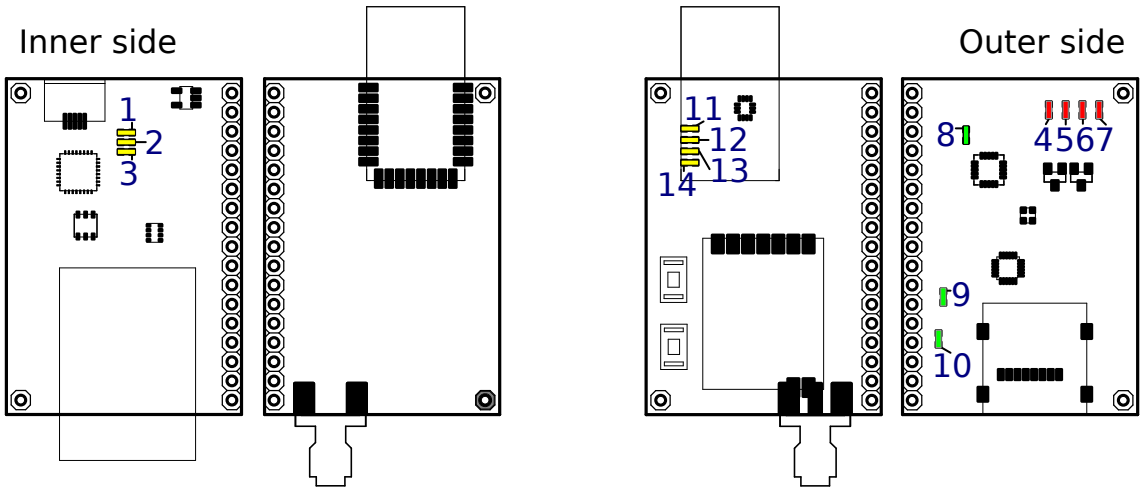



Figure 6: SensorBoard LED meanings

1. **LED_CBUS3**: (yellow) FTDI LED [16], USB power indication

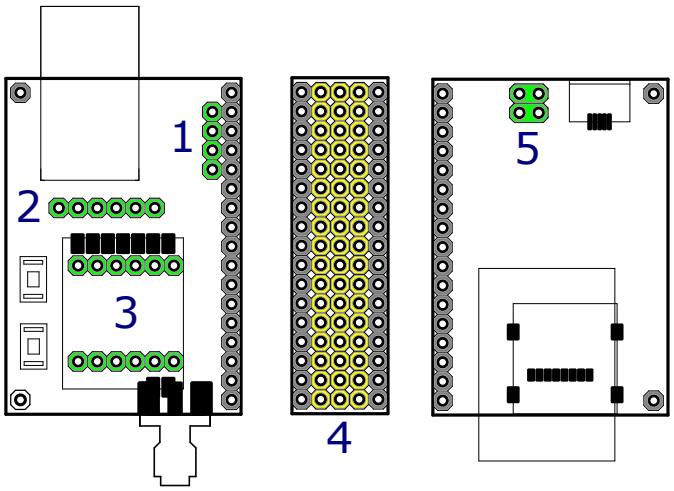
2. **LED_CBUS2**: (yellow) FTDI LED [16], USB connection indication
3. **LED_CBUS4**: (yellow) FTDI LED [16], USB data transfer indication
4. **LED_5V**: (red) 5 V power LED
5. **LED_3V3**: (red) 3.3 V power LED
6. **LED_VCCIO**: (red) VCCIO power LED
7. **LED_USB**: (red) USB power LED
8. **LED6**: (green) Charging the battery indication
9. **LED4**: (green) Software configurable LED
10. **LED5**: (green) Software configurable LED
11. **LED7**: (yellow) DWM1000 TXLED [12]
12. **LED3**: (yellow) DWM1000 RXLED [12]
13. **LED2**: (yellow) DWM1000 SFDLED [12]
14. **LED1**: (yellow) DWM1000 RXOKLED [12]

## Internal Connections

The prototype of the SensorBoard contains several internal ports a lot of SMD pads. The pads are designed mainly for testing. The connection of each pad can be found in the schematics and in the PCB layout. The internal ports are dedicated for connection of internal devices that are not mounted on the board. The internal devices are battery, BMF055 [21] extension board and HM-TRP [23] radio.

The figure 7 and the table 5 describe all the internal ports. The yellow ports are designed for external connections and the grey ports are used for mechanical assembling. Only green ports are internal.

Figure 7: Internal (green) and external (yellow) ports on the SensorBoard, the grey pins are used for mechanical assembling



| Internal and external ports on the SensorBoard | | |
|---|---|---|
| Number | Connection | Description |
| 1 | Internal | BMF055 board connector (UART) |
| 2 | Internal | HM-TRP 433/868 MHz radio connector |
| 3 | Internal | GY-953 connector |
| 4 | External | External pins |
| 5 | Internal | Battery connector |

Table 5: Internal and external ports on the SensorBoard

### Connector for BMF055 board (UART)

The BMF055 connector is a 3.3 V UART connector. Its pinout is presented in figure 8 and table 6.

Figure 8: Connector for BMF055 board (UART)

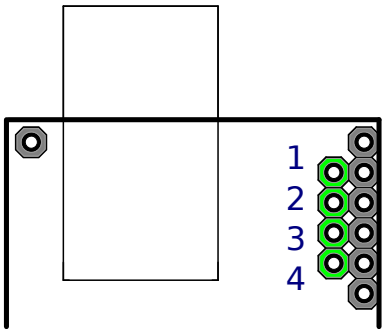



Table 6: Connector for BMF055 board (UART)

| Connector for BMF055 board (UART) | | |
|---|---|---|
| Number | Name | Description |
| 1 | GND | Ground |
| 2 | +3V3 | 3.3 V output |
| 3 | RX | Receive pin (UART) |
| 4 | TX | Transmit pin (UART) |

### HM-TRP radio connector

Pinout of the HM-TRP radio connector is presented in figure 9 and table 7. The HM-TRP [23] radio is able to transmit or receive data at 433 MHz or 868 MHz frequency.

