Final Degree Report

Universitat de les Illes Balears

escola politècnica superior

DEGREE

Title

BEDŘICH SAID

Tutor
Antoni Jaume i Capó

Supervisor
Supervisor Name

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, May 01, 2018

CONTENTS

ACRONYMS

ABSTRACT

The attitude sensors are more and more widely used and their price is decreasing. We can find them in many electronic systems designed for specific purposes. Examples are in wearable devices, navigation or control systems. We can use some of the devices for covering new use cases and developing new systems. I still didn't find any device that fulfills all my requirements for development new data processing algorithms.

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 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 selected a task about movement analysis of a horse. The devices were placed on the horse's body as wearable devices and I was developing algorithms for determination of the type of its movement – stand, walk, trot or gallop. The algorithm first determines very basic types of movements and then it's looking for more advanced ones based on previous results. The types of movement are defined as statements, so new types of movement of any subject can be detected without modification of the computation code.

In general, the developed device can be used for sensor data capturing, on-board data processing, navigation or control of moving mechanics. The device works independently, so it is easy to install on the measured subjects.

1

INTRODUCTION

//todo

HARDWARE

The hardware design is split into several steps in this paper. First, I had to decide if it's 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 don't find any suitable device on the market, we can start to think about creating a new one. In my opinion it's not a good way to start any development before doing the previous steps.

When we already start a development of a new hardware we can think about adding some additional requirements for increasing the versatility of the final hardware. Of course, the new requirements mustn't rapidly increase the final prize. But if the hardware will be used only in the specific task, it's usually unnecessary to add any other functionality. In this case I'm 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'm decided to develop a 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's time for testing. The tests should find as many errors as possible and they should proof us if the requirements were already met. The section 2.7. The whole process is shown in the figure 2.

Figure 2.1: Flowchart of the hardware design process
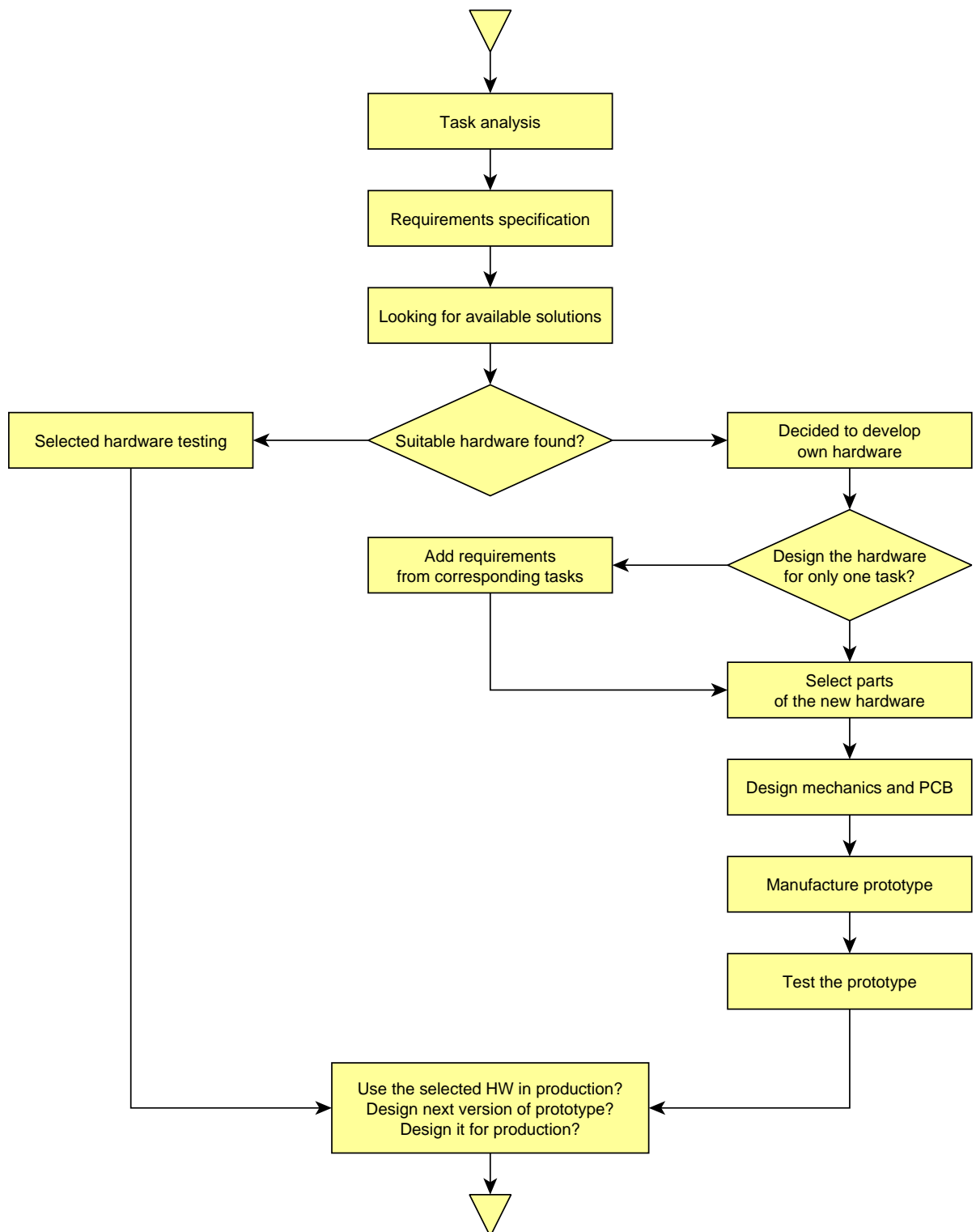
2.1 Task Analysis

When we define our complete task in the first step, then 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.

> **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 of the movement and naming the specific categories of movements

The subject is an animal – horse in this paper – 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 logged data. For example the categories of movement of a horse are: stand, walk, trot, gallop, . . .

2.2 Requirements

The development of a new hardware or software is usually driven by specified requirements. In this paper 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.

2.2.1 High level requirements

Measuring of the movement of a subject: The solution should work outdoor. So, we cannot take into account any studio or laboratory based technologies like Motion Capture (Mo-cap). On the other hand we can replace the passive or active markers by 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 in outdoor use, so we can log the data to internal memory or transmit them via any wireless technology. The wireless systems may be not fully reliable in complex areas with many objects. So, when we want to transmit the data directly it's 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's not very important during the hardware design. But this requirement defines what data we need to measure and this is a hardware requirement.

For the movement reconstruction we need primarily data about position and attitude of every sensor. Some methods for movement classification don't 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 what 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 according to animals.

Finally, I would like to add as many interesting sensors as possible, because it will give us more data sources and we are less limited by the data sources. The other advantage is multifunctionality of the hardware. On the other hand these additional sensors should be added only into additional requirements with lower importance. Otherwise the selection of an appropriate hardware (based on the requirements) will be manipulated by number of probably unnecessary sensors.

2.2.2 Low level requirements

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

2.2.3 Additional requirements

The additional requirements are shown in table 2.2.3.

2.3 Available solutions

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

2.4 Selection of devices

//todo

2.5 PCB design

The PCB design should meet both electrical and mechanical requirements. It's 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 for production use.

Here in this process I will merge the first and the second version together. I will create a larger device which is still usable as wearable device. This implies that I can do the software

Table 2.1: Sensor Board 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	Comment
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: Sensor Board low level requirements 2

Low level requirements 2			
Category	Requirement	Importance	Comment
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: Sensor Board additional requirements 2

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

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

development, laboratory tests and the first outdoor tests with the same board designed during the first iteration. I'm 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) [?] during this process. I've finally chosen this editor because it has the availability of the libraries for the devices I wanted to use. I would probably use KiCad [?] if there was the same availability of the libraries.

2.5.1 Schematics and board layout

The schematics and the board layout was created in CadSoft EAGLE [?]. The figures 2.5.1 and 2.5.1 shows the exported schematics split into two sheets. The exported PCB layout drawing in scale 1:1 is in figure 2.5.1 and in scale 2:1 with more details in figure 2.5.1.

2.5.2 Mechanical layout and connectors

The mechanical layout is shown in figure 2.5.2.

