Optimising Measurements using Low-Power Headworn Sensors

Jeffrey Pronk



TU Delft

Optimising Measurements using Low-Power Headworn Sensors

by

Jeffrey Pronk

to obtain the degree of Master of Science
in Computer Science, Software Technology track, at the Delft University of Technology,
to be defended publicly on Tuesday June 18th, 2024 at 10:00 AM.

Student number:	4839617	
Project duration:	November 13, 2023 – June 18, 2024	
Thesis committee:	dr. Przemysław Pawełczak	TU Delft, Thesis advisor
	Vivian Dsouza, M. Sc.	TU Delft, Daily co-supervisor
	dr. Burcu Özkan	TU Delft, External committee member

The contents of this thesis are confidential and cannot be made public until June 18th, 2026.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

When I first met Przemek and Vivian to discuss a possible master thesis, I could have never imagined what I signed up for. I would have never thought I would be working on a research project that would be published as part of a paper. After a lot of work, I think we published a paper we can be proud of. I would like to thank my supervisors, Przemysław Pawełczak and Vivian Dsouza, for the trust they put into me to deliver and to voice my opinions. I am thankful for the opportunity they gave me. Because of how we worked, it sometimes felt more like we were co-workers, rather then a student trying to finish their thesis and two supervisors overseeing the process. I would also like to thank Burcu Özkan for joining my thesis committee on short notice.

Finally, I would like to thank my family, my mom and dad, my sister Stefanie and my girlfriend Amber, for their love and support.

Jeffrey Pronk
Delft, June 2024

Contents

1 Introduction **1**

2 Background **3**

2.1 Related work . . . 3

2.1.1 Jaw movement analysis . . . 3

2.1.2 Tongue movement analysis . . . 3

2.1.3 Body position analysis during sleep . . . 3

2.1.4 Head concussion detection . . . 4

2.1.5 Dental applications . . . 4

2.1.6 Body temperature analysis . . . 4

2.1.7 Respiratory rate and sleep analysis . . . 4

2.1.8 Saliva analysis . . . 4

2.2 Densor background . . . 4

2.3 Hardware overview . . . 6

2.3.1 Accelerometer and temperature sensor . . . 6

2.3.2 Photo-diode . . . 7

2.3.3 NFC interface . . . 7

2.3.4 Power storage and management . . . 7

2.3.5 MCU . . . 7

2.3.6 Real-time clock . . . 7

2.4 Novel applications of Densor . . . 8

2.4.1 Bruxism detection . . . 8

2.4.2 Mouth opening state . . . 8

2.4.3 Head position while laying . . . 8

2.4.4 Sleep stage detection . . . 8

2.4.5 Fluid intake analysis . . . 9

2.4.6 Speaking detection . . . 9

2.4.7 Orthodontic compliance . . . 9

3 Problem statement **10**

3.1 Functional requirements . . . 10

3.1.1 Determining sample times . . . 10

3.1.2 Run control . . . 10

3.1.3 Charge indication . . . 10

3.1.4 Data compression . . . 11
3.1.5 Voltage-drop compensation . . . 11
3.1.6 Data labeling . . . 11
3.2 Non-functional requirements . . . 11
3.3 Research questions . . . 12

4 Software solutions **13**
4.1 Timestamping . . . 13
4.2 Charge indication . . . 14
4.3 Run control . . . 15
4.4 Data compression . . . 16
4.5 Voltage-drop compensation . . . 16
4.6 Data labeling . . . 17

5 System design **19**
5.1 Firmware . . . 19
5.2 Android app . . . 20
5.3 Data labeling . . . 22
5.3.1 Experiment master . . . 22
5.3.2 Data master . . . 22
5.3.3 Syncing with Apple Watch . . . 23
5.3.4 Ear-mounted accelerometer labeling . . . 23
5.4 Machine learning models . . . 23

6 Experiment methodology **25**
6.1 Data analysis and classification . . . 25
6.2 Electrical characteristics and operation lifetime . . . 26
6.3 Firmware size . . . 26
6.4 Labeled data collection for classification . . . 26
6.5 Sleep stage classification . . . 28
6.6 Comparison between ear and jaw acceleration . . . 29
6.7 Test subjects . . . 30

7 Results **31**
7.1 Electrical characteristics and operation lifetime . . . 31
7.2 Firmware size . . . 32
7.3 Classifications . . . 32
7.4 Comparison between ear and jaw acceleration . . . 35

8 Conclusion **37**

8.1 How power- and memory-efficient can the firmware for Densor be made, while fitting all set requirements? . . . 37

8.2 How can the sensors of Densor be utilized to detect the following classification classes? . 37

8.3 What is the difference in signal quality and classification capability between Densor and a comparable ear-mounted accelerometer in detecting mouth opening state? . . . 38

9 Recommendations **39**

9.1 Higher sampling rates . . . 39

9.2 Variable sampling rates . . . 39

9.3 On-board classification . . . 39

9.4 Firmware updates through NFC . . . 40

9.5 System interrupts . . . 40

A Data compression **41**

1 Introduction

Body worn sensors, such as smartwatches and smart-glasses allow for a variety of health related data to be recorded. Smartwatches, for example, are capable of detecting breathing patterns during sleep by detecting micro-movements of the wrist [1]. Another example of body worn sensors are ear-mounted sensors, like the OpenEarable [2], which can detect chewing patterns from the ear-canal. There is however a lack of sensors that can do measurements from within the mouth. This is unfortunate, as the mouth can be a rich source of health related information. Sensors mounted inside the mouth could detect jaw movement to determine grinding, detect water intake to monitor drinking habits and sense a variety of signals determine sleep quality.

Creating an intra-oral sensing platform is challenging. An intra-oral sensing platform that can be widely adopted has to meet to following requirements. First, it should be small in size, as it needs to fit within the mouth without obstructing the user. It should also be self contained, making it more comfortable to wear, as users will wear the systems longer if they are more comfortable. This also means the system should not contain external components that are wired through the mouth opening, as this will be less comfortable. Next, the platform should be easy to use and should not need specialists to operate. This would help in sleep studies as patients currently have to be supervised overnight by a professional. As intra-oral sensors should be mounted in the mouth, it should also be safe to use. It should not contain big batteries to minimize the risk of fires. It should also be designed to minimize the risk of shocks. Finally, it should be long lasting and should be able to record meaningful data for a full night of sleep. Such a system should be able to record when samples are taken, to give the samples meaning. Although some active and passive intra-oral sensing platforms do exist, to the best of our knowledge—they do not meet all the aforementioned requirements.

Therefore Densor is introduced, a small flexible PCB with multiple sensors that can be attached to a commodity dental aligner. The Densor contains a temperature sensor, an accelerometer and a photo-diode. The Densor has an on-board Microcontroller Unit (MCU) and a Real-Time Clock (RTC) that enables the Densor to completely shutdown in-between readings. This saves power, which is at a premium since the system is powered by two 11 mF capacitors. These capacitors are charged by a custom Near Field Communication (NFC) antenna which is embedded into the Densor's PCB. This NFC tag also handles the storage of all samples taken by the system and act as the only interface with the system to obtain data and charge the capacitors.

The first version of Densor was build by Christian Peppelman in 2022. The system contains an accelerometer and temperature sensors. It has capacitors for power storage. The energy of the system is harvested using a Mercury energy harvester which uses two Thermal Electric Generators (TEGs) to gain energy through temperature differences while drinking. It features an NHS3152 MCU and an NFC antenna for communication. The measurements are stored on an FRAM memory module. Although the results of this version seemed promising, it was not able to measure for a full night of sleep, after which the project was transferred to Vivian Dsouza, who build the second version of Densor in 2023. The development of this version was helped by the contributions presented in this thesis.

The main focus of the second version is on using NFC to harvest energy, rather then a thermal electric generator. Instead of FRAM, it contains an NFC tag that can both be used for storage and energy harvesting. This version isolates the power and introduces an RTC, which should significantly reduce power consumption [3], [4]. It also includes a photo-diode, over the previous system. Both versions of the system were supervised by Przemysław Pawełczak who also helped in the development of both systems.

Although the hardware was already created by the supervisors of this thesis and mostly working, the software was still lacking. Some small projects were made by the supervisors of this thesis to test individual sensors, but no overall software eco-system was created. Therefore, this thesis focused on designing the software eco-system for Densor from the ground up. This included creating firmware, an android app to charge and control the Densor as well as being able to export data from its NFC chip. On top of this, a system to handle and label data, generated by Densor, should be created. This labeled data can then be used to research how the sensors can be utilized.

Designing the software, and especially the firmware, for Densor is a more challenging task then one might expect. The firmware has to fit within the MCU's 16 KB of flash storage and may only use 2 KB of RAM. Besides, the system is designed to be ultra-low power, so the longer the firmware takes to make a sample, the less amount of sample the system can take in a single life-span. As the amount of power storage is limited, it is crucial that the amount of time the firmware takes to make a sample is limited. Besides this, there is a multitude of design challenges, which will be introduced in Chapter 3.

After designing the software, multiple experiments have to be performed to collect data from the Densor. This data should then be accurately labeled, such that it can be discovered how the sensors can be utilized to perform a multitude of classifications. These classifications then give an indication of the effectiveness of the hardware and will present the researchers how to proceed with the development of the system.

The main research questions for this thesis is *"How can the software for Densor be made as efficient as possible and how can its sensors be utilized to maximize activity detection effectiveness?"*, which results in more specific research questions posed in Section 3.3.

The remainder of this thesis will introduce the background of the Densor and will give some inside into the possible capabilities of the system in Section 2.4. Some related work will be presented in Section 2.1, which will showcase the importance of the Densor project. Then, the thesis will focus on the problems of the system that need to be solved in software, followed by non-functional requirements of the system in Chapter 3. From this, the research questions will be presented in Section 3.3, which were drawn from the aforementioned problems and requirements. Then, solutions to the aforementioned problems will be presented in Chapter 4, followed by the overall design of the software eco-system in Chapter 5. Finally, the methodology of all experiments will be described in Chapter 6 before presenting the results of all the experiments in Chapter 7, followed by conclusions on the results in Chapter 8 and a presentation of possible future work and recommendations in Chapter 9.

2

Background

As mentioned before, there is a lack of platforms that allow a user to measure signals from within the mouth to collect health related data on a daily basis. For example, in sleep related studies a test subject has to come into a special facility where they have to spend the night. The test subject will then be wired to monitoring systems and has to be video monitored by a researcher through the night. This is a highly privacy invasive and expensive way of research that only yields data for a signal night of sleep.

This chapter will first present some related work showcasing the importance of Densor like platforms in Section 2.1. Next, the background of Densor will be presented in Section 2.2, followed by an overview of Densor's hardware in Section 2.3. Finally, the possible novel applications of Densor will be introduced in Section 2.4.

2.1. Related work

2.1.1. Jaw movement analysis

Jaw movement and pressure applied on the jaw can be a rich source of information when it comes to researching bruxism. Passive approaches to analyse bruxism do exist, but these approaches do not give insight into when a subject was grinding their teeth. An example of such approaches is [5].

Active approaches, like [6], also exist. These systems rely on EMG and ECG data. They can only be used once after which the electrodes have to be replaced. On top of that, they are quiet cumbersome as the user will have to wear electrodes with wires attached. These electrodes are attached to both the users chest and head.

Jaw movement can also be instrumental in speaking detection. Speaking can already be detected by audio recording apps like [7]. This approach can be privacy invasive for both the test subjects and other people around them as these apps can record any sound made.

2.1.2. Tongue movement analysis

Tongue thrusting is another condition that can be monitored with intra-oral sensing platforms. To detect tongue thrusting and biting, sensor can be included on an intra-oral sensing platform to detect tongue movement. Platforms that detect tongue thrusting from within the mouth include [8].

2.1.3. Body position analysis during sleep

Body position during sleep provides insightful information for sleep studies. Platforms like [9] use accelerometers in smartwatches to determine the body position during sleep. These platforms can also measure breathing related micro-movements. These can be used to estimate breathing patterns.

2.1.4. Head concussion detection

Athletes of impact-heavy sports could benefit from mouth-worn accelerometers to measure accelerations induced by impacts. These accelerations could cause concussions. In-ear accelerometers are already being used in motor sports to measure the acceleration of a drivers head in the event of a crash [10]. Such accelerometers help determine the magnitude of a crash. This helps in researching and developing new safety measures to make motor sports safer and helps medical staff in assessing head injuries of drivers.

Since the drivers already wear ear-pieces to protect their hearing and for communication with their team, they do not need extra equipment to do these measurements. Accelerometers can be embedded into the ear-pieces that are already used for the aforementioned use cases. A similar approach can be applied to sports where athletes wear mouth guards and have a high risk of head injuries. By applying an accelerometer to the mouth guards, head injuries can be more easily assessed by collecting acceleration data during the impact. Examples of such mouth-worn accelerometers are [11], [12].

2.1.5. Dental applications

Intra-oral sensing platforms can be used for a variety of dental applications. Such applications include monitoring of orthodontic retainers [13] and teeth occlusion [14], [15]. Other orthodontic applications are bite pressure [16] and mouth pressure [17] sensing.

2.1.6. Body temperature analysis

Another signal that can be measured from the mouth is body temperature. By measuring body temperature during the night, users could gain insight into diseases they might have and the progression of their recovery. An example of research that monitors body temperature from within the mouth for health reasons is [18].

2.1.7. Respiratory rate and sleep analysis

As discussed in Chapter 1, wrist worn sensors can be used to determine the respiratory rate of the user while sleeping. This is done by measuring the micro-movements which result from the breathing motion. These movements are then analysed to determine the respiratory rate of the user. This can be insightful in determining the sleep stages of the user which can be used for sleep analysis. A wrist-worn system to determine these sleep stages could help a patient by being able to collect more data in a less intrusive and privacy invasive way then having to go into a clinic. Examples of such research includes [1], [19].

2.1.8. Saliva analysis

Saliva is a rich source of biological signals. An intra-oral sensing platform can do measurements on saliva-based biomarkers like pathogenic bacteria [20], lactic and uric acid [21], [22], sodium [23] and glucose [24]. These measurement can then be used to do a multitude of detections on health, performance and stress levels.

2.2. Densor background

As can be concluded from Section 2.1, creating intra-oral sensing platforms is challenging. Most of the presented systems only have a single use case. Systems like [6] are cumbersome and require wires to be attached to the user. Some systems, like [8], contain a non-rechargeable and non-replaceable battery. Some systems, such as [11] and [12] are uncomfortable to wear. From the shortcomings of the aforementioned work, a list of requirements for intra-oral sensing platforms that can be widely adopted can be concluded.

First, the system should be small and self contained. It should fit within the mouth and no wires

should go to an external system to increase comfort and minimize obstruction to the user. On top of this, the systems should be as comfortable as possible to ensure a user can wear it for multiple nights without fatigue or annoyance.

To minimize cost and increase data recorded, the system should be easy to use. As mentioned before, in current sleep studies a professional operator is necessary to setup the monitoring equipment and then monitor the patient during the night. Because of this, only a single night of sleep can be monitored. If an intra-oral sensing platform does not need any special equipment or training it can be used to collect more data than a sleep study in a clinic.

As an intra-oral system should be fully mounted in the mouth, it should not contain (potentially toxic) batteries to ensure a safe system. By not using a battery the risk of fires or chemical leaks is minimized, which could cause big issues as the system is fully mounted inside the mouth. To minimize the risk of shocks, the power storage should be completely sealed and the power rail should not be exposed. Therefore, the system should also harvest energy without exposing any charging pads or pins.

