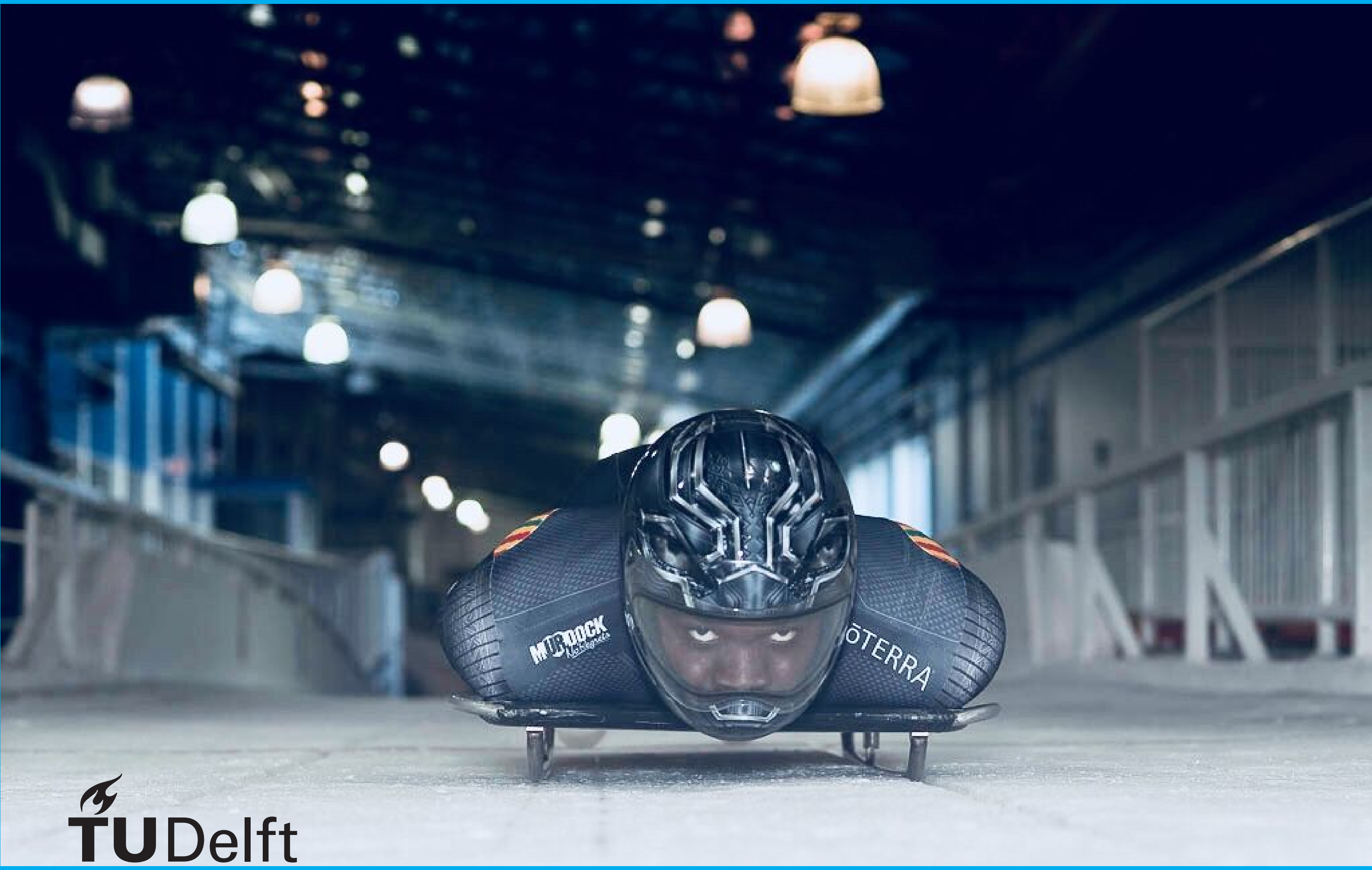
Instrumented Skeleton Sled

Bachelor thesis

A.W.G. Hunter
T. Moree

Focussing on the data processing
and the user interface



TU Delft

Instrumented Skeleton Sled

Bachelor thesis

by

A.W.G. Hunter
T. Moree

focussing on data processing and user interface,

in partial fulfilment of the requirements for the degree of

Bachelor of Science

at the Delft University of Technology.

Project duration:	April 22 – July 5, 2019
Supervisor:	prof. dr. P. J. French
Thesis committee:	dr. ir. Z. Al-Ars (chair)
	prof. dr. P. J. French
	dr. ir. B. Gholizad

To be defended on Wednesday 3 July 2019 at 13.30.

TU Delft

Abstract

This report details the design of an instrumentation system to be used on a skeleton sled. The system will measure several quantities on a skeleton track for the athlete to learn from. This data is stored during the run using several sensors on the sled and processed and visualised immediately afterwards on a mobile device. The project is tailor-made for skeleton athlete Akwasi Frimpong: he requested a way to get some insight in his steering using his knees and shoulders and its results on his achievements.

This thesis focuses on the data processing, communication and visualisation of the data and the software integration of all sensors. The communication is done using Wi-Fi and the visualisation is realised with a web page. The whole system is implemented with an ESP32 microprocessor which functions as a Wi-Fi access point and web server for the user interface.

Preface

This thesis is written in context of the Bachelor Graduation Project of Electrical Engineering. The project was commissioned by Akwasi Frimpong, a professional skeleton athlete, with the goal to come up with a way to provide him with information on his performance when practising his sport. His goal is to use the developed system to eventually compete for a medal on the Winter Olympics of 2022 in Beijing.

We would like to express our gratitude to our supervisors, prof. dr. Paddy French, dr. ir. Andre Bossche, and ing. Jeroen Bastemeijer for their guidance and advice during the project. Furthermore, we would like to thank Akwasi Frimpong for the topic and wish him good luck. Finally we would like to thank our colleagues: Werner van Dijk, Martijn Heller, Jan de Jong and Karen van der Werff for a productive, but most of all enjoyable collaboration!

William Hunter & Tijs Moree
Delft, June 2019

Contents

1 Introduction

1.1. Background

Skeleton is a winter sport in which an athlete lies in a prone position (face down and head first) on a small sled with two metal runners underneath, and goes down a winding track which is approximately 1800 meters long and covered in ice. An example of such a track can be seen in Fig. 1.1, which depicts a computer mock-up of the track that will be used at the 2022 Winter Olympics in Beijing. The sport has a high intensity: during a run, the athlete is subject to high g-forces and speeds which can exceed 130 km/h [1]. The steering of the sled is done by pushing the shoulders or knees into the sled: "the sled contorts as a response to the athlete's steering control movements. When this happens, the left or right runner knife is forced into the ice, creating an asymmetry in ice friction resulting in a steering moment. That is, when the left runner is forced into the ice, the sled will turn left. Athletes thus use their shoulders and knees to contort the sled; for a more dramatic steering movement, they 'tap a toe' onto the ice, creating a larger steering moment" [2].

A run starts with the athlete sprinting from the starting point with the sled, the so-called push start, a critical part of the run (this feature also appears in bobsled racing, but is absent from the similar sport of luge). This can be seen schematically in Fig. 1.2: after about fifteen to thirty metres, the athlete mounts the sled at full running speed and manoeuvres it around a series of (often) high-banking corners in the desired path, to maximise his speed [2].

1.2. Problem definition

Akwasi Frimpong is an Olympic skeleton athlete, who aims to be the first African athlete to win a medal at the Winter Olympics. He is relatively new to the sport, with his background being track-and-field. In order to accomplish his goal, he therefore wants to shorten his learning curve in the skeleton sport. At the moment, only video imagery and visual feedback from the coach are used as feedback on his performance. Because of the high speeds at which he is travelling during the largest part of a run, the important details from the run can be very hard to spot. This is evident from Fig. 1.3: grainy images like this are the best the average athlete has to evaluate his performance on the sled. Besides the previously mentioned means of feedback, there are no quantitative elements that are readily measured except for the elapsed time, measured by timing eyes mounted in set intervals on the track.

The important thing for an athlete to know is how he influences the sled during the run. This is especially relevant in curves, since the way these are traversed is the most impacting factor on the run time besides the push start [4]. As mentioned in §1.1, the steering of the sled is done by exerting forces on it using the shoulders and the knees. These movements of the shoulders and knees are practically impossible to see on video, or on

Fig. 1.1: Computer mock-up of the skeleton track to be used at the 2022 Beijing Winter Olympics [3]

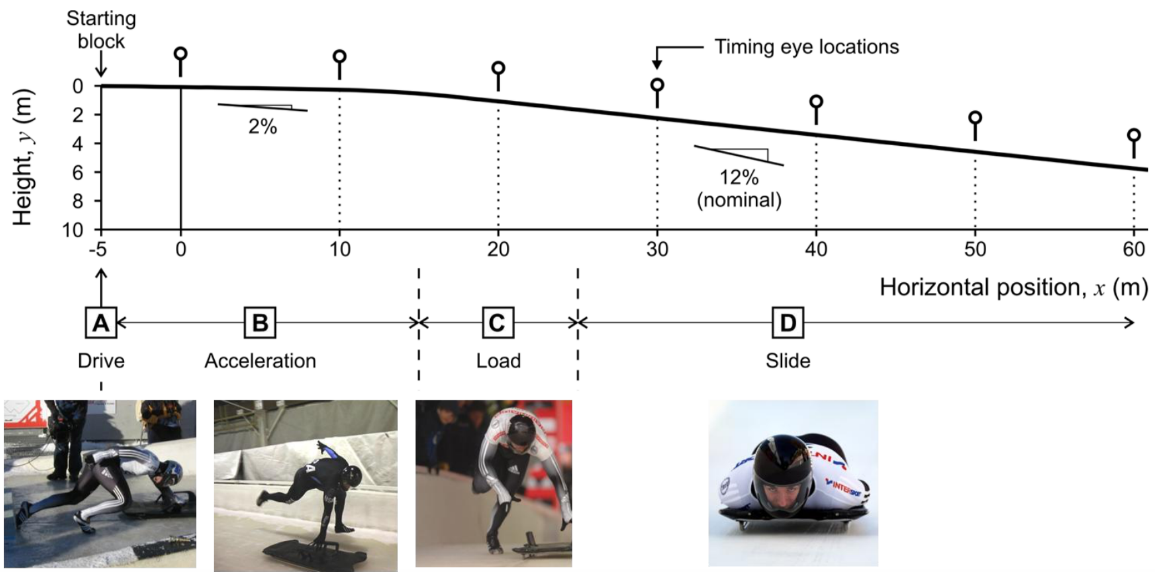


Fig. 1.2: Overview of the height profile at the beginning of a skeleton practice track, illustrating the push start [1]

images like Fig. 1.3. The solution to this problem could be offered by integrating a measurement system onto the sled (forming a so-called "instrumented sled"), which can measure the forces applied by the athlete, along with other relevant performance parameters and which can couple the measured forces to the location of the sled on the track—all these measurements would then need to be presented to the athlete in a way that he can quickly access. The use of an instrumented sled would result in a significant increase in useful feedback that the athlete can use to improve his run time, thus increasing the chances of the athlete shortening his learning curve. The primary goal of the product would thus be providing detailed, useful feedback on an athlete's run, in a way that is easy for both the athlete and the coach to understand and work with.

Fig. 1.3: An example of the visual feedback that is currently available to a skeleton athlete. This image was used by a coach to illustrate a momentary lifting of the shoulders off the sled, which is considered to be a bad habit (image courtesy of Akwasi Frimpong).

1.3. State-of-the-art analysis

Despite the fact that the sport of skeleton is not very widely practised, a number of studies have been done on the measurement of the forces, speeds and acceleration involved in the sport. These studies were primarily performed in order to create a better understanding of the dynamics at play in the sport, instead of having a goal of being used to actively improve athlete performance. Roberts [1] showed in his work that from measurements of the acceleration of the sled in three axes, velocity and traversed distance can be derived, which can provide useful information about the push start of the run. This is valuable, as it has been shown that the (effectiveness of the) push start has a large impact on the eventual time taken to complete a run [5]. After the push start and later in the run, however, the noise on these measurements becomes too large due to vibrations and other factors [1], making it impossible to integrate this to obtain a meaningful speed and distance reading during the whole descent. Sawade *et al.* [2] studied the factors influencing skeleton steering, showing a correlation between the applied steering force by the athlete and measured accelerometer and gyroscope data.

The aforementioned studies examined a number of relevant parameters involved in a skeleton run by attaching sensors to the sled and logging their output data. Although this is useful data, these studies provide only a limited tangent to the solution to the problem put forth in §1.2, as further processing of the data and visualisation (to produce the graphs from which conclusions could be drawn) was done after the fact. No special consideration was given to making the data easily and quickly available to the athlete (and his coach) from which the measurements were sourced, relegating their results to the status of reference work for an athlete instead of a product they can use themselves. The general idea of combining computing, sensing and communication as is required for this project, however, isn't new. This way of integrating information processing into user objects without the user being actively aware of the hardware behind it is known as "ubiquitous" or "pervasive" computing and has seen a rapid increase in the past decade [6]. In sports in general, "these ubiquitous computing technologies are utilised to acquire, analyse and present performance data without affecting the athletes during training and competitions" [6].

Nevertheless, these technologies aren't yet prevalent in the skeleton world; the closest similar system was studied by Lee *et al.* for use on a bobsled. This involved fitting an elaborate system of sensors and cameras on the sled, that together produced video imagery overlaid with sensor data, which was then wirelessly transmitted to a monitor on a remote site in real time [7]. This is useful functionality, but it isn't practically usable in the skeleton case: the system is bulky, as can be seen in Fig. 1.4, requiring (amongst other things) a relatively large and heavy control unit, which wouldn't fit on a skeleton sled.

It can thus be concluded that although modern research into the topic of skeleton dynamics is available, none exists that cover the scope of the system required to provide skeleton run data accurately and quickly in an (from the athlete's point of view) easy-to-understand format.