2.6 Manufacturing

//todo: pridat vsude fotky jednotlivych fazi

The manufacturing process of the prototype consists of these steps:

1. Collecting source data – schematics and board layout (in this paper 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 (SMT)

   a) Export data for template for applying solder paste

   b) Export partlist – list of all devices with their coordinates

   c) Export bill of materials (BOM)

   d) Manufacturing of the template for applying solder paste

   e) Order all devices according to BOM

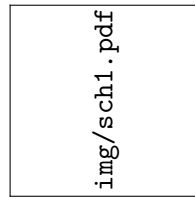   f) Sending template for applying solder paste, packages with devices and partlist to the manufacturer
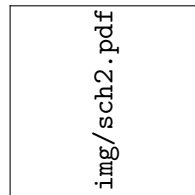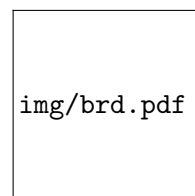
img/sch1.pdf
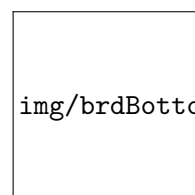
Figure 2.2: Schematics of the Sensor Board sheet 1

img/sch2.pdf

Figure 2.3: Schematics of the Sensor Board sheet 2

img/brd.pdf

Top and Bottom layer

img/brdTop.pdf

Only Top layer

img/brdBottom.pdf
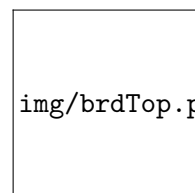
Only Bottom layer

Figure 2.4: Sensor Board layout in scale 1:1

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

2.6.1 Collecting source data

The source data may be in various formats according to the used tools. I used CadSoft EAGLE [?] during the development, but many other tools are available on the market. It depends on the manufacturer company if they accept the source data in our format. They usually accept the source data, but they usually apply an exporting fee. All PCB editors should be able to export the manufacturing data in standardized formats. (Sometimes it's 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 by machine we should add a calibration marks on the final panel. I have followed the instructions from SMTplus company [?].

First, I have panelized the seprated boards and I have added the calibration marks according to the rules of SMTPlus company [?]. Finally, I have exported the gerber files using a CAM job [?] in Eagle. I have followed the design rules for class 3 by Gatema a.s. company [?].

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 creating 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 my small SMDs I used $100\,\mu m$, but I recommend to discuss the parameters with the manufacturer.

The BOM and partlist 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 partlist and then import it on the seller's website. I recommend to buy 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 have to recommend to do this job by ourself. It allows us to perform some tests before packaging and we don't have to spend time and money with preparation of the documentation about how to assemble and package our prototype. Otherwise the different software developers and testers can receive the different parts of the prototype for their next work.

Table 2.4: Available solutions

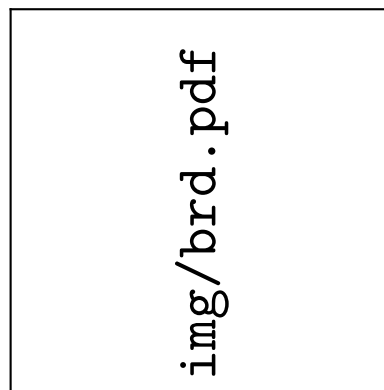| Device | Datasheet | Price | Number of unfullfilled requirements | | |
			Low	Medium	High
MetaWear ArduPilot 2.6 X-IMU wearable...					

img/brd.pdf

Figure 2.5: Sensor Board layout in detail scale 2:1

Figure 2.6: Sensor Board dimensions

img/HWdimensions.pdf
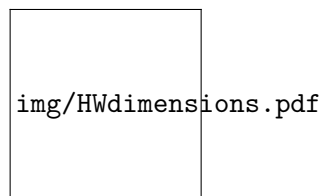
Figure 2.7: Template for soldering paste in scale 1:2

img/smtPasteTemplate.pdf
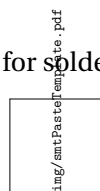
2.7 Testing

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 in section 2.7.1. I did another tests against the specification of used devices. The results show us what devices should be used in the next version of the hardware and what devices are redundant. The results are shown in the section 2.7.2. The outdoor tests didn't find any significant errors, but there were found several recommendations for the future versions of the Sensor Board. These recommendations are mentioned in section 2.7.3.

2.7.1 Errata

The table 2.5 shows all errors I have found during software development and laboratory testing of the Sensor Board prototype.

2.7.2 Device testing

The device testing may select the devices used in the future versions and remove all spare parts from the prototype. The table 2.6 shows the test results and the table 2.7 shows the devices that I recommend to use in future versions.

2.7.3 Recommendations for the next version

Recommendations for the future versions of the Sensor Board based on the outdoor testing:

– Replace software LEDs by one or more RGB LEDs. For example WS2812 RGB LED [?].

– Move the sensor fusion and computations to Atmel SAM D21 [?] processor inside the BMF055 [?] chip. Keep the ESP32 [?] processor only for communication and user computations.

– Decrease board dimensions using 4 layer PCB. The increased cost for 4 layer board may be saved on smaller surface.

– Sometimes it is better to have all SMDs at 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 for 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. The USB on board actually supports only UART communication on virtual COM port. The File Transfer protocol for the SD card should be supported, too. (If it is needed we can add full USB host support or USB-C compatibility.)

Table 2.5: Errata of the Sensor Board

Severities	
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

Number	Description	Error created during	Image	Severity
1	Bridge between boards under micro USB connector What happens: 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 What happens: 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 What happens: The pins IO34 – IO39 are input only, so they cannot drive a LED	schematics design		Spare
4	Missing pull-up resistors on SD card What happens: The SDIO interface needs external pull-up resistors to work properly. I have added these resistors later by hand.	schematics design		Medium
5	The temperature measurement is placed very close to the main processor What happens: The processor heating affects the measured temperature	board layout		Spare
6	Low capacity capacitor placed on power supply What happens: When we use WiFi some brownouts can be detected. I have added a bigger capacitor later by hand.	schematics design		Medium
7	The light sensor is placed on inner side of the board What happens: The measured values are affected by	board layout		Spare

– Add specialized pads for 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 of other parts (sensors). The sensors support sleep modes, but we cannot completely switch them off to save more energy from battery.

//todo: pridat informaci o ALB32-WROVER

2.7.4   Analysis of additional costs

//todo
   //todo: citovat vetsi verzi ESP WROOM 32 a vypsat vyhody

Table 2.6: Device tests

Device	Test	Result
MPU9250	SPI interface	Warning
MPU9250	I2C interface	Pass
//todo	...	...

Table 2.7: Device selection

Decision		Device	Decision	
Selected	recommended for use in future versions	MPU9250	Removed	
Possible	Not necessary, but has use-cases	BMF055	Selected	
Removed	Not recommended or will be spare	//todo	...	...

3

SOFTWARE

The Sensor Board hardware has two programmable CPUs and several configurable chips. The main ESP32 CPU (two low-power Xtensa 32-bit LX6 microprocessors) [?] is programmable via USB or Bluetooth or JTAG. The JTAG connector is not present on the Sensor Board. The second microcontroller is a part of BMF055 [?] multifunctional chip. It is Atmel SAMD20 [?] with ARM Cortex-M0+ CPU programmable via SWD interface [?].

Microcontrollers:

1. Espressif ESP-WROOM-32 [?]:

   - 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 [?]:

   - ARM Cortex-M0+ CPU, 48 MHz, 32 kB SRAM, 256 kB flash memory
   - Designated for processing inertial data and computing sensor fusion, 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 Sensor Board has to be programmed separately via its interface. Only ESP32 [?] (main processor) can be programmed over the air via Bluetooth. This feature is disabled by default. The pinout an other information about the SensorBoard and BMF055 board are described in appendix **??**.