Finally, an intra-oral system should finally be able to last for a full night of sleep, while being capable of collecting a significant amount of meaningful data. To achieve this, the system should contain multiple sensors that can be used to measure a variety of tasks. It should also be possible to determine when the samples were taken for the data to be meaningful.

Since, to the best of our knowledge, no system exists that addresses all the aforementioned requirements, development of the first version of Densor was started in 2022 by Christian Peppelman and Przemysław Pawełczak [25]. This version contains three small capacitors for power storage to mitigate the risks of using a battery. It contains a Mercury energy harvester [26] and two CUI Devices CP076581 thermal electric generators (TEGs) which are used to harvest energy. These TEGs convert difference in temperature, while drinking, between water and the body of the user into usable energy. It has an STMicroelectronics LIS2DW12 accelerometer to measure accelerations of the jaw and head. The LIS2DW12 also also contains a temperature sensor. Samples are taken by an NXP NHS3152, which also contains an NFC interface for communication. The NFC interface allows the system to be completely sealed, while also providing a way to communicate. Finally, an Infineon CY15B104QI FRAM module is included for data storage. This module contains 4 Mbit of storage to save samples.

The first version of Densor was not able to measure a full night of sleep. Although energy could be harvested through the TEGs, it only provided enough power for a single measurement. The overall system became cumbersome to wear because of them. After this version, Vivian Dsouza developed the hardware for the second version of Densor together with Przemysław Pawełczak. The MCU was replaced by a separate MCU and NFC tag such that the MCU can be switched off in-between readings to save power. The TEGs were removed and replaced by the energy harvesting capability of the NFC tag. Because of this, the second version of Densor is charged using the NFC connection from a smartphone. To accommodate the energy harvesting through the NFC tag, the size of the NFC antenna was increased. An RTC is introduced and the power rail is isolated to reduce power consumption [3], [4]. A photo-diode is included to measure light intensity. This increases the amount of signals that can be measured at the same time from two to three signals. The second version of Densor is further developed and analysed by the work in this thesis. A full overview of the second version of Densor can be found in Section 2.3.

Both versions of Densor are designed such that no external tools or specialist are necessary to run the system. A user just needs their NFC enabled smartphone to setup the Densor, charge its capacitors and collect its generated data. Because the systems do not contain a battery, it is safer to use within the mouth than battery powered systems. A downside of this approach is that this results in a shorter overall lifetime. To make sure Densor can be widely adopted it needs to be mounted comfortably within the mouth. Since many people already wear aligners overnight, they are used to mount both versions of Densor. By using a commodity aligner to mount Densor they become comfortable to wear. Using these aligners, which are already widely adopted, helps the adoptability of the system. To increase comfort, it is ensured the system can sit flush on the aligner by making the the Printed Circuit Board (PCB) flexible. Because of this, it can bend to the shape of the aligner. Figure 2.1, shows the fully fabricated second version of Densor embedded onto a commodity aligner.



Figure 2.1: Picture of a fully fabricated Densor embedded onto a commodity aligner.

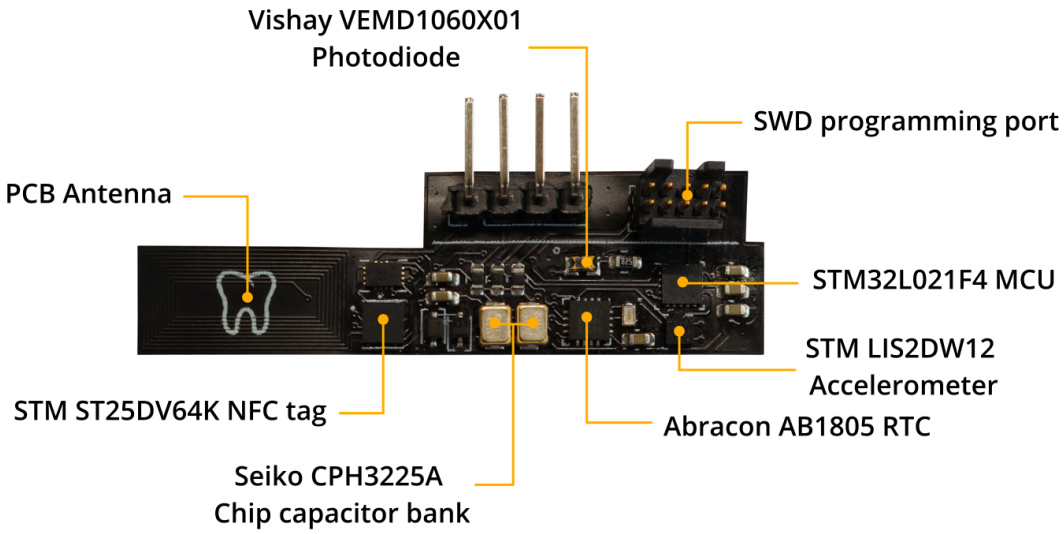


Figure 2.2: Overview of all components on the battery-free Densor.

2.3. Hardware overview

As mentioned before, Densor is an intra-oral battery-free sensing platform. It is designed to be open-source such that any researcher or dental technician can adopt it to their needs. The default setup contains three sensors, an NFC interface, an RTC and an MCU.

All components are soldered on a flexible PCB, made out of flexible polyamide substrate. The solder paste used is lead-free as this is a safer approach then using lead-based solder paste. This is important, considering the board is designed to go into the mouth for a prolonged period of time and lead-based solder paste is toxic. After soldering, the flexible PCB is mounted onto a dental aligner or retainer using a food-grade epoxy resin. By using this, the PCB can sit flush against the aligner and be water tight while also not obstructing the photo-diode from light.

The rest of this section will focus on all separate components of the Densor version used in the remainder of this paper.

2.3.1. Accelerometer and temperature sensor

An STMicroelectronics LIS2DW12 accelerometer is used as it has the ability to measure both temperature and 3-axis acceleration. This sensor can be used to measure intra-oral temperature, head rotation and head acceleration. The temperature can be read in both 8-bit and 12-bit resolution, while the accelerometer readings are supplied in 12- or 14-bit resolution. The sensor can be interfaced through an I2C interface in slave mode. The sensor supports various data rates and power modes such that the user can decide between minimized power consumption or increased performance. The sensor can

also be configured in ”single data conversion mode” which will take a single reading on command. The readings can then be requested through the I2C interface.

2.3.2. Photo-diode

Densor also contains a Vishay VEMD1060X01 photo-diode. This photo-diode can be used to measure intra-oral light intensity. The photo-diode is used in photoconductive mode and is placed as close to the top of the PCB. Because of this placement there is a lower chance of the photo-diode being covered by the lower lip. The photo-diode is connected to one of the Analog to Digital Converter (ADC) inputs of the MCU which can be used to read the sensors value.

2.3.3. NFC interface

For communication and data storage, Densor has an STMicroelectronics ST25DV64K NFC tag with a custom PCB antenna. The NFC tag has an I2C interface in slave mode for communication with the MCU. The NFC tag has 8 KB of EEPROM storage which is used as the main storage of the Densor. This storage can be accessed through both the NFC and the I2C interface. The NFC interface also supports energy harvesting which is used to charge the system. When an NFC enabled smartphone tries to communicate with the tag, it will power up the NFC side of the chip and charge the power capacitors through the energy harvesting pin.

2.3.4. Power storage and management

The energy harvested from the NFC interface is stored on two Seiko CPH3225A capacitors. These capacitors have a maximum voltage of $3.6\,\mathrm{V}$ and a capacitance of $11\,\mathrm{mF}$ This results in a total storage capacity of $0.019\,80\,\mathrm{mW}$. The power stored on these capacitors is used to power all components of the system. Two capacitors are chosen over a (rechargeable) battery as this is safer and more acceptable to use in the mouth. To keep the system as power efficient as possible, Densor does not have any complicated power management circuit. To prevent over-voltage, the system has a Fairchild Semiconductor MM3Z3V6C Zener diode, with a typical Zener voltage of $3.6\,\mathrm{V}$. The system also contains a Nexperia 1PS76SB40 Schottky diode to prevent current backfeed.

2.3.5. MCU

Densor is controlled by an STMicroelectronics STM32L021F4 MCU. This MCU is very low power and has a 16 KB flash and 2 KB of RAM. It contains an I2C interface in master mode to communicate with all other components of Densor and an ADC to read the photo-diode. It is responsible for setting up the sensors, taking samples and storing them on the NFC tag. The MCU can power itself down by calling the RTC which controls the ground connection of the MCU and all other components.

2.3.6. Real-time clock

An Abracon AB1805 RTC is used to control the power of Densor. The RTC is the only component in the systems that is always powered by the power capacitors. This is because it can power up and down all other components. The ground connection of all components are completely isolated and only connected to the RTC which can close the circuit on-demand. The RTC can be configured using an I2C interface in slave mode. Through the I2C interface, an alarm can be set. When the alarm is set, all other components are powered down until the alarm time has elapsed. When the alarm time has elapsed, all components will be powered up again.

The RTC has two different clock sources: an internal power-optimized RC oscillator and an external (XT) crystal. The internal RC oscillator runs at 128 Hz, while the XT crystal runs at 32.768 kHz. The clock source can be chosen by the user through the RTC’s I2C interface. The advantage of the RC clock is its lower power consumption, while the XT crystal provides a higher accuracy. Since the goal of Densor is to obtain a power consumption as low as possible and the RC is accurate enough for our needs, all testing in this paper was done using the RC clock. It needs to be noted that if the XT crystal

is not present on the system and the RTC is set to use the XT crystal, it can not count. This means that, because the MCU is powered down, the Densor will lock up. This lock can be reset by power-cycling the system.

To make sure no voltage is leaked through the I2C lines of the NFC interface, which may be powered through its antenna, a TS3A44159 switch is placed between the MCU and NFC interface. This switch is controlled by the MCU. When the MCU is powered down, it can not close the switch and thus, the NFC can not leak current to the MCU. This switch also controls the power of the NFC tag. Because of this, the I2C interface of the tag is powered down while the MCU is powered down. The NFC side of the tag can still be reached, as this side of the tag does not need external power to function, other then the power generated by the NFC reader.

2.4. Novel applications of Densor

Even though the initial scope of the Densor is on sleep related studies, this is not the only capability of an intraoral sensing platform. Densor could possibly be used for the following applications.

2.4.1. Bruxism detection

Densor could provide more insight than the passive systems presented in Section 2.1.1 by timestamping grinding events. This would allow researchers to fuse the bruxism events with other signals like mouth opening state and sleep stage. This way, they can possibly gain better insight into causes of bruxism and when someone grinds their teeth. On top of this, Densor is less cumbersome to use then the active systems presented in Section 2.1.1 which makes it easier to use. A user just needs their NFC enabled smartphone.

2.4.2. Mouth opening state

Mouth opening state could possibly give an indication if a subject is not grinding their teeth. If the mouth is opened, there is obviously a lower chance the subject is grinding their teeth. On top of this, sleep with the mouth opened could be an indication of sleep apnea [27]. Therefore, analysing if the mouth is opened or closed during sleep could help research into sleep apnea. By fusing this data with temperature data, this could also give insight into possible diseases by detecting a fever and detecting if someone has their mouth open during sleep and is possibly breathing through their mouth.

2.4.3. Head position while laying

Head position during sleep could be an indicator of sleep quality and sleeping habits. Although smartwatches can already detect body position, they can not tell which way the head is orientated. As head rotation can significantly influence the severity of sleep apnea [28], this could prove to be insightful information. On top of this, Densor could potentially do these measurements with sparse sampling. Because of this, the position of the head during sleep can be determined using only a single sample.

2.4.4. Sleep stage detection

Densor could possibly measure micro movements caused by breathing to determine the sleep stage, just like the wrist-mounted accelerometers mentioned in Section 2.1.7 ([1], [19]). The advantage of Densor is that it can fuse these detections with other signals like body temperature and intraoral light intensity to further determine the sleep quality of a user in one place. If Densor is capable of performing this classification, a user would only have to wear their aligner to sleep, rather then also having to wear a smartwatch to determine their sleep quality.

2.4.5. Fluid intake analysis

Another application of Densor could be monitoring fluid intake. By detecting temperature drops on the temperature sensors and by detecting head tilt on the accelerometer, Densor could possibly detect if the user is drinking and for how long they are drinking. This could then be used to estimate the amount of fluid taken in.

2.4.6. Speaking detection

An intraoral sensing platform, like Densor, could detect speaking without recording audio. This would result in a more privacy oriented speaking detection. Especially with sparse sampling, Densor could detect speaking without the ability of tracing what the wearer originally said. This could be instrumental in detecting sleep-talking in a more privacy orientated way. This would also minimize the risk of false positives if two people in the same room talk while sleeping, as the jaw of the test subject will only move in a speaking manor when they are actually speaking.

2.4.7. Orthodontic compliance

The dental aligners, to which Densor is mounted, are designed to slowly align teeth in a more comfortable, practical and invisible way then regular braces. To help dental technicians assess the effectiveness of the aligners, the photo-diode and temperature sensor can be used to determine if the user wore the aligner and for how long they wore it. This can give the dental technician objective information on the effectiveness of the aligner. If an aligner proves to be ineffective, a dental technician can use this data to determine if there is a flaw in its design or if the user did not wear it enough.

Wearing detection can also help other models in determining if they should classify. If the aligner is not within the mouth, it is not required to determine if the mouth is opened or closed for example.

3

Problem statement

Now that Densor has been introduced, the problem statement will be presented. First some problems that need to be solved in software will be presented in Section 3.1, followed by some non-functional requirements of the software stack in section Section 3.2. This chapter is then concluded by presenting the research questions for this thesis in Section 3.3.

3.1. Functional requirements

This section presents all functional requirements of the software stack of Densor. These are all challenges that need to be solved in software for the system to become fully functional. Solutions to these problems will be presented in Chapter 4.

3.1.1. Determining sample times

For the data generated by Densor to be meaningful, it is necessary to record when the samples were taken. The issue is that the only interface of the system is the NFC interface. Densor can not connect with an Network Time Protocol (NTP) server to sync time. Besides, the MCU is powered down the majority of the time to safe power which means it can't keep a synchronised time. It also needs to be considered that, since timestamps are relatively big in size, a way has to be found to timestamp the data in a memory-efficient way.

3.1.2. Run control

As the system does not have a start and stop button, this has to be done through the NFC interface. It is necessary that the taking of samples can be paused such that the user can change settings and time sync the system. Failing to pause the system while performing the aforementioned tasks could result in corrupted samples or causing the system to crash. This is a more difficult task then it may seem as there is no direct phone-to-MCU communication. All communication has to go through the storage of the NFC tag. It needs to be noted that the NFC interface has a fast-transfer mailbox which could allow direct phone-to-MCU communication by connecting an interrupt pin from the NFC tag to the MCU, but this pin is not connected on the current board design.

3.1.3. Charge indication

For ease of use and for consistency of the experiments, Densor will have to give a charge indication. As Densor does not have a battery-management system with a charge indication, it has to be given through the app as this is the only interface of the system. The minimum solution gives the user an indication when the phone is charging the system. On top of this, it would be preferred if the charge

level of the system or current draw from the phone can be displayed. This way, the user will also have an indication if they are holding their Densor in the optimal position and if the system is done charging. If the system is not held in the optimal position of the phone's antenna, this will result in a lower charge and thus a lower lifetime.

3.1.4. Data compression

Since the NFC tag of Densor only has 8 KB of storage, the data stored on this chip should probably be compressed. The challenge is that the firmware of Densor should be made as power-efficient and quick as possible, such it can make as many samples as possible. Since compression algorithms are relatively complex, a compression algorithm that is as simple as possible has to be found. A decision has to be made between a lossless and lossy algorithm. It also needs to considered that it takes significant time to read from and write to memory. This means that this algorithm should access the NFC as minimal as possible.