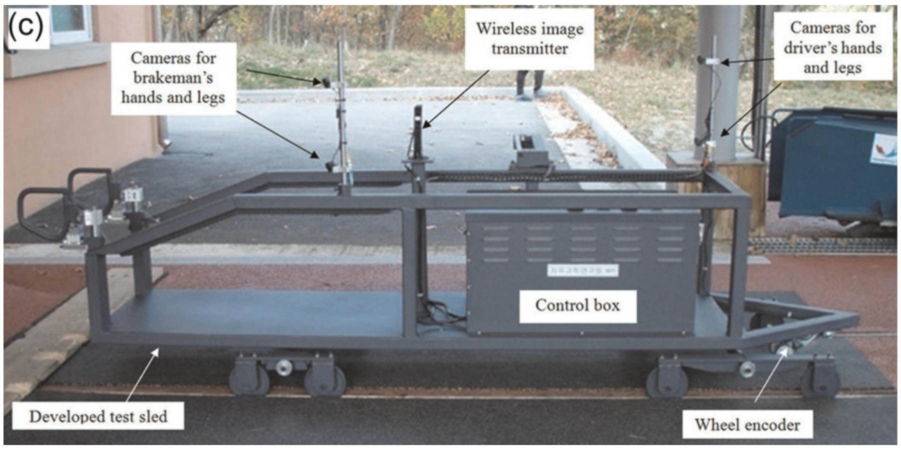

(c)
Cameras for brakeman's hands and legs
Wireless image transmitter
Cameras for driver's hands and legs
Control box
Developed test sled
Wheel encoder

Fig. 1.4: Example of an instrumented bobsled, as developed by Lee *et al.* [7].

1.4. Subdivision of the system

In order to realise the instrumented sled system, the product is divided into three subgroups, each with its own responsibilities: a schematic overview is given by Fig. 1.5. The work of each of the three groups is documented in separate theses: "sensor group A" refers to the work done by Heller and De Jong [8] and "sensor group B" refers to the work done by Van Dijk and Van der Werff [9]. Group A is responsible for the measurement of the (steering) force that is applied by the athlete and the measuring of the g-forces and orientation of the sled. Group B is responsible for the localisation of the sled, measurement of the ice temperature and power management of the system. The data group (and thus this thesis) focusses on integration of the total system at software level, data storage, processing and visualisation at the end of a run, to produce the desired user interface.

The work of all three subgroups is eventually combined to form the instrumented sled system: the eventual prototype of the system will consist of a printed circuit board (PCB) containing all necessary parts (developed by the different subgroups) integrated into one system, along with the software required to run it and produce the visualised data. The responsibility of designing this PCB prototype and corresponding hardware-level considerations of the integrated system lie with sensor group B; the software integration falls under the responsibility of the data group.

1.5. Thesis outline

As was touched on in §1.4 and illustrated by Fig. 1.5, this thesis covers the design process and choices made with regards to the software side of integrating the instrumented sled system, with special consideration of the aspects of data storage, data processing/visualisation and user interface. Chapter 2 lays out and defines a programme of requirements to which the designed system will be tested. Chapters 3 to 5 describe the design process and the justification of choices made: Chapter 3 covers the data storage and system integration aspects, with Chapter 4 dealing with reading out stored data and communication with mobile devices, and Chapter 5 addressing visualisation of the transmitted data and the resulting user interface. Testing and corresponding results are presented in Chapter 6, as well as a discussion of these results. Finally, Chapter 7 rounds the thesis off with general conclusions, recommendations and ideas for future work.

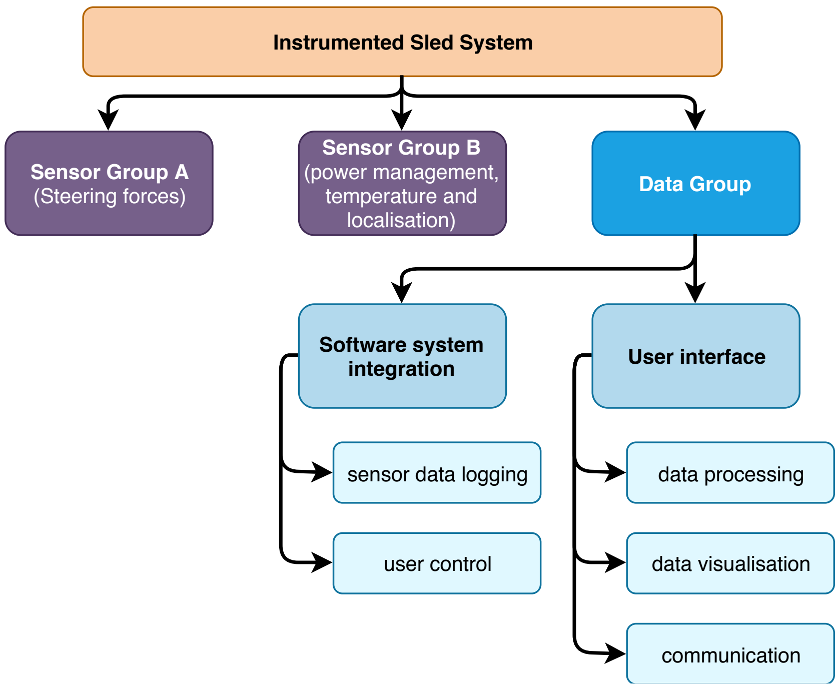


Fig. 1.5: Subdivision of the project into groups, and their responsibilities

2 Program of requirements

As has been described in Chapter 1, this project consists of creating an instrumented skeleton sled, to enable the athlete to train faster and smarter. In order to achieve this, a couple of aspects must be monitored, such as the location, the velocity and the pressure. These values must be stored and visualised for the athlete and his coach. This system is subject to various requirements, which will be listed below. It is split in general requirements for the system as a whole and specific ones for this subsystem. Each design choice will be made with these requirements in mind and the final product will be tested against them.

2.1. General requirements (GR)

General requirements are those requirements that are relevant for the entire system and that should be met by every subgroup. They can be listed as follows:

GR.1 The product must be able to measure g-forces, rotation, force applied by the athlete, ice temperature of the track and must be able to determine the location of the skeleton sled.

GR.2 The product must be able to work in a temperature range from -20 °C to 40 °C, since it will be used in an area with temperatures in this range.

GR.3 The product must be able to withstand momentary accelerations of up to 5 g [4, p. 198].

GR.4 The complete system should not weigh more than 1.5 kg, to prevent the characteristics of the skeleton sled being different from match conditions during training.

GR.5 The dimensions of the product cannot exceed a box of dimensions 31.5 cm × 14.7 cm × 2 cm, since this is the size of the available box inside the skeleton sled.

GR.6 The update rate of the force sensors and localisation system should be such that data points are at most 1 metre apart. Working with a maximum speed of 147 km/h [1], this gives a minimum frequency of 41 Hz.

GR.7 It should not be necessary to open the space inside the skeleton sled, where the circuitry will be located, in between runs. Therefore, the user must be able to start and stop the measurement from the outside.

GR.8 The product must be easily installed or removed from the sled, without leaving any (permanent) traces on the sled.

GR.9 The product should influence neither the aerodynamic properties nor the mechanical properties, apart from the weight, of the sled.

GR.10 The product cannot have any wired connections outside the sled and must be able to operate continuously for the time it takes to do 3 runs and the time in between runs.

GR.11 The system must be robust, being able to handle the vibrations of the sled during a run.

GR.12 The acquired data must be available within 5 minutes after each run for the athlete and the coach to use.

GR.13 The product should be easy to use.

GR.14 The total cost of making the prototype must fit in a budget of €250,-.

2.2. Specific requirements (SR)

There are several requirements specific for the subsystems described in this thesis. For the user interface, the following specifications must be met:

SR.1 The acquired data must be accessible from a mobile device, such as a smartphone or a tablet.

SR.2 The data must not be accessible by other athletes.

SR.3 The data should be visualised in a manner that can be interpreted well by the athlete and his coach.

SR.4 The athlete should be able to select previous runs to visualise as well for comparison.

SR.5 The acquired data should be visible for each time point with a selector to scroll through time.

SR.6 The velocity, g-force and pressure data should be visible graphically over the course of all time points.

Not all general requirements will be addressed in this thesis specifically, but they will be kept in mind when taking certain decisions. All requirements are listed again in Chapter 7, to check whether they were met.

3
Software system integration

In Chapter 2, the global specifications of the instrumented sled system are given. From requirement GR.1 it follows that a number of sensors are needed, so that the mentioned quantities can be accurately observed. Furthermore, GR.12 stipulates that the measured values need to be accessed after a run, so a form of storage is needed. This gives an outline of the (input/output) peripherals of the system that is required to perform the desired functionality. In between the sensors and the storage lies a microprocessor, to which the sensors initially supply their data during a test run. The microprocessor needs to perform a number of operations on this data as it comes in, before saving those values to an external storage unit, from where they can be accessed for further processing and visualisation after completion of a run, as is described by Chapters 4 and 5. The microprocessor further serves as the "brain" and control centre of the whole system: it must therefore also ensure correct working of the system while taking into account requirements GR.7 and GR.13, concerning user-friendliness and interaction with the user. The system as a whole should have a low power consumption, so as to ease satisfying requirement GR.10, be able to withstand the temperatures dictated by GR.2 and finally, requirement GR.5 should be kept in mind, so the system should be compact.

This chapter describes the design choices arising from the mentioned requirements, leading to a system that integrates sensor readout, consisting of the microprocessor, the sensors and the external storage unit, and describes its working and the software controlling these different components.

3.1. Overview

As mentioned previously, the sensor readout system broadly consists of three parts: sensors, supplying data to the rest of the system; a microprocessor, controlling and processing all data streams; and a storage unit, to save the measurement data for later use. Fig. 3.1 gives a schematic overview of this system. In this figure, "GNSS", "IR", "IMU" and "FSR" (connected to the microprocessor via an analog-to-digital converter "ADC") indicate the type of sensors that are used; the terms above these boxes indicate the type of quantity that the sensors measure. A detailed look into the working of these sensors is outside the scope of this thesis: this, along with elaboration on the design choices that led to the specific sensors that were used, can be found in concurrent work by Van Dijk and Van der Werff [9] and Heller and De Jong [8]. The italic terms in Fig. 3.1 signify the communication protocols with which the sensors, microprocessor and storage unit exchange data; this is illustrated in §3.2.

The GNSS sensor periodically receives signals from a number of global navigation satellite systems (GNSSs) and supplies the system with information on position and (absolute value of the) velocity. An infrared (IR) sensor is used to measure temperature, and an inertial measurement unit (IMU) measures the g-force and orientation. Within the IMU, an accelerometer derives the g-force from its acceleration readings and a gyroscope keeps track of the orientation/rotation, specifically the pitch and roll quantities. Finally, five force-sensing resistors (FSR) measure perhaps the most important quantity: the pressure applied to the sled by the athlete to effect steering movements. With this collection of sensors, requirement GR.1 is thus satisfied, as all mentioned quantities are sensed.

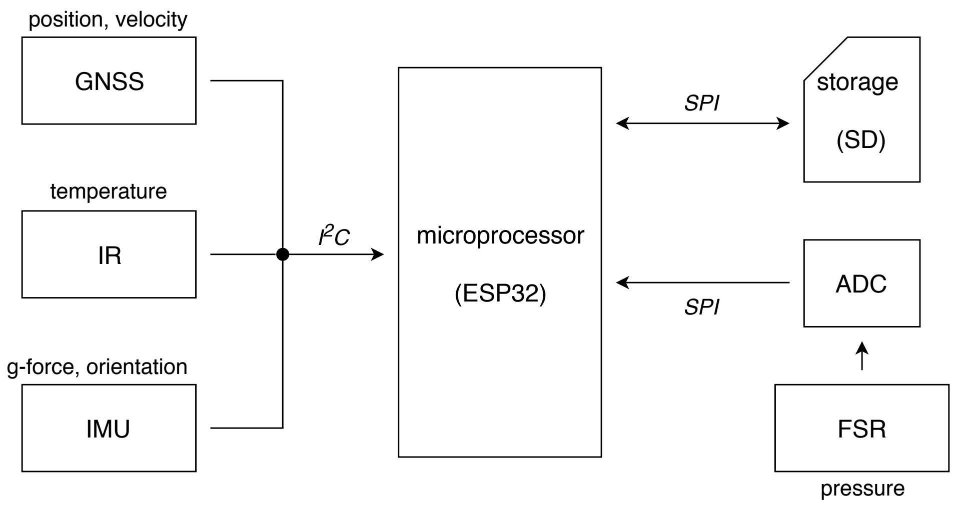


Fig. 3.1: Overview of the sensor readout system, with used communication protocols indicated in italic

As GR.12 stipulates that the data must be available post-measurement, a reliable form of data storage is required, which is indicated in the top-right corner of Fig. 3.1. Though this could be achieved by using the memory on the microprocessor, an external storage unit in the form of a Secure Digital (SD) card was chosen instead. This provides a number of advantages: first of all, the data-logging capabilities of the system are not limited by the (relatively small) storage capacity of the microprocessor. Modern SD cards can have capacities of up to 128 terabytes [10], ensuring that there will practically always be storage space available for a new measurement. Second, the storage provided by an SD card is non-volatile [11], so data is conserved after power-down of the system and can thus be accessed long after a measurement is performed. As far as external storage formats go, an SD card is furthermore relatively easy to work with, cheap, readily available, and, importantly, can be very small—the used micro-SD format has dimensions of 15 mm × 11 mm. The choice of a micro-SD card as external storage thus satisfies the availability requirement GR.12, with a size satisfying the dimensional requirements of GR.5 and the user-friendliness required by GR.13.

At the heart of the system, as is indicated in Fig. 3.1, lies the microprocessor. A lot of options are available to fulfil this role: for this system, the ESP32 by Espressif Systems was chosen. For its low price, it has a good performance properties and offers a number of advantages over contemporary counterparts [12]. With twelve (12-bit) ADC channels, two Inter-Intergrated Circuit (I^2C) interfaces, four Serial Peripheral Interfaces (SPI) and 32 general purpose input/output (GPIO) pins [13], it offers much more input/output capabilities than most comparable counterparts [12]. It has extensive software support and offers its own integrated development environment (IDE), but can also be programmed easily with the Arduino IDE—Arduino library support for the ESP32 is extensive. Furthermore, the ESP32 has built-in Wi-Fi and Bluetooth capabilities, which enable transfer of data to a mobile device, needed to satisfy requirement SR.1. The temperatures it will face according to GR.2 don't pose a problem: the ESP32 has an operating temperature range of -40 °C to 125 °C [13]. It has a low power consumption, similar to comparable microprocessors [12], which helps in satisfying requirement GR.10. Summing up, this all makes the ESP32 a powerful, easy-to-work-with choice to control the sensor readout system.