Figure 3.1: Schema of the available modules inside the SensorBoard hardware

`img/SensorBoardSchema.pdf`
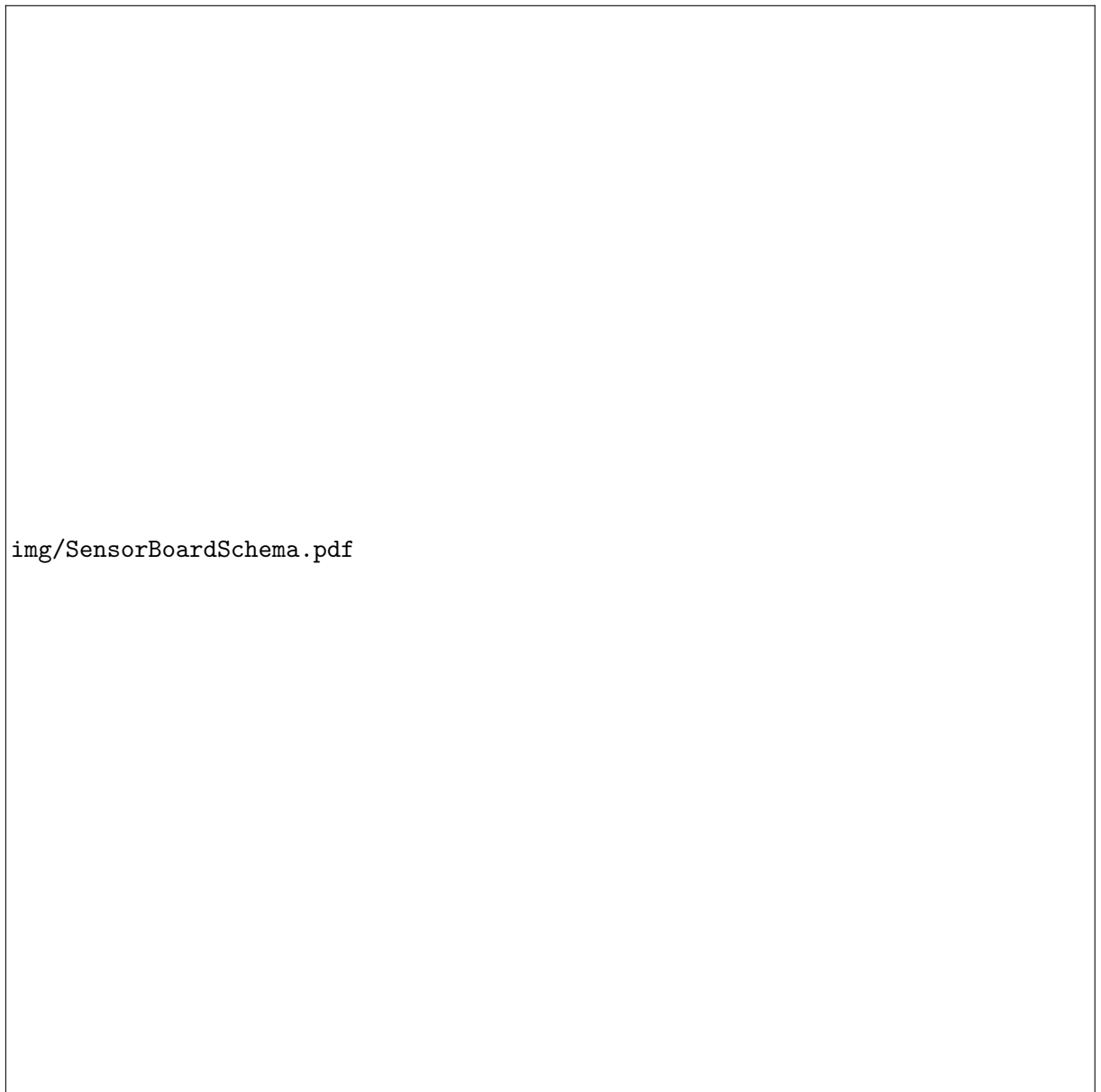
3.1.1 Programming ESP32

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

The chip can be programmed using Arduino compatible framework [?] 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 [?] firmware directly to the ESP32 controller. The Python libraries, scripts and other data are stored on the SD card. The MicroPython firmware supports most of the hardware functionality, but there are still some restrictions.

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

ESP-IDF Framework

The ESP-IDF framework is almost POSIX compatible. [?] It supports many POSIX compatible functions, but it still doesn't 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 [?] to setup the environment and program the board. The user can do everything from terminal (command line). The figure **??** 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 [?]. In my opinion, the compilation of our programs is very slow when we use this option. The figure **??** shows the SensorBoard project opened in Eclipse IDE.

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

Arduino Compatibility

The Arduino compatible framework for ESP32 [?] is dependent on ESP-IDF framework [?]. 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 begineers and for fast prototyping. I don't recommend to use this framework for advanced applications, because it doesn't cover
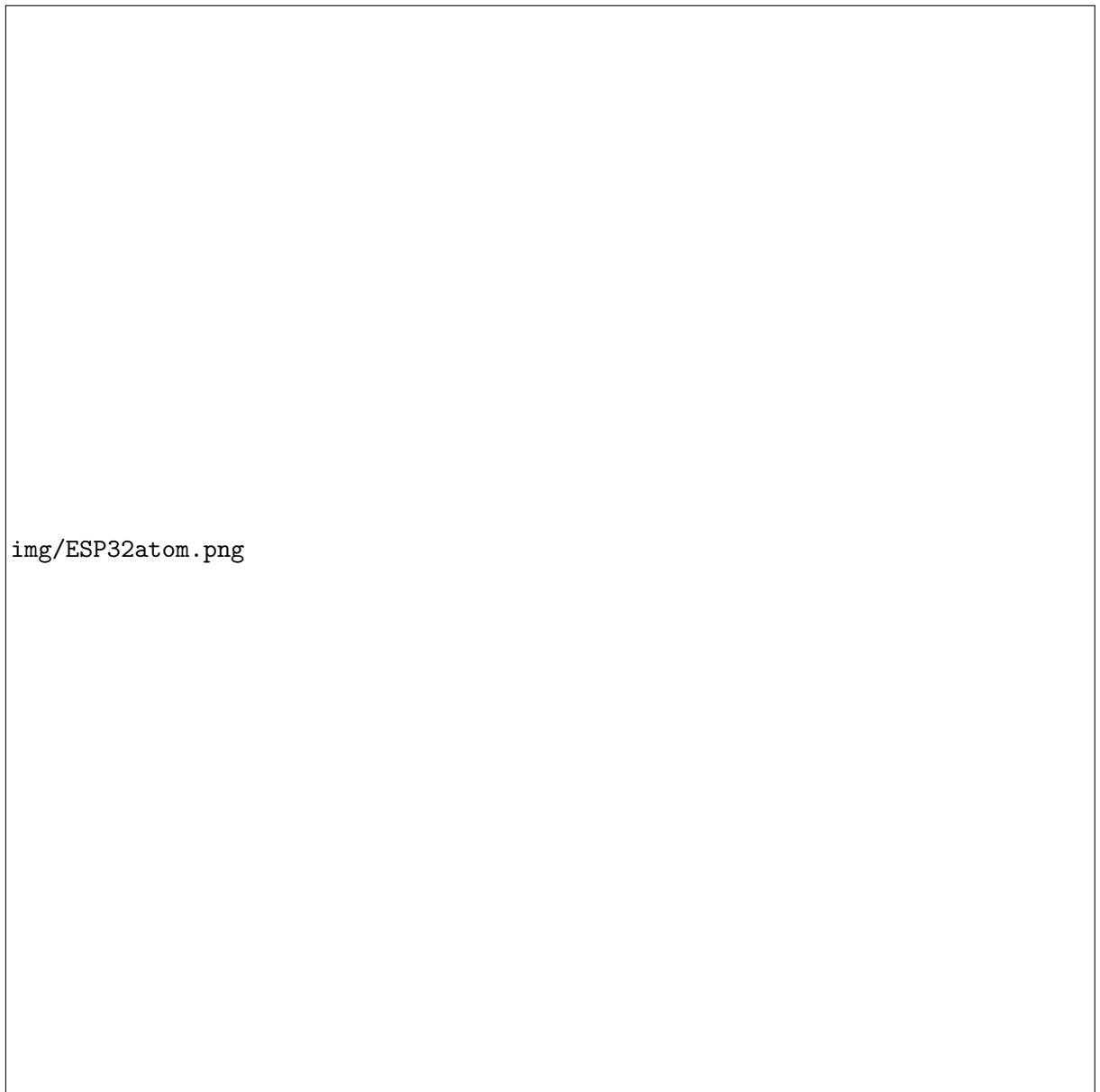
Figure 3.2: Configuration of the ESP-IDF program in terminal before the first compilation

img/ESP32menuconfig.png

Figure 3.3: The SensorBoard project opened in Eclipse IDE

img/ESP32eclipse.png

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

img/ESP32atom.png

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 **??**. All libraries are donwloaded and installed automatically inside the PlatformIO. We have to set only a `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 builded MicroPython binary for ESP32 can be directly downloaded from MicroPython website. [**?**] We can directly flash this binary to our ESP32 controller (to the SensorBoard) and then diretcly 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 separate command via this serial terminal or run the whole 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 **??**.