3.1.5. Voltage-drop compensation

While testing the sensors it was found that the temperature sensor does not compensate for the voltage drop on the input voltage. This means that, as the voltage slowly drops, the temperature readings slowly drops while readings a consistent temperature. The same issue occurs with the photo-diode readings because of the characteristics of an ADC: as the input voltage drops, the output of the photo-diode also drops. To mitigate this, these readings have to be compensated on Densor or offline when requested by the user.

3.1.6. Data labeling

As mentioned in Section 3.1.1, it is hard to get and keep an exact time of samples on Densor. This is due to the lack of a possibility to connect to a NTP server and because the MCU is powered down most of the time. This also makes it hard to label data offline, since it needs to be absolutely certain that a label corresponds to a certain sample. It is also not possible to connect to the system while performing experiments, as the system is completely embedded in an aligner. The only interface is the NFC interface, but it is impossible to keep a reliable NFC connection during the experiments. A solution had to be found to reliably and consistently label the data generated by Densor, with labels generated on a different system.

3.2. Non-functional requirements

Apart from the functional requirements presented in Section 3.1, there are also some non-functional requirements for the software stack. These requirements can be found in the following list.

- All software needs to become open-source with the rest of the project and thus needs to be properly documented.
- The firmware of Densor needs to fit inside the 16 KB flash memory of the STM32L021F4.
- The firmware of Densor needs to be able to run on the 2 KB RAM of the STM32L021F4.
- The firmware of Densor needs to be designed with power consumption in mind. It should be capable of running for an entire night of sleep while fully charged (using a smartphone) and using a sampling interval with which meaningful data can be collected.
  - All sensors need to be disabled when not in use.
  - Delays in the firmware need to be brought to a minimum.
  - Ones the MCU is done taking and saving a sample, it needs to go back to sleep as soon as possible.

  - The runtime of the MCU should be as short as possible.

- The firmware of Densor needs to be as resilient as possible on incorrect settings and components not responding, as it can not be easily power cycled once embedded in the aligner.
- The system should be fully operable by just an NFC enabled smartphone.
- Densor needs to function independently, without the need for the SWD connector or any other breakout pin as these will be cut before installation.
- Densor needs to run independently after charging, changing the settings and starting.

3.3. Research questions

Following the requirements presented in Section 3.1 and Section 3.2 the research questions can be developed. It needs to be noted that this master thesis is part of a bigger research into intra-oral sensing for which the Densor has been created. Because of this, there are also some questions that came forth from this research that preferably would be answered in this thesis. The final research questions for this thesis are:

1. How power- and memory-efficient can the firmware for Densor be made, while fitting all set requirements?
2. How can the sensors of Densor be utilized in detecting classification classes?
3. What is the difference in signal quality and classification capability between Densor and a comparable ear-mounted accelerometer in detecting mouth opening state?

For the first research question, the hypothesis is that it is possible to create the firmware to fit all requirements, while still being able to collect meaningful data. It is expected that an inter-sample interval of at least some minutes is necessary to obtain a lifetime of a full night of sleep.

For the second research question, the goal is for all classifications to run on Densor hardware. Because a future version of Densor might be able to dynamically adapt its sampling frequency, it is preferred that every model can run on its hardware. This future proof way of classification can later be used to determine if the sampling frequency should be increased when certain events do or do not occur. Since the current system is not capable of running at higher frequency it is not necessary to run the models on the hardware already, but they should be kept simple enough such that they can be ported to Densor hardware in the future. It also needs to be noted that the specific classification classes have to be determined after collecting the first data and analysing it. From this analysis it will be derived what classes might be possible to classify. The hypothesis for this question is that Densor will be able to classify *head position while laying* and *mouth opening state*, but it is uncertain if other classes can be classified since it needs to be researched if sensors can be combined to come to these conclusions.

The third research question stems from the fact that ear-mounted accelerometers are capable of detecting chewing. This is possible, since the ear canal slightly moves and deforms while chewing. The researchers behind Densor have the hypothesis that an intraoral sensing platform would be better at detecting jaw movement then an equivalent ear-mounted accelerometer, even while using sparse sampling. The goal of this research question is to test this hypothesis.

4

Software solutions

All functional requirements presented in Section 3.1 have a challenge attached to them that needs to be solved. With the solutions to these challenges, the overall software ecosystem can be designed. After this, data can be collected and analysed to determine how the sensors can be utilised. This chapter will present solutions to all challenges presented in Section 3.1.

4.1. Timestamping

As described in Section 3.1.1, time stamping the data is a harder task than one might expect. Since the only interface of Densor is its NFC interface, it can not connect to an NTP server to sync time. On top of this, the MCU shuts down in between readings, which means it can not keep time either.

Further the NFC tag only has 8 KB of storage, so the size of the samples has to be minimzed. A sample in this context is defined as all readings of all enabled sensors from one boot cycle. Timestamping each measurement is unfortunately not feasible. Since a regular epoch seconds timestamp takes 4 bytes of storage, adding a timestamp per sample would increase the sizes of the dataset between 40 and 200% (Samples are between 2-10 bytes in size, depending on which sensors are enabled). A visual representation of this and all following approaches can be found in Figure 4.1.

Since not the entire timestamp is necessary for every samples, another approach would be to save the time of the first sample and, for every next sample, save the interval between a sample and the first sample. This would mean every sample would need at least two bytes to save its timestamp. The reason for this is that one byte would not be sufficient, as it would only result in a maximum runtime of approximately 4 minutes. Using two bytes for this interval would result in a maximum runtime of 18 hours. The downside of this approach is, again, that the size needed for the dataset would increase. For this approach, the size of the dataset would increase between 20 and 100%, depending on which sensors are enabled. Although this a significant decrease in memory needed over the first approach, this would still significantly increase the size of the dataset.

For the aforementioned approaches, the MCU will need to know the actual time while taking samples, such that it can be used to timestamp the samples. This can be done by using the RTC. Since the RTC is constantly powered on, it can be used to keep a consistent clock source. The downside is that the time still has to be synced to the RTC through the NFC interface. This means that a connected phone would have to sync its time to the NFC tag, after which the MCU would have to detect this on its next boot up and set the correct time on the RTC. The issue is that the MCU is not powered on at all times. In between samples, the MCU powers down and only powers up once it is time to take the next sample. This would mean the clock on the RTC would never be exactly correct as it is never known when the MCU will boot up next. Ones the phone uploads the time to the NFC tag, it is unknown when the MCU was powered on last and when it will be powered on again. This could be fixed by having the phone constantly check for new samples. When a new sample is found, it means that the MCU just

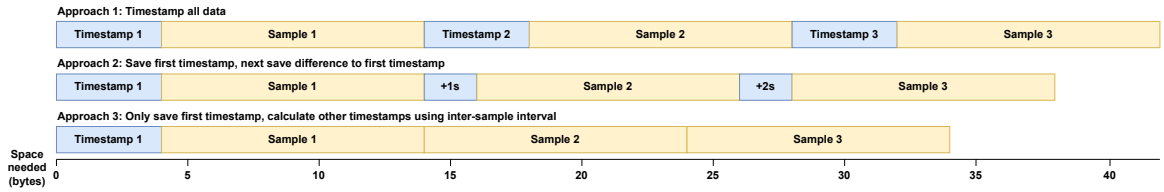


Figure 4.1: Visual representation of all three presented timestamping approaches from top to bottom. Depicts how all samples are timestamped and how much space it would take on the Densor's memory with samples taken using a one second interval. "+1s" represents the one second difference from the timestamp of the first sample. Will be stored in memory as "1", but "+1s" is used to clarify the functioning of the algorithm.

powered up. Because the inter-sample interval is known, this can be used to calculate the time when it will power up again. This inter-sample interval can then be added to the time when the new sample was found. This calculated timestamp can then be synced to the NFC tag, which can be used by the MCU to sync time to the RTC once it powers up. The issue with this approach is, sending too many requests on the NFC line will cause interference on the I2C line. This would cause the MCU to not be able to reach the NFC interface anymore, which also means it can not save samples. Because of this, the phone will never detect when the MCU was powered on.

The final approach, which is the approach that is implemented, is only saving the timestamp of the first sample. Since the NFC interface is the only interface of the system, part of the EEPROM of the NFC tag will be used to store the settings of Densor. This will also include the inter-sample interval. By using the timestamp of the first sample, combined with the inter-sample interval, every timestamp of the i-th sample (t_i) can be calculated using to the following formula,

$$t_i = t_s + ix, \tag{4.1}$$

where t_s is the start time of the system (time of the first sample) in epoch seconds, i is the index of the sample (zero-indexed) and x is the inter-sample interval in seconds. The downside is that, if a sample is missed, the timestamps of all next samples will be shifted forward by one interval. As it is not expect this will happen under normal conditions, this is not an issue in regular operation. This is also the simplest option as Densor does not need to communicate with the RTC and/or NFC tag to determine the time of the sample. It also does not need to write more bytes to the NFC tag then strictly necessary which results in the system being quicker and simpler, resulting in a shorter runtime and less power consumption. A visualisation of all approaches can be found in Figure 4.1

4.2. Charge indication

To keep power consumption down, as described in Section 3.1.3, Densor has a simple power management system without any charge indication. This means that the hardware has no way of telling the user if Densor is charging or when it is fully charged. This is unfortunate, since the user will have to align the antennas of Densor and their smartphone to charge the system. When Densor is not correctly positioned on the phone, it will not charge. Therefore the first step is to find a way to tell the user if the NFC interface is still connected to the smartphone.

To keep things simple, the NFC tap app from STMicroelectronics is used as the basis for the Densor's app. The first thing that was tried is to check if there is a callback that is triggered once the connected NFC tag disconnects. Such a callback does exist for when an NFC tag connects, as a sound plays and the app is changed to view tag dependent settings. A disconnect-callback, combined with the connection-callback, could then change an indicator on the screen, indicating if the tag is in range and thus charging or not. Unfortunately, the framework does not support such a disconnect-callback.

Instead, a different approach was found. The app can periodically read a byte on the NFC tag. If the tag is not in range, reading a byte will throw an error. The thrown error can then serve as an indication that the antenna of Densor is not in range. The app could read this byte in a loop of 1 second. A try/catch frame can catch the error and if an error is thrown, it can change an indicator on screen,

telling the user that the tag is not in range anymore. The opposite can be done when the byte is read correctly, in which case the tag is still connected. The downside of this is that the NFC tag can cause interference on the I2C bus when communication takes place on the NFC interface. To minimize the risk of issues caused by this interference, the loop to check connectivity on the app should not poll more frequent then 1 Hz.

Even though this gives the user an indication if the Densor is in range and charging, it does not give any information on the the progress of the charging. The app could start a timer after the first connection which pauses when the tag is not in range. This still does not give any indication of the quality of the connection and could result in the Densor not being charged optimally. If the antennas are not aligned optimally, this results in a lower voltage on the energy harvesting pin, meaning the capacitors are not charged as high as possible. Preferably, the voltage on the capacitors should be measured, which could give an indication of the voltage that is being supplied to the capacitors through the energy harvesting pin of the NFC tag.

The MCU has an internal voltage reference that is wired to one of its ADC inputs. Using this, the ADC can do a measurement on this voltage reference, which is essentially the voltage input [29]. The MCU contains some hard-coded registers, implemented during the production process, which contain calibration values for this voltage reference, with which the supply voltage can be calculated. This supply voltage is roughly the voltage on the capacitors and can thus be used to calculate the charge level. The supply voltage of the MCU can be calculated with the following formula,

$$v_{\text{dda}} = \frac{a_{\text{calibration}} v_{\text{calibration}}}{a_{\text{vref}}}, \tag{4.2}$$

where v_{dda} is the supply voltage in millivolts, $a_{\text{calibration}}$ is the output of the ADC during calibration, $v_{\text{calibration}}$ the supply voltage during calibration and a_{vref} the measurement of the ADC. By storing this voltage on the NFC chip, the smartphone can read this value and indicate to the user the supply voltage on the last boot cycle of the MCU. This can then pose as an indicator of the quality of the connection between the phone and the Densor and could give an indication if the system is fully charged or not.

4.3. Run control

As described in Section 3.1.2, the system should be start- and stoppable through the NFC interface. This needs to be possible to ensure settings can be changed without creating incorrect samples. If, for example, the inter-sample interval is changed while the system is running, the timestamps of the samples will be incorrect, since the new interval is used to calculate the timestamps of all samples. This includes the samples that are already stored, before changing the interval. The same holds for changing sensors that are enabled, as the length of the samples depends on this. If this is changed while the system is already sampling, it will result in datasets that can not be read, as it is unclear when samples start and end. Further, the solution to the time-stamping issue presented in Section 4.1 requires the first reading to be taken at the exact time the time sync took place. This means that the system should be temporarily stopped such that the memory can be erased, after which the time can be synced and the first reading can be taken.

Since the start trigger has to take place right after the time has been synced, the most efficient way to perform this trigger is to use the registers that holds the timestamp. The MCU can check the timestamp on every cycle and if the timestamp is invalid it will go back to sleep. A timestamp can be marked as invalid when the two most significant bytes are equal to zero, which would result in a time on January 1st, 1970. Since this is in the past, Densor can never reach this timestamp and therefore it save to label these timestamps as invalid. If the timestamp is valid, it will take a sample.

Similarly, to stop the system we can simply overwrite the timestamp with zeros and use that as a stop indication as this would result in an invalid timestamp (using zeros on the two most significant bytes results in a time on January 1st, 1970 which is in the past and thus invalid). What should also be considered, is that a place is needed to store the charge indication presented in Section 4.2. Since taking the charge measurement and storing it on the NFC tag takes significant time, the system would preferably not do this while it is not charging. This can be fixed by using the timestamp registers for

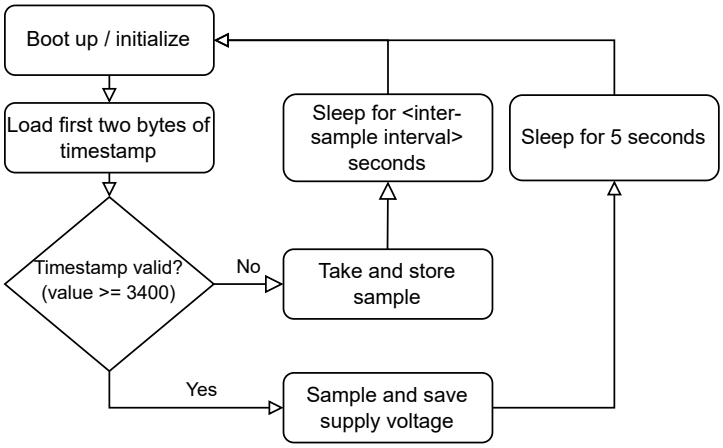


Figure 4.2: Flowchart depicting how charge mode works on Densor. Shows how either charge mode or regular mode is selected. Regular mode is simplified in this flowchart.

the charge indication while it is charging. Telling the Densor to charge would then also tell the system to stop taking samples.

The final solution works as follows, the firmware will load the first two bytes of the timestamp and verify them. If they are below 3400 (3.4V), it will measure the voltage, store it and go back to sleep. If the bytes are above 3400, the timestamp is valid and the system will take a sample as normal. The system can then be stopped by writing zeros to the first two bytes of the timestamp, after which the Densor will measure and calculate the supply voltage. The system can be started by writing a valid timestamp to the timestamp register. A flowchart depicting this process can be found in Figure 4.2. The first two bytes of the timestamp are save to use as the voltage of the system will never exceed 3.4V under normal circumstances. The maximum value the timestamp can take with the voltage being exactly at 3.4V would result in a date of 23th of January, 1977. Since this is in the past and time-travel has not yet been invented, it is safe to use the first two bytes of the register in this way.