3.2. Communication

The different parts of the sensor readout system need to be able to effectively, quickly and accurately transfer data. From Fig. 3.1, it can be seen that this communication between devices is effected by the I^2C and SPI protocols. Although other recognised serial protocols do exist that have their own handy features [14], the main attractive features of I^2C and SPI for this system are their widespread implementation and ease of use. The GNSS sensor, IR sensor and IMU mentioned in §3.1 all provide digital outputs and have built-in communication capability via I^2C. As mentioned in §3.1, the ESP32 microprocessor has good support for this protocol and the fact that the protocol needs just two buses (a serial data and a serial clock line [15]) to communicate with any number of devices reduces the circuit complexity. A handy addition is the fact that Arduino library support for I^2C is extensive, easing high-level programming. With I^2C, data rates of 100 kbit/s (in the so-called "standard mode") or 400 kbit/s (in the "fast mode") can be achieved [15]: as will be elaborated in §3.3, this is more than fast enough to cope with the data streams that the sensors transmit. This all makes I^2C the obvious go-to option for sensor communication in the sensor readout system.

Although data transfer with an SD card is possible using its own SD Bus interface, the SD card also supports SPI [16], which is much more widely used. Just as with I^2C, this poses no problem for the ESP32, with elegant Arduino libraries making for an uncomplicated way to control communication. Communication with the FSR goes via SPI as well: the FSR's initially analog output goes to the ADC, which outputs its data via the serial protocol. Although the ESP32 has on-board ADCs, the choice was made to use an external ADC: this is explained in Chapter 6. Further elaboration on this topic and an in-depth analysis of the FSR can be found in work by Heller and De Jong [8].

3.3. System integration and program algorithm

With all components of the system depicted in Fig. 3.1 (globally) known, they can be put together to implement the desired functionality. The different parts (sensors, ESP32 and the SD card) are integrated on and connected via a printed circuit board, along with a number of additional elements required to correctly run the system. This includes a battery to act as a power source, a power management system, readout circuitry to obtain correct sensor output and a number of auxiliary electrical components, e.g. pull-up resistors required for I^2C operation. A detailed description of these items falls outside the scope of this thesis; the reader is referred to Van Dijk and Van der Werff [9] for a comprehensive look at the complete electronic system (i.e., all hardware that is required), as well as the battery and the power management system. Readout circuitry for the FSRs is detailed by Heller and De Jong [8]. This section focusses on the integration of the sensor readout system on a software level.

As mentioned in §3.1, some design choices of the sensor readout system were made with ease of programmability in mind: numerous libraries and extensive Arduino IDE support exist for both the ESP32 as well as the used sensors. These libraries, as well as the Arduino framework that is used to program the ESP32, are written in C++; therefore, this language was used to write the software controlling the functionality of the system—the source code can be found in its entirety in Appendix A. As was the case for the system hardware in §3.1, the software is subject to the requirements put forth in Chapter 2. Most points mentioned in Chapter 2 are covered by the processing and visualisation algorithm mentioned in Chapters 4 and 5; the relevant points that remain for the sensor readout system are GR.6, GR.7 and GR.13. To satisfy GR.6, a sensor sample rate of 45 Hz is set, resulting in a corresponding "sensor readout period" of approximately 22 ms. The sample rate is deliberately kept below 50 Hz to avoid electromagnetic interference from mains-connected equipment. To make the system user-controllable, a push button is included, readily accessible by the athlete, with which the measurement of a run can be initiated (the status of a measurement is correspondingly indicated by an LED).

The rest of the software can best be described in a number of steps, which are detailed below.

3.3.1. Initialisation

At power-up, a function is called that initialises the system. An adaptation of Arduino's "Wire" library for the ESP32[1] is used to control I^2C communication and is initialised by passing along the pin numbers of the ESP32 that are used as serial data and serial clock lines. Similarly, an ESP32 version of Arduino's "SD" library[2] controls communication with the micro-SD card and is initialised by passing on the chip select (CS) pin of the ESP32 used for SPI communication. Dedicated libraries control manipulation of and communication with the GNSS[3] and IR sensors[4], as well as the IMU[5]; each of these is initialised with relevant circuit parameters. The ESP32's WiFi functionality is turned off for the time being, to limit the microprocessor's power consumption.

3.3.2. Sensor data logging

Pressing the aforementioned push button starts the run measurement routine of the program. First, a new text file is created on the SD card: all files follow a naming convention, so each new file gets a name consisting of the date and the run number of that day. If the file can be opened successfully, execution proceeds. To prevent unnecessary measurements while the athlete hasn't started his run yet, the system enters a wait state, continually sensing the velocity data, until the measured velocity exceeds a given threshold. The sensor readout now begins: each time after the given sensor readout period has elapsed, new sensor values are read. For each such time point, a string is constructed in which all relevant quantities are concatenated (separated by tabs). First, using the library of the IMU, a number of instructions are carried out to prepare the IMU data. Next, the sensor values are successively read out and added to the string, in the following order: a timestamp, pressure, speed, g-forces, pitch and roll, temperature and finally latitude and longitude. These last two quantities are read out only if the GNSS sensor has four or more satellites in its view, so that "obviously" erroneous readings are excluded. By writing to the text file once per time point (instead of after reading each sensor value), the execution time is drastically reduced, ensuring a quicker-running program. The end result of a measurement is thus a text file on the SD card consisting of a number of lines, corresponding to the number of time data points, with in each line all sensor values.

Data logging stops when the measured velocity drops below a given threshold, ensuring that a measurement automatically stops at the end of a run. The file on the SD card is closed and the run measurement LED is turned off. At this point, the sensor values that were read out during a run have successfully been logged and are ready to be processed and visualised, which is described in Chapters 4 and 5. If the push button is pressed, the program loops and a new measurement routine is started to capture the next run.

[1]https://github.com/espressif/arduino-esp32/tree/master/libraries/Wire
[2]https://github.com/espressif/arduino-esp32/tree/master/libraries/SD
[3]https://github.com/sparkfun/SparkFun_Ublox_Arduino_Library
[4]https://github.com/sparkfun/SparkFun_MPU-9250_Breakout_Arduino_Library
[5]https://github.com/adafruit/Adafruit-MLX90614-Library

4 Data communication

After acquiring the data from all sensors and saving it to the SD card, it has to be made available to the athlete in some way. Most useful to the end user would be to have a system that graphically gives information on the whole run. This could be achieved using an on-board screen, but there are weight limits and maximum dimensions to the whole system according to GR.5, and there is a better solution using external devices. Since the athlete or his coach do not want to carry around a laptop during the test runs according to SR.1, the data must be read out in a mobile manner, i.e. using a mobile phone or a tablet. This already rules out the more conventional ways of plotting data, e.g. using MATLAB. Thus, a different solution to plot graphs and perform other visualisation tasks needs to be found.

4.1. Requirements

Since the sensors cannot be connected to the mobile device directly, they have to be implemented in a system of some sorts with the microprocessor before transferring their data to the mobile device. This implies there must be some kind of communication system. To keep the system as universally usable as possible, it has to use a USB cable or either wireless option of Bluetooth or Wi-Fi to communicate the data to the mobile device, because these are the communication protocols available on most mobile devices nowadays. The wireless options are by their very nature the most user-friendly ones. The data has to be read out between each run according to GR.12; using a USB cable would require plugging this cable into the sled each time and that is not easy, because the circuit cannot be accessed in between test runs according to GR.7. This means the communication will be done either through Wi-Fi or Bluetooth.

Working with mobile devices is a little harder software-wise, since there are two prevailing and rather different operating systems among them: Android and iOS. Designing an application that works on both systems is a lot of work and requires a lot of knowledge that is outside the scope of the Electrical Engineering bachelor's degree. There probably are a lot of available applications that can display data in some way, but the most universal and most customisable method would be to implement the visualisation as a web page. This way, it can be accessed from any operating system, since web browsers exist for all of them.

Since the user now will interact with the data via a web page and since Bluetooth is not conventionally accepted by browsers, the obvious communication system becomes Wi-Fi. Browsers always function as a so-called web client. Such a client needs a web server, that serves the data, and a way of data visualisation. This system would mean that the microprocessor extracting data from the SD card will act as a web server. The communication protocol between a web client and server is the Hypertext Transfer Protocol (HTTP). This protocol describes the way the web page itself and the data to be visualised are transferred to the mobile device from the microprocessor. Using this system, it is possible for the athlete to read out the data immediately after going down the bobsled track, which is required by GR.12.

The programming languages that can be interpreted by browsers are Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript. The first is used to describe the layout of the page, the second determines the style and the last is used to make the page interactive. This should end up in a nice user interface which is necessary to comply with SR.3.

4.2. Implementation

As described earlier, the microprocessor will function as a web server, meaning it will serve data when asked to over HTTP. To make this work, the microprocessor and the mobile device have to be connected in some way over Wi-Fi. An option would be to connect them both to a network available at the bobsled track, but this would mean the Wi-Fi credentials of the network had to be inserted into the microprocessor in some way. Since the system should be easy to use according to GR.13, this option was rejected. The other option is to run a local network on the microprocessor with a password to keep the other competitors outside, complying with SR.2.

4.2.1. Wi-Fi

As described in Chapter 3, the microprocessor used is an ESP32. This chip has Wi-Fi integrated [13] and the breakout circuit used in the prototype has a small PCB trail functioning as an antenna, so this makes half of the communication rather simple. Espressif Systems, the manufacturer of the ESP32, has a library[1] that is able to create a local Wi-Fi network using the chip. Now, any device can log in to the network if using the right credentials. An Internet Protocol (IP) address is set as well, which is the location the eventual user interface will run from. This IP address is a private address that is restricted to fall in the ranges of either 192.168.0.0 to 192.168.255.255, 172.16.0.0 to 172.31.255.255 or 10.0.0.0 to 10.255.255.255 [17]. In this project the IP address was chosen to be 10.0.0.0 for simplicity reasons.

One thing to note here is that connecting to this network does not mean there is a connection with the internet, since neither the server nor the client then is connected to it. This means they can only talk to each other and not to any external devices or servers. This comes with a minor problem as will become clear later, but a solution is available.

4.2.2. Web server

The other half of the communication system consists of the data transferred over Wi-Fi. This data transfer is done using the set of rules that is called HTTP. As said earlier, the microprocessor will not only serve as a Wi-Fi access point, but as a web server as well. Since the communication protocol is HTTP, it is common to run this server on the default port for HTTP, 80 [18]. This is done using a library[2] created by Espressif. Setting up the server results in a web server that can be accessed if connected to the designated Wi-Fi network on *http://10.0.0.0:80/*, which can be abbreviated in most browsers to *10.0.0.0*.

4.2.3. HTTP

HTTP always consists of a response to a request. This request has a Uniform Resource Locator (URL) and a certain method, which in this product will always be *GET*. There are also other methods, like *POST* and *DELETE*, meant for changing data on the server side, but this product will only need data from the server for display. A request has headers and optionally parameters as well. The parameters are used to request more specifically; in this product it is used to select a certain run, for example. The headers can be used to pass

[1]https://github.com/espressif/arduino-esp32/tree/master/libraries/WiFi
[2]https://github.com/espressif/arduino-esp32/tree/master/libraries/WebServer

Table 4.1: This table shows all possible requests and their responses. The URL is shortened to everything after the base of *10.0.0.0*.

Request			**Response**			**Description**
URL	**Method**	**Param**	**Status**	**Content-type**	**Data**	
/	GET	-	200	text/html	*The web page*	This loads the whole HTML and CSS
/script.js	GET	-	200	text/javascript	*The scripts*	The scripts are loaded from the HTML
/list	GET	-	200	application/json	*The list of runs*	This is loaded from the Javascript
/date	GET	date	200	application/json	-	This sets the date of the ESP32
/file	GET	run	200	application/json	*The selected run*	This is requested when selecting a certain run
otherwise	*any*	-	404	text/plain	“Not found!”	When requesting anything else it responds with a 404

additional information as well, but are left untouched, meaning they are set to default.

The response consists of a status code, data and headers again. The status code describes whether the request was received and processed well. When everything is fine, this code will be 200. When the server cannot find what is requested it will return with the infamous code of 404. The headers are mostly left untouched, except for the *content-length* and *content-type* headers (these headers speak for themselves). The length signals to the client what the size of the response is and the type will be HTML or JavaScript, for example.

4.2.4. Requests

All possible requests and their corresponding responses are listed in Table 4.1. The first two rows of the table will both simply return a string: the first containing all HTML and CSS and the second one all the JavaScript. The JavaScript could have been embedded in the first one, but this way the browser will remember the script file in cache, meaning it does not always have to request it, saving time. These two strings will be covered extensively in Chapter 5.

The */list* request returns a list of all saved runs on the SD card for the user to select from. This simply goes over all files on the SD card and responds with this list in the JavaScript Object Notation (JSON). Since the data has to be interpreted using JavaScript, JSON is the best choice to do so. The list could also be sent without formatting it this way, but that would mean that the client-side scripts have to format it themselves, which probably costs more time.

For the athlete, it would be best to have the runs be named with the date and increasing numbers for the run number on the day itself. However, as mentioned before, it is not possible for the ESP32 to know date or time, since it is not connected to the internet in any way. Therefore, all runs will be saved to a default date first and renamed when a mobile device connects to the local Wi-Fi access point and sends the current date over. This is done using the */date* request. The JavaScript can extract the date from the mobile device and sends it over as a parameter.

The */file* request will be a file selected from the list. The selected run is communicated as a parameter, which will look like *10.0.0.0/file?run=2019-06-14-2*, for example. This chooses the second run of June the 14th of 2019. If the web server cannot find the run, it will return with an empty response with status code 404. If it does find the file, it will eventually give status code 200, but it has to process the file before being able to transfer all the data.