3.1.2 Programming BMF055

The BMF055 [**?**] 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 separate board which can be optionally mounted to the SensorBoard or it can be used independently. The figure **??** shows the separate 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 [**?**] implemented this single chip autopilot solution for multicopters.

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

3.2 Application Programming Interface

The programming interface for the SensorBoard can be split to 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 **??**.
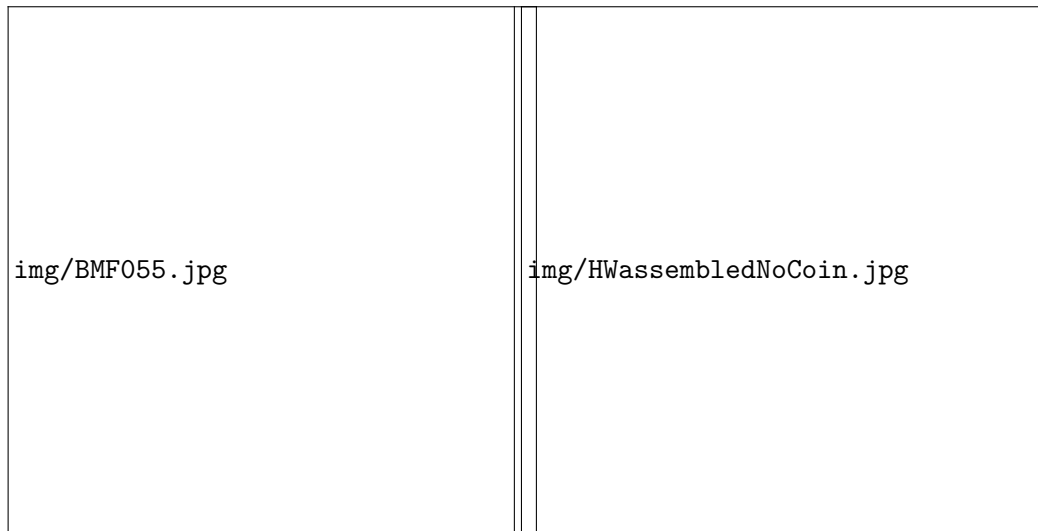
I have implemented a new library for easier working with the sensors on the SensorBoard. This library is described in section **??**.

23

Figure 3.5: The running Python serial terminal on the SensorBoard
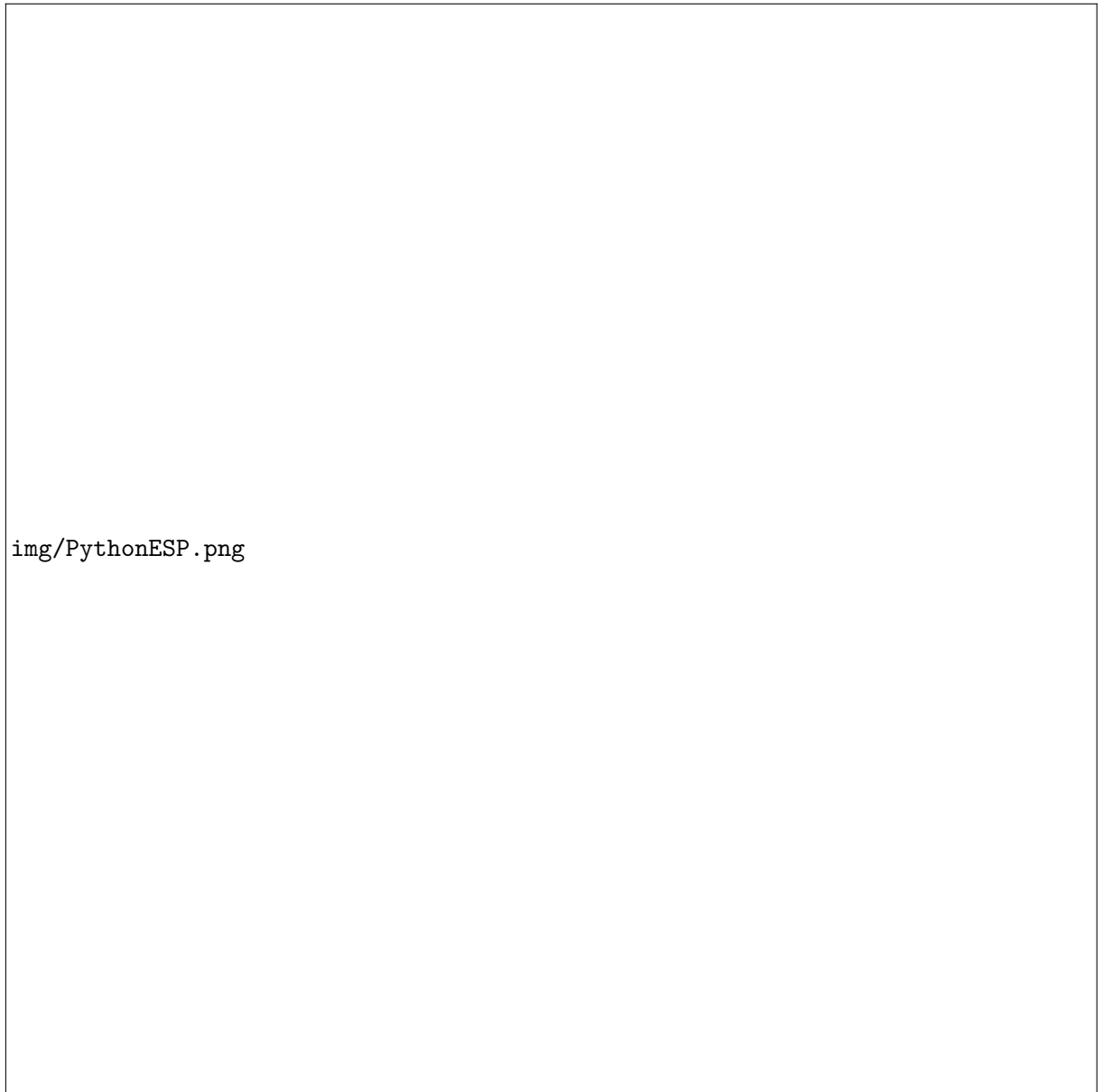
`img/PythonESP.png`

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

3.2.1 SensorBoard API

The ESP32 and Atmel SAM controller have defined their API by the manufacturer, but these APIs don't implement a communication with specific sensors and peripherials connected to the controlers. 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. It allows for example a FreeRTOS usage, because the FreeRTOS is a part of ESP-IDF framework [?]. 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

Figure 3.7: An opened BMF055 project in Atmel Studio 7

img/BMF055AtmelStudio.png

- 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 controlers used on the SensorBoard have an API defined and implemented by the manufacturer.

Atmel SAM D21: [**?**] 2

- 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: [**?**] 2

- 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.

3.3.1 Sensors data logger

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

When we need to measure some inertial data, we sometimes need more sensors placed in different places. Then, it's 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 **??**.

I'm using this advanced configuration during the analysis of a movement of a horse. This solution is explained in detail in section **??**. The goal of this task is to recognize the type of the movement of a horse (stand, walk, trot or gallop). So, we can get some results from only one sensor, but we get better results when we can measure each leg and the body separately. How the SensorBoards are used in this task is shown in figure **??**.