4.4. Data compression

As Densor only has 8 KB of storage for its samples, it was considered Densor might need a compression algorithm to save storage. This would have been necessary if the system generated more samples in a single lifetime than it can store. To find a suitable compression algorithm for Densor, a literature survey was conducted and two possible compression algorithms were found. After verifying the lifetime of Densor and how much data it can create in this lifetime, it was found that data compression is not necessary for the current iteration of Densor. More on the survey performed and the algorithms found can be found in Appendix A.

4.5. Voltage-drop compensation

While verifying the sensors of Densor in a controlled environment, it was found that there is a difference in readings on both the photo-diode and temperature sensor when varying the supply voltage. This difference in readings when the voltage drops can be seen in Figure 4.3. To mitigate this issue, the supply voltage of the sensors at sample-time needs to be stored. Then, the voltage can be compensated offline when needed. This will save power on Densor, compared to compensating online, and will allow the user to review the raw readings if they desire so.

For the temperature sensor, first an experiment with the Densor connected to an external power supply was performed where a consistent temperature is read. First, 100 samples at a supply voltage of $3.1\,\mathrm{V}$ were taken and next 100 samples at a supply voltage of $1.9\,\mathrm{V}$ were taken. Between the two

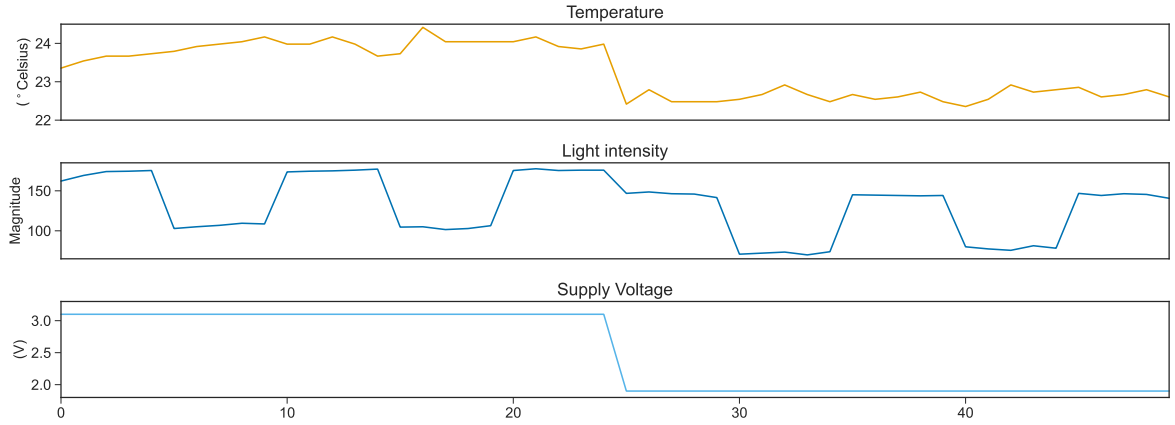


Figure 4.3: Visualisation of the difference in temperature and photo-diode readings when the voltage drops. Shows 25 readings at a supply voltage of $3.1\,\mathrm{V}$, followed by 25 readings at $1.9\,\mathrm{V}$. For the photo-diode, it is constantly not-covered for 5 readings, followed by it being covered for 5 readings.

sets of readings, a drop of $1.67\,°\mathrm{C}$ is found. This means that the temperature will drop $1.4\,°\mathrm{C}$ when the voltage drops by $1\,\mathrm{V}$. The readings can then be compensated as

$$t_\mathrm{c} = t_\mathrm{o} - 1.4(V_\mathrm{d} - 2.5), \tag{4.3}$$

where constant 2.5 in (4.3) stems from the assumption that the correct measured value is at $2.5\,\mathrm{V}$, i.e. center value between $1.8\,\mathrm{V}$ and $3.3\,\mathrm{V}$ (minimum and maximum operating voltage of Densor, respectively), with the capacitor voltage at runtime V_d, which is estimated and stored by the Densor at a precision of 0.1 V. The value of 1.4 in Equation 4.3 is found by the aforementioned experiment. t_o is the original temperature and t_c is the compensated temperature.

For the photo-diode, readings are obtained through an ADC which returns a 12-bit reading, p_o. Because of the characteristic of an ADC, its readings are proportional to the source voltage at runtime. The compensated photo-diode value, p_c, can be calculated as

$$p_\mathrm{c} = \frac{1000(V_\mathrm{d} p_\mathrm{o})}{2^{12}}, \tag{4.4}$$

where the value of 2^{12} is the maximum value p_o can take and V_d is the capacitor voltage at runtime.

4.6. Data labeling

To create and test models to perform classifications, the Densor generated data has to be labeled. Since it is unfeasible to keep Densor connected at all times during an experiment, it is not possible to perform this labeling online. Therefore, an offline solution has to be found. By keeping timestamps while performing the experiments, the labels can be added offline, after the experiment, since Densor's data is time-stamped as described in Section 4.1. These timestamps can be kept by hand, but to make it more precise the *experiment master* was made that performs this task automatically. This program runs on an external computer and is started at the same time as Densor, after which it will present the user the current task to perform, including a count-down timer until the next task. The next task is also presented, so that the user can be prepared. All tasks are shuffled at random to make sure the experiments are performed in different orders. This mitigates possible bias in the data caused by the order of the tasks. This bias can occur as the Densor will slowly warm up after being inserted into the mouth. If the order of tasks is fixed, this also means that the temperature readings are always higher during the later tasks. When the experiment is finished, the *experiment master* presents the user with the name of the file to which the Densor data should be saved. Next, it will store the used protocol, including its order, together with the name of the Densor data file in an index file.

When the Densor data file is exported and stored on the computer, another program called *data*

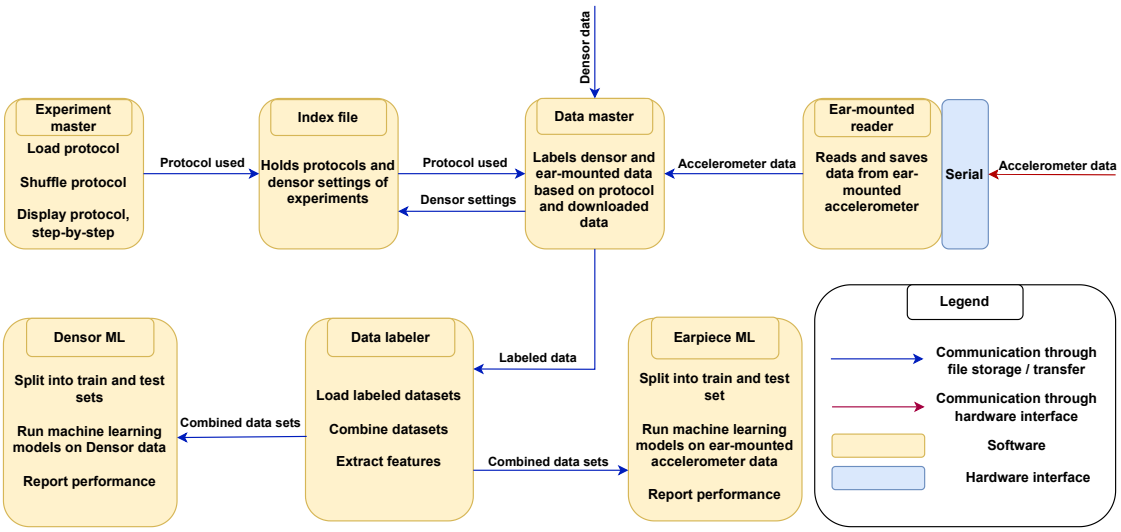


Figure 4.4: Overview of all software related to labeling and classifying data.

master can be used. This program takes the protocol from the index file and loads the data from the ,e data file. It then creates a csv file with all samples lab eled.

It should be noted that after time-syncing, Densor can take upto five seconds to take its first sample. This is, because while charging, the MCU boots up and powers down every five seconds. Therefore, for every task, there is a five second window at the end of where it is uncertain what the label of the data point is. These data points are labeled as *unknown* to make sure they are excluded from any further processing. If the inter-sample interval used for the experiment is longer then 5 seconds, no data points will be labeled as *unknown*.

If an experiment is performed without the *experiment master*, the user can create their own protocol and store it in the index file. Then, the *data master* can generate a labeled csv file in the aforementioned way. An overview of all aforementioned programs can be found in Figure 4.4.

5

System design

With the design solutions presented in Chapter 4, the software stack of for Densor can be created. The software stack consists of four sections: the firmware running on Densor, the android app used to communicate with the hardware, the programs responsible for data labeling and the classification models. This chapter will describe the design of these systems, how they function and how they interact. All code is documented and is being made open-source such that anyone can adopt Densor to their needs [30].

5.1. Firmware

The firmware for the Densor hardware is written in C++ using the STM32CubeIDE and the STM32_HAL libraries. This is chosen as it is a versatile way of written software for the STM32 MCU of the Densor.

The Densor uses its NFC tag as non-volatile storage for settings, system states and sample storage. The 8 KB of memory on the NFC chip is split into two sections: 9 register bytes and 8183 data bytes. The register bytes are used to store settings and system states. The data section is used to store generated samples. A full memory overview of the NFC chip can be found in Figure 5.1. The *Sensor enable* byte determines which sensors should be enabled. The *Time stamp / charge voltage register* is used for enabling and disabling the system. If the system is in charge mode, this will hold the sampled supply voltage. If the system is in regular operation mode, this will hold the timestamp of the first sample. This register is used to control if the system should take samples or not as described in Section 4.3. The next register, the *inter-sample interval register*, holds the interval in between samples in binary-coded decimals. The first bit, which is not used for the time indication, determines if the interval is in seconds (0) or minutes (1). The *startup delay register* holds the startup delay that should be used before starting. Again, the first bit is not used for the delay itself, so it is utilised as the trigger flag. If the flag is set to 1, the delay should be triggered on the next startup. If the bit is set to 0, the delay either already triggered or no delay should be applied, thus the firmware will ignore it. The final register is the *memory pointer register*. This register holds the address to the next byte that has not yet been written to. This is the location of the first byte of the next sample. Since the MCU powers down in between readings, it is necessary to store this on the NFC as this will be deleted from the RAM of the MCU ones it powers down.

A state diagram showcasing the functionality of Densor can be found in Figure 5.2. When Densor boots up, it will first sleep for 5 milliseconds and then retrieve the first two bytes of the timestamp. This delay is to let the NFC tag boot up, since it will be powered up and down together with the MCU. It will then proceed by verifying the first two bytes of the timestamp. If these two bytes are below 3400, it means the Densor is in charge-mode. From there, the firmware will sample the supply voltage as described in Section 4.2, store the sample in the first two bytes of the timestamp and try to set the alarm on the RTC for 5 seconds. This path will also be taken if loading the bytes from the timestamp register fails, e.g. if the NFC tag is not responding. The only difference is that in case of an error, it does not

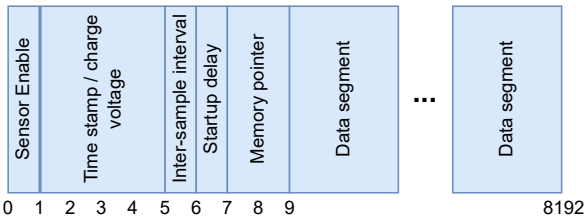


Figure 5.1: Overview of the memory structure introduced in the Densor's NFC tag.

sample the supply voltage, but immediately enters sleep for 5 seconds. This is to protect the system from accidentally taking samples if not desired.

If the first two bytes of the timestamp are above 3400 and the NFC tag responds, the firmware will enter regular operation. It will then load all other necessary registers and check if the startup delay is already triggered. This delay can be set to a maximum of 59 minutes and will delay the first sample. This can be useful if a user wants to sample while they are already sleeping, as this will save power by not taking samples that are unnecessary. If the startup delay is not triggered yet, the system will reset the trigger flag and try to set the alarm on the RTC for the set amount of minutes. If no delay is needed, the trigger flag needs to be reset when first setting up the system.

If no delay is set, the system will enable all sensors as described in the *sensor enable register* and sleep for five milliseconds to let the sensors boot up. After this delay, the sample is taken and stored on the NFC tag if there is enough space to store the sample. The MCU will then ping the NFC tag until it responds, at which point the samples are stored. This check is performed to ensure the system does not shutdown while the NFC tag is still saving the sample. The memory pointer is updated on the NFC tag and the MCU will set an alarm for the set inter-sample interval on the RTC.

A fail-safe is built into the alarm setting. The MCU will instruct the RTC to set the alarm until it needs to boot up again and then activate the alarm. This activation should power down the MCU, until the alarm triggers after the set alarm time. If the MCU does not power down, it will repeat the aforementioned steps to set the alarm 9 more times. If after, in total, 10 tries the MCU does not power down, it will reboot itself. Since activating the alarm on the RTC shoud power down the MCU, this would break the aforementioned loop. If, for some reason, this does not happen, the MCU can reset itself to avoid a lock-up.

Both delays that are introduced in the firmware are minimized as much as possible to decrease power consumption, but to ensure correct operation. The delays that are introduced are necessary, as removing them was found to cause incorrect behavior. The delays were then introduced and step-wise minimized to find the minimal value that would ensure correct functionality.

5.2. Android app

To communicate with Densor, an Android app is created. This app communicates with Densor through its NFC interface and is needed to change settings, sync time and download generated data. For simplicity, the NFC tap app from STMicroelectronics is used as a basis for the Densor app. This app is designed to communicate with the ST25 NFC tag on the Densor. The app already contained screens to read and dump the memory of the NFC tag, which are reused for Densor's app. Some screens that are not necessary are excluded. Two screens are added, the time-sync screen and the settings screen.

The time-sync screen contains multiple functions to control the memory of the Densor. First of, it contains a sync-time function which will load the current epoch seconds time from the phone to the timestamp register of the tag. This will also trigger the first sample to be taken, the next time the MCU boots up. Second, there is a wipe function. This function will clear all memory in the data section that was previously used. This will also overwrite the memory pointer with the value nine, which is the first address of the data section. All bytes before that are configuration and system-state registers. The

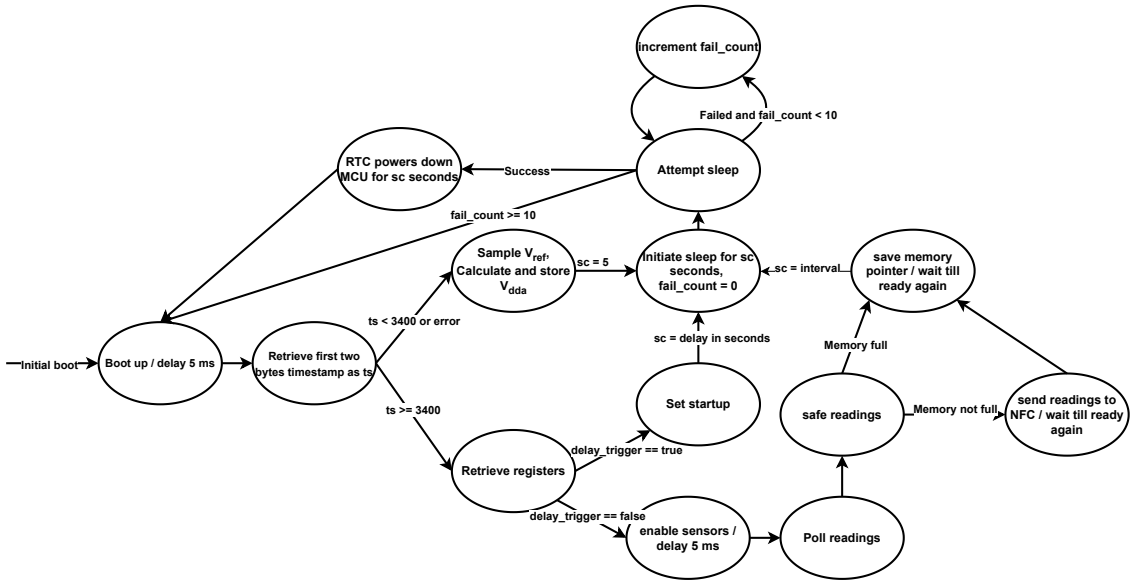


Figure 5.2: State diagram of the firmware of Densor.