4.2.5. Data processing

Since the files on the SD card can be very large depending on the length of the run, it is possible that the file does not fit in the cache of the microprocessor. The data will therefore be transferred in chunks using the chunked transfer encoding of HTTP. This is enabled in the response headers, so the mobile device knows what to expect. Then, each line of the file is read and formatted using JSON. The response data will contain the date of the run, the number of the run on that day and the actual data consisting of a list of all timestamps of the run with the sensor data.

Some of the sensor data will first be processed using a running average. This is implemented with a library by Matt Fryer[3]. Every time a line is read from the SD card, its value will be saved to a so-called `Smoothed` object and read from it again to be inserted in a long string that is in the JSON format. 100 lines of processed SD lines will be one chunk that is sent to the mobile device. When the algorithm reaches the end of the file, all data should be sent, after which the JavaScript can use the data in the interface. Using this chunk encoding it is possible to have a run last very long, with the limit only being the capacity of the SD card.

4.2.6. Error handling

One thing to keep in mind is that the precision of the GNSS data depends on the amount of satellites in sight. According to one of the other subgroups, the data is only reliable if more than three satellites are in sight [9]. If this condition is not met, the data is rejected and the last valid location measurement is used. If there has not even been valid localisation data, it will send `null`, which will be interpreted by the JavaScript as invalid and therefore rejected as well.

Since the SD card can be removed and its data can be altered, it can happen that some data corrupts. When a line differs from the expected value, the system should not immediately crash. This can happen during a run as well: when the battery drains for example, it can show unexpected behaviour. Therefore, an extra safety mechanism is built in which checks whether the data in the file is valid. If not, it will stop reading the file and respond to the mobile device with the data up to the last valid time point.

[3]https://github.com/MattFryer/Smoothed

5 Visualisation

As described in Chapter 4, it is possible to connect to a Wi-Fi access point, visit a certain IP address and receive an HTML web page which in turn can receive a list of runs and the data of each run. The last step is to design this web page and give it some functionality so the received data is of any use to the athlete and his coach to comply with SR.3. The resulting source code can be found in appendix A.

5.1. Requirements

The interface should have a method to select any recorded run according to SR.4. There should be a page with a time point selector to comply with SR.5 and there should be a page that graphically shows the whole run, as stated in SR.6. The last requirement is that the interface should be easy to use according to GR.13. This resulted in the two web pages that can be seen in Fig. 5.1 and 5.2.

5.2. Layout

Both pages (Fig. 5.1 and Fig. 5.2) have all controls on the top of the page to keep it simple, complying with GR.13. Both pages have a selector on the right to choose the run that will be investigated. The default run is the last-recorded one. The first page has a time slider, a play button and two buttons to go back or forward one time point. The second page has one big slider that can be used to select a range of time. The upper left corner shows the time of the run; the run number can be seen in the run selector and switching between both pages is done by clicking on “Akwasi Frimpong Sensors”.

Both pages have the data shown in three panels at the bottom of the page. The left panel is the same for both pages: it shows the GNSS data and the current location on this map. The data that is shown here is a car drive around the southern part of the Delft University campus. The second page shows the selected time range on the map as well. The first page has a rotation and g-force panel in the middle. The picture of the athlete is drawn from his feet, showing his rotation relative to earth. The arrow is the size and direction of the g-force acting on him. The right panel shows the sled from above with the size of the red circles representing the amount of force applied to the knees and shoulders, the actual interesting data for the athlete according to GR.1.

The first page shows some variables as text as well. There is a current time and temperature of the ice on the left; the middle shows the absolute g-force acting on the sled and the slope downwards; the right has the velocity of the athlete and the average pressure applied to the sled as a number. The second page shows the time range and the current ice temperature on the left as well.

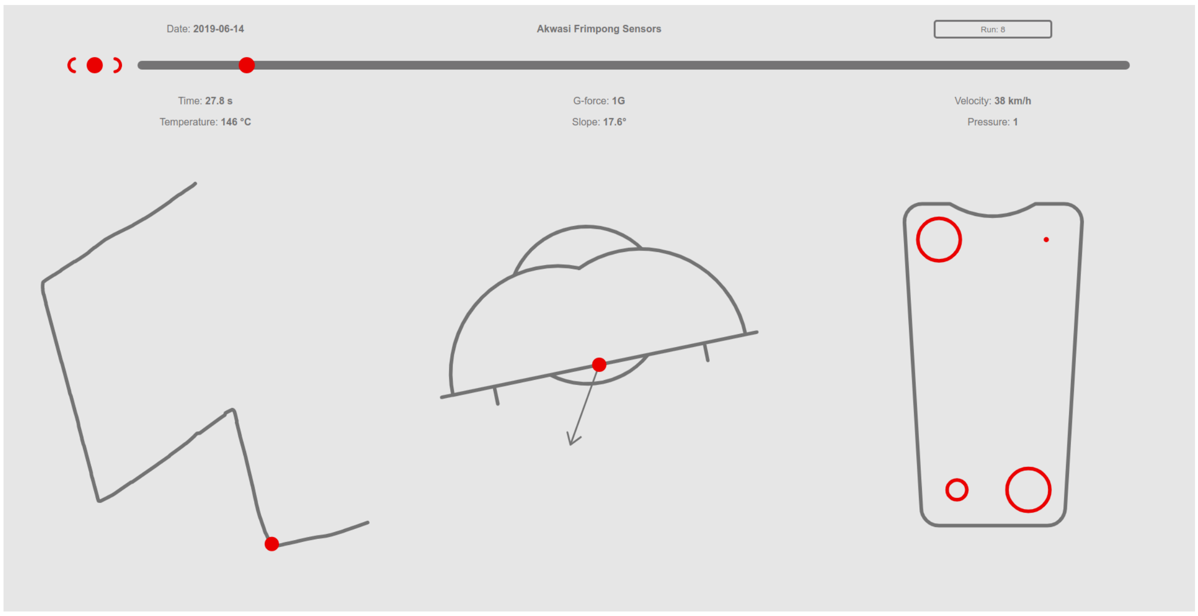


Fig. 5.1: First page of the user interface. The visible data is placeholder data and therefore not representative of a skeleton run.

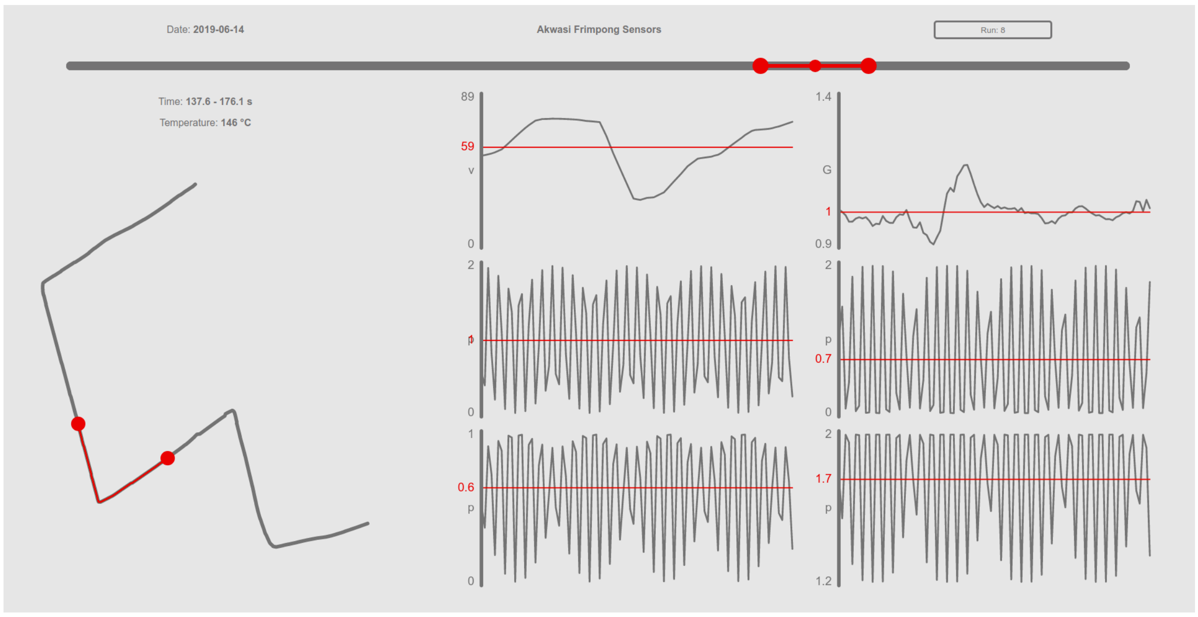


Fig. 5.2: Second page of the user interface. The visible data is placeholder data and therefore not representative of a skeleton run.

The middle and right panel of the second page are different from the first page. There are six graphs showing the fluctuations of the velocity, g-force and all pressure points over the selected time range. The pressure data that is visible consists of placeholder data now, because the whole system was not integrated yet at the moment this thesis was written.

5.3. Functionality

When the web page loads, it request the HTML string of the microprocessor. This string contains all HTML and CSS. The HTML defines all components present on the web page: text elements, such as the date; input ranges, such as the time sliders and so-called canvasses for the interaction buttons and the graphical renderings of the data. All components get an `id` for the JavaScript to be able to interact. The CSS decides on the colouring and sizes of everything, this makes sure all is visible on small and bigger screens. The CSS is used to make the time sliders work as well. The slider on the second page consists of three stacked sliders and the CSS makes sure it looks like one.

When the HTML page loads, it requests */script.js* as mentioned in §4.2.4 as well. The JavaScript first defines some variables and useful functions. When the whole HTML is loaded, a certain function is triggered, which contains the rest of the functionalities. The page switching mechanic using the title on top is fixed, the play button and forward and backward button are drawn and the actual functionality behind the play button is implemented.

The current date is sent to the web server using the date request*/date* mentioned in §4.2.4, so all runs without the correct date can be fixed. Then, a list of runs is loaded from */list* to fill the run selector in the upper right corner of the pages. The server side of this request is covered in §4.2.4. The last run of this list is loaded using */file*, mentioned in §4.2.4 as well. When all data is loaded, all time sliders reset and the images are drawn in the canvasses using the requested data; all text fields are updated as well. These steps also occur when selecting another time point in the time slider: when keeping the mouse pressed and sliding through time, this can happen multiple times a second. The redrawing also happens in real-time when pressing the play button. The graphical data will update as fast as it actually happened during the run down the track. This should be roughly 40 Hz, since this is the frequency all sensors are read, as can be read in §3.3. Depending on the mobile device, this can be a bit slower, but it still gives a good representation. Selecting another run will request the corresponding data from the server after which the sliders reset again and everything will be redrawn.

Clicking the title only hides one page and shows the other, clicking again does the opposite. The second page works roughly the same as the first. Every time the time slider changes, the canvasses are redrawn, which can happen multiple times a second. The second page also has to do some calculations to fit the data in the window as well; a red line is drawn to show the average of the data as well.

5.4. System size reduction

The whole JavaScript has been designed in such a way that it reuses the code as much as possible by creating functions and global variables. UglifyJS[1] is used to compress the script as much as possible by renaming variables to shorter ones and by removing all unnecessary characters. This reduced the size of the script from 23 kilobytes to 6.7 kilobytes. This makes the whole system a tad faster, since it lowers the time it takes to load the web page. The HTML and CSS are stripped from any unnecessary white spaces for the same reason.

[1]https://github.com/mishoo/UglifyJS2

6 Discussion

Since the subsystem covered by this thesis is at the end of the chain of this project so to say, it requires a lot from the other subsystems to work. The system integration literally combines the other two subsystems on the software side, and the visualisation needs data to execute. Therefore, it is a little harder to test this subsystem on its own. This chapter describes the test plan of the system and the integration of this subsystem in the rest of the instrumented sled project.

6.1. Visualisation

The visualisation has been developed locally by running the local HTML file in the browser. An external web server has been used to request the list and files from. Eventually, this web server has been used to serve the HTML and such from as well. This way, the interface can also be tested on mobile devices. Former versions of the interface were not apt for smaller devices, because half the page would be filled with the controls instead of the actual data, for example. This problem has been solved by moving some components around on the page.

6.2. Communication

Later on, it was possible to integrate the visualisation with some data files uploaded to the SD card. This way, the Wi-Fi and web server were being tested as well. It turned out that the microprocessor could not handle more than one request at the time, which led to some problems that have since been solved. It can still cause problems when accessing the server with multiple devices at the same time, but since that has not been required, it is not considered a major issue.

Another problem that occurred with the communication is that the Wi-Fi sometimes causes the whole system to crash, and then restart. This issue sometimes comes up when reprogramming the microprocessor and it has not become clear where it comes from. This is something that would require more time to fix.

6.3. System integration

The tests for the system integration could start when some of the sensors started to work. The GNSS and IMU were available at some point to test: after attaching all components together and implementing the code, it was possible to drive around and test the location, rotation and g-force sensing. The location was pretty

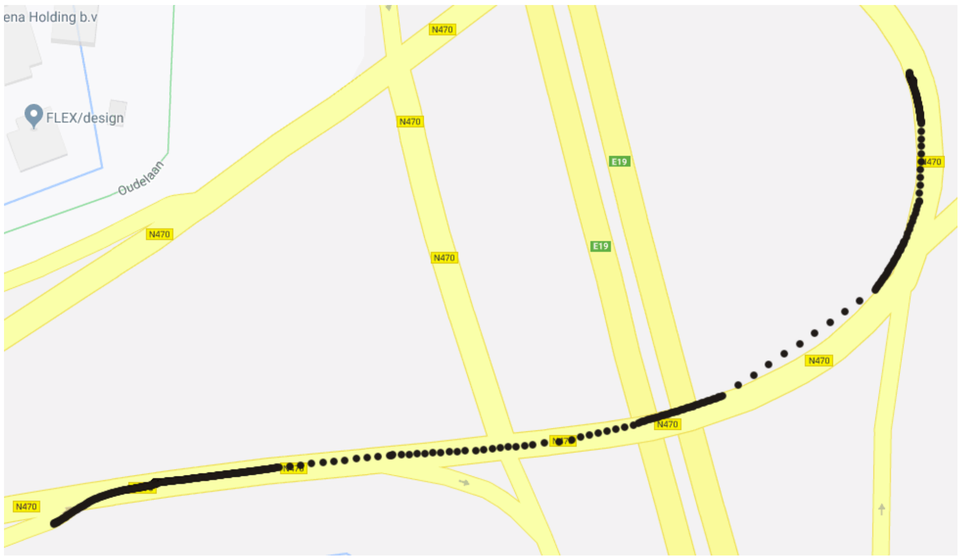


Fig. 6.1: Location data gathered while driving around Delft, laid over Google Maps. The spots with less frequent dots is where the GNSS sensor fails.

accurate, but the GNSS sensor sometimes loses connection and it occurs that it takes way too much time for the GNSS sensor to acquire the location, which is evident from Fig. 6.1. The spots where the frequency of the dots is significantly lower is where the GNSS sensor loses connection. The straight line of dots is merely the running average trying to compensate for the failing sensor.