Figure 9: HM-TRP radio connector

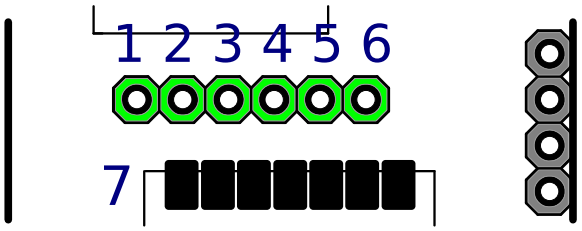



Table 7: HM-TRP radio connector

| HM-TRP radio connector | | |
|---|---|---|
| Number | Name | Description |
| 1 | GND | Ground |
| 2 | CONFIG | Config pin, LOW for configuration mode, HIGH for communication [23] |
| 3 | +5V | 5 V output |
| 4 | RX | Receive pin |
| 5 | TX | Transmit pin |
| 6 | ENABLE | Enable pin, LOW for normal mode, HIGH for sleep mode |
| 7 | SMD port | Port for SMD version of the HM-TRP radio [23], pinout in HM-TRP datasheet |

### GY-953 connector

Pinout of the GY-953 connector is presented in figure 10 and table 8.

Figure 10: GY-953 connector

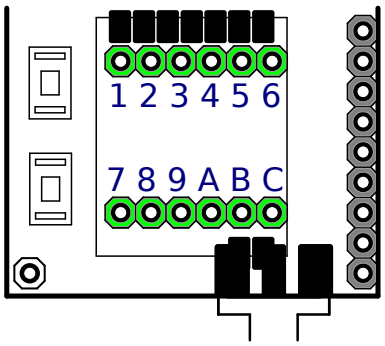


Table 8: GY-953 connector

| GY-953 connector | | |
|---|---|---|
| Number | Name | Description |
| 1 | SWC | Serial Wire Clock (SWD interface), not connected |
| 2 | SWD | Serial Wire Debug (SWD interface), not connected |
| 3 | RX | Receive (UART) |
| 4 | TX | Transmit (UART) |
| 5 | GND | Ground |
| 6 | +5V | 5 V output |
| 7 | B0 | Not connected |
| 8 | INT | Interrupt |
| 9 | MOSI | SPI data out |
| A | MISO | SPI data in |
| B | SCK | SPI clock |
| C | CS | SPI chip select |

### External pins

The external port is discussed in section 5.

### Battery connector

Pinout of the battery connector is presented in figure 11 and table 9.

Figure 11: Battery connector

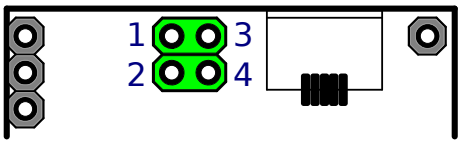



Table 9: Battery connector

| Battery connector | | |
|---|---|---|
| Number | Name | Description |
| 1 | SWITCH_A | ON/OFF switch (internally connected to BATT+) |
| 2 | SWITCH_B | ON/OFF switch |
| 3 | BATT+ | Battery positive (internally connected to SWITCH_A) |
| 4 | BATT- | Battery negative (Ground) |

## BMF055 Extension Board

The BMF055 extension board contains the BMF055 chip [21] with its mandatory mainly Resistor (R), Inductor (L), Capacitor (C) (RLC) accessories. The BMF055 chip was not placed directly onto the SensorBoard because it was very new on the market. I decided to use this chip, but to

place it outside the main board. When any problem with the BMF055 chip occurs, I can simply disconnect this extension board from the main electronics.

### Pin Connection

The pinout of the BMF055 extension board is shown in figure 12 and explained in table 10.

Figure 12: BMF055 extension board pinout

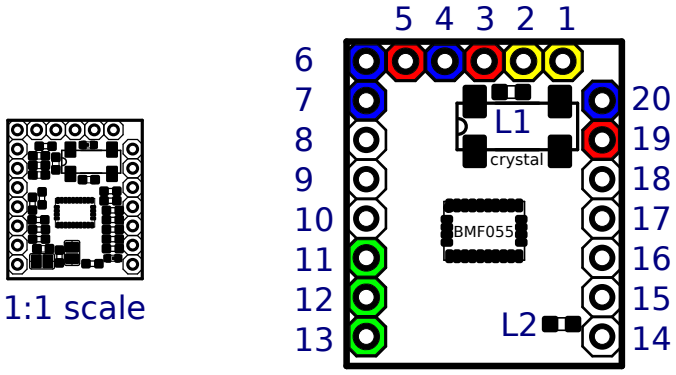



Table 10: BMF055 extension board pinout

| BMF055 extension board pinout | | |
|---|---|---|
| Number | Name | Description |
| 1 | RX | UART receive pin |
| 2 | TX | UART transmit pin |
| 3 | VCC | 3.3 V input/output supply |
| 4 | GND | Ground |
| 5 | VCC | 3.3 V input/output supply |
| 6 | GND | Ground |
| 7 | GND | Ground |
| 8 | PB02 | Atmel SAM d20 GPIO (PB02) [42] |
| 9 | PB01 | Atmel SAM d20 GPIO (PB01) [42] |
| 10 | PB00 | Atmel SAM d20 GPIO (PB00) [42] |
| 11 | SWD | SWD interface, data pin |
| 12 | SWC | SWD interface, clock pin |
| 13 | $\overline{RESET}$ | Reset when LOW |
| 14 | PA28 | Atmel SAM d20 GPIO (PA28) [42] |
| 15 | PA23 | Atmel SAM d20 GPIO (PA23) [42] |
| 16 | PA22 | Atmel SAM d20 GPIO (PA22) [42] |
| 17 | PA21 | Atmel SAM d20 GPIO (PA21) [42] |
| 18 | PA20 | Atmel SAM d20 GPIO (PA20) [42] |
| 19 | VCC | 3.3 V input/output supply |
| 20 | GND | Ground |

### LED Meanings

The BMF055 board has two LEDs, one power LED and one software driven LED. The positions of the LEDs are shown in figure 12 and marked as L1 and L2. Their meaning is explained in table 11.