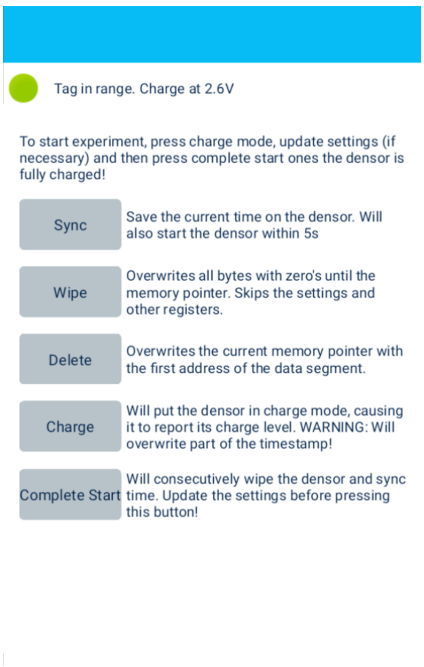
NFC tag can only handle communication in blocks of four memory-aligned bytes. If a part of a block has to be written, the app will determine the other bytes of the block it belongs to and write back the entire block with the updated bytes. This is the default behavior of the write function provided with the framework, but some errors occurred while testing where incorrect values were written back. Therefore, the created wipe function will align itself and will only provide entire blocks to write, rather then providing partial blocks. This way, the write function provided will not have to retrieve any data before starting a write transmission. The aforementioned issues where not noticed since.

The delete function is similar to the wipe function, but instead of writing zeros to all previously written data addresses, it will just set the memory pointer to 9. This way, the Densor will automatically overwrite all previous samples when necessary. The downside is, if for any reason the system saves the memory pointer but not a sample, the value from the previous lifetime will be displayed. In case of a wipe, all values will be displayed as zero in this occasion.

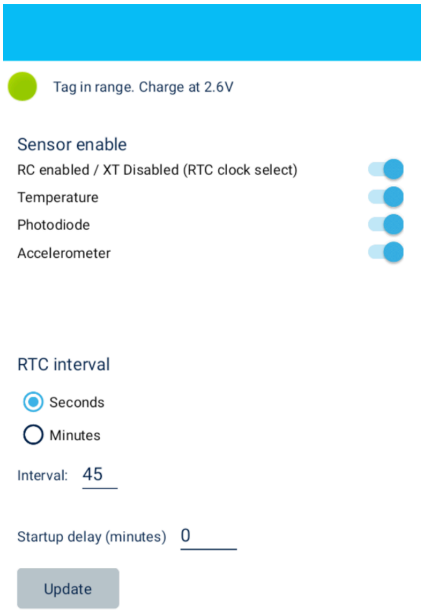
The charge function writes zeros to the first two bytes of the timestamp register. This will cause the Densor to enter charge mode, after which the charge indication can be shown in the app. Finally, the complete start function will run the wipe and sync function consecutively. A screenshot of the time-sync screen can be found in Figure 5.3a.

Next, the settings screen is used to update the settings of the Densor. This screen can be used to set which sensors should be enabled, set the crystal to be used by the RTC, which startup delay should be used and the inter-sample interval to be used. For the startup delay, it is checked that the value is an integer in the range [0, 59]. The same check if used for the interval input. If any of these checks fails, the update will not be conducted. The second and minute radio buttons tell the app if the number input should be interpreted in seconds or minutes. After pressing the *update* button, the settings are converted and pushed to the Densor. A screenshot of the settings screen can be found in Figure 5.3b.

The app also contains a connection status, which is added to the top of all screens. A background task is attached to this indication, which checks the connection status of the Densor every second. If the system is in charge mode, it will display the charge indication. If the system is in regular operation, it will display the memory pointer. By introducing this indication the user knows the charge progress, as described in Section 4.2, and if the system is running.



(a) The added time-sync screen of the Densor app. Used to control Densor and wipe the data section of its memory.



(b) The added settings screen of the Densor app. Used to change settings on the Densor.

Figure 5.3: Screenshot of the screens specifically created for the Densor app. Used for time-syncing, wiping data, entering charge mode (stopping data collection) and updating settings.

5.3. Data labeling

As previously described in Section 4.6, to determine how the sensors data from Densor can be utilized for classification, data generated by Densor has to be labeled. This section will describe all the programs necessary to perform this task

5.3.1. Experiment master

The *experiment master* will be used to load a set of steps the user should perform called a protocol. It will shuffle the protocol and display it, step-by-step, to the user including a countdown timer. The labeling process is implemented according to Section 4.6. The user has to start the experiment master and the Densor at the same time. There might be a small difference between the start of the two systems, but since the maximum sampling frequency of the Densor is 1 Hz and the difference will be within a second, this difference is negligible. The protocol is then saved in the *index.json* file, which is used as a dictionary for all experiments. The experiment master determines the next file name for the Densor data file based on the filenames that are stored in the *index.json* file.

The experiment master is written in python and the user interface is made using a library called *PySimpleGui*, as this allowed to quickly setup a simple interface that could be used for the experiments. A screenshot of the user interface can be found in Figure 5.4

5.3.2. Data master

When the data is downloaded by the phone and transfered to the laptop running the experiment master, the *data master*, also written in Python, can load this file. It will then lookup the protocol in the *index.json* file using the data file's name and label the data accordingly. This creates a new csv file, which stores all data from the data file, with their respective labels. As mentioned before in Section 4.6, all data points within 5 seconds of the end of an activity are labeled as "unknown". This script will also store the registers from the data file in the index file. The reason for this, is that this data can not be stored

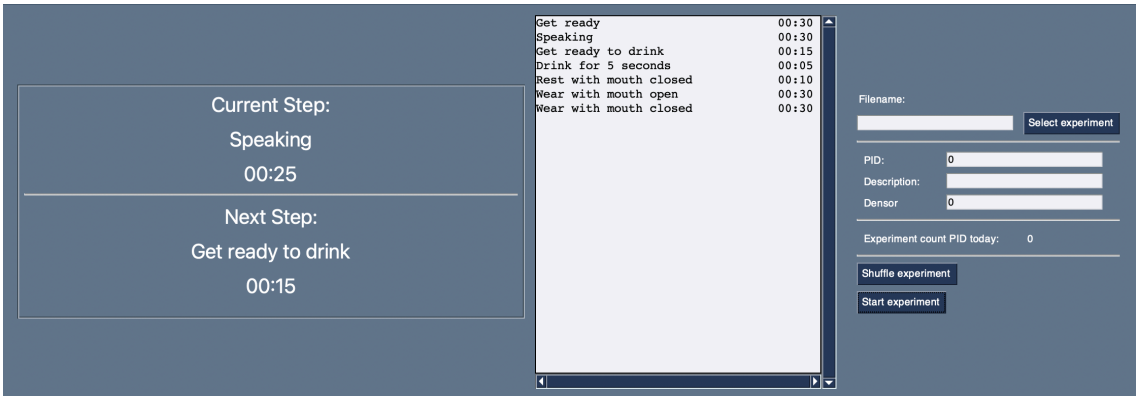


Figure 5.4: Screenshot of the *experiment master*, used to give the test subject an indication of when to perform what activity. Contains a countdown timer per task, can shuffle tasks and stores the used protocol such that it can later be used to label data.

in the csv file and might be useful in later steps. The script can also loop over all data files in the folder they are normally stored in. If they are already labeled previously, the script will skip them. If they are not yet labeled, the script will do so. By default, the script will compensate the raw temperature and photo-diode readings for the voltage drop while running in the way mentioned in Section 4.5.

5.3.3. Syncing with Apple Watch

For the experiments involving sleep, data from an Apple Watch will be used for determining the labels. The reasons for this will be mentioned in Chapter 6. All data collected by the Apple Watch will be exported from a user's phone using the Health app. This returns an xml-file which contains all sleep stages with their start and stop times. This data is then converted into a protocol by a custom conversion program. These protocols are equivalent to the protocols created by the *experiment master*. This protocol can then be used to label the overnight data in the same way mentioned in Section 5.3.2.

5.3.4. Ear-mounted accelerometer labeling

An ear-mounted accelerometer is created for the third research question presented in Section 3.3. The *data master* script can also take data from the ear-mounted accelerometer and label the data in the same way described in Section 5.3.2. Just as with the Densor data, the program can loop through the folder with ear-mounted accelerometer data and label the data files that are not yet labeled. It needs to be noted that, contrary to the Densor data, the ear-mounted accelerometer data is labeled in-place, which means the labels are added to the original csv file, as the ear-mounted data is already stored in csv files.

5.4. Machine learning models

After labeling all data, it can be used to train models. For this, two programs are made. These programs are written in Python and use scikit-learn (sklearn) for the machine learning [31]. The reasoning for this, is that sklearn is a library that is used to easily implement models and extract different plots and scores from them. One of the programs works on Densor generated data, while the other works on ear-mounted accelerometer data. The programs load all labeled datasets and add features, derived from the raw data. It will split the data into a train and test set, after which it can be used to train and verify the models. The train to test split of the models is set to 2:1. The programs, by default, only run a Decision Tree model, but they can also run K-nearest Neighbors and Support Vector Machine models if desired. The programs allow to perform a permutation-test on the Decision Trees to determine if the models are overfitting. The programs can also generate a variety of plots to visualize the models and data it is running on like ROC curves and scatter plots. An overview of the entire pipeline, for all experiments, can be found in Figure 5.5

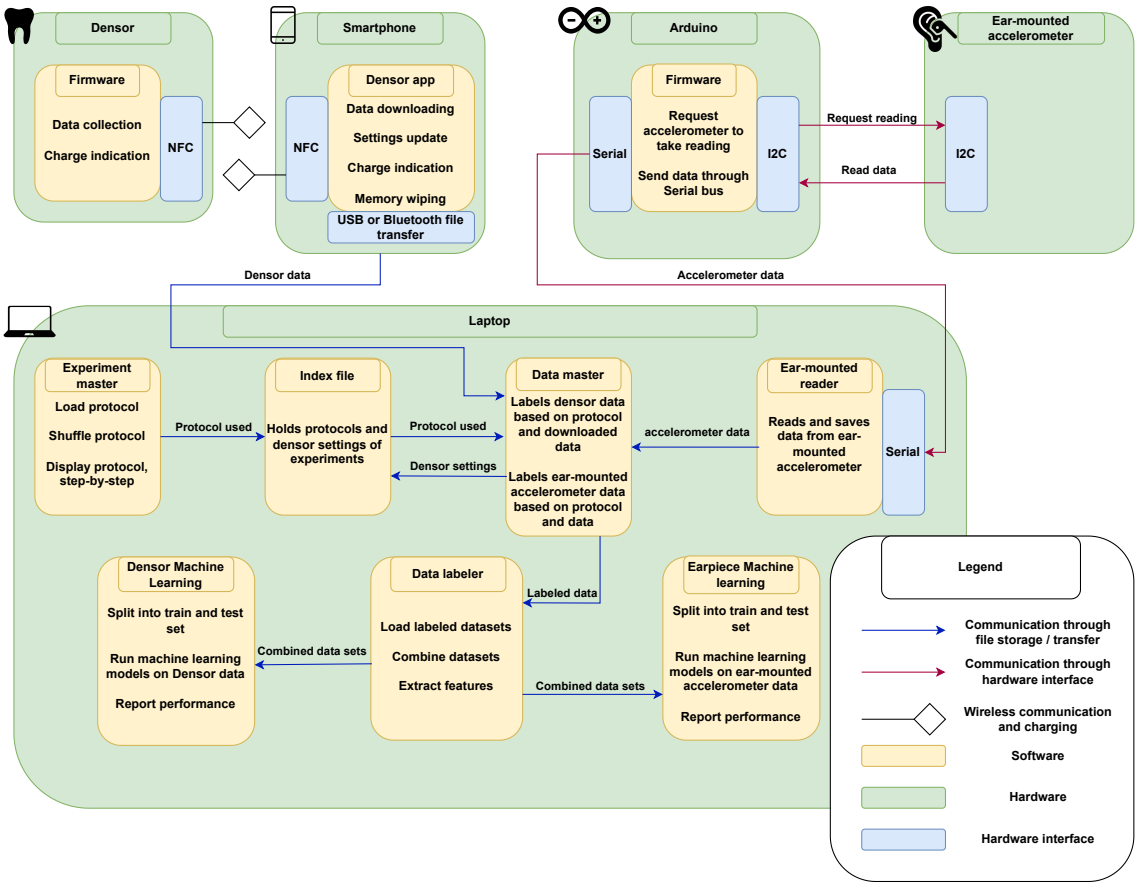


Figure 5.5: Overview of the entire software eco-system of Densor, including the data flow between different parts used both for regular Densor operation and data collection for Densor performance evaluation.

6

Experiment methodology

To answer the research questions stated in Section 3.3, four experiments had to be conducted. First the initial data had to be analysed to determine which classification classes could possibly be detected. The results for this analysis will be given in the first section. Next, the first experiment had to be performed to determine the power usage and run-times of Densor. The second experiment collected data for the classifications. The third experiment involved wearing Densor overnight to collect sleep information. The fourth experiment was used to determine the difference between Densor and an equivalent ear-mounted accelerometer system. Three test subjects were available for these experiments, each with their own Densor. Some experiments were not performed on certain test subject. The reasoning behind this will be described in the respective experiment explanation. This chapter will describe how all the experiments are performed.

6.1. Data analysis and classification

After creating the first Densor, some tests were conducted to verify the sensors and the user wore the system multiple times to collect data. In these sessions, the user performed a multitude of tasks, like taking a walk with it outside, speaking with it on a call and wearing it while traveling. Some other tasks, the researchers thought would be visible on the signals, were explored separately like drinking, head position when laying down and grinding. The Densor was also worn overnight to determine if anything specific could be detected during sleep. For these tests, rough labels were kept such that the signals could be manually analysed later. The classification classes that were selected after these test are:

- Mouth opening state
- Head position while laying
- Sleep stage
- Fluid intake
- Speaking
- Orthodontic compliance

For **mouth opening state**, a clear signal difference was found for the photo-diode signal between opening and closing the mouth. A smaller difference was also visible on the accelerometer signals.

Head position while laying was obviously visible on the accelerometer data, as the accelerometer detects gravity as an acceleration. Therefore, the readings y and z axis changed when rotating the head while laying.

As previously mentioned in Section 2.4.4, wrist-mounted accelerometers can determine sleep stages by measuring micro-movements of the wrist, caused by breathing. These micro-movements can be used to reproduce breathing patterns, which can then be used to detect to different sleep stages. Since Densor also contains an accelerometer, it was questioned if this system would also be able to detect these **sleep stages** using the same approach as the wrist-mounted accelerometers.