The implementation of the rotation and g-forces was wrong at first, as became clear from the acquired data while driving around Delft. This was eventually easily fixed by adjusting some factors. Afterwards, the results were very good and the system, without pressure and ice temperature sensors, worked. At the moment of writing this thesis, it was not yet possible to implement the pressure sensors in the system.

To increase the accuracy of the location, a Kalman filter is being implemented, combining readings from the GNSS sensor and the IMU; this is detailed in the thesis of Van Dijk and Van der Werff [9]. This filter needs test data to work, so the system integration was tested in some roller coasters to achieve correct functioning.

7 Conclusion

After exploring different ways of visualising the acquired data, a web page appeared to be the best way to implement this product. This way the athlete can simply visit the web page to inspect his achievements during his practice runs. This meant using an ESP32 as a Wi-Fi access point and as a web server. The sensor data is saved to an SD card during the run, and afterwards it is immediately available on a web page hosted on the local Wi-Fi network. Most requirements that were postulated in Chapter 2 have been met using this system:

GR.1 The product measures location, g-forces, rotation, applied pressure by the athlete and ice temperature; this is captured in the other two subsystems [8] [9], so all necessary quantities are measured.

GR.6 Apart from an issue in the GNSS sensor, the sensor readout rate is 45 Hz, which satisfies the minimum of 41 Hz—this is covered in §3.3. The problem with the GNSS sensor is explained in more detail in the thesis of subgroup B [9].

GR.7 To make sure the sled does not have to be opened in between runs, there is a control button outside the sled. Furthermore, the data inspection occurs wirelessly, so that does not involve accessing the circuit inside the sled either.

GR.12 The data should be available within five minutes after finishing the run. With the Wi-Fi solution it is actually available practically immediately, so this requirement is complied with. It will take half a minute at most for the data to be downloaded by the mobile device.

GR.13 The product is easy to use, since starting the measurement only requires pressing a button and inspection of the data simply works on a tablet or mobile phone with easy controls.

GR.14 The total costs of making the prototype should be within the budget of €250. This requirement was not met: the overall costs of the prototype are approximately €400.

SR.1 The Wi-Fi and web page solution make it possible that the data can be accessed from a mobile device.

SR.2 Other athletes cannot access the data, since the Wi-Fi network is secured with a password.

SR.3 All acquired data is visible in a pleasant and interactive manner on any device that can connect to Wi-Fi.

SR.4 Previous runs are inspectable using the selector in the upper right corner.

SR.5 The first page of the interface uses a time slider to scroll through the data in the time dimension.

SR.6 The second page makes it possible to see the fluctuations of certain quantities over a period of time.

The other requirements were mostly covered by one of the other subgroups. The system has been tested at g-forces of up to approximately 5 *g*, which is covered in more detail by Heller and De Jong [8]. The system was able to acquire data in these conditions, but due to an error in the code the measured data could not easily be interpreted. This means that it cannot be verified whether requirement GR.3 is met. It can be said that the final product will be more robust than the prototype used for testing: this prototype was not mounted on the dedicated PCB designed by subgroup B [9], while this will be the case with the final product. The prototype “survived” tests that were performed in a number of roller coasters: therefore, the even more robust final product will partially satisfy requirement GR.11.

The prototype has not yet been developed to such a state that it can withstand the forces exerted on a skeleton sled, so GR.8 has not yet been complied with either. Since the means to measure the influence of the temperature of the environment in which the product will be used and the vibrations on the track due to irregularities in the surface of the ice are not available, it cannot be checked whether requirements GR.2 and GR.11 are fully met.

The dimensional requirements listed in GR.5 and GR.4 have been met: the dimensions of the part of the product that should fit inside the sled are 11.5 cm × 12.5 cm × 1.8 cm and the weight of the total system is 261 grams. The product, however, does have a minimal, but present, influence on the aerodynamics of the sled due to parts of the system being mounted on the outside of the sled, such as the GNSS sensor [9] and the switch for starting the run. Also, the weight of the sled is thus impacted slightly; this is practically unavoidable. This means the product fails to meet requirement GR.9 entirely.

7.1. Recommendations

It has not been possible to combine all subsystems mentioned in §1.4, due to the long time spent waiting for some needed components to arrive and the resulting time problems. The pressure sensors and the ice temperature sensors have not been implemented with the rest of the system. It was not possible to test the feature automatically starting measurement based on the velocity either, because the GNSS sensor (supplying the velocity data) had some issues to overcome first. It was not possible to test the system in the environment it is eventually meant for either. Using the system on a bobsled track might result in some unexpected behaviour, which would then have to be solved as well.

Future recommendations are therefore testing to make sure the GNSS sensor works without the quirk of taking several seconds to determine its location every once in a while. The location improvement algorithm through the use of a Kalman filter has not been implemented either. This probably makes a big difference on the precision of the location as well, but requires some more tweaking beforehand. The system also needs a more robust casing to make sure it withstands the use on an actual sled. It is now merely a prototype, which cannot be used by the athlete immediately.

The visualisation could use some extra features as well. This will of course become clear once the athlete actually starts using the system and can pinpoint the useful and useless parts. One addition to the visualisation could be to be able to compare two runs directly. A more three-dimensional display of the sled could improve the user experience as well, but has not yet been implemented, for simplicity reasons.

References

[1] I. Roberts, “Skeleton Bobsleigh Mechanics: Athlete-Sled Interaction”, PhD dissertation, Univ. Edinburgh, Edinburgh, United Kingdom, 2013.

[2] C. Sawade, S. Turnock, A. Forrester, and M. Toward, “Assessment of an Empirical Bob-Skeleton Steering Model”, *Procedia Engineering*, vol. 72, pp. 447–452, 2014, The Engineering of Sport 10, ISSN: 1877-7058. DOI: https://doi.org/10.1016/j.proeng.2014.06.078. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877705814005943.

[3] S. Xiaochen. (Jun. 1, 2018). Plans Unveiled for Building All 2022 Olympic Venues, [Online]. Available: http://www.chinadaily.com.cn/a/201806/01/WS5b1081d9a31001b82571d8c3.html (visited on 06/21/2019).

[4] F. Braghin, F. Cheli, S. Maldifassi, S. Melzi, and E. Sabbioni, *The Engineering Approach to Winter Sports*. Springer, 2016, ISBN: 978-1-4939-3019-7. DOI: 10.1007/978-1-4939-3020-3.

[5] F. M. Impellizzeri, A. La Torre, G. Merati, E. Rampinini, and C. Zanoletti, “Relationship Between Push Phase and Final Race Time in Skeleton Performance”, *J. Strength Cond. Res.*, vol. 20, no. 3, pp. 579–583, 2006.

[6] A. Baca, P. Dabnichki, M. Heller, and P. Kornfeind, “Ubiquitous Computing in Sports: A Review and Analysis”, *J. Sports Sciences*, vol. 27, no. 12, pp. 1335–1346, Oct. 2009. DOI: 10.1080/02640410903277427.

[7] S. Lee, T. Kim, S. Lee, S. Kil, and S. Hong, “Development of Force Measurement System of Bobsled for Practice of Push-off Phase”, *Proc. IMechE Part P: J. Sports Engineering and Technology*, vol. 229, no. 3, pp. 192–198, 2015. DOI: 10.1177/1754337114565383.

[8] M. J. Heller and A. J. de Jong, “Instrumented Skeleton Sled: Focusing on Force and Orientation Sensing”, BSc thesis, Delft Univ. Technol., Delft, Netherlands, 2019.

[9] W. M. van Dijk and K. N. van der Werff, “Instrumented Skeleton Sled: Focusing on Power Management and the Sensors for Localisation, Velocity and Temperature”, BSc thesis, Delft Univ. Technol., Delft, Netherlands, 2019.

[10] SD Association. (Jun. 27, 2018). SD Express – A Revolutionary Innovation for SD Memory Cards, SD Association, [Online]. Available: https://www.sdcard.org/press/SD_EXPRESS_A_REVOLUTIONARY_INNOVATION_FOR_SD_MEMORY_CARDS.pdf (visited on 06/19/2019).

[11] (2019). SD Standards Family, [Online]. Available: https://www.sdcard.org/developers/overview/family/index.html (visited on 06/19/2019).

[12] A. Maier, A. Sharp, and Y. Vagapov, “Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things”, in *2017 Internet Technol. and Appl. (ITA)*, Sep. 2017, pp. 143–148. DOI: 10.1109/ITECHA.2017.8101926.

[13] *ESP32 series*, ESP32, Version 3, Espressif Systems, 2019.

[14] J. Mankar, C. Darode, K. Trivedi, M. Kanoje, and P. Shahare, “Review of I2C Protocol”, *Int. J. Res. Advent Technol.*, vol. 2, no. 1, pp. 474–479, Jan. 2014, ISSN: 2321-9637.

[15] *I2C-Bus*, UM10204, Rev. 6, NXP Semiconductors, Apr. 2014.

[16] *SD Specifications*, Part 1: Physical Layer, Simplified Specification, Version 6.00, SD Card Association, Aug. 2018.

[17] R. Moskowitz, D. Karrenberg, Y. Rekhter, E. Lear, and G. J. de Groot, *Address Allocation for Private Internets*, RFC 1918, Feb. 1996. DOI: 10.17487/RFC1918. [Online]. Available: https://rfc-editor.org/rfc/rfc1918.txt.

[18] J. Reynolds and J. Postel, *Assigned Numbers*, RFC 1340, Jul. 1992. DOI: 10.17487/RFC1340. [Online]. Available: https://rfc-editor.org/rfc/rfc1340.txt.

A
Code

This appendix contains all code written for the software integration system and the visualisation system.

A.1. Software integration system

A.1.1. Main code

```
#include "Akwasi.h"

void setup() {
    // Initialise the I2C connection
    Wire.begin(I2C_SDA_PIN, I2C_SCL_PIN);

    // Stall if either SD, GPS or IMU is not connected
    while (!SD.begin(SD_SELECT_PIN) || !myGPS.begin() || myIMU.readByte(MPU9250_ADDRESS, WHO_AM_I_MPU9250)
      != 0x71) UVP();

    // Setup the GPS
    myGPS.setNavigationFrequency(GPS_HZ, 0);
    myGPS.setI2COutput(COM_TYPE_UBX);
    myGPS.saveConfiguration();

    // Setup the IMU
    pinMode(IMU_INT_PIN, INPUT);
    digitalWrite(IMU_INT_PIN, LOW);
    myIMU.MPU9250SelfTest(myIMU.SelfTest);
    myIMU.calibrateMPU9250(myIMU.gyroBias, myIMU.accelBias);
    myIMU.initMPU9250();
    myIMU.initAK8963(myIMU.magCalibration);

    // Setup the thermometer
    myTherm.begin();

    // Setup the external ADC for the pressure data
    pinMode(ADC_SELECT_PIN, OUTPUT);
    digitalWrite(ADC_SELECT_PIN, HIGH);

    // Initialise buttons, LEDs and buzzers
    pinMode(RUN_BTN_PIN, INPUT);
    pinMode(RUN_LED_PIN, OUTPUT);
    pinMode(BAT_BUZZ_PIN, OUTPUT);
    pinMode(BAT_READ_PIN, INPUT);

    // initial state of the run start button pin
    buttonState = digitalRead(RUN_BTN_PIN);
    digitalWrite(RUN_LED_PIN, LOW);

    // Setup the web server and its request points
    server.on("/", HTTP_GET, homePage);
    server.on("/script.js", HTTP_GET, scriptFile);
    server.on("/list", HTTP_GET, listFiles);
    server.on("/file", HTTP_GET, showFile);
    server.on("/date", HTTP_GET, setDate);
    server.onNotFound(pageNotExplicit);
    server.begin();
}

void loop() {
    // Turn on the Wi-Fi
    WiFi.mode(WIFI_AP);
    WiFi.softAPConfig(ip, ip, IPAddress(255, 255, 255, 0));
    WiFi.softAP("Akwasi Frimpong", "secretpassword");

    // Wait for the start run button to be pressed
    // Keep the battery buzzer running and the web server responsive
    while (digitalRead(RUN_BTN_PIN) == buttonState) {
        UVP();
        server.handleClient();
    }
    delay(BOUNCE_DELAY_MS);
    buttonState = !buttonState;
    digitalWrite(RUN_LED_PIN, HIGH);

    // Turn off the Wi-Fi
    WiFi.disconnect();
    WiFi.mode(WIFI_OFF);

    // Write sensor data to the SD until the button is pressed again
    writeSD();
}
```

A.1.2. Header code

```
#ifndef AKWASI_H
#define AKWASI_H

// include the relevant libraries
#include <WebServer.h>
#include <SD.h>
#include <Smoothed.h>
#include <Wire.h>
#include <Adafruit_MLX90614.h>
#include <MPU9250.h>
#include <SparkFun_Ublox_Arduino_Library.h>
#include "quaternionFilters.h"

// button settling time in milliseconds
#define BOUNCE_DELAY_MS 25

// Amount of sensor data output
#define SENSORS 14

// Chunk size of the HTTP responses
#define CHUNK 100

// speed threshold at which the run measurement stops, in km/h
#define SPEED_THRESHOLD 1

// minimum sensor data rates in Hz
#define GPS_HZ 18
#define IMU_GYRO_HZ 4000 // 4k for the gyro, 8k for the accelerometer
#define IMU_ACC_HZ 8000
#define THERM_HZ 10000 // between 10k and 100k