3.3.2 Sensor fusion and AHRS

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

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

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 **??**. 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 flight controllers must be very reliable devices. They usually consists of two parts. The first part controls the pitch, roll, yaw, velocity, 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 as 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

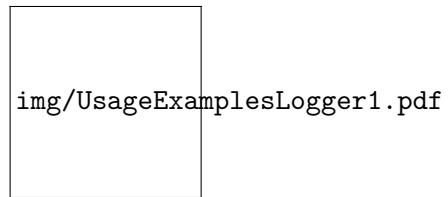Figure 3.8: Simple logging application with SensorBoard

img/UsageExamplesLogger1.pdf

Figure 3.9: Advanced logging application with SensorBoard

img/UsageExamplesLogger2.pdf
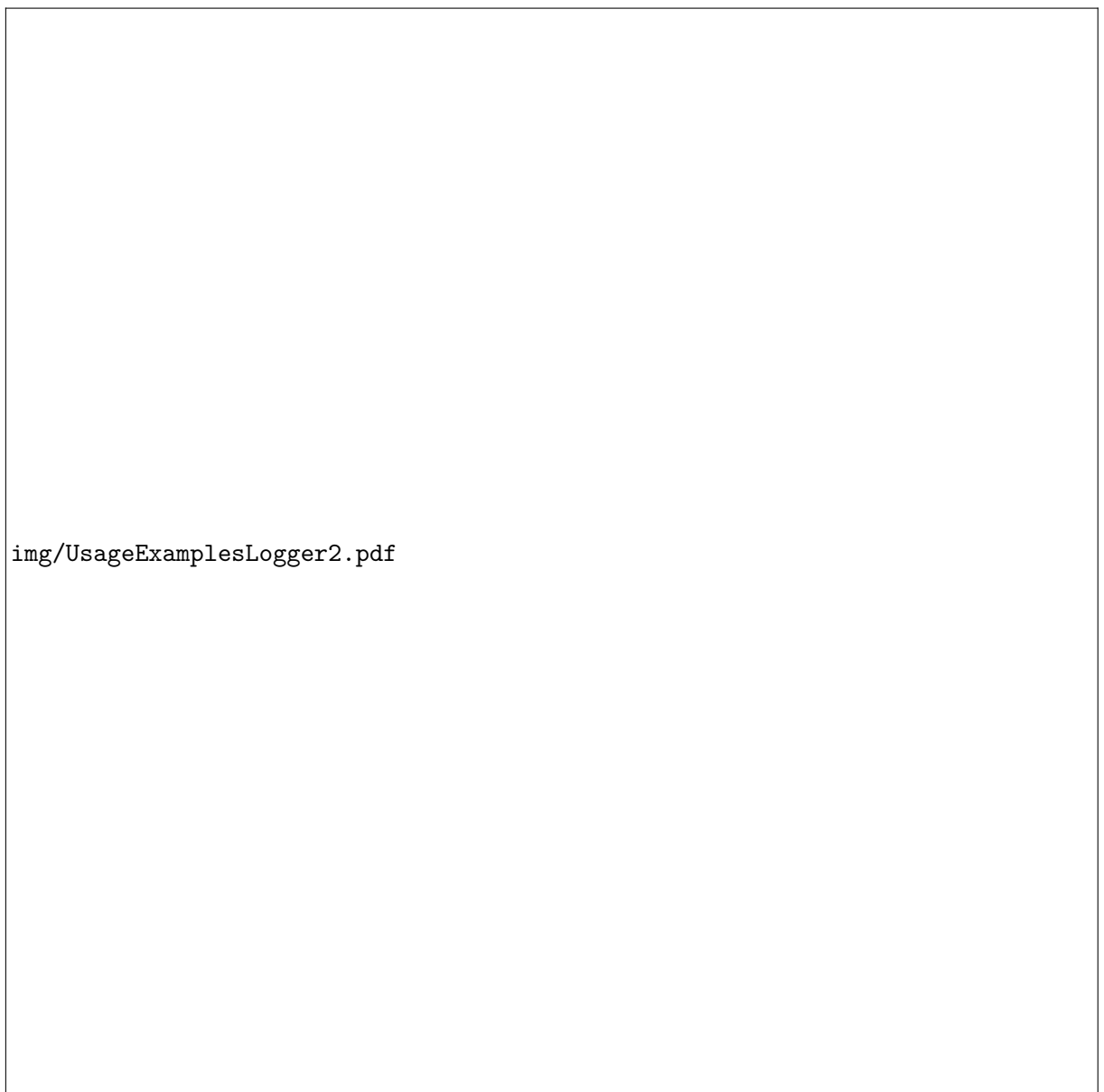
Figure 3.10: Configuration of the SensorBoards during measurement of a movement of a horse

img/UsageExamplesLoggerHorse.pdf
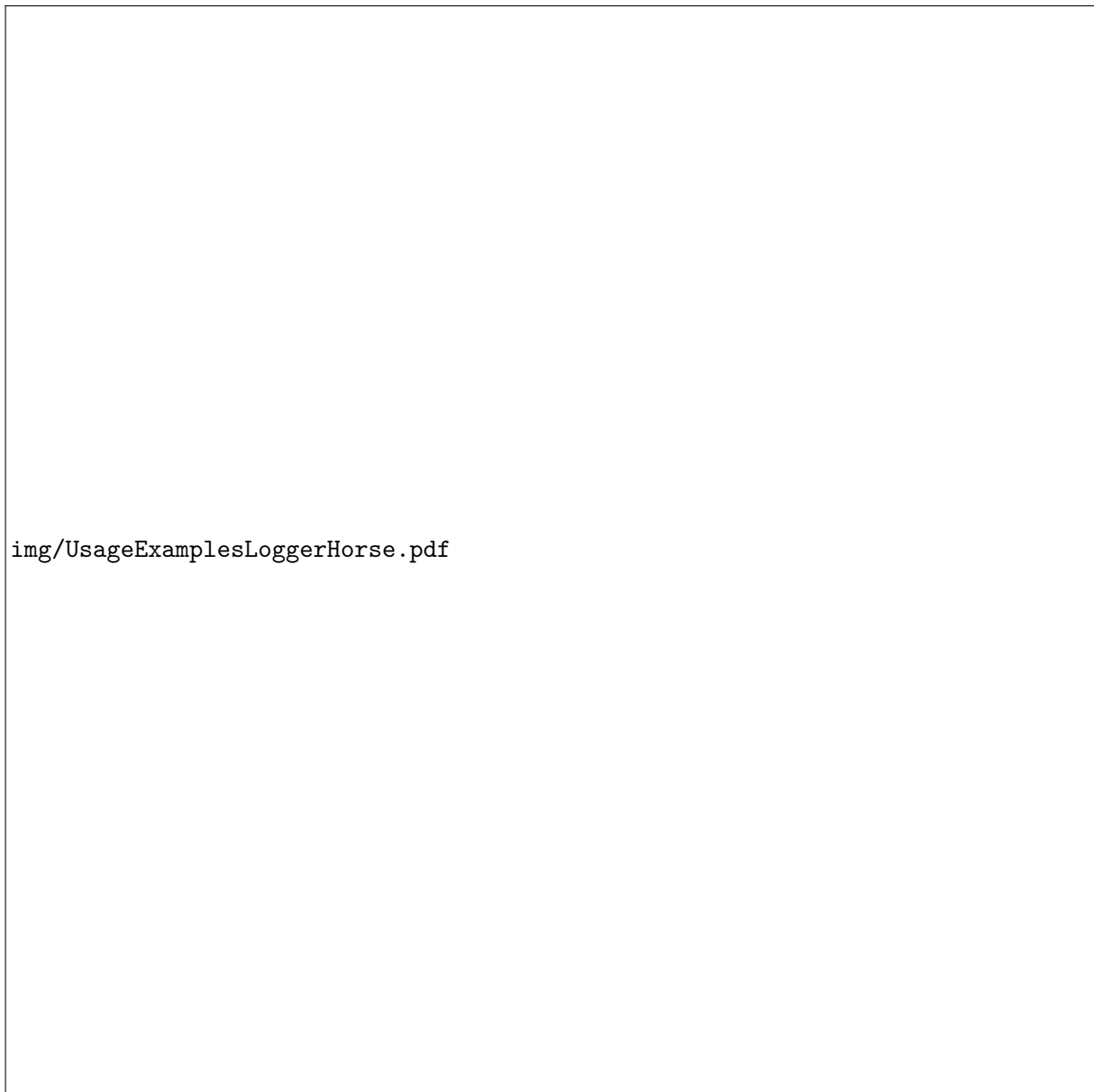
Figure 3.11: Sensor fusion example on custom AHRS

img/UsageExamplesAHRS.pdf
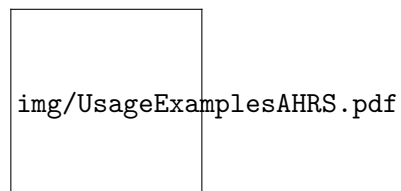
non-critical program on the second controller is usually a navigator or user program. The figure **??** shows the possible configuration of the flight controller on the SensorBoard. This configuration uses BMF055 Flight Controller [**?**] by Lukas Blocher. His code can be compiled directly for the SensorBoard without any modification.