For **fluid intake**, it was found when cold water was kept in the mouth, this would be visible on the temperature readings. It was therefore questioned if it is possible to detect fluid intake by finding a dip in temperature on the temperature signal.

For **Speaking** detection, because the mouth is mostly opened while talking, it was expected there would be no difference on the photo-diode signal as compared to having the mouth opened, but a signal with different levels of light intensity was found while speaking. This meant that speaking can be detected by analysing the variance in light intensity and was therefore included as a separate class.

Last, a rise on the temperature signal was found when the Densor is inserted in the mouth. Similarly, a drop on temperature was found when the Densor was removed. This sudden difference in the signal was not seen when, for example, walking outside with the mouth opened. It was therefore deemed plausible that **orthodontic compliance** could be detected from the temperature signal.

6.2. Electrical characteristics and operation lifetime

To find the power used by the system, the current draw had to be measured. This was done by powering the system through a Keithley 2450 source meter. This can measure the current drawn while also providing power. Densor has two different stages with different current draws, namely the run- and idle-time. At run-time, the MCU is powered on and taking samples, while at idle-time only the RTC is powered on.

The current draw at idle-time was measured by running the Densor in regular operation. The current could then easily be measured by the source meter in between readings. A high inter-sample interval was used for this test as this makes it easier to measure the current drawn at idle-time, as there is more time to perform the measurement.

The current drawn at run-time was harder to measure, as the runtime of Densor is not long enough to perform an accurate measurement. Therefore, the code that turns on the alarm on the RTC, e.g. powers down the MCU, was temporarily removed. This means that the MCU operated in run-time, all the time. The source-meter could then easily be used to measure the current draw of the system.

Next, it was necessary to know how long the MCU is powered on. This can be used to estimate how long the system can run before running out of power. This was done by using a Salea Pro 16 logic analyser. This can be used to probe the switch that controls the power to the MCU. By measuring the logic state of this pin, we could determine for how long the MCU is turned on and thus how long the run-time of the system is.

6.3. Firmware size

The STM32CubeIDE was used to analyse the size of the firmware and the usage of the RAM. This can be done as the STM32CubeIDE, which is also used for developing the firmware, can analyse the compiled firmware and report the total flash and RAM used. With this it can be determined if the firmware can be further expanded if necessary in the future.

6.4. Labeled data collection for classification

To determine how the sensor data can be utilized to classify the classification classes mentioned in Section 6.1, an experiment had to be conducted that can collect this data. First, a protocol had to be created that can collect all necessary data as efficient as possible. A description of this protocol can be found in Table 6.1.

Task description	Duration (seconds)
Wear Densor with mouth closed	30
Wear Densor with mouth opened	30
Speaking with Densor worn	30
Get ready to drink, Drink while wearing Densor, Rest	15, 5, 10

Table 6.1: Protocol used to collect data to determine how sensors can be utilized for most classification classes. Order of each task listed in each row will be randomized, noting that the sub-tasks of the task in the last row (Get ready to drink, drink while wearing Densor, Rest) is always executed in the same specified order.

Task description	Duration (seconds)
Densor laying on table out of mouth with no human shadows casting over system	60
Walk around with Densor in hand	60
Walk around with Densor in bag	60

Table 6.2: Protocol used to collect ”Not wearing” data to determine how sensors can be utilized to classify orthodontic compliance. Tasks are randomly shuffled before each experiment run.

Every experiment was started with a 30 second *Get ready* task in which the user is expected to insert the Densor in their mouth. The protocol was designed such that all classes in th datasets are equally distributed by design. Besides, the tasks were randomized before the start of every experiment run to mitigate any possible bias in the data caused by the order in which they were performed. An example of this could be the temperature slowly rising when the system is inserted in the mouth. Some tasks would have a higher temperature then others if the order was kept equal every time. By shuffling the tasks, this bias is mitigated. The data for most classification classes could be obtained using this protocol.

For the *Orthodontic compliance* class, not wearing data had to be obtained in conjunction with this data. The reason that this data is not included in the aforementioned protocol is that we did not need the help of every test subject to collect this data. The protocol to collect ”Not wearing” data is as described in Table 6.2. The tasks of the ”not wearing” protocol are intentionally designed to include a variety of tasks where Densor is not worn to increase the correctness of the model. Only recording data while Densor is laying on a desk for example would not reflect real-life ”not wearing” data where it might lay flat on a desk, but also might be carried in a bag. The last protocol was used to detect head position while laying. This protocol was separated from the first protocol as it was found easier to do one experiment where the user lays down and one experiment where the user sits. The tasks for this protocol can be found in Table 6.3.

To perform experiments with all three protocols, the *experiment master* described in Section 5.3 was used to display the tasks one-by-one and store the protocol in the index file. Then, all data was labeled using the *data master*, again as described in Section 5.3.

Then, to make a separation between the different models, every model got their own label column while adding features. This label column contains either a classification specific label or *NA*. All samples that should be excluded from the model were labeled as *NA* and were filtered from the dataset when retrieving the train and test sets.

Task description	Duration (seconds)
Head laying center	30
Head laying left	30
Head laying right	30
Head laying center	30
Head laying left	30
Head laying right	30

Table 6.3: Protocol used to collect data to determine how sensors can be utilized to classify head position while laying. Tasks are randomly shuffled before each experiment run.

Classification	Accelerometer	Temperature	Photo-diode
Mouth opening state	-	-	Raw
Head position while laying	X, Y, Z, Resultant	-	-
Fluid intake	-	Min temp diff	-
Speaking	-	-	Variance
Orthodontic compliance	-	Raw	-

Table 6.4: Features used per classification to research how Densor's sensors can be utilized to detect the described classifications. Sensors marked red are ignored, sensors in green are included by the feature described in the cell. *Min temp diff* denotes the minimal of all difference between a temperature and its preceding temperature in a time-series window. Variance denotes the variance between all values in the time-series.

Classification	Wear with mouth opened	Wear with mouth closed	Speaking	Drinking	Not wearing	Laying left	Laying center	Laying right
Mouth opening state	Open	Closed	NA	NA	NA	NA	NA	NA
Head position while laying	NA	NA	NA	NA	NA	Left	Center	Right
Fluid intake	Not drinking	Not drinking	Not drinking	Drinking	NA	NA	NA	NA
Speaking	Not speaking	Not speaking	Speaking	Not speaking	NA	NA	NA	NA
Orthodontic compliance	Wearing	Wearing	Wearing	Wearing	Not wearing	NA	NA	NA

Table 6.5: Labels used per classification to research how Densor's sensors can be utilized to detect the described classifications. Tasks marked red are ignored, while tasks marked in green are labeled according to the label in the cell.

For every classification class, the necessary features and labels were filtered before creating and validating the model. Table 6.4 describes which features from which sensors were used for the classifications. All features were selected by analysing the raw outputs of the experiments and looking for markers that could be used as features for the models. Table 6.5 shows how the labels of the tasks were converted for the classifications and which tasks were included and excluded per classification class. Per task, it displays which classification-specific label the tasks is given. For simplicity, all tasks from the ”Not wearing” protocol are combined under the ”Not wearing” column, as they are always bundled together.

Both the sitting and laying protocols were performed on all three test subjects. The out-of-mouth protocol was only performed on the Densor of test subject 1. All test for this experiment were performed with all sensors enabled and using an inter-sample interval of 1 second. No startup delay was used for these experiments.

Only Decision tree models were trained for this experiment, as this is the simplest model that can be trained. This way, the model can potentially run on Densor hardware in the future. For that same reason, all decision tree models were set to a max depth of one. This minimized the chance of overfitting and minimized the tree size, such that it has a higher chance of running on Densor hardware. K-nearest neighbors (KNN) and Support vector machine (SVM) models were also considered, but were not further explored for the following reasons. K-nearest neigbors can never run on the current Densor hardware as the entire dataset has to be loaded to run this model, which would not fit on the flash storage. Besides, since the model grew big and might grow bigger in the future, KNN may prove to be too inefficient. SVM is also discarded as it is more complex then Decision Tree. It was also found, after examining the data, that there is a clear separation between the data points, for which decision tree is the perfect, low power solution.

To make sure the models don't overfit, p-testing was applied to all trained models. If the p-value was bigger then 0.01, the set was regarded as overfitting and it was assumed that no relation could be found in the data.

6.5. Sleep stage classification

Since it was necessary that the test subject was actually sleeping during the experiments to collect sleep stage data, it was not possible to collect data in the same way described in Section 6.4. Therefore a different way had to be found to collect the labels for the sleep stage classification. As described before in Section 2.4.4, Apple Watches can already perform this classification. This data can be exported into an xml file using the Health app on the iPhone connected to the Apple Watch. Since two Apple Watches

were already available during the research period, it was opted to use this as a source for the labels.

For this experiment, the test subjects wore both the Densor and an Apple Watch overnight. The Apple Watch automatically collected this data, so no setup was necessary, other then making sure the battery was charged. The test subject charged the Densor right before going to sleep and inserted it immediately after charging and timesyncing. All sensors on the system were enabled, the inter-sample interval was set to 2 minutes and the startup delay was disabled. When the test subject woke up, they would take the Densor out and downloaded the data. Next, they exported the XML file containing the sleep stage data through the Health app. Both files were then transferred to a laptop with the Densor labeling and classification programs.

On the laptop, labels could be extracted and the data could be labeled as described in Section 5.3. The following labels were used for the classification: Awake, Core, Deep and REM. The features used for the classification were the raw temperature samples, the x, y and z outputs of the accelerometer, as well as the resultant of all readings from the accelerometer.

This experiment was only performed on test subject 1. Because of a mistake in the manufacturing process, the Densors of test subjects 2 and 3 were not comfortable enough to wear overnight. Test subject 2 reported that part of the Densor's PCB cut into their lower lip, which caused discomfort. Test subject 3 reported that the aligner was too tight, causing his gums to hurt after a prolonged period of wearing.

Again, the models were trained using a Decision tree with a max depth of 1. The same reasons for this settings as mentioned in Section 6.4 apply here. A p-test was performed again, with a maximum p-value of 0.01. It was assumed that there is no relation between the labels and the data in the dataset when this value was above 0.01. In the performed p-tests, the p-value was either significantly bigger then 0.01 or 0.0099. Therefore, only the models with a p-value of 0.0099 were considered to have a relation.

6.6. Comparison between ear and jaw acceleration

To answer the research question if there is a difference between Densor and an equivalent ear-mounted accelerometer in detecting mouth state, the comparison between ear and jaw acceleration experiment was created.

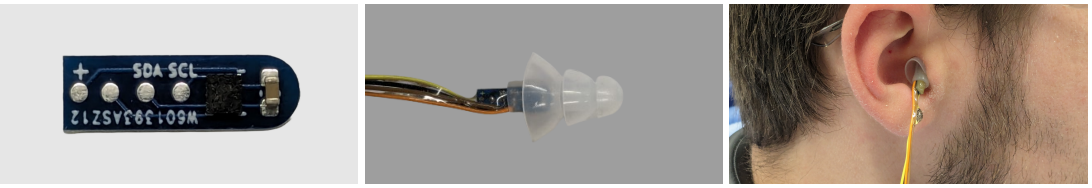
For the ear-mounted accelerometer, a custom PCB was made, which can be seen in Figure 6.1b. This PCB contains the same STMicroelectronics LIS2DW12 accelerometer used on Densor. Wires were soldered to the PCB to provide power and communicate with an external microcontroller. The PCB's size was minimized such that it can fit within the ear. Next a commodity earplug, used to block loud music at concerts and festivals, was used to mount the PCB. The audio filter was removed from the earplug and replaced with the PCB.

Next, the accelerometer was connected to an Arduino Leonardo. This Arduino set the accelerometer and polled for readings at a frequency of 25 Hz. The readings were then written to its serial bus. The Arduino was connected to a laptop through serial, which ran a python program that listened on the serial bus. When a new readings was available, it would convert this from the two bytes provided into a floating point number, display this number in a matplotlib plot and then store the reading in a csv file, together with a timestamp at which the reading arrived at the laptop.

The same computer would run the *experiment master* and collect samples in a similar fashion to Section 6.4. The Densor would run with all sensors enabled, no startup delay and an inter-sample interval of one second. Then, after labeling the Densor data, the ear-mounted accelerometer data could be labeled. To do so, the timestamp of the first reading was taken as the start of the protocol. Since the *experiment master* and Densor were started at the exact same time, this first timestamp is also the time when the protocol started. Since the protocol works on a precision of seconds, the milliseconds of the timestamps were ignored when labeling the ear data. It needed to be verified, before doing the experiment, that the time between the phone and the laptop used in the experiment is synchronized.

Task description	Duration (seconds)
Mouth closed, Mouth open	15, 15
Mouth closed, Mouth open	15, 15
Mouth closed, Mouth open	15, 15
Mouth closed, Mouth open	15, 15
Mouth closed, Mouth open	15, 15

Table 6.6: Protocol used to collect data to determine difference between Densor and an equivalent ear-mounted accelerometer system. Tasks are intentionally designed such that randomly shuffling them has no effect.



(a) PCB of the ear-mounted accelerometer, used to compare a equivalent ear-mounted accelerometer to Densor.

(b) Ear-mounted accelerometer board attached to the ear tip.

(c) Complete setup in use.

Figure 6.1: In-ear system for measuring acceleration within an ear.

The protocol used for this experiment can be found in Table 6.6. In case of this experiment, the order was fixed between the experiments to ensure an equal amount of opening and closing events took place for every experiment run.

As test subject 1 was not available for these experiments, this experiment was only performed on test subject 2 and 3.

To determine the difference in classification capability, a model was trained on both the ear and Densor collected data. As the ear-mounted system only has an accelerometer, it was only trained on the x, y and z output of this accelerometer, as well as the resultant of all outputs. To make a fair comparison, two models were trained on Densor data: one with the same features as the ear-mounted accelerometer and one with the photo-diode readings included. Again, a Decision tree was used for all models, but for this experiment no max depth was introduced. This gave both boards the highest chance of obtaining a working model. The same p-test was performed and again, it was assumed there is no relation in the dataset if the p-score is bigger then 0.1.

6.7. Test subjects

As mentioned before, only three test subject were available for these experiments, each with their own Densor. Preferably, there would be more test subjects to create a dataset with more variety, but this was not feasible within the timeline of this thesis. It takes significant time and cost to produce a prototype Densor and therefore, it was not feasible to produce more at the time. The tests are preformed on the author of this thesis and his two supervisors. For privacy reasons, they are numbered 1, 2 and 3 in the remainder of this paper.

7

Results

After making the experimental setup as described in Chapter 6, three experiments were performed. One to deterimine the lifetime of Densor, another one to obtain data to create classification models and a third to compare Densor to an equivalent ear-mounted accelerometer. In total 57 experiments were performed to collect data for the classification models, spread over multiple days with varying environmental conditions. This chapter will describe all results from these experiments, with which the research questions, presented in Section 3.3, can be answered.

7.1. Electrical characteristics and operation lifetime

After performing the measurements as described in Section 6.2, the following results where found. As described in the aforementioned section, the system has two different levels of current draw, one at idle-time and one at run-time. Using the Keithley 2450 source meter, it was found that the current draw at idle-time is 22 nA (while using the RC clock source on the RTC). When switching to the external crystal as the clock source for the RTC, the clock source becomes 70 nA.