```

```
// actual (used) sensor readout rate in Hz
#define READOUT_RATE_HZ 45

// ADC sample rate in Hz
#define ADC_RATE 6000

// ESP32 pin definitions
#define SD_SELECT_PIN 26
#define ADC_SELECT_PIN 5
#define RUN_BTN_PIN 39
#define RUN_LED_PIN 32
#define BAT_BUZZ_PIN 21
#define BAT_READ_PIN 34
#define I2C_SDA_PIN 23
#define I2C_SCL_PIN 22
#define IMU_INT_PIN 12

// server declarations
IPAddress ip(10, 0, 0, 0);
WebServer server(80);

// default date
String date = "2019-06-14";

// infrared sensor variables
const uint8_t temperatureAddress = 0x5A;

// variable indicating led button pin state
bool buttonState;

// variable to save

// global variables for undervoltage protection
bool UVP_buzz = LOW;
bool UVP_flag = true;
unsigned long previousUVPtime = 0;
float UVP_interval = 2000;

// global sensor objects
MPU9250 myIMU;
SFE_UBLOX_GPS myGPS;
Adafruit_MLX90614 myTherm(temperatureAddress);

#endif
```

A.1.3. IMU code

```
// Prepare the data of the IMU using all kinds of calculations
void prepareIMU() {
    myIMU.readAccelData(myIMU.accelCount);
    myIMU.getAres();

    myIMU.ax = (float)myIMU.accelCount[0] * myIMU.aRes;
    myIMU.ay = (float)myIMU.accelCount[1] * myIMU.aRes;
    myIMU.az = (float)myIMU.accelCount[2] * myIMU.aRes;

    myIMU.readGyroData(myIMU.gyroCount);
    myIMU.getGres();

    myIMU.gx = (float)myIMU.gyroCount[0] * myIMU.gRes;
    myIMU.gy = (float)myIMU.gyroCount[1] * myIMU.gRes;
    myIMU.gz = (float)myIMU.gyroCount[2] * myIMU.gRes;

    myIMU.readMagData(myIMU.magCount);
    myIMU.getMres();

    myIMU.magbias[0] = +470.;
    myIMU.magbias[1] = +120.;
    myIMU.magbias[2] = +125.;

    myIMU.mx = (float)myIMU.magCount[0] * myIMU.mRes * myIMU.magCalibration[0] -
               myIMU.magbias[0];
    myIMU.my = (float)myIMU.magCount[1] * myIMU.mRes * myIMU.magCalibration[1] -
               myIMU.magbias[1];
    myIMU.mz = (float)myIMU.magCount[2] * myIMU.mRes * myIMU.magCalibration[2] -
               myIMU.magbias[2];

    myIMU.updateTime();

    MahonyQuaternionUpdate(myIMU.ax, myIMU.ay, myIMU.az, myIMU.gx * DEG_TO_RAD,
                           myIMU.gy * DEG_TO_RAD, myIMU.gz * DEG_TO_RAD,
                           myIMU.my, myIMU.mx, myIMU.mz, myIMU.deltat);

                           const float* q = getQ();

    myIMU.yaw = atan2(2.0f * (q[1] * q[2] + q[0] * q[3]),
                      q[0] * q[0] + q[1] * q[1] - q[2] * q[2] - q[3] * q[3]);
    myIMU.pitch = -asin(2.0f * (q[1] * q[3] - q[0] * q[2]));
    myIMU.roll = atan2(2.0f * (q[0] * q[1] + q[2] * q[3]),
                       q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3]);

    myIMU.pitch *= RAD_TO_DEG;
    myIMU.yaw *= RAD_TO_DEG;
    myIMU.yaw -= 8.5;
    myIMU.roll *= RAD_TO_DEG;

    myIMU.sumCount = 0;
    myIMU.sum = 0;
}
```

A.1.4. ADC code

```
double adcRead(int channel) {
    // Enable ADC reading from SPI
    digitalWrite(ADC_SELECT_PIN, LOW);

    // Request data
    SPI.transfer(0x06 | (channel >> 2));
    int firstrequest = SPI.transfer(channel << 6);
    int secondrequest = SPI.transfer(0);

    // Disable ADC reading from SPI
    digitalWrite(ADC_SELECT_PIN, HIGH);

    // Turn raw data into kilograms
    return 17.6066 - 0.00482858262 * ((((firstrequest | 0xE0) & 0x0F) << 8) + secondrequest);
}
```

A.1.5. File read code

```
// The JSON format
String jsonGlue[] = {"{\"t\":", ",\"p\":[", ",",        ",", ",",
                     ",",        "],\"v\":", ",\"g\":[", ",", "],\"r\":[",
                     ",",        "],\"c\":", ",\"l\":[", ",", "]}"};

// The amount of decimals to be sent to the web client
int json7[] = {1, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 7, 7};

// Whether a running average should be applied to the sensor data
bool jsonToSmooth[] = {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};

// A function that imports data from the SD, applies a running average where
// necessary and sends the data to the web client in chunks
bool sendFile(String fileName) {
    // If the file does not exist it returns false which will respond with 404
    if (!SD.exists(fileName)) return false;

    File file = SD.open(fileName, FILE_READ);

    // Start the HTTP chunk encoding
    server.setContentLength(CONTENT_LENGTH_UNKNOWN);
    server.send(200, "application/json", "");

    // Create enough Smooth objects for the running averages
    int arrayIndex[SENSORS];
    int nSmooth = 0;
    int nRegular = 0;
    for (uint8_t i = 0; i < SENSORS; i++) {
        if (jsonToSmooth[i]) {
            arrayIndex[i] = nSmooth++;
        } else {
            arrayIndex[i] = nRegular++;
        }
    }
    nSmooth++;
    nRegular++;

    double regularValues[nRegular];
    for (uint8_t i = 0; i < nRegular; i++) {
        regularValues[i] = 0.0;
    }

    Smoothed<double> smoothValues[nSmooth];
    for (uint8_t i = 0; i < nSmooth; i++) {
        smoothValues[i].begin(SMOOTHED_AVERAGE, 10);
    }

    // Format the date and run number
    String chunk = "{\"date\":\"";
    chunk += fileName.substring(1, 11);
    chunk += "\",\"run\":";
    chunk += fileName.substring(12, 13);
    chunk += ",\"data\":[";

    String value = "";

    uint8_t iCol = 0;
    unsigned int iRow = 0;

    // While not at the end of the file, keep reading
    while (file.available()) {
        char c = file.read();

        if (c == '\t' || c == '\n') {
            // Prevent corrupted files breaking the system
            if (iCol >= SENSORS) break;

            // Parse string from the SD to a real number
            if (!value.equals("")) {
                if (jsonToSmooth[iCol]) {
                    smoothValues[arrayIndex[iCol]].add(value.toDouble());
                } else {
                    regularValues[arrayIndex[iCol]] = value.toDouble();
                }
            }

            value = "";
            iCol++;

            // If at the end of the file, add whole parsed string to the chunk
            if (c == '\n') {
                // Prevent corrupted files breaking the system
                if (iCol < SENSORS) break;

                iCol = 0;

                // Add comma if not first datapoint
                if (iRow > 0) chunk += ",";

                // Parse all data to JSON
                chunk += jsonGlue[0];
                for (uint8_t i = 0; i < SENSORS; i++) {
                    double val;
                    if (jsonToSmooth[i]) {
                        val = smoothValues[arrayIndex[i]].get();
                    } else {
                        val = regularValues[arrayIndex[i]];
                    }

                    // Check if value is valid
                    if (isnan(val) || isinf(val)) {
                        chunk += "null";
                    } else {
                        chunk += String(val, json7[i]);
                    }
                    chunk += jsonGlue[i + 1];
                }

                // Send data per 100 rows
                if (!(++iRow % CHUNK)) {
                    server.sendContent(chunk);
                    chunk = "";
                }
            }
        } else if (c != '\r') {
            value += c;
        }
    }

    file.close();

    chunk += "]}";

    server.sendContent(chunk);
    server.sendContent("");  // end transmission

    return true;
}

// Determines the new file name
String getFileName() {
    uint8_t runNo = 0;
    String file = "";

    do {
        runNo++;
        file = String("/" + date + "-" + runNo + ".txt");
    } while (SD.exists(file));

    return file;
}
```

A.1.6. File write code

```
void writeSD() {
    // Open the file to be written
    File file = SD.open(getFileName(), FILE_WRITE);

    // Initialise some time variables
    // And the string that will be written to the SD
    unsigned long startTime = millis();
    int lastWrittenTime = 0;
    String line = "";

    // Wait for the velocity to pass the threshold
    while (myGPS.getGroundSpeed(0) / 1000.0 * 3.6 < SPEED_THRESHOLD);

    // Keep writing the data to the SD until the button is pressed
    while (digitalRead(RUN_BTN_PIN) == buttonState && myGPS.getGroundSpeed(0) / 1000.0 * 3.6 >
     SPEED_THRESHOLD) {
        if (millis() - startTime - lastWrittenTime > 1000 / READOUT_RATE_HZ) {
            // Initialise the current data of the IMU
            prepareIMU();

            // Write the time to the SD
            lastWrittenTime = millis() - startTime;
            line = String(lastWrittenTime) + '\t';

            // Write the pressure data to the SD
            line += String(adcRead(0)) + '\t';
            line += String(adcRead(1)) + '\t';
            line += String(adcRead(2)) + '\t';
            line += String(adcRead(3)) + '\t';
            line += String(adcRead(4)) + '\t';

            // Write the velocity to the SD
            line += String(myGPS.getGroundSpeed(0) / 1000.0 * 3.6, 7) + '\t';

            // Write both acceleration measurements to the SD
            line += String(myIMU.ay, 7) + '\t';
            line += String(myIMU.az, 7) + '\t';

            // Write both rotation measurements to the SD
            line += String(myIMU.pitch, 7) + '\t';
            line += String(myIMU.roll, 7) + '\t';

            // Write the placeholder temperature data to the SD
            line += "146\t";

            // Write both location coordinates to the SD
            // If the data is of high enough quality
            if (myGPS.getSIV(0) > 3) {
                line += String(myGPS.getLatitude(0) / 10000000.0, 7);
                line += '\t';
                line += String(myGPS.getLongitude(0) / 10000000.0, 7);
            } else {
                line += '\t';
            }

            file.println(line);
        }
    }
    delay(BOUNCE_DELAY_MS);
    buttonState = !buttonState;
    digitalWrite(RUN_LED_PIN, LOW);

    // Close file
    file.close();
}
```

A.1.7. Web server code

The `html` and `js` variables in the second and fifth line are simplified strings of the code shown in §A.2, respectively the HTML code and JavaScript code.

```
// Serve the web page itself
void homePage() { server.send(200, "text/html", html); }

// Serve the script file
void scriptFile() { server.send(200, "text/javascript", js); }

// Serve the list of available files of individual runs in JSON
void listFiles() {
    File root = SD.open("/");
    File entry = root.openNextFile();

    String page = "[";

    while (entry) {
        String fileName = entry.name();
        fileName.remove(0, 1);
        fileName.remove(fileName.length() - 4, 4);

        page += ",\"";
        page += fileName;
        page += "\"";

        entry.close();
        entry = root.openNextFile();
    }

    root.close();

    page.remove(1, 1);
    page += "]";

    server.send(200, "application/json", page);
}

// Serve a specific run file in JSON
void showFile() {
    String fileName;
    String content;

    if (server.hasArg("run")) {
        fileName = "/";
        fileName += server.arg("run");
        fileName += ".txt";
    }

    if (!fileName || !SD.exists(fileName) || !sendFile(fileName)) {
        server.send(404, "application/json", "null");
    }
}

// Set the date to the date sent over from the web client
void setDate() {
    if (server.hasArg("date")) {
        String getDate = server.arg("date");

        if (getDate.length() == 10) {
            date = getDate;
        }

        String fileName;
        for (uint8_t i = 0; i < 99; i++) {
            fileName = "/2019-06-14-" + String(i) + ".txt";

            if (SD.exists(fileName)) {
                SD.rename(fileName, getFileName());
            }
        }
    }

    server.send(200, "text/plain", "true");
}

// Throw an error when another page is requested
void pageNotExplicit() { server.send(404, "text/plain", "Not found!"); }
```

A.1.8. Undervoltage code

```
// This function makes the under voltage buzzer signal
// the user if the battery is low
void UVP() {
    unsigned long currentMillis = millis();
    float Vbat = 3.6 * 3.2 / 4095 * analogRead(BAT_READ_PIN);
    const float Valarm = 7.2;

    if (Vbat <= Valarm && Vbat > Valarm - 0.15) {
        UVP_flag = false;
        if (currentMillis - previousUVPtime >= UVP_interval) {
            previousUVPtime = currentMillis;
            if (UVP_buzz == LOW) {
                UVP_buzz = HIGH;
                UVP_interval = 200;
            } else {
                UVP_buzz = LOW;
                UVP_interval = 4000;
            }
            digitalWrite(BAT_BUZZ_PIN, UVP_buzz);
        }
    } else if (Vbat <= Valarm - 0.15 && Vbat > Valarm - 0.3) {
        UVP_flag = false;
        if (currentMillis - previousUVPtime >= UVP_interval) {
            previousUVPtime = currentMillis;
            if (UVP_buzz == LOW) {
                UVP_buzz = HIGH;
                UVP_interval = 500;
            } else {
                UVP_buzz = LOW;
                UVP_interval = 2000;
            }
            digitalWrite(BAT_BUZZ_PIN, UVP_buzz);
        }
    } else if (Vbat <= Valarm - 0.3) {
        if (!UVP_flag) {
            UVP_interval = 500;
            UVP_flag = true;
        }
        if (currentMillis - previousUVPtime >= UVP_interval) {
            previousUVPtime = currentMillis;
            if (UVP_buzz == LOW) {
                UVP_buzz = HIGH;
                if (UVP_interval > 30) {
                    UVP_interval = UVP_interval - 0.7;
                }
            } else {
                UVP_buzz = LOW;
                if (UVP_interval > 30) {
                    UVP_interval = UVP_interval - 0.7;
                }
            }
            digitalWrite(BAT_BUZZ_PIN, UVP_buzz);
        }
    } else {
        digitalWrite(BAT_BUZZ_PIN, LOW);
    }
}
```