3.3.4 Indoor navigation using TDOA

Indoor navigation requires high accuracy positioning. We usually need higher accuracy than given by GPS. I know so many projects approaching to the indoor positioning. There were 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 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. [**?**]. The figure **??** shows how the relative positioning works on the SensorBoard.

The DWM1000 [**?**] module is present on SensorBoard and when it connects to at least three other boards, then 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 **??**, we have all data about current position and attitude of the SensorBoard. We can use some other sensors on the SensorBoard like BMP280 barometer [**?**] to improve the current postition 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 common operating system. The common operating system doesn't 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. [**?**] So, 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 ESP32 controller like is shows in figure **??** and optionally we can use the sensors as data sources and as resources.

3.3.6 MicroPython Robot Controller

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

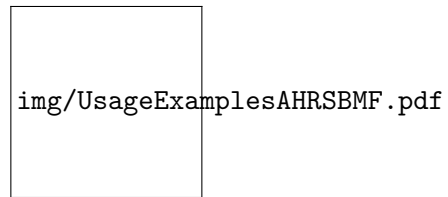Figure 3.12: Sensor fusion AHRS example with two controllers

img/UsageExamplesAHRSBMF.pdf

Figure 3.13: SensorBoard used as a quadcopter flight controller

img/UsageExamplesFlightController.pdf
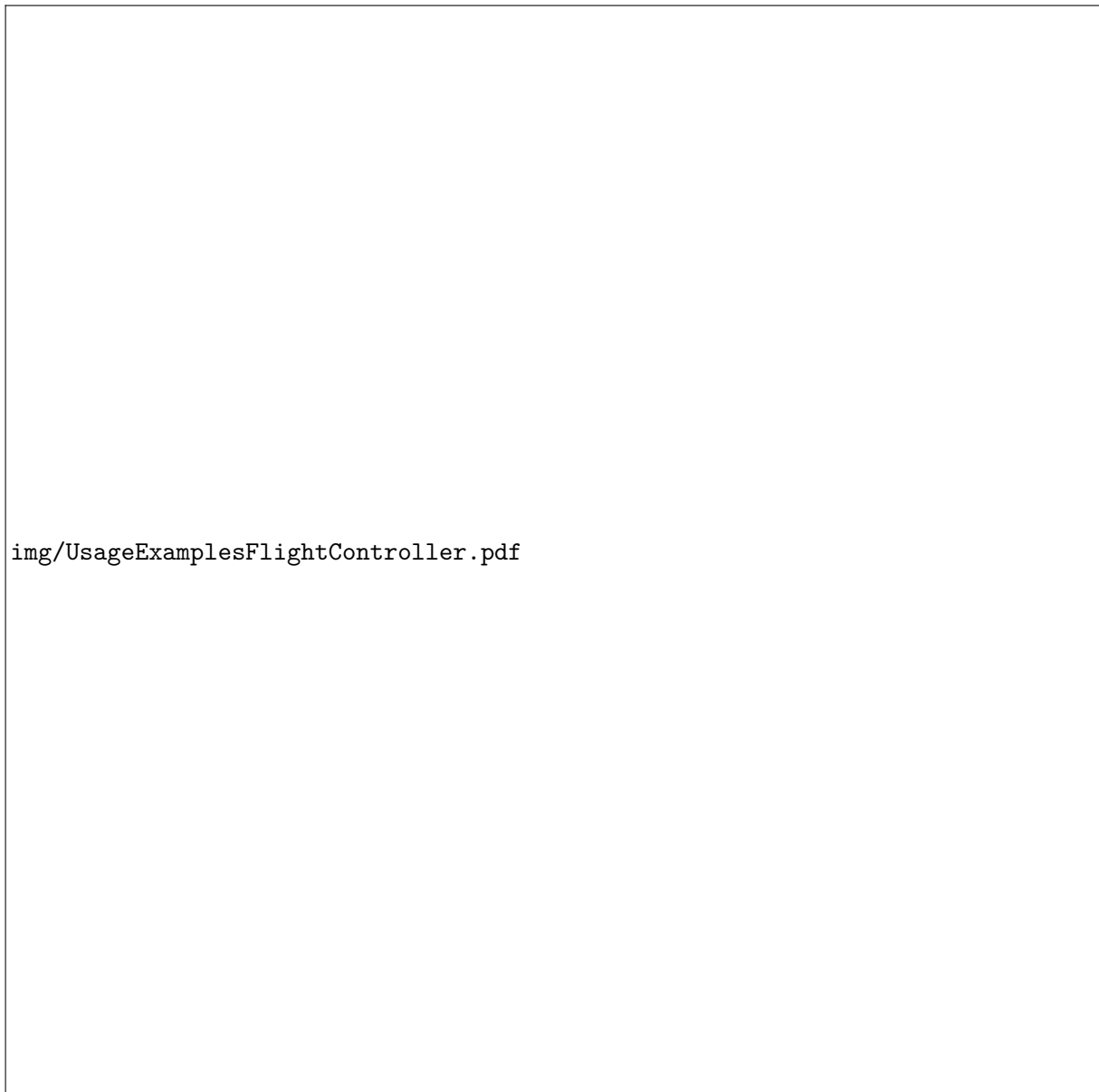
Figure 3.14: TDOA relative location service on the SensorBoard

img/UsageExamplesTDOA.pdf
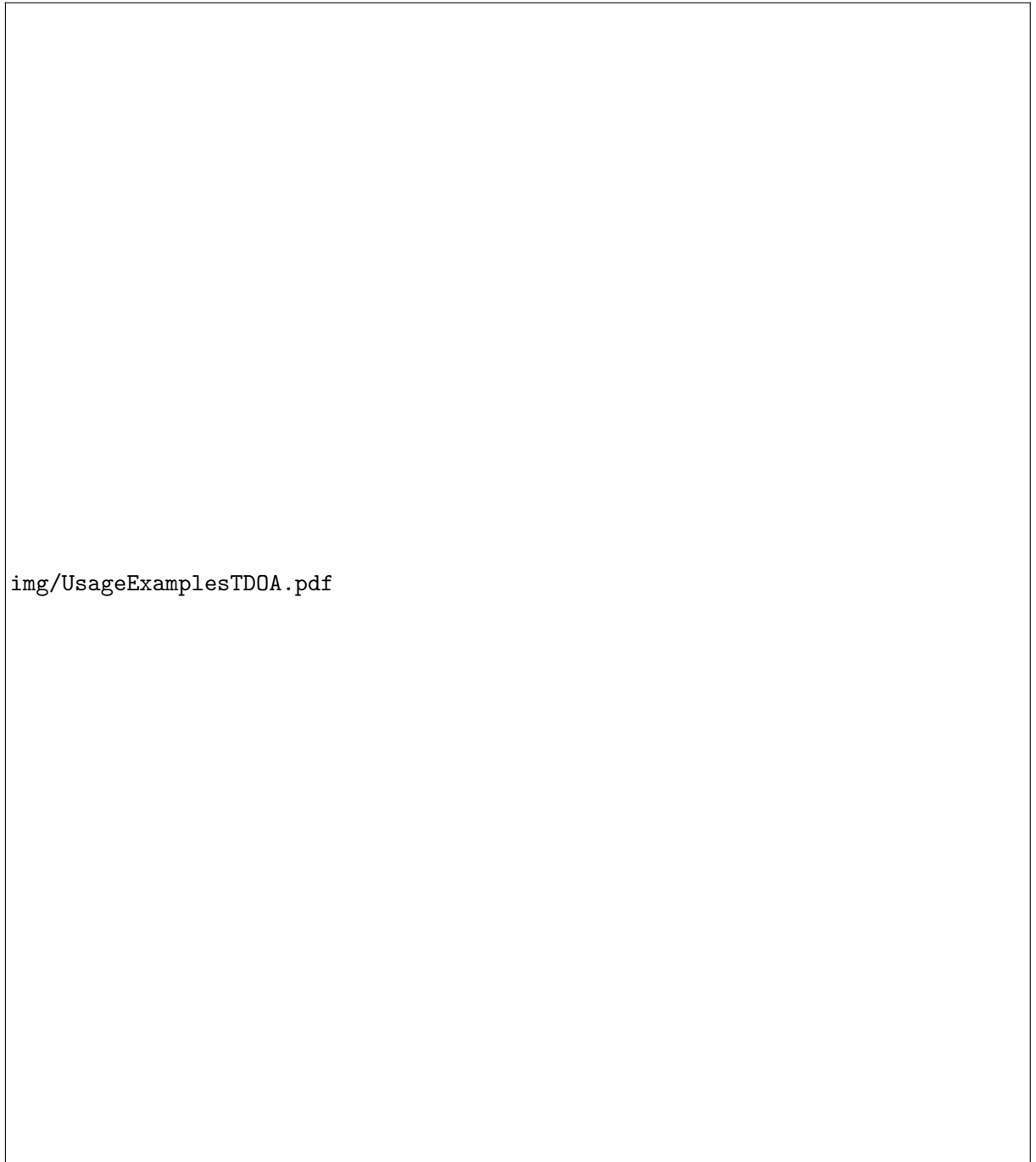
In my opinion, this setup is an easy way how to teach the basics of programming (in Python) with some dynamic hardware. This way of teaching is probably easier for the students and it probably creates more interactivity between students and the device. The figure **??** shown an example of using MicroPython inside 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 [**?**] or FTP server [**?**] or implement any other WiFi based service like is shown in figureUEWiFi.