After reprogramming the system and running it under normal operational conditions, it was found that the current draw at run-time is $822\,\mu\mathrm{A}$. Since the difference in current draw between the two clock sources of the RTC is insignificant compared to the current draw of the entire system at run-time, this is only tested with the RC as the clock source. After finding the current draws, the Salea Pro 16 logic analyser was used to find how long a single sample takes, e.g. for how long the system is powered on per sample. With all sensors enabled and the RC used as a clock source, it was found that the MCU is powered on for $41\,\mathrm{ms}$. All aforementioned metrics are summarized in Table 7.1, where the current consumption of separate components is also presented. It needs to be noted that the current consumption of the separate components are taken from their respective data sheet, rather then measuring it.

Component	Current
Schottky diode leakage (Nexperia 1PS76SB40) [32, Fig. 2]	$7.5\,\mathrm{nA}$
NFC chip isolating switch (TS3A44159) [33, Sec. 6.5]	$0.5\,\mathrm{nA}$
MCU (STM32L021F4) [34]	$805\,\mu\mathrm{A}$
Accelerometer (STM LIS2DW12) [35, Table 4]	$0.38\,\mu\mathrm{A}$
RTC (Abracon AB1805) [36, Table 8]	$14\,\mathrm{nA}$ (RC oscillator); $55\,\mathrm{nA}$ (XT crystal)
Measured: total system at idle	$22\,\mathrm{nA}$ (RC oscillator); $70\,\mathrm{nA}$ (XT crystal)
Measured: total system at run	$822\,\mu\mathrm{A}$

Table 7.1: Current consumption of Densor, broken down into separate components (as per specification) and total system (as measured)

From these measurement, it can be calculated how long Densor can theoretically last on a single charge. First, it needs to be known how much usable power can be stored on Densor's capacitor bank. This can be calculated as follows,

$$E_{\text{stored}} = \frac{NC(V^2 - V_{\text{L}}^2)}{2}, \tag{7.1}$$

where E_{stored} denotes the usable energy of the Densor's capacitor bank in Joule, N is the amount of capacitors on the system, C is the capacitance of the capacitors, V is the voltage to which the system is charged and finally V_{L} is the lowest voltage at which the system can operate. In the case of the current iteration of Densor, $N = 2$, $C = 11\,\mu\text{F}$ and $V_L = 1.8\,\text{V}$.

Next, the energy consumption of the system per second needs to be calculated as

$$P_{\text{consumed}} = V\left(\frac{I_{\text{r}}T_{\text{r}} + I_{\text{o}}(T_{\text{s}} - T_{\text{r}})}{T_{\text{s}}}\right), \tag{7.2}$$

where P_{consumed} denotes the consumed power of the Densor in Watt (Joule per second), V is the voltage to which the system is charged, I_{r} and I_{o} are the current consumption at run- and idle time respectively, and finally T_{s} and T_{r} are the inter-sample interval and power-on time of the system respectively. To calculate the lifetime of Densor in seconds, Equation 7.1 and Equation 7.2 can be combined as

$$L = \frac{E_{\text{stored}}}{P_{\text{consumed}}} = \frac{N\frac{1}{2}C(V^2 - V_{\text{L}}^2)}{V\left(\frac{I_{\text{r}}T_{\text{r}} + I_{\text{o}}(T_{\text{s}} - T_{\text{r}})}{T_{\text{s}}}\right)}. \tag{7.3}$$

It can be found, with $N = 2$, $C = 11\,\mu\text{F}$ and $V_L = 1.8\,\text{V}$ and the found current draws and runtime in Table 7.1, that the system has a theoretical total lifetime of approximately 7 minutes when using an inter-sample interval of 1 second and approximately 13.4 hours when using an inter-sample interval of 2 minutes.

7.2. Firmware size

According to the analyser in the STM32CubeIDE, the firmware is 9.73 KB in size and uses 1.73 KB of RAM when building in release mode. Building in release mode removes all debugging code from the firmware, resulting in a smaller firmware size. The firmware takes up 60.84% of the flash storage and utilizes 86.72% of the available RAM.

7.3. Classifications

For the classification classes described in Section 6.1, before creating the machine learning models using a decision tree, it was first researched if a different, simpler classification system existed. The only class where this was the case was for the *Head position while laying* class. Since the Densor contains an accelerometer, this can be utilized to measure gravity while the accelerometer is not moving. Then, by taking this measurement, trigonometry can be used to determine the roll angle of the accelerometer, β, in degrees with respect to earth's gravity. The roll angle is calculated as [37, equation 7]

$$\beta = \tan^{-1}\left(\frac{(-y)}{180\left(\frac{z}{|z|}\right)\sqrt{x^2 + z^2}/\pi}\right), \tag{7.4}$$

where x, y and z are gravitation in the x, y and z axis. The advantage of this calculation is that it reports the angle at which the user is laying rather then the just if the user is laying with their head at their left, center or right side. The downside is that it will take up more storage when used on Densor hardware, but if a user only wants to measure head position, this would still be more efficient then storing the raw acceleration outputs. It needs to be noted that, because of the different placement of the Densor

Test subject	Calibration value (roll)
1	32
2	-12
3	8

Table 7.2: Roll angles in degree per test subject that should be added to the measured roll angles to calibrate the values. Results in the calibrated roll angle being 0 when laying with head at center.

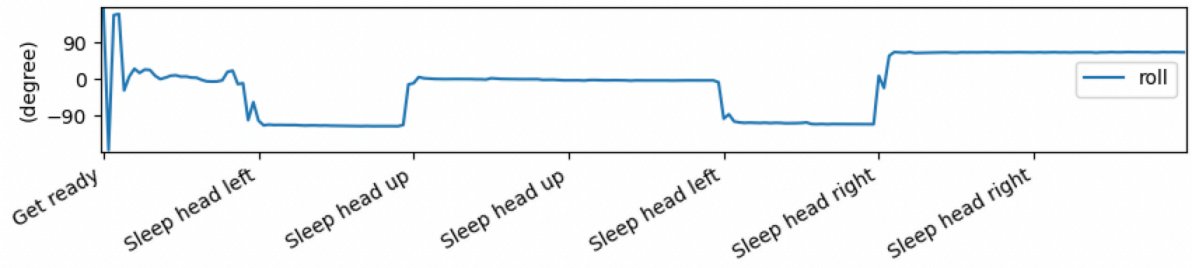


Figure 7.1: Example trace of the calculated roll from a ”head position while laying experiment” in which a test subject laid down and moved the head in the requested position shown on the x-axis. From this, it can be seen that the calculated roll perfectly follows the head rotation.

board on the aligners, a calibration value has to be added to the calculations to obtain an accurate angle. This angle was found by taking a measurement from each Densor while holding it in a center laying orientation. The calibration values per test subject can be found in Table 7.2. Figure 7.1 shows the calculated roll from one of the experiments. From this figure, it can be seen that the calculated roll perfectly follows the ground truth, displayed on the x-axis.

For all other classifications, machine learning models are created. All models are decision trees with a maximum depth of one to both minimize the risk of overfitting and minimizing the size such that it can potentially run on Densor hardware. The features used per model can be found in Table 6.4 and the tasks that are included in the models, including the classes to which the tasks are categorized, can be found in Table 6.5. For the models, apart from the sleep experiments, 35 experiment runs are executed between the three test subjects. This resulted in 3465 usable data samples. For the sleep experiments, the Densor was worn for 4 nights, resulting in 575 usable samples. A figure containing the signals from all tasks used for the experiments can be found in Figure 7.2.

After creating the models, their performance is evaluated by calculating the accuracy-, f1- and p-score. The p-score is calculated using 5-fold cross validation. If a model returned a p-score bigger then 0.1, it is assumed that the model was not able to find any relation between the provided features and the labels. Table 7.3 reports all results from the machine learning models per test subject and for

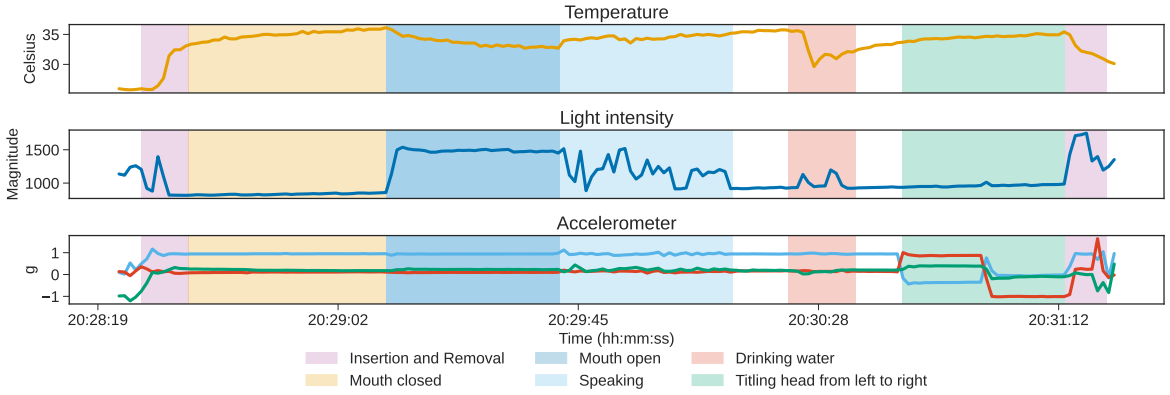


Figure 7.2: Example traces of all signals recorded from Densor, for all tasks used in the experiments.

Test subject	Mouth opening state	Fluid intake	Speaking	Orthodontic compliance	Sleep stage
P1	acc: 0.79 f1: 0.78 p: 0.01	-	-	acc: 1.00 f1: 1.00 p: 0.01	acc: 0.29 f1: 0.18 p: 0.73
P2	acc: 0.91 f1: 0.91 p: 0.01	acc: 1.00 f1: 1.00 p: 0.01	acc: 1.00 f1: 1.00 p: 0.01	acc: 1.00 f1: 1.00 p: 0.01	-
P3	acc: 0.92 f1: 0.92 p: 0.01	acc: 0.70 f1: 0.64 p: 0.73	acc: 0.80 f1: 0.82 p: 0.01	acc: 1.00 f1: 1.00 p: 0.01	-
All	acc: 0.90 f1: 0.90 p: 0.01	acc: 1.00 f1: 1.00 p: 0.01	acc: 0.91 f1: 0.89 p: 0.01	acc: 1.00 f1: 1.00 p: 0.01	acc: 0.29 f1: 0.18 p: 0.73

Table 7.3: Accuracy-, F1- and p-scores for all decision tree models trained on Densor data. All models are a decision tree with maximum depth of one. Fluid intake model trained on all, only includes data from participant 2. Sleep stage model trained on all only, only includes data from participant 1. All models in red are labeled as ”incorrect”. All models in green are labeled as ”correct”. All models in grey were discarded or not created due to a lack of data.

all test subjects.

As can be seen from these results, most classes can be classified in the aforementioned setup with Densor generated data. For *mouth opening state* and *orthodontic compliance* all scores are high enough to be significantly better then guessing. The p-scores are also small enough to suspect a relation between label and features is found, as a model that generates random answers will result in a higher p-score in the permutation tests.

For the *fluid intake* and *speaking* detection, the models for P1 were discarded. The reason behind this is that the features are time-series based. That is, instead of using single samples, the feature is taken from all samples per task execution. For example, for the *speaking* model, the feature used is the variance in photo-diode readings per task, per experiment run. This results in a single value per task per experiment which is then used in the classification. A visual representation of this feature extraction can be found in Figure 7.3. This results in significantly less features to use in the classification (30 to 1). Since less data was collected for P1, there was not enough data left after processing the features to perform the classification reliably for these two models. Therefore, the results of these two models are discarded for test subject 1. On top of this, for the *fluid intake* model, the data from P1 and P3 was discarded from the model that is supposed to be trained on all subjects. For this task, a clear difference in temperature was visible when test subject 2 drank, but no such difference was visible for P1 and P3. The reasoning behind this, is that the PCB of P2’s Densor is positioned slightly higher. Therefore, the water touches the accelerometer, resulting in a more dramatic decrease in temperature which is more easily detectable. This is also the reason that the model trained on P2 has a correlation between features and labels, while the model trained on P3 does not.

For the *sleep stage* classification, data was only collected by P1 for the reasons mentioned in Section 6.5. Unfortunately, no correlation between the labels and the data could be found. The idea was that, since wrist mounted accelerometers can be utilized to determine sleep stage by detecting micro-movements caused by breathing, Densor should also be able to detect these micro-movements. Even though some micro-movement is visible on the data, sampling at a 2 minute interval is not enough to reproduce the breathing patterns. It is possible that a future iteration of the Densor, capable of sampling

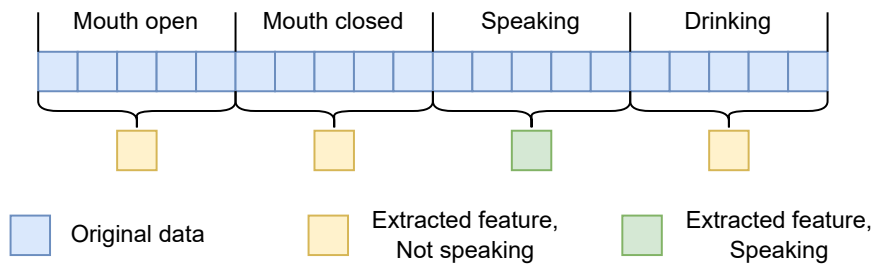


Figure 7.3: Visual representation of feature extraction for time-series based models for a single experiment run. All features of all separate tasks are combined into a single sample. This results in three ”Not speaking” and one ”Speaking” labeled sample.

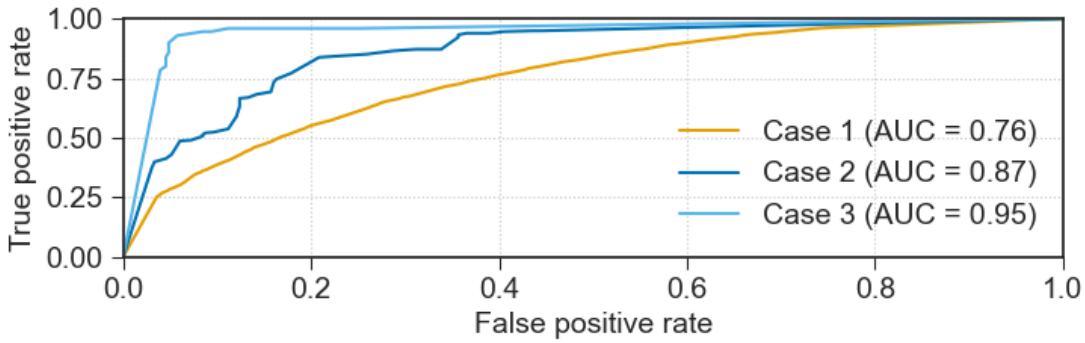


Figure 7.4: Receiver operating characteristic (ROC) curve for detecting mouth opening state, i.e. open or close. Case 1: ear-mounted accelerometer, Case 2: Densor (accelerometer used only), Case 3: Densor (accelerometer and photodiode used). It can be conclude that Densor outperforms an ear-mounted accelerometer, while using a lower sampling frequency; AUC: area under the ROC curve.

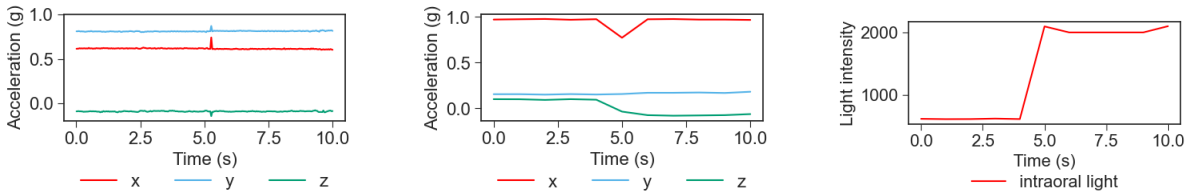
at a higher frequency overnight, could be able perform this classification, but the current iteration of the hardware is incapable of doing so.