A.2. Visualisation

A.2.1. HTML code

The `base64 data` mentioned in line 8 has been omitted, because it is unreadable to a human. It represents the icon of the page in the browser.

```
1 <!DOCTYPE html>
2 <html>
3
4 <head>
5     <title>Akwasi Sensors</title>
6     <script src="index.js"></script>
7     <link rel="shortcut icon"
8         href="data:image/jpeg;base64,... (base64 data) ..." />
9     <style>
```

```
    body {
        font-family: sans-serif;
        background-color: #e7e7e7;
        color: #747474;
        text-align: center;
    }

    #head {
        display: flex;
    }

    #title {
        cursor: pointer;
    }

    .select {
        margin: 25px 0;
    }

    select {
        width: 30%;
        height: 30px;
        margin-top: 10px;
        background-color: #e7e7e7;
        outline: none;
        border: 3px solid #747474;
        border-radius: 5px;
        color: #747474;
        size: 20px;
        -webkit-appearance: none;
        text-align-last: center;
        cursor: pointer;
    }

    select::-ms-expand {
        display: none;
    }

    select:disabled {
        background-color: #747474;
        color: #e7e7e7;
    }

    .canvasses {
        width: 100%;
        display: flex;
    }

    .third,
    .twothird {
        width: 33.33%;
        padding: 10px 0;
        box-sizing: border-box;
    }

    .twothird {
        width: 66.66%;
    }

    #n-controls {
        width: 90%;
        display: flex;
        margin: 10px 5%;
    }

    #n-slider {
        -webkit-appearance: none;
        width: 100%;
        margin-left: 10px;
        margin-top: 6px;
        height: 14px;
```

```
            background: #747474;
            outline: none;
            border-radius: 7px;
        }

        #n-slider::-webkit-slider-thumb {
            -webkit-appearance: none;
            appearance: none;
            width: 26px;
            height: 26px;
            background: #e92100;
            border-radius: 13px;
            cursor: pointer;
        }

        #n-slider::-moz-range-thumb {
            width: 26px;
            height: 26px;
            background: #e92100;
            border-radius: 13px;
            cursor: pointer;
        }

        .icon {
            margin-right: 15px;
            cursor: pointer;
        }

        .g-controls {
            width: 90%;
            height: 26px;
            margin: 10px 5%;
            position: relative;
        }

        .g-controls input {
            pointer-events: none;
            position: absolute;
            left: -4px;
            outline: none;
            -webkit-appearance: none;
            width: 100%;
            margin-top: 6px;
            height: 14px;
            background: #747474;
            border-radius: 7px;
        }

        .g-controls input::-webkit-slider-thumb {
            pointer-events: all;
            position: relative;
            z-index: 2;
            -webkit-appearance: none;
            appearance: none;
            width: 26px;
            height: 26px;
            background: #e92100;
            border-radius: 13px;
            cursor: pointer;
        }

        .g-controls input::-moz-range-thumb {
            pointer-events: all;
            z-index: 10;
            width: 26px;
            height: 26px;
            background: #e92100;
            border-radius: 13px;
            cursor: pointer;
        }

        #g-slider-m::-moz-range-thumb {
            z-index: 9;
            width: 20px;
            height: 20px;
        }

        #g-slider-m::-webkit-slider-thumb {
            z-index: 1;
            width: 20px;
            height: 20px;
        }

        #g-line {
            background-color: #e92100;
            z-index: 0;
            height: 6px;
            position: absolute;
            margin-top: 10px;
        }
    </style>
</head>

<body>
    <div class="all">
        <div class="canvasses">
            <div class="third">
                <p>Date: <b id="date"></b></p>
            </div>
            <div class="third">
                <p id="title"><b>Akwasi Frimpong Sensors</b></p>
            </div>
            <div class="third">
                <select id="select"></select>
            </div>
        </div>
        <div id="normal" style="display: block;">
            <div id="n-controls">
                <canvas class="icon" id="n-back" width="16" height="26"></canvas>
                <canvas class="icon" id="n-play" width="26" height="26"></canvas>
                <canvas class="icon" id="n-forw" width="16" height="26"></canvas>
                <input type="range" id="n-slider" autofocus>
            </div>
            <div class="canvasses">
                <div class="third">
                    <p>Time: <b id="n-time"></b> <b>s</b></p>
                    <p>Temperature: <b id="n-temperature"></b> <b>&deg;C</b></p>
                    <canvas id="n-location"></canvas>
                </div>
                <div class="third">
                    <p>G-force: <b id="n-force"></b><b>G</b></p>
                    <p>Slope: <b id="n-angle"></b><b>&deg;</b></p>
                    <canvas id="n-gforce"></canvas>
                </div>
                <div class="third">
                    <p>Velocity: <b id="n-velocity"></b> <b>km/h</b></p>
                    <p>Pressure: <b id="n-pressure"></b></p>
                    <canvas id="n-board"></canvas>
                </div>
            </div>
        </div>
        <div id="graphs" style="display: none;">
            <div class="g-controls">
                <input id="g-slider-a" type="range">
                <input id="g-slider-b" type="range">
                <input id="g-slider-m" type="range">
                <div id="g-line"></div>
            </div>
            <div class="canvasses">
                <div class="third">
                    <p>Time: <b id="g-time-left"></b> <b>-</b> <b id="g-time-right"></b> <b>s</b></p>
                    <p>Temperature: <b id="g-temperature"></b> <b>&deg;C</b></p>
                    <canvas id="g-location"></canvas>
                </div>
                <div class="twothird">
                    <canvas id="g-graph"></canvas>
                </div>
            </div>
        </div>
    </div>
</body>

</html>
```

A.2.2. JavaScript code

```
// Define some variables
var jsDoc = document;
var jsWindow = window;
var math = Math;
var min = math.min;
var max = math.max;
var pow = math.pow;
var PI = math.PI;
var abs = math.abs;
var colorGrey = "#747474";
var colorRed = "#e92100";
var sRound = "round";
var normal = true;
var playing = true;
var delay = 10;

// Define some useful functions
function spreadArray(fn, arr) {
    return arr.reduce(function (x, y) {
        return fn(x, y);
    });
}
function mapper(arr, a, b, fn) {
    return arr.map(function (x) {
        return (fn || Number)(x[a][b] !== undefined ? x[a][b] : x[a]);
    });
}
function average(arr) {
    return arr.reduce(function (a, b) {
        return a + b;
    }, 0) / arr.length;
}
function pythagoras(arr) {
    return math.sqrt(pow((arr[0]), 2) + pow((arr[1]), 2));
}

function decimalRound(number, decimals) {
    return math.round(number * pow(10, decimals || 0)) / pow(10, decimals || 0);
}

function getHTML(name) {
    return jsDoc.getElementById(name);
}
function getCanvas(name) {
    return getHTML(name).getContext("2d");
}
function setHTML(name, data) {
    getHTML(name).innerHTML = data;
}

function xmlRequest(url, fn) {
    var request = new XMLHttpRequest;
    request.open("GET", url);
    request.onreadystatechange = function () {
        if (request.readyState !== 4 || request.status !== 200) return;

        fn(JSON.parse(request.responseText));
    }
    request.send();
}

function clearRect(ctx, x, y, width, height) {
    ctx.clearRect(x, y, width, height);
}
function beginPath(ctx) {
    ctx.beginPath();
}
function rotate(ctx, angle) {
    ctx.rotate(angle);
}
function lineTo(ctx, x, y) {
    ctx.lineTo(x, y);
}
function moveTo(ctx, x, y) {
    ctx.moveTo(x, y);
}
function arc(ctx, x, y, radius, startAngle, endAngle, anticlockwise) {
    if (radius) ctx.arc(x, y, radius, startAngle, endAngle, anticlockwise);
}
function stroke(ctx) {
    ctx.stroke();
}
function fill(ctx) {
    ctx.fill();
}
function arcTo(ctx, x1, y1, x2, y2, radius) {
    ctx.arcTo(x1, y1, x2, y2, radius);
}
function fillText(ctx, text, x, y) {
    ctx.fillText(text, x, y);
}
function setLineWidth(ctx, width) {
    ctx.lineWidth = width;
}
function setStrokeStyle(ctx, color) {
    ctx.strokeStyle = color;
}
function setFillStyle(ctx, color) {
    ctx.fillStyle = color;
}
function onclick(obj, fn) {
    obj.canvas.onclick = fn;
}
function oninput(obj, fn) {
    obj.oninput = fn;
}
function getValue(obj) {
    return obj.value;
}
function setValue(obj, val) {
    obj.value = val;
}
var assign = Object.assign;

function validLocations(timeslot) {
    return timeslot.l[0] && timeslot.l[1];
}

// When the HTML page loads, this will trigger
jsWindow.onload = function () {
    // More variables
    var runSelector = getHTML("select");
    var timeSlider = getHTML("n-slider");
    var iconCanvasses = ["n-back", "n-play", "n-forw"].map(getCanvas);
    var imageCanvasses = ["n-gforce", "n-location", "n-board"].map(getCanvas);
    var timeSliders = ["g-slider-a", "g-slider-b", "g-slider-m"].map(getHTML);
var timeLine = getHTML("g-line");
var locationCanvas = getCanvas("g-location");
var graphCanvas = getCanvas("g-graph");
var styles = ["block", "none"];
var allData = {};
var currentData;
var canvasWidth;
var canvasHeight;
var graphHeight;
var canvasSize;
var lineWidth;
var locationWidthOffset;
var locationHeightOffset;
var locationFactor;
var rotationFactor;
var boardFactor;
var length;
var slidersWidth;

// Make the switching of pages possible
getHTML("title").onclick = function () {
    getHTML("normal").style.display = styles[normal | 0];
    normal = !normal;
    getHTML("graphs").style.display = styles[normal | 0];
}

// Settings for the control button canvasses
iconCanvasses.forEach(function (canvas) {
    assign(canvas, {
        fillStyle: colorRed,
        strokeStyle: colorRed,
        lineWidth: 5,
        lineCap: sRound
    });
});

// Draw the control button canvasses
beginPath(iconCanvasses[0]);
arc(iconCanvasses[0], 13, 13, 10.5, .55 * PI, -.55 * PI);
stroke(iconCanvasses[0]);
onclick(iconCanvasses[0], function () {
    timeSlider.value--;
    timeSlider.oninput();
});

beginPath(iconCanvasses[2]);
arc(iconCanvasses[2], 3, 13, 10.5, -.45 * PI, .45 * PI);
stroke(iconCanvasses[2]);
onclick(iconCanvasses[2], function () {
    timeSlider.value++;
    timeSlider.oninput();
});

// The play button functionality
function iterate() {
    timeSlider.value++;
    if (playing) {
        if (getValue(timeSlider) !== timeSlider.max) {
            setTimeout(iterate, delay);
        } else {
            goPlay();
            timeSlider.value = 0;
        }
    }
    timeSlider.oninput();
}
function goPlay() {
    playing = !playing;
    clearRect(iconCanvasses[1], 0, 0, 26, 26);
    beginPath(iconCanvasses[1]);