Table 11: BMF055 extension board LED meaning

| BMF055 extension board LED meaning | | |
|---|---|---|
| Number | Name | Description |
| L1 | Power LED | (red) The LED is ON if and only if the power is applied |
| L2 | Software LED | (red) The LED is fully driven by firmware inside the BMF055 |

# SCHEMATICS AND PCB LAYOUT OF THE SENSORBOARD

## Schematics

The schematics and the board layout was created in CadSoft EAGLE [34]. The figures 13, 14, 15 and 16 show the exported schematics split into four sheets.

## Board layout

The exported PCB layouts in scale 1:1 are in figure 17 for the top layer and in figure 18 for bottom layer. The same layouts with printed parts in scale 2:1 are in figure 19 for top layer and in figure 20 for bottom layer.

## Template for soldering paste

The scaled template for soldering paste is dosplayed in figure 21.

## SensorBoard Drawings

The SensorBoard drawings shows the correspondence between the manufactured prototype and its schematic drawings. The comparison is shown in figure 22.

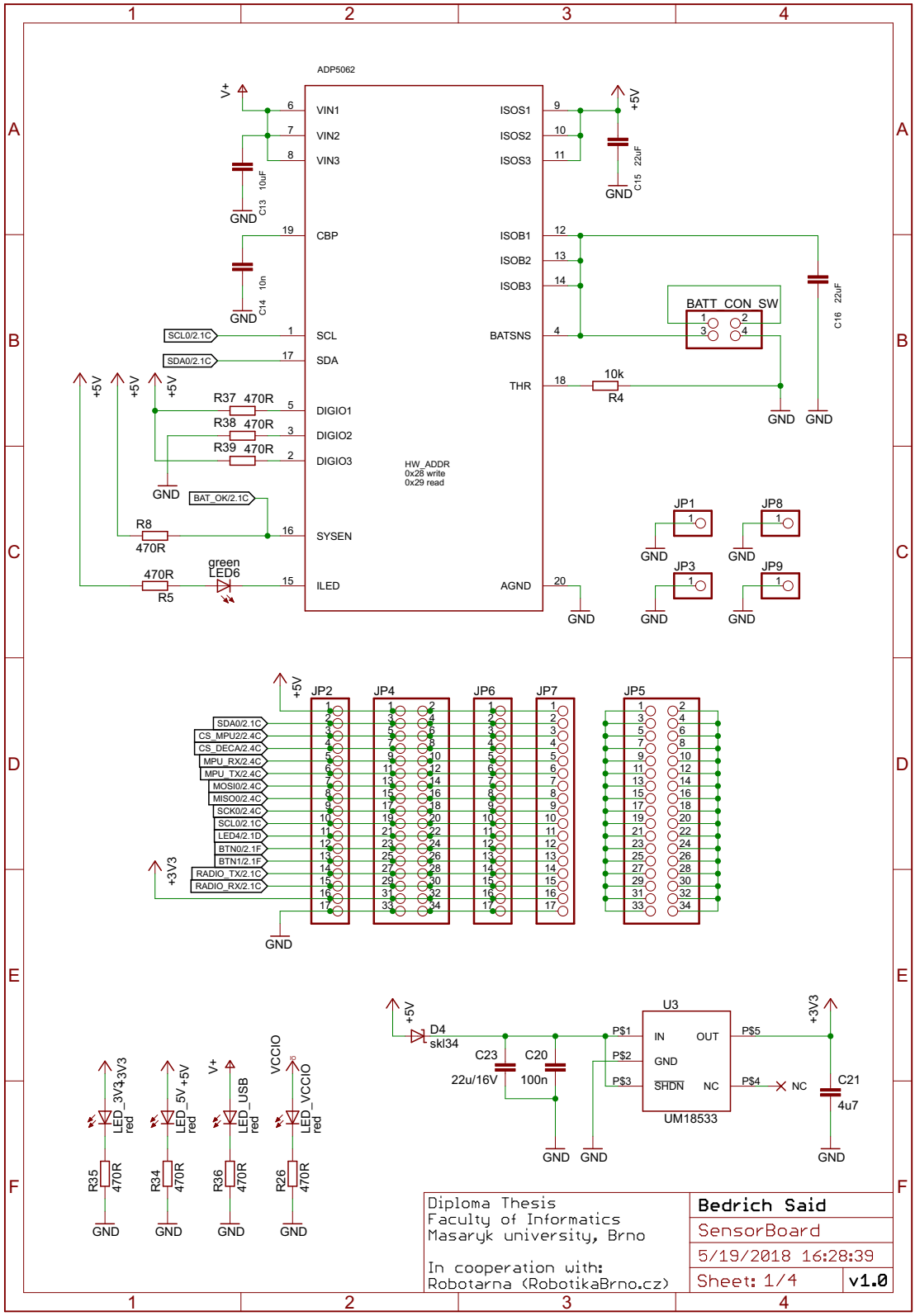



Figure 13: Schematics of the SensorBoard sheet 1

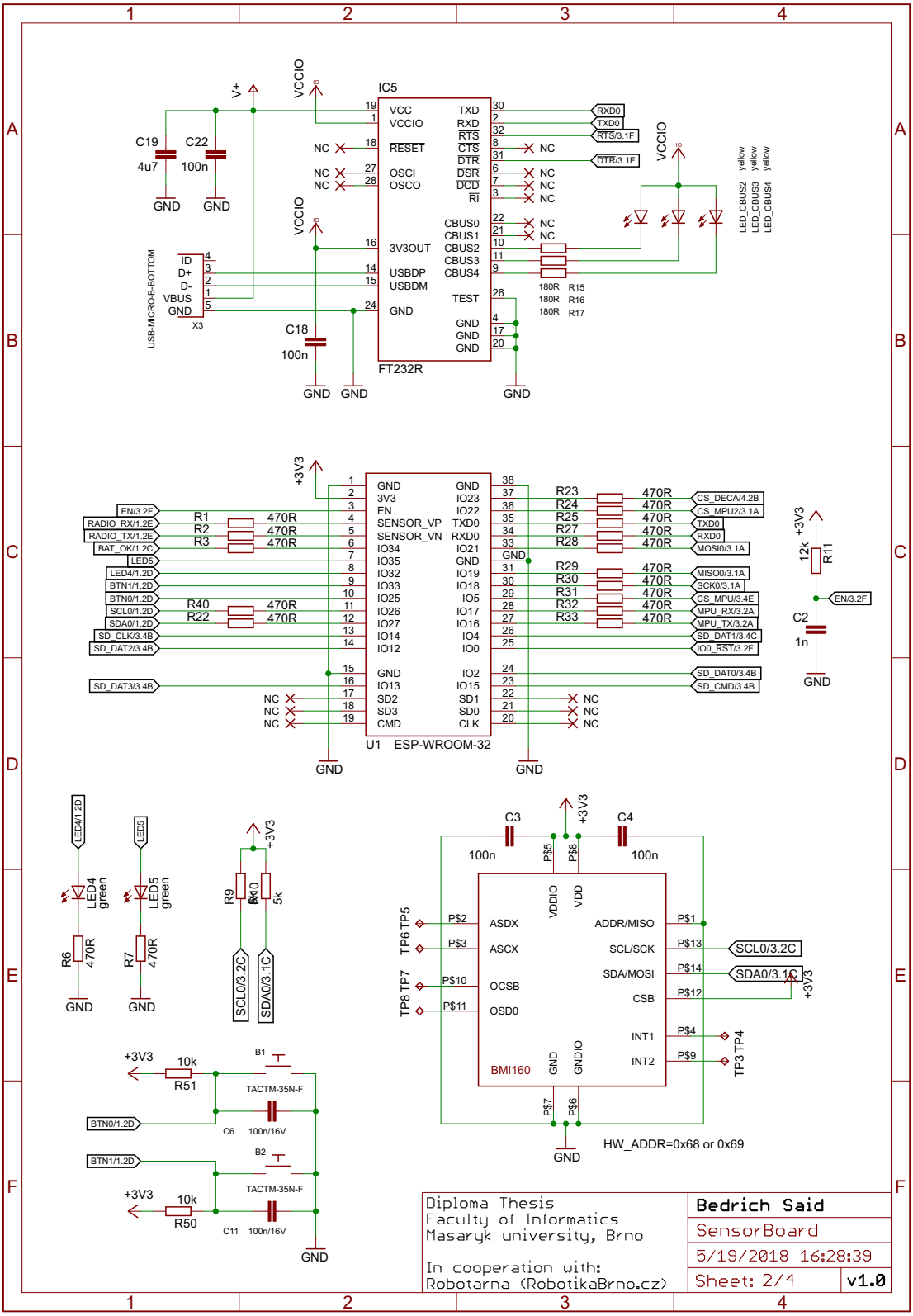



Figure 14: Schematics of the SensorBoard sheet 2

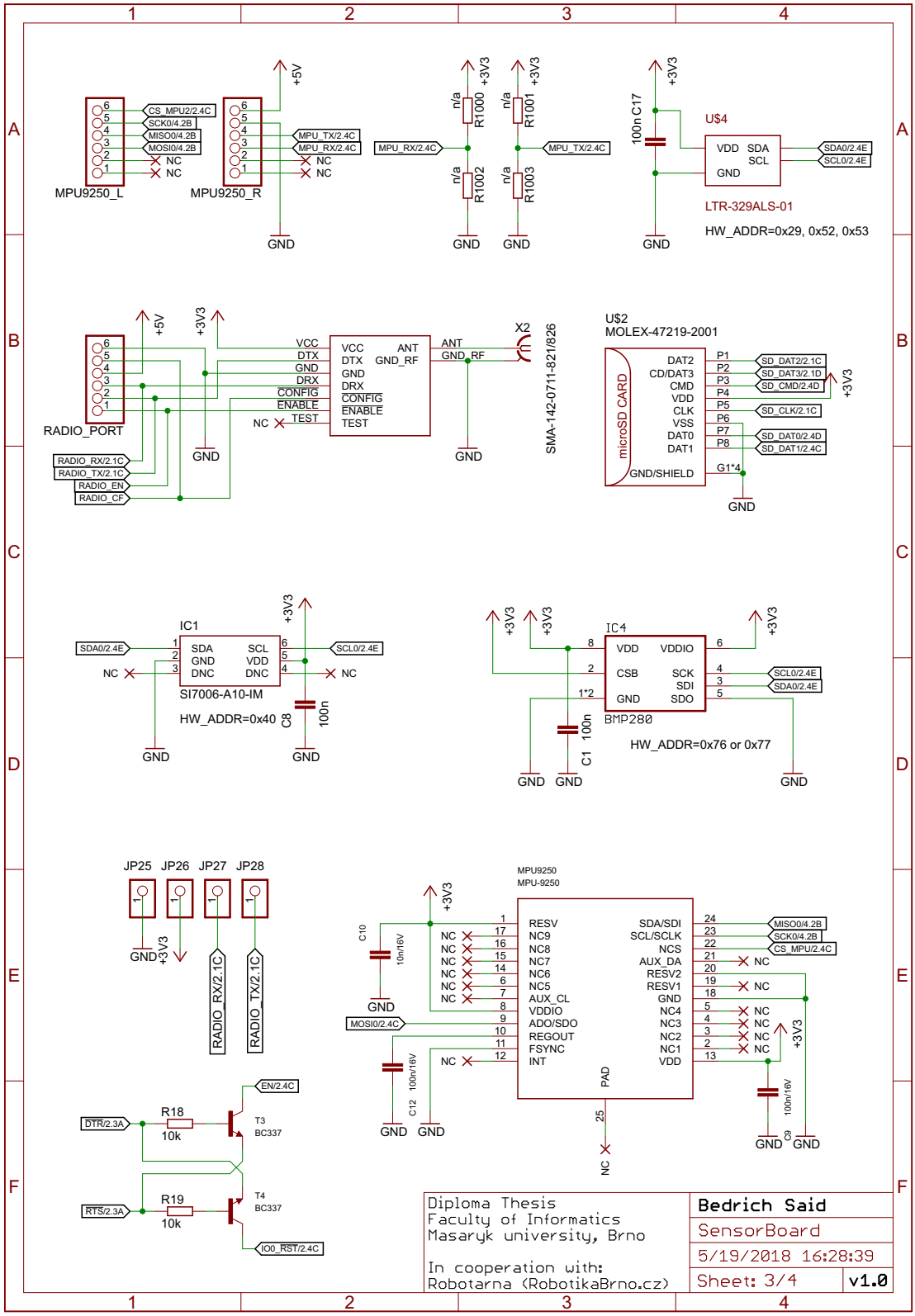



Figure 15: Schematics of the SensorBoard sheet 3

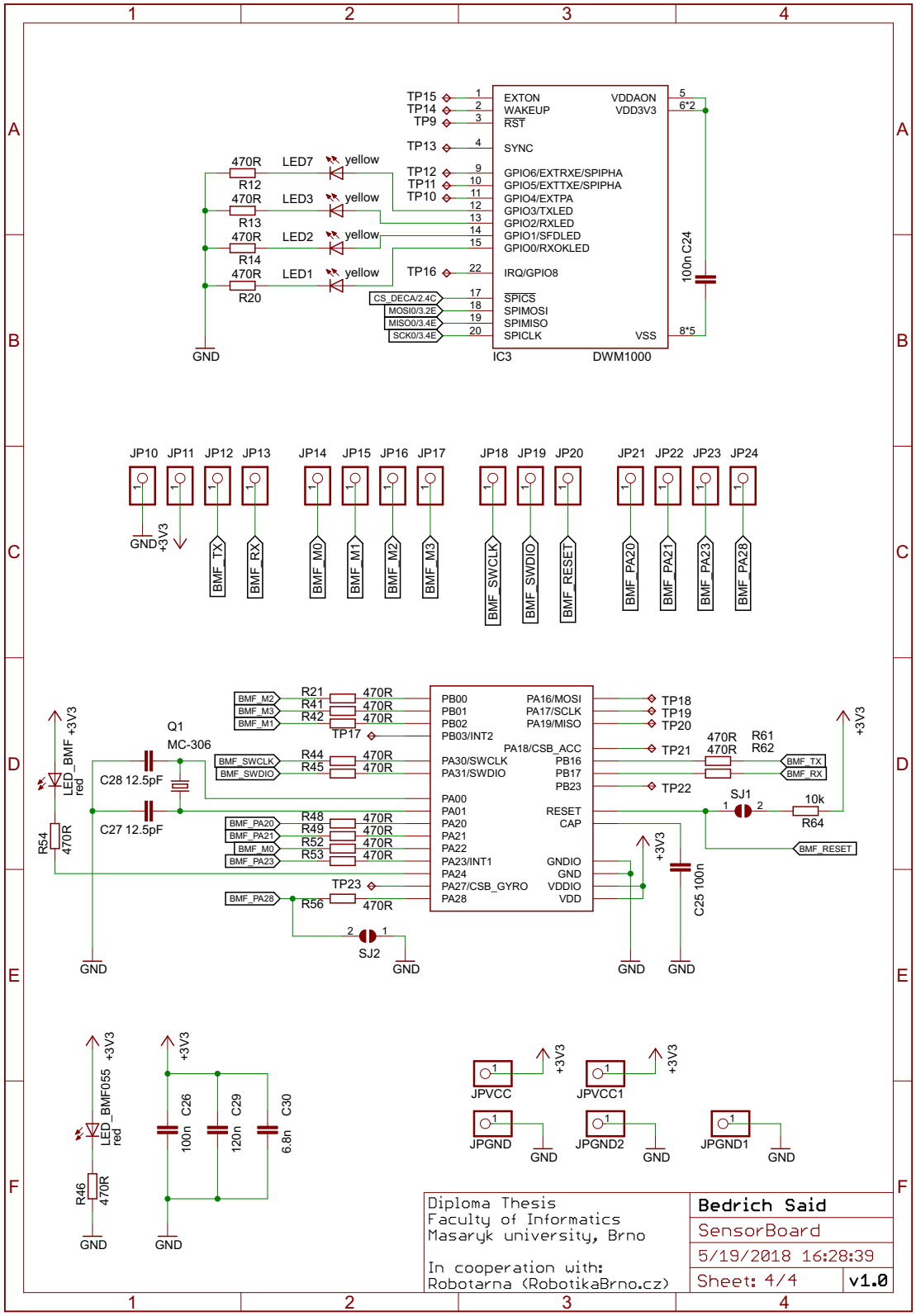



Figure 16: Schematics of the SensorBoard sheet 4

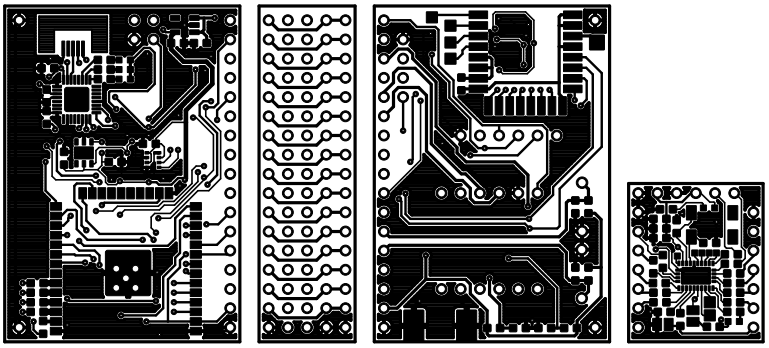


Figure 17: SensorBoard layout in scale 1:1 (top layer)