3.3.8   Grid computing education board

The SensorBoard has a real time system, wireless and independence abilities. These advantages create a grid from multiple items. Then we can use the whole grid as one virtual unit. When we solve the problems with failures and accessibility, we have a grid solution with swappable items. An example of a grid with two failures is shown in figure **??**.

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 on-board. The SensorBoard fulfilles both conditions and it can be used directly this way. The figure **??** shows an example of using the SensorBoard as a device reading sensors data and streaming them via GSM netowrk or via WiFi connection.

3.4   Movement Analysis Firmware

The firmware is a sensor data logging application similar to the example in section **??**. 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 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.

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.

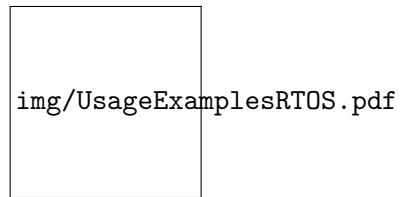Figure 3.15: FreeRTOS application diagram on ESP32 on the SensorBoard

img/UsageExamplesRTOS.pdf

Figure 3.16: MicroPython robot controller example on the SensorBoard

img/UsageExamplesPythonRobot.pdf
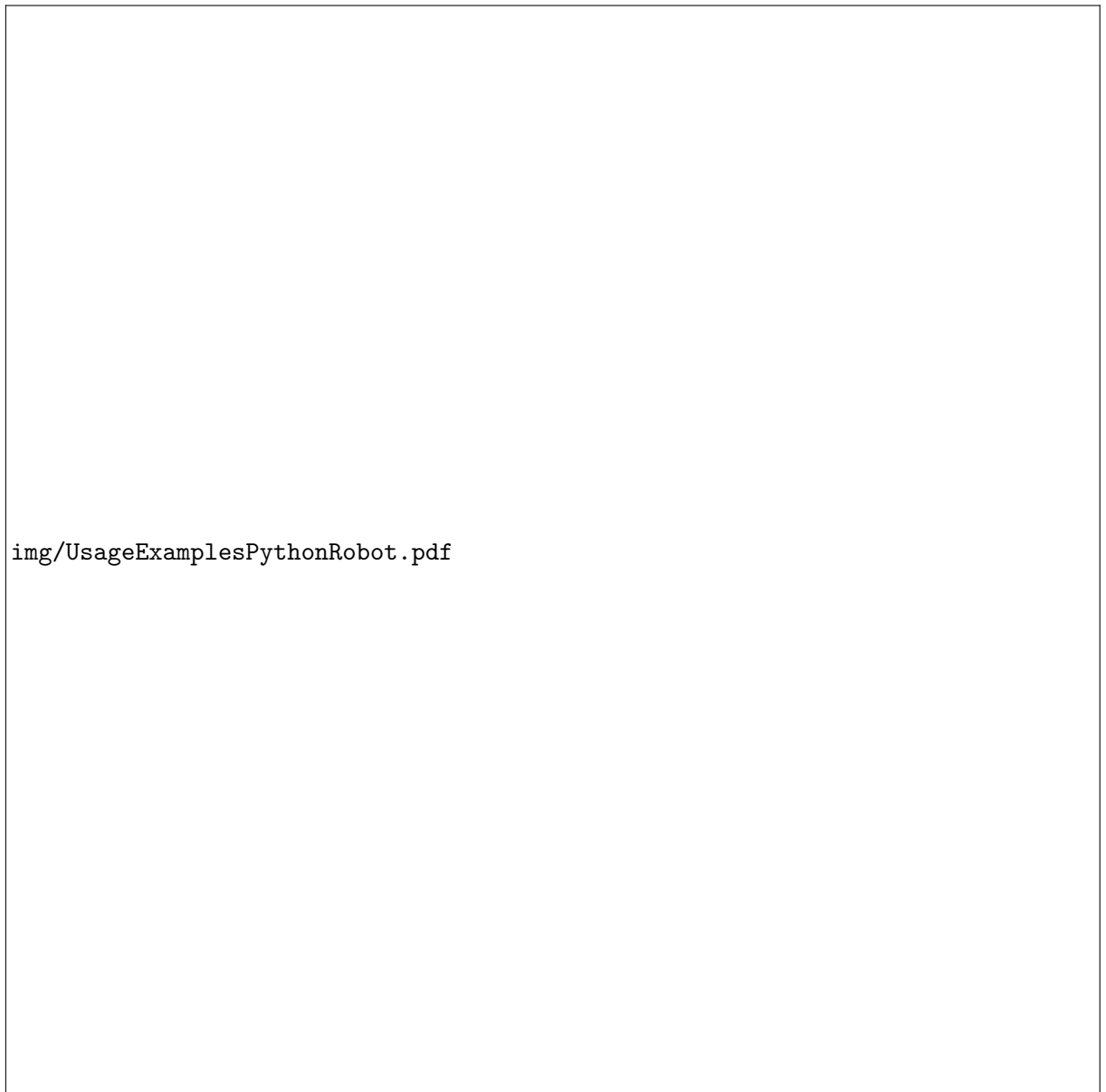
Figure 3.17: WiFi based services example on the SensorBoard