```

```
        if (playing) {
            arc(iconCanvasses[1], 13, 13, 10.5, 0, 2 * PI);
            stroke(iconCanvasses[1]);

            if (getValue(timeSlider) === timeSlider.max) {
                timeSlider.value = 0;
            }

            iterate();
        } else {
            arc(iconCanvasses[1], 13, 13, 13, 0, 2 * PI);
            fill(iconCanvasses[1]);
        }
    }
    onclick(iconCanvasses[1], goPlay);
    goPlay();

    // Play when pressing `space`
    jsDoc.addEventListener("keydown", function (keyEvent) {
        if (keyEvent.keyCode == 32 && normal) {
            goPlay();
        }
    });

    // Set variables upon opening and resizing the page
    jsWindow.onresize = function () {
        canvasWidth = jsWindow.innerWidth * .3;
        canvasHeight = jsWindow.innerHeight - 250;
        canvasWidth = canvasWidth;
        graphHeight = canvasHeight / 3 + 28;
        canvasSize = min(canvasWidth, canvasHeight);
        lineWidth = canvasSize * 0.01;
        slidersWidth = jsWindow.innerWidth * .9;

        var widthRange = mapper(currentData, "l", 0);
        var heightRange = mapper(currentData, "l", 1);
        var locationWidthRange = [spreadArray(max, widthRange), spreadArray(min, widthRange)];
        var locationHeightRange = [spreadArray(max, heightRange), spreadArray(min, heightRange)];

        locationWidthOffset = .5 * (locationWidthRange[0] + locationWidthRange[1]);
        locationHeightOffset = .5 * (locationHeightRange[0] + locationHeightRange[1]);
        locationFactor = min(canvasWidth / (locationWidthRange[0] - locationWidthRange[1]), canvasHeight / (locationHeightRange[0] - locationHeightRange[1])) / 1.1;
        rotationFactor = canvasSize / 3 / max(spreadArray(max, mapper(currentData, "g", 0, abs)), spreadArray(max, mapper(currentData, "g", 1, abs)));
        boardFactor = canvasSize * .06 / spreadArray(max, currentData.map(function (timeslot) { return spreadArray(max, timeslot.p); }));

        imageCanvasses.forEach(function (canvas) {
            canvas.canvas.width = canvasWidth;
            canvas.canvas.height = canvasHeight;
            canvas.translate(canvasWidth / 2, canvasHeight / 2);
        });

        graphCanvas.canvas.width = canvasWidth * 2;
        graphCanvas.canvas.height = graphHeight * 3;

        locationCanvas.canvas.width = canvasWidth;
        locationCanvas.canvas.height = canvasHeight;
        locationCanvas.translate(canvasWidth / 2, canvasHeight / 2);

        [imageCanvasses[0], imageCanvasses[1], imageCanvasses[2], graphCanvas, locationCanvas].forEach(function (canvas) {
            assign(canvas, {
                lineWidth: lineWidth,
                lineCap: sRound,
                lineJoin: sRound,
                strokeStyle: colorGrey,
                fillStyle: colorRed
            });
        });

        assign(graphCanvas, {
            font: canvasSize / 30 + "px sans-serif",
            fillStyle: colorGrey,
            textAlign: "right",
            textBaseline: "middle"
        });

        timeSlider.oninput();
        timeSliders[2].oninput();
    }

    // Reset time sliders on loading a new run
    function newRun(data) {
        currentData = data.filter(validLocations);

        length = currentData.length - 1

        assign(timeSlider, {
            min: 0,
            max: length,
            value: 0
        });

        assign(timeSliders[0], {
            min: 0,
            max: length,
            value: 0
        });

        assign(timeSliders[1], {
            min: 0,
            max: length,
            value: length
        });

        assign(timeSliders[2], {
            min: 0,
            max: length,
            value: length / 2
        });

        jsWindow.onresize();
        timeSlider.focus();
    }

    // Load run on selecting a new one from the upper right selector
    oninput(runSelector, function () {
        var currentRun = getValue(runSelector);

        if (currentRun in allData) {
            newRun(allData[currentRun]);
        } else {
            runSelector.disabled = true;

            xmlRequest("https://tijsmoree.nl/akwasi/file/?run=" + currentRun, function (data) {
                runSelector.disabled = false;

                setHTML("date", data.date);

                newRun(allData[currentRun] = data.data);
            });
        }
    });

    // Redraw the first page on changing the time slider of the first page
    oninput(timeSlider, function () {
        var currentTime = getValue(timeSlider);
        var currentTimeslot = currentData[currentTime];
        var gForceAngle = math.atan(currentTimeslot.g[0] / currentTimeslot.g[1]);
        var angle = -currentTimeslot.r[0] * PI / 180;
    var gForce = pythagoras(currentTimeslot.g);
    delay = currentTime < length - 1 ? currentData[Number(currentTime) + 1].t - currentTimeslot.t : 10;

    setHTML("n-temperature", decimalRound(currentTimeslot.c, 1));
    setHTML("n-time", decimalRound((currentTimeslot.t - currentData[0].t) / 1e3, 1));
    setHTML("n-force", decimalRound(gForce, 1));
    setHTML("n-velocity", decimalRound(currentTimeslot.v, 1));
    setHTML("n-angle", decimalRound(-currentTimeslot.r[1], 1));
    setHTML("n-pressure", decimalRound(average(currentTimeslot.p), 1));

    setLineWidth(imageCanvasses[0], lineWidth);
    clearRect(imageCanvasses[0], -canvasWidth / 2, -canvasHeight / 2, canvasWidth, canvasHeight);
    beginPath(imageCanvasses[0]);
    rotate(imageCanvasses[0], angle);
    arc(imageCanvasses[0], 0, -.17 * canvasSize, .22 * canvasSize, 0, PI * 2);
    stroke(imageCanvasses[0]);
    clearRect(imageCanvasses[0], -.5 * canvasSize, -.29 * canvasSize, canvasSize, .29 * canvasSize);
    beginPath(imageCanvasses[0]);
    lineTo(imageCanvasses[0], -.45 * canvasSize, 0);
    lineTo(imageCanvasses[0], .45 * canvasSize, 0);
    moveTo(imageCanvasses[0], -.3 * canvasSize, 0);
    lineTo(imageCanvasses[0], -.3 * canvasSize, .05 * canvasSize);
    moveTo(imageCanvasses[0], .3 * canvasSize, 0);
    lineTo(imageCanvasses[0], .3 * canvasSize, .05 * canvasSize);
    moveTo(imageCanvasses[0], -.15 * canvasSize, 0);
    arc(imageCanvasses[0], -.118 * canvasSize, 0, .3 * canvasSize, PI, PI * 1.63);
    moveTo(imageCanvasses[0], .3 * canvasSize, 0);
    arc(imageCanvasses[0], .118 * canvasSize, 0, .3 * canvasSize, 0, PI * 1.37, true);
    stroke(imageCanvasses[0]);
    rotate(imageCanvasses[0], gForceAngle - angle);
    beginPath(imageCanvasses[0]);
    moveTo(imageCanvasses[0], 0, 0);
    setLineWidth(imageCanvasses[0], lineWidth / 2);
    lineTo(imageCanvasses[0], 0, gForce * rotationFactor);
    lineTo(imageCanvasses[0], -.02 * canvasSize, max(0, gForce * rotationFactor - .03 * canvasSize));
    moveTo(imageCanvasses[0], 0, gForce * rotationFactor);
    lineTo(imageCanvasses[0], .02 * canvasSize, max(0, gForce * rotationFactor - .03 * canvasSize));
    stroke(imageCanvasses[0]);
    beginPath(imageCanvasses[0]);
    rotate(imageCanvasses[0], -gForceAngle);
    arc(imageCanvasses[0], 0, 0, lineWidth * 2, 0, PI * 2, true);
    fill(imageCanvasses[0]);

    clearRect(imageCanvasses[1], -canvasWidth / 2, -canvasHeight / 2, canvasWidth, canvasHeight);
    beginPath(imageCanvasses[1]);
    currentData.forEach(function (timeslot) {
        lineTo(imageCanvasses[1], (timeslot.l[0] - locationWidthOffset) * locationFactor, (timeslot.l[1] - locationHeightOffset) * locationFactor);
    });
    stroke(imageCanvasses[1]);
    beginPath(imageCanvasses[1]);
    arc(imageCanvasses[1], (currentTimeslot.l[0] - locationWidthOffset) * locationFactor, (currentTimeslot.l[1] - locationHeightOffset) * locationFactor, lineWidth * 2, 0, PI * 2, true);
    fill(imageCanvasses[1]);

    clearRect(imageCanvasses[2], -canvasWidth / 2, -canvasHeight / 2, canvasWidth, canvasHeight);
    setStrokeStyle(imageCanvasses[2], colorGrey);
    beginPath(imageCanvasses[2]);
    moveTo(imageCanvasses[2], -.12 * canvasSize, -.45 * canvasSize);
    imageCanvasses[2].quadraticCurveTo(0, -.38 * canvasSize, .12 * canvasSize, -.45 * canvasSize);
    arcTo(imageCanvasses[2], .25 * canvasSize, -.45 * canvasSize, .25 * canvasSize, 0, .05 * canvasSize);
    arcTo(imageCanvasses[2], .2 * canvasSize, .45 * canvasSize, 0, .45 * canvasSize, .05 * canvasSize);
    arcTo(imageCanvasses[2], -.2 * canvasSize, .45 * canvasSize, -.2 * canvasSize, 0, .05 * canvasSize);
    arcTo(imageCanvasses[2], -.25 * canvasSize, -.45 * canvasSize, 0, -.45 * canvasSize, .05 * canvasSize);
    lineTo(imageCanvasses[2], -.12 * canvasSize, -.45 * canvasSize);
    stroke(imageCanvasses[2]);
```

```
        setStrokeStyle(imageCanvasses[2], colorRed);
        beginPath(imageCanvasses[2]);
        arc(imageCanvasses[2], -.15 * canvasSize, -.35 * canvasSize, currentTimeslot.p[0] * boardFactor,
 0, 2 * PI);
        stroke(imageCanvasses[2]);
        beginPath(imageCanvasses[2]);
        arc(imageCanvasses[2], .15 * canvasSize, -.35 * canvasSize, currentTimeslot.p[2] * boardFactor, 0,
 2 * PI);
        stroke(imageCanvasses[2]);
        beginPath(imageCanvasses[2]);
        arc(imageCanvasses[2], -.1 * canvasSize, .35 * canvasSize, currentTimeslot.p[1] * boardFactor, 0,
 2 * PI);
        stroke(imageCanvasses[2]);
        beginPath(imageCanvasses[2]);
        arc(imageCanvasses[2], .1 * canvasSize, .35 * canvasSize, currentTimeslot.p[3] * boardFactor, 0, 2
 * PI);
        stroke(imageCanvasses[2]);
    });

    // Redraw the second page on changing the sliders on the second page
    function tripleInput() {
        var range = abs(getValue(timeSliders[1]) - getValue(timeSliders[0])) / 2;
        var mid = min(length - range, max(getValue(timeSliders[2]), range));

        setValue(timeSliders[0], mid - range);
        setValue(timeSliders[1], mid + range);
        setValue(timeSliders[2], mid);

        var left = min(getValue(timeSliders[0]), getValue(timeSliders[1]));
        var right = max(getValue(timeSliders[0]), getValue(timeSliders[1])) + 1;
        var dataRange = currentData.slice(left, right);
        var end = dataRange.length - 1;

        assign(timeLine.style, {
            left: left / length * (slidersWidth - 25) + 5 + "px",
            right: ((length - right) / length + 0.01) * (slidersWidth - 25) + "px"
        });

        setHTML("g-temperature", decimalRound(average(mapper(dataRange, "c")), 1));
        setHTML("g-time-left", decimalRound((dataRange[0].t - currentData[0].t) / 1e3, 1));
        setHTML("g-time-right", decimalRound((dataRange[end].t - currentData[0].t) / 1e3, 1));

        clearRect(locationCanvas, -canvasWidth / 2, -canvasHeight / 2, canvasWidth, canvasHeight);
        setLineWidth(locationCanvas, lineWidth);
        setStrokeStyle(locationCanvas, colorGrey);
        beginPath(locationCanvas);
        currentData.forEach(function (timeslot) {
            lineTo(locationCanvas, (timeslot.l[0] - locationWidthOffset) * locationFactor, (timeslot.l[1]
 - locationHeightOffset) * locationFactor);
        });
        stroke(locationCanvas);
        beginPath(locationCanvas);
        arc(locationCanvas, (dataRange[0].l[0] - locationWidthOffset) * locationFactor, (dataRange[0].l[1]
 - locationHeightOffset) * locationFactor, lineWidth * 2, 0, PI * 2, true);
        fill(locationCanvas);
        beginPath(locationCanvas);
        arc(locationCanvas, (dataRange[end].l[0] - locationWidthOffset) * locationFactor, (dataRange[end].
l[1] - locationHeightOffset) * locationFactor, lineWidth * 2, 0, PI * 2, true);
        fill(locationCanvas);
        beginPath(locationCanvas);
        setLineWidth(locationCanvas, lineWidth / 2);
        setStrokeStyle(locationCanvas, colorRed);
        dataRange.forEach(function (timeslot) {
            lineTo(locationCanvas, (timeslot.l[0] - locationWidthOffset) * locationFactor, (timeslot.l[1]
 - locationHeightOffset) * locationFactor);
        });
        stroke(locationCanvas);

        clearRect(graphCanvas, 0, 0, canvasWidth * 2, graphHeight * 3);
        [
            [0, "v", "v"],
            [1, "p", "p", 0],
            [1, "p", "p", 1],
            [1, "G", "g", null, pythagoras],
            [1, "p", "p", 2],
            [1, "p", "p", 3]
        ].forEach(function (args, index) {
            var thisData = mapper(currentData, args[2], args[3], args[4]);
            var maxData = spreadArray(max, thisData);
            var minData = spreadArray(min, thisData);

            var mid = average(thisData.slice(left, right));
            var midTranslated = graphHeight * (index % 3 + .5) + ((maxData + minData) / 2 - mid) * (graphHeight - lineWidth * 6) / (maxData - minData);

            beginPath(graphCanvas);
            setLineWidth(graphCanvas, lineWidth / 2);
            thisData.slice(left, right).forEach(function (timeslot, i) {
                lineTo(graphCanvas,
                    lineWidth * 12 + (canvasWidth - lineWidth * 13) / end * i + (index > 2 ? canvasWidth : 0),
                    graphHeight * (index % 3 + .5) + ((maxData + minData) / 2 - timeslot) * (graphHeight - lineWidth * 6) / (maxData - minData)
                );
            });
            stroke(graphCanvas);

            beginPath(graphCanvas);
            setStrokeStyle(graphCanvas, colorRed);
            setLineWidth(graphCanvas, lineWidth / 3);
            moveTo(graphCanvas, lineWidth * 12 + (index > 2 ? canvasWidth : 0), midTranslated);
            lineTo(graphCanvas, lineWidth * 12 + (canvasWidth - lineWidth * 13) + (index > 2 ? canvasWidth : 0), midTranslated);
            stroke(graphCanvas);

            beginPath(graphCanvas);
            setStrokeStyle(graphCanvas, colorGrey);
            setLineWidth(graphCanvas, lineWidth);
            moveTo(graphCanvas, lineWidth * 12 + (index > 2 ? canvasWidth : 0), graphHeight * (index % 3) + lineWidth * 2);
            lineTo(graphCanvas, lineWidth * 12 + (index > 2 ? canvasWidth : 0), graphHeight * (index % 3 + 1) - lineWidth * 2);
            stroke(graphCanvas);

            setFillStyle(graphCanvas, colorGrey);
            fillText(graphCanvas,
                decimalRound(maxData, args[0]),
                lineWidth * 10 + (index > 2 ? canvasWidth : 0),
                graphHeight * (index % 3) + lineWidth * 3
            );
            fillText(graphCanvas,
                args[1],
                lineWidth * 10 + (index > 2 ? canvasWidth : 0),
                graphHeight * (index % 3 + .5)
            );
            fillText(graphCanvas,
                decimalRound(minData, args[0]),
                lineWidth * 10 + (index > 2 ? canvasWidth : 0),
                graphHeight * (index % 3 + 1) - lineWidth * 3
            );
            setFillStyle(graphCanvas, colorRed);
            fillText(graphCanvas,
                decimalRound(mid, args[0]),
                lineWidth * 10 + (index > 2 ? canvasWidth : 0),
                midTranslated
            );
        });
    };
    oninput(timeSliders[0], tripleInput);
    oninput(timeSliders[1], tripleInput);
    oninput(timeSliders[2], tripleInput);

    // Send the current date to the microprocessor and request the list to fill the run selector
    xmlRequest("https://tijsmoree.nl/akwasi/date/?date=" + (new Date()).toJSON().slice(0, 10), function ()
      {
        xmlRequest("https://tijsmoree.nl/akwasi/list/", function (list) {
            var sortedList = list.sort().reverse();

            sortedList.map(function (file) {
                return file.slice(0, -2);
            }).filter(function (date, i, arr) {
                return arr.indexOf(date) === i;
            }).forEach(function (date) {
                var optgroup = jsDoc.createElement("optgroup");
                optgroup.label = date;
                sortedList.filter(function (file) {
                    return file.slice(0, -2) === date;
                }).forEach(function (file) {
                    var option = optgroup.appendChild(jsDoc.createElement("option"));
                    option.value = file;
                    option.innerHTML = "Run: " + file.slice(-1);
                });
                runSelector.appendChild(optgroup);
            });

            runSelector.oninput();
        });
    });
}
```