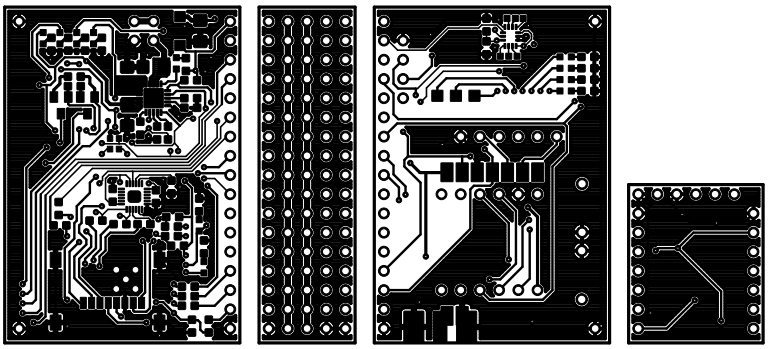


Figure 18: SensorBoard layout in scale 1:1 (bottom layer)

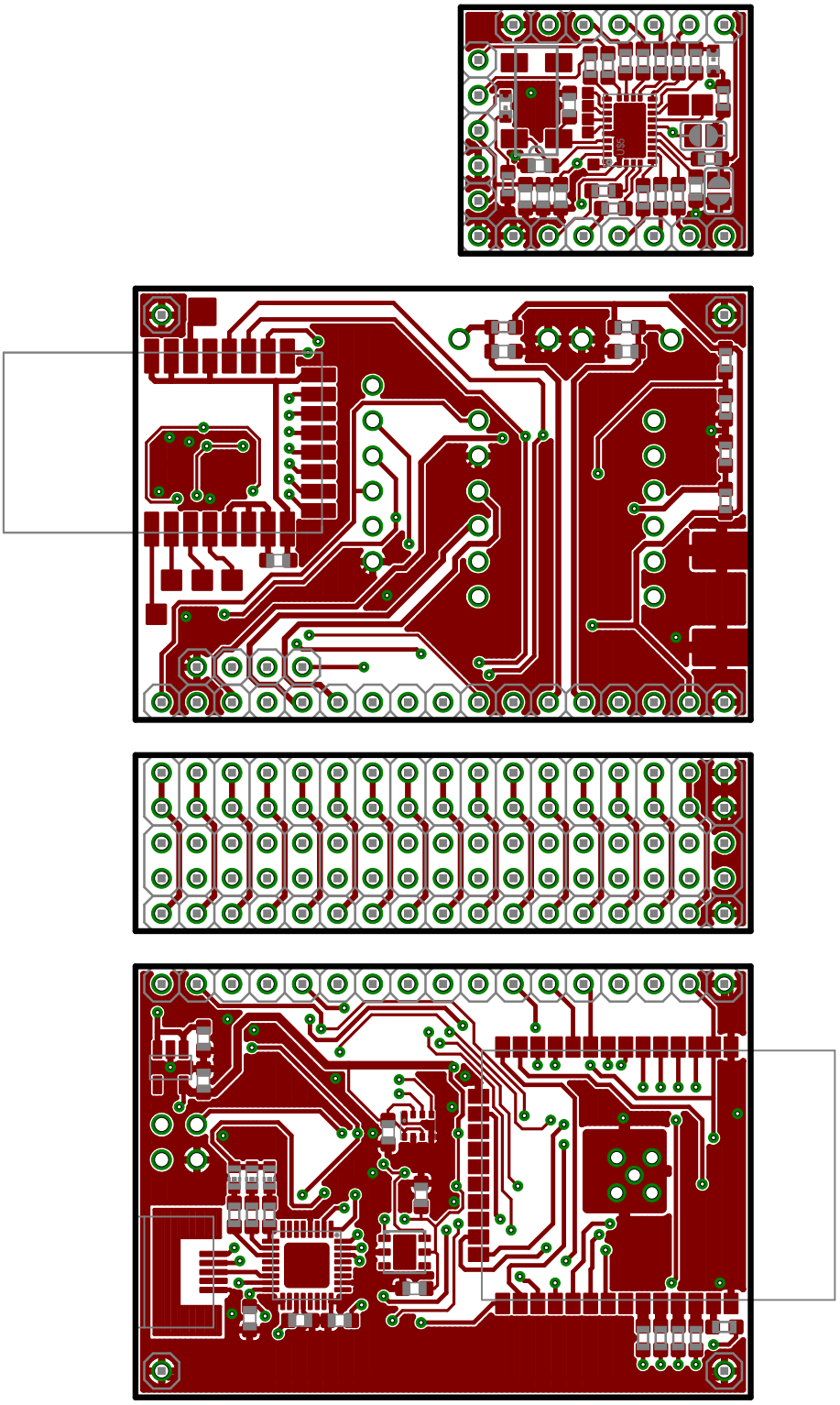


Figure 19: SensorBoard layout in detail scale 2:1 with parts (top layer)