img/UsageExamplesServer.pdf
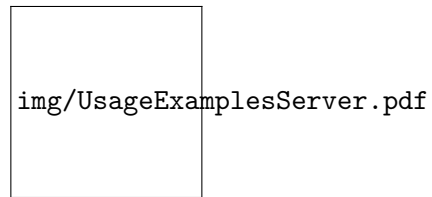
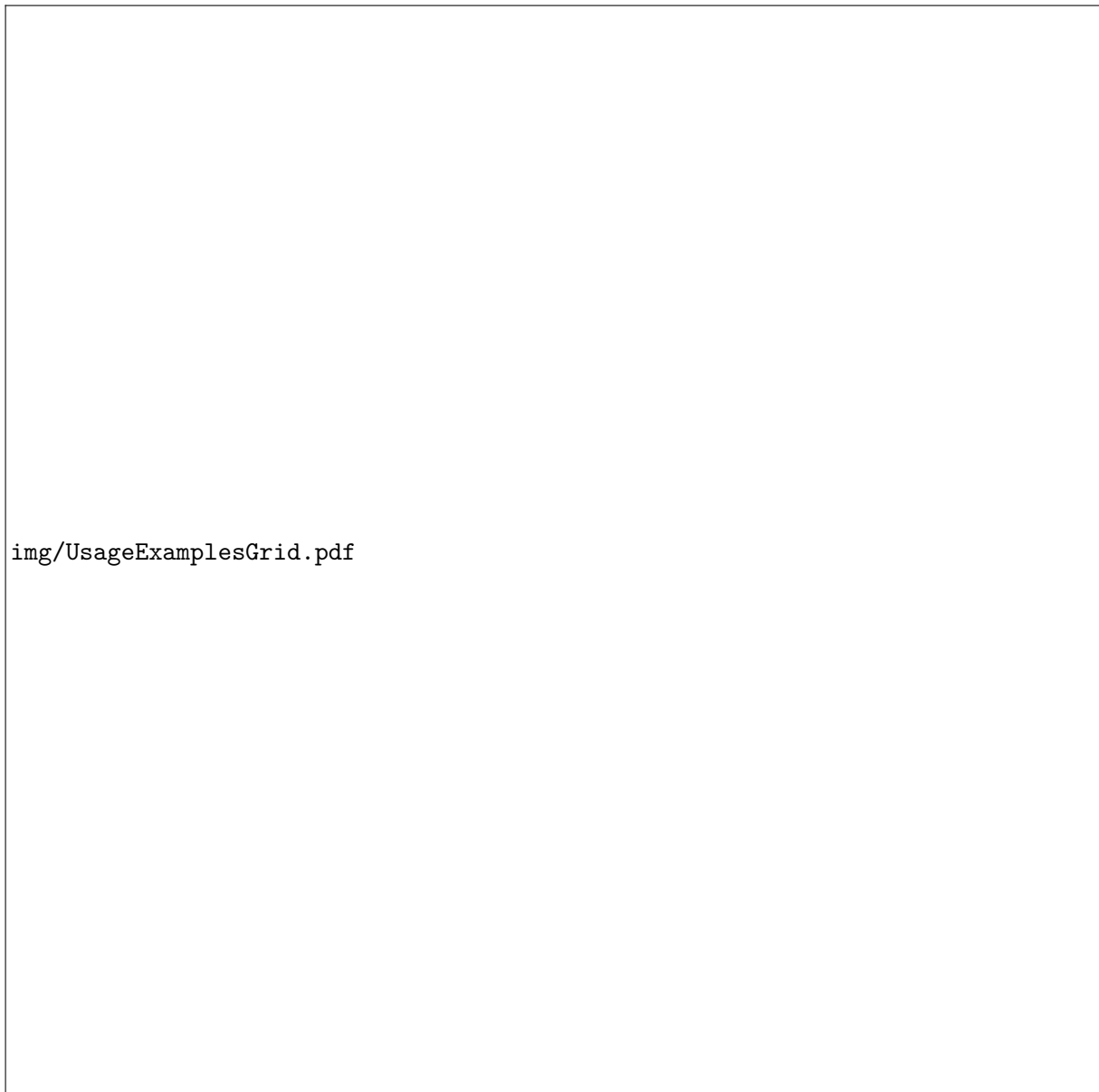Figure 3.18: Example of a grid with two failures created from multiple SensorBoards

img/UsageExamplesGrid.pdf

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

`img/UsageExamplesIoT.pdf`

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 every time when the SensorBoard is powered on. The configuration file is never modified by the SensorBoard. An example of the configuration file is in figure **??**. 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 **??**.

- **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 incerements 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 **??**.

- **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 behaviour of the program can be described using the state machine presented in figure **??**.

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 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.

Figure 3.20: Example of the configuration file on SD card

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

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

% Network SSID
% When initialized as station: tries to connect to existing netowrk
% When initialized as AP: creates new network
WiFiSSID = SensorBoard

% WiFi network password
WiFiPassword = 1234567890

% IP address of the device,
% if initialized as access point, can be used 192.168.1.1
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
```
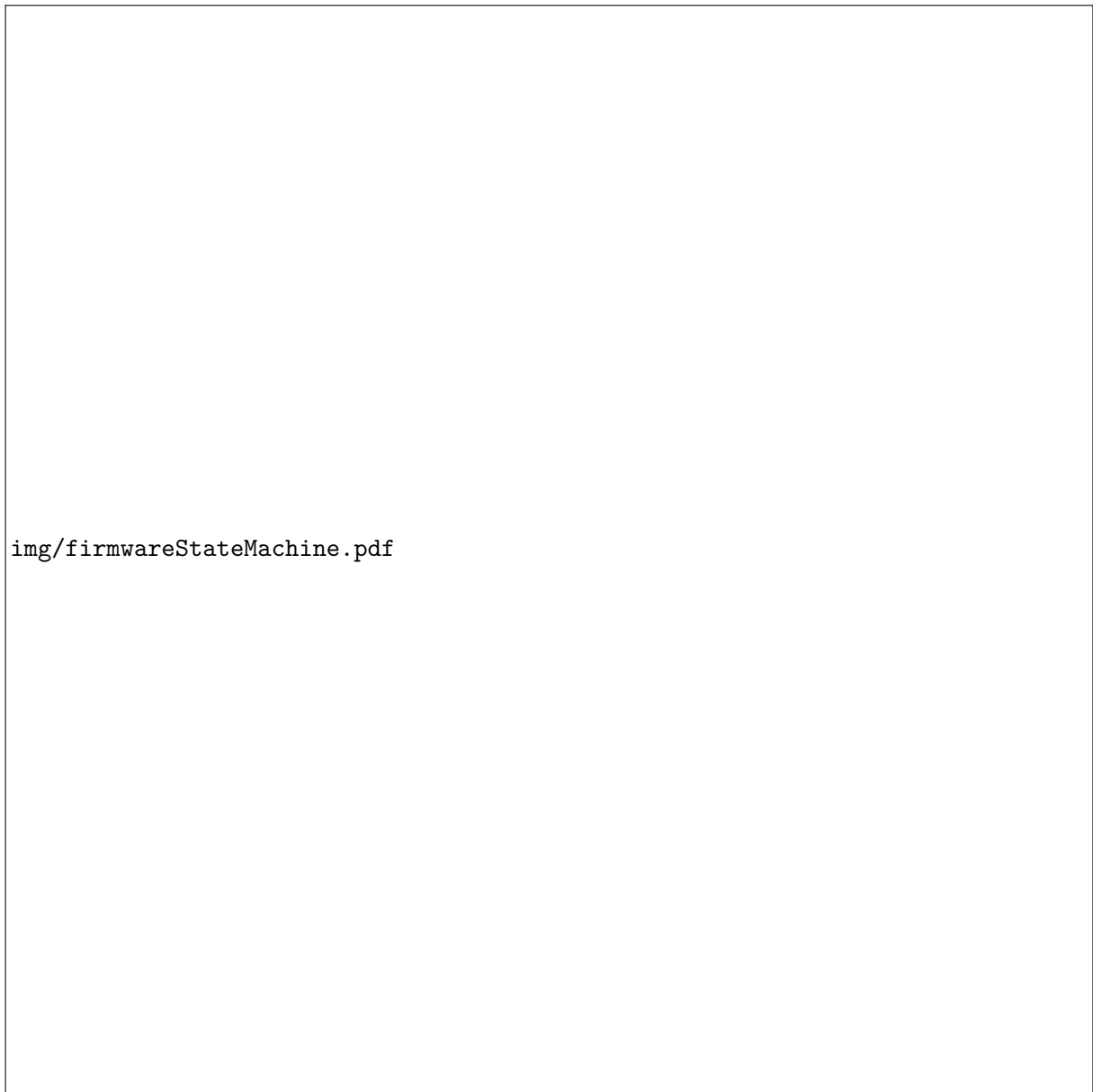42

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


```
img/firmwareStateMachine.pdf
```

- **Downloading data** The data can be donwloaded directly from the SD card or to a connected device.

- **Running** The Running state starts the measuring loop and allows to log, stream or analyse the measured data. It's 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 Idle state. The Running state can be activated automatically according to the configuration file.

3.4.4 Horse movement analysis feature

The application has build-in movement analysis tool. When the SensorBoard is places on a horse body, the tool can determine when the horse is moving and what kind of movement it does.

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 connection

The application can be controlled via TCP connection using single letter commands. I recommend to implement a web-page based user interface in the future versions of the application. One letter commands are actually used because it's the easiest way how to control the board from tablet with on-screen keyboard only. All the communication is possible via the TCP connection or via 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 **??**. Then all the hardware connected togedher 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 tethering/hotspot). All the SensorBoards except one (must be configured in the configuration file 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 synchronyously. The log files are stored on the SD card on each board 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 synchronyous 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 for example like in figure **??**.

//todo: ukazaka pripojeneho TCP terminalu na tabletu

CONCLUSION

The work 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 didn't fulfilled 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 hardware 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 a working hardware with C++ libraries for easier communication with sensors and other parts of the hardware. 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 movement of a horse. 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 begining. 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.

If there is an interest on future work with this hardware, I would recommend to create the second version of the SensorBoard with attention to the list of hardware errors presented in this thesis. The SensorBoard was used for example as a simple autopilot of a small quadcopter, so I believe that 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