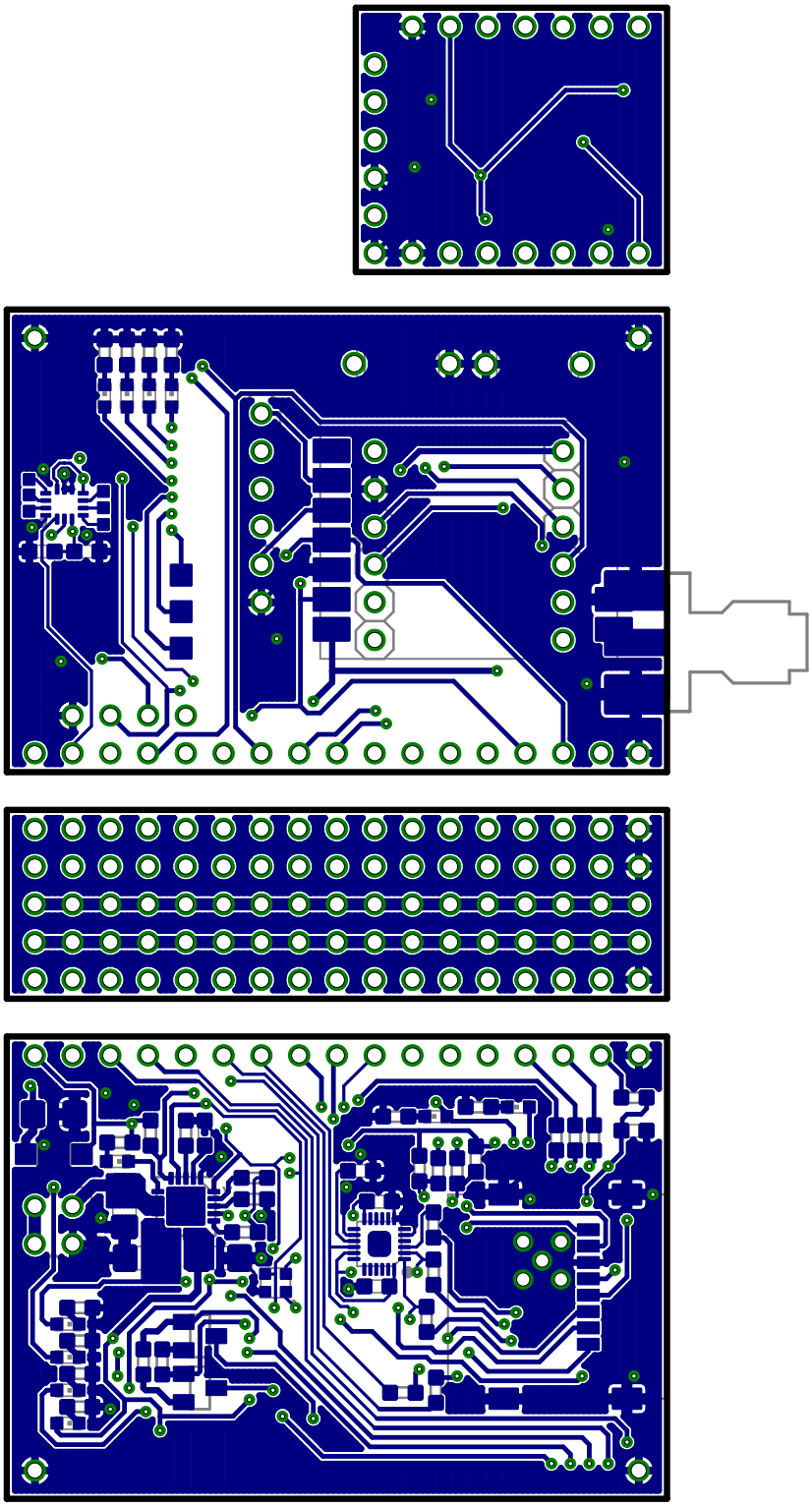


Figure 20: SensorBoard layout in detail scale 2:1 with parts (bottom layer)

Figure 21: Template for soldering paste (resized)

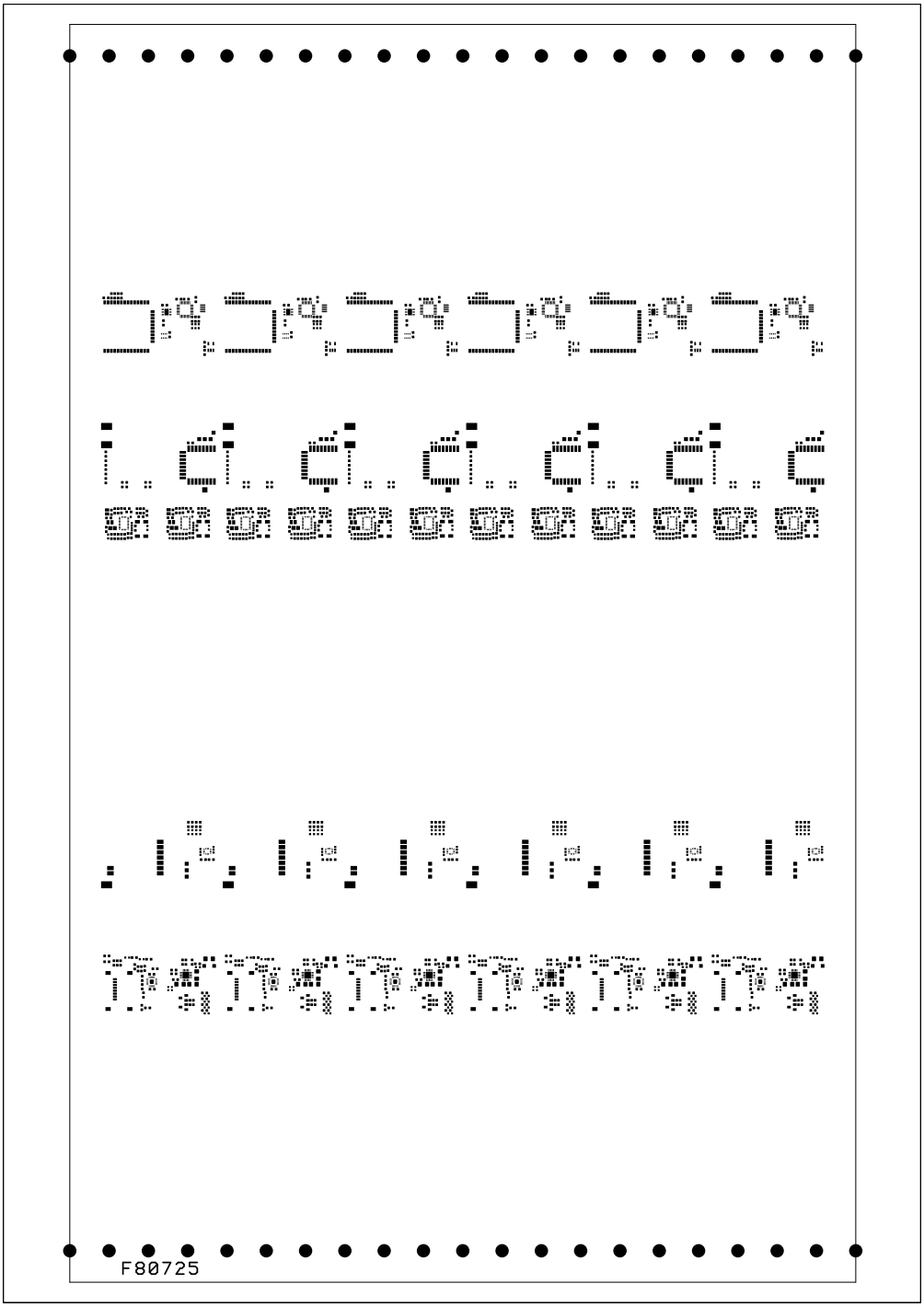


Figure 22: Drawings of the SensorBoard

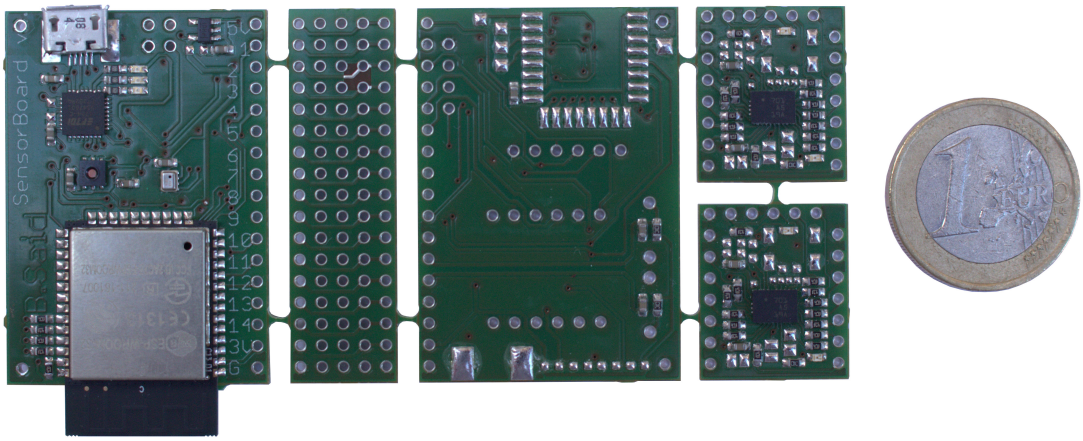
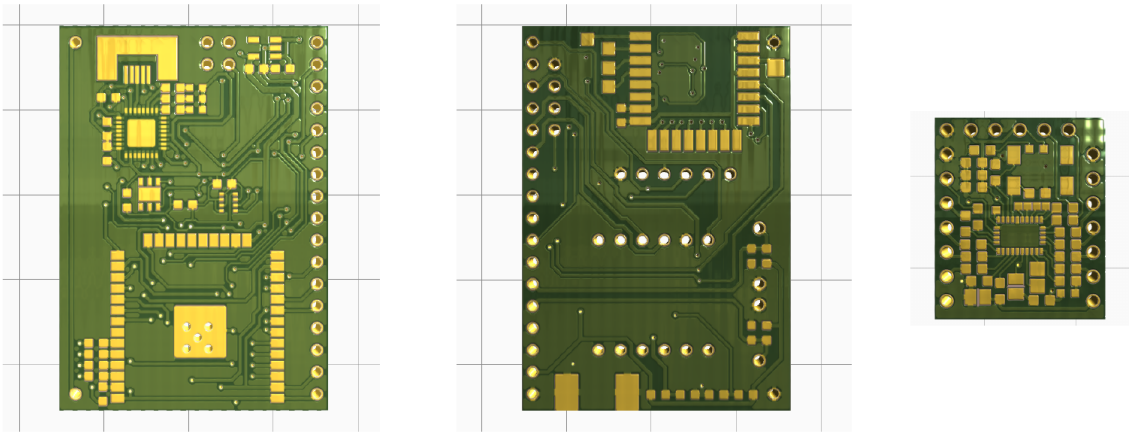
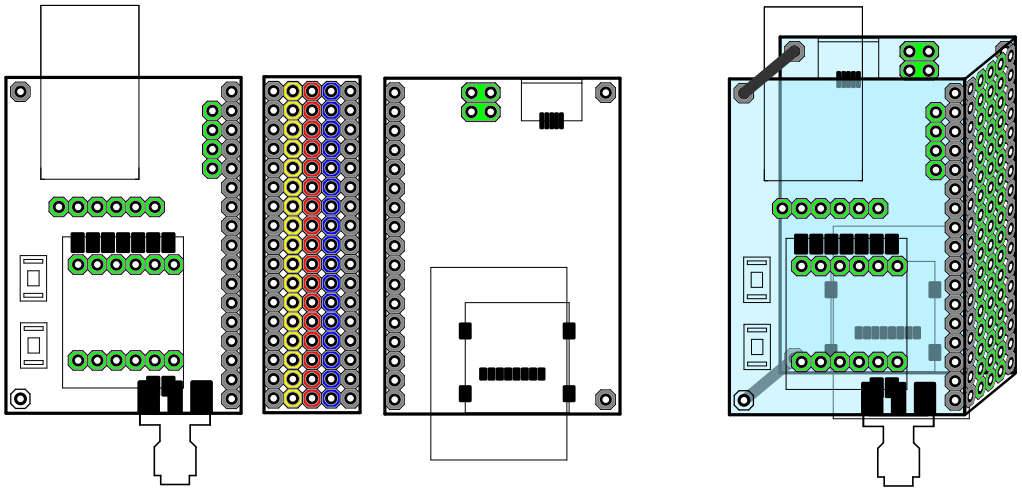


Photo of the disassembled prototype



Renders of the PCB



Schematical drawing