7.4. Comparison between ear and jaw acceleration

To verify the effectiveness of Densor in determining the mouth opening state as compared to an equivalent ear-mounted system, the following experiment was conducted, as described in Section 6.6. In total, 18 experiment runs were conducted where either test subject 2 or 3 wore both their Densor and an ear-mounted accelerometer. These 18 experiment runs total 1800 Densor data samples and 45,000 ear-mounted data samples. After collecting the data, three decision tree models where created. Since it is not necessary to run these models on Densor hardware, it was opted to not limit the maximum depth of the tree to give both models the best possibility of finding a good model. All models are only trained on data that was collected in the *”comparison between ear and jaw acceleration”* experiment and did not contain any samples from different experiments, which also containt opening and closing events. This is to not give Densor an unfair advantage, as it would otherwise have more data available.

The Receiver Operating Characteristic (ROC) curves of all three models can be found in Figure 7.4. From these curves, it can be seen that the ear-mounted accelerometer performs the worst when determining the opening state from a single sample. The middle best model is the Densor model trained on just accelerometer features (x, y, z and the resultant of the gravitations). The best performing model, is the one where the photo-diode readings are included in the model. This is as expected when comparing it to the traces of all tree signals.

All three signals from a single opening event can be viewed in Figure 7.5. As can be seen, it is possible to detect the closing event on the ear-mounted accelerometer data, but it is a small peak in the signal (at the five second mark) compared to the peak seen on the Densor accelerometer data.



(a) Acceleration measured by in-ear accelerometer.

(b) Acceleration measured by Densor.

(c) Increase in intraoral light intensity measured by Densor.

Figure 7.5: Example traces (simultaneously collected) from in-ear accelerometer and Densor during a mouth opening event. The mouth is opened at the five second mark. The in-ear accelerometer only records a change during the transition. Densor can recover the jaw orientation (pitch) directly—note the permanent change of the z direction once the mouth is open (center figure) compared to flat z direction for in-ear accelerometer (left figure). Additionally, Densor can measure intraoral light intensity to help the decision, which makes determining mouth opening state easier with Densor.

Besides, there is no difference visible before and after the spike, meaning it is impossible to determine the state without detecting the spike and knowing the initial state. It is possible to determine this from both the Densor's accelerometer and photo-diode signals. The accelerometer signals show a difference on one of the axis before and after the event, even though this difference is slim. This signal can also, theoretically, be changed by moving the head instead of the jaw resulting in false positives (misclassifying opened for closed). There is also a bigger difference before and after the event on the photo-diode signal as more light falls on the diode once the mouth is opened. This signal is more reliable as it is harder to create false-positives by moving the jaw as can be done with the accelerometer. It should be noted is that this opening event is an outlier where the opening of the mouth was exaggerated. In most cases, opening or closing the mouth results in a spike in acceleration that is either hardly or not visible on the ear-mounted accelerometer compared to the noise on the signal.

8

Conclusion

The goal of this thesis was to find out if it was feasible to create the firmware for Densor with the given constraints, determine how the sensors of Densor could be utilised to detect a multitude of classification classes and what the difference between Densor and an equivalent ear-mounted accelerometer is in detecting mouth state. After performing multiple experiments with three different test subjects, collecting approximately 5840 different samples, the following conclusions can be drawn. This chapter will conclude on all research questions, one-by-one.

8.1. How power- and memory-efficient can the firmware for Densor be made, while fitting all set requirements?

For the first research question ”How power- and memory-efficient can the firmware for Densor be made, while fitting all set requirements?”, it can be concluded that the firmware can be made to fit in 9.73 KB of storage, while also fitting all set requirements. This results in a 60.86 % usage of the available flash storage on the MCU, meaning the firmware can be expanded if necessary. The firmware needs 1.73 KB of RAM, which results in 86.72% usage of the available RAM of the MCU. In terms of power consumption, the firmware's runtime was minimised to $41\,\mathrm{ms}$, resulting in a theoretical lifetime of 13.4 hours when using an inter-sample interval of 2 minutes.

8.2. How can the sensors of Densor be utilized to detect the following classification classes?

The next research question ”How can the sensors of Densor be utilized to detect the following classification classes?” can be answered using the results from the classification models. For *mouth opening state*, the raw output of the photo-diode can be used to detect the mouth opening state with an accuracy of 90%. It should be taken into account that this only works when there is enough light to detect in the environment, but the results can be augmented by the accelerometer data as shown in the ear-vs-mouth experiment. *Fluid intake* can be detected with an accuracy of 100% while using the minimal difference between a temperature and its preceding temperature within a window as its feature. It should be noted for this classification that this depends on the placement of the temperature sensor, as this only worked for one specific Densor. *Speaking* classification is possible using the variance in photo-diode readings in a time-series window. This classification can be done with an overall accuracy of 91%. Raw temperature readings can be used as a feature to detect *orthodontic compliance* with an accuracy of 100%, even though it should be noted that this is not verified in an environment with temperatures equivalent to body temperature. It is expected that the model will not work in these circumstances, but further research should be performed to determine if the model fails to work in these environments. *Head position while laying* can be calculated from the raw accelerometer readings us-

ing trigonometry. Finally, *sleep stage* can unfortunately not be detected with the current setup of the Densor, but if a Densor is made which is capable of higher sampling rates, it might be possible.

8.3. What is the difference in signal quality and classification capability between Densor and a comparable ear-mounted accelerometer in detecting mouth opening state?

For the final research question ”What is the difference in signal quality and classification capability between Densor and a comparable ear-mounted accelerometer in detecting mouth opening state?”, it can be concluded that Densor is significantly better at detecting mouth opening state while using sparse sampling. From a single sample, Densor is more capable at determining if the mouth is opened. This statement holds while only using accelerometer related features from Densor, as well as combining this data with photo-diode readings.

9

Recommendations

During and after the research presented in this thesis, some possible improvements came up that could not be realised within the project timeline. This chapter describes all these improvements such that they can, hopefully, be implemented in future iteration of this research.

9.1. Higher sampling rates

The current iteration of Densor can only sample at a maximum sampling rate of 1 Hz. This is because sampling with a higher frequency, although technically possible, would drain the battery too quick. Since the maximum theoretical lifetime of Densor is only 7 minutes while sampling at 1 Hz, it is currently not feasible to sample at a higher sampling rate. If a future iteration of the system contains more energy storage, the sampling frequency should be increased. This would possibly allow for different classification classes, like sleep stage, to become detectable by Densor.

9.2. Variable sampling rates

To save power, while also allow for collecting more meaningful data, it would be beneficial if the sampling rate could be adapted dynamically. It would help if, for example, the system runs on a low frequency by default to safe power. If an event is detected during a boot cycle, the system can temporarily increase its sampling frequency to collect more data from the event. A different way of timestamping the data needs to be found for this approach to work, as the current system relies on the fact that all samples are collected on a consistent sampling frequency. This could, for example, be achieved by allowing for a lower and higher sampling rate. Every sample could then contain a flag, indicating which of the two intervals is used after the sample. To implement the variable sampling rates, either the data storage of Densor should be increased or compression should be implemented. This is necessary to account for the increase in sample sizes from the aforementioned interval flag, as well as the additional samples generated by the higher frequency.

9.3. On-board classification

To make the variable sampling rates work, on-board classification is necessary to determine which sampling rate should be used. Even though the models created during this thesis are designed to be as lightweight as possible, they have not been tested on Densor hardware yet. Running on-board classification could also help in saving storage by only storing the classification predictions, rather then the raw data. The downside of this approach is that the predictions can not be verified offline, as the original data is lost.

9.4. Firmware updates through NFC

Currently, there is no way to update the Densor's firmware after it has been embedded in the aligner. Since the programming connectors are removed before epoxying the board to the aligner, there is no way to interface with the Densor, outside of the NFC interface. The downside of this approach is that, once the system is made its functionality can not be changed. This becomes an issue once bugs are found or new functionality needs to be introduced, as a new aligner and board has to be made to accommodate these firmware changes. To mitigate this issue, it would be preferable if the Densor's firmware can be updated through the NFC interface. The NFC Tap app, used to design the Densor app, already contains functionality to handle the phone side of this, but a boot-loader has to be created to handle the Densor side. It needs to be researched if this is possible with the current setup, as adding a custom bootloader to handle the firmware updates could exceed the available flash storage.

9.5. System interrupts

With the current setup, if the phone wants to communicate with the MCU, it has to wait for the RTC to boot it up. The NFC tag has an interrupt pin that is pulled high once a NFC reader is connected. If this pin could be connected to the RTC and the MCU, it could be used to temporarily boot the MCU to handle connections from the phone. This could be useful in case of the aforementioned firmware updates through NFC, since with the current setup the phone will have to wait for the MCU to boot up before the update can take place. In case the system is in run mode, this can take up to an hour to respond as this is the maximum inter-sample interval in this mode and the MCU is turned off in between samples. This would also help with the 5 seconds of inaccuracy caused by charge mode being active. In the case the interrupt pin is connected, the MCU can be powered up until the timestamp is updated, at which point it can immediately power down. The downside is that this interrupt pin only works when a voltage is applied to the I2C side of the NFC tag. If the NFC tag is just powered through the NFC interface, the interrupt pin does not function. This voltage is intentionally not applied when the MCU is powered down, to save power. A way should be found to power the NFC chip ones a phone is connected. A solution could be to have a switch that enables power to the NFC chip when the energy harvesting pin is pulled high, but it should be researched if this approach is feasible.

A

Data compression

As Densor only has 8 KB of storage for its samples, it might be necessary to compress samples. This is necessary if the system generates more samples in a single lifetime than it can store. To find a suitable compression algorithm for Densor, a literature survey was conducted.

Since Densor has very little power storage, it is necessary that a compression algorithm uses as little power as possible. Besides, the algorithm can not use much space, as the flash storage of Densor's MCU is only 16 KB in size. This also needs to fit the rest of the firmware, including all necessary libraries and the hardware-abstraction layer.

Although many studies were found that focus on energy-efficient compression by developing simple algorithms, many of them are designed for devices that transmit data through Bluetooth Low Energy (BLE). These devices have more power storage then Densor and the algorithms for them focus on minimizing transmission power. The issue is that these algorithms do not have to be as power-efficient as possible, as long as they are more power-efficient then directly transmitting the data through BLE. One example of such algorithms is [38].

What was also found, is that many power-efficient algorithms rely on dictionaries to perform the compression. These dictionaries are optimized for space-constrained devices. The issue is that most devices described in such papers are less space constrained then Densor. One example of such algorithms is [39], which needs a 20 KB dictionary. Such a dictionary could not fit on Densor hardware and it is possible that such dictionaries can only compress a single signal type. Since Densor collects more then one signal. the size of the dictionary would probably increase beyond the mentioned 20 KB. From the found algorithms, two possibilities were considered: Zero-Order Hold (ZOH) and delta compression.

ZOH is a lossy compression algorithm that compares all generated values to its previous value [40]. If the difference between the current and previous value is smaller then a set delta, the current value will be discarded. The bigger this delta, the more aggressive the compression. The downside of this algorithm is that it is lossy, and thus the original signal can never be completely reconstructed. Another downside of this algorithm is that it needs timestamps for all samples. It uses these timestamps to reconstruct the signal, as it needs to know how many samples where discarded. Since Densor does not store timestamps per sample, another way to reconstruct the signal has to be found. This could possibly be done by storing the data asynchronous. When doing this, every bit will be preceded by a 1-bit indicating a new value. Then after the value, every 0-bit will indicate a similar value. Only when a 1 is read, the next bytes will be evaluated as a new value.

Delta compression is similar to ZOH, as it also uses the difference between values for compression. The difference is that delta compression is a lossless algorithm that stores the difference between values. It keeps the first value as a baseline and then only stores the differences between two consecutive values. If the vector (1, 2, 3) would be compressed using delta compression, this would result in the vector (1, 1, 1). If the differences can be stored in smaller values then the actual value, this algorithm

will compress the data. Since all current sensors on Densor produce 2-byte readings, this can only compress the data by 50% at max. On top of this, it will only work for signals like temperature where the difference between readings is small and can be stored in one byte.

Finally, no signal compression was developed for Densor. After developing the firmware, it was found that the current iteration of Densor can never exceed its storage on a single charge. When enabling all sensors, Densor will produce ten bytes of data every sample. Excluding the nine bytes that are allocated registers, the data section of the storage is 8183 bytes large. This means, the storage can save a maximum of 818 samples. When using an inter-sample interval of 2 minutes, which is used for most in-vivo experiments overnight, the system would have to run for over 27 hours to produce enough data to fill the storage. As will be discussed later in Chapter 7, the current Densor can never reach this on a single charge. Therefore, it was found unnecessary to develop a compression algorithm for Densor, as this would limit the amount of samples that can be taken by using already limited power stored in the capacitors. The two found algorithms may still prove useful in the future when a Densor with longer run times is developed.

Bibliography

[1] Apple Inc., *Estimating sleep stages from apple watch*, https://www.apple.com/healthcare/docs/site/Estimating_Sleep_Stages_from_Apple_Watch_Sept_2023.pdf, Sep. 2023.

[2] T. Röddiger, T. King, D. R. Roodt, C. Clarke, and M. Beigl, "OpenEarable: Open hardware earable sensing platform," in *Proc. UbiComp/ISWC'22 Adjunct*, https://doi.org/10.1145/3544793.3563415, Cambridge, UK: ACM, 2022, pp. 246–251.

[3] G. Kazdaridis, N. Sidiropoulos, I. Zografopoulos, P. Symeonidis, and T. Korakis, "Nano-things: Pushing sleep current consumption to the limits in IoT platforms," in *Proc. IoT*, https://doi.org/10.1145/3410992.3410998, Malmö, Sweden: ACM, Oct. 2020, pp. 1–8.

[4] J. de Winkel, C. Delle Donne, K. S. Yıldırım, P. Pawełczak, and J. Hester, "Reliable timekeeping for intermittent computing," in *Proc. ASPLOS*, https://doi.org/10.1145/3373376.3378464, Lausanne, Switzerland: ACM, Mar. 2020, pp. 53–67.

[5] Scheu Dental GmbH, *Bruxchecker*, https://www.scheu-dental.com/en/scheu-dental/products/pressure-moulding-technique/pressure-moulding-material/brux-checkerr-acc-to-prof-sato/0760, Last accessed: May. 16, 2024, May 2024.

[6] K. Saczuk, B. Łapińska, P. Wilmont, L. Pawlak, and M. Łukomska-Szymańska, "The bruxoff device as a screening method for sleep bruxism in dental practice," *Journal of Clinical Medicine*, vol. 8, no. 7, 2019, https://doi.org/10.3390/jcm8070930, ISSN: 2077-0383. [Online]. Available: https://www.mdpi.com/2077-0383/8/7/930.

[7] Thestr, *Sleep talk recorder*, https://site.sleeptalkrecorder.com/, Last accessed: May. 16, 2024, May 2024.

[8] H. Matsumoto, K. Tomoto, G. Kawase, *et al.*, "Real-time continuous monitoring of oral soft tissue pressure with a wireless mouthguard device for assessing tongue thrusting habits," *Sensors*, vol. 23, no. 11, 2023, https://doi.org/10.3390/s23115027, ISSN: 1424-8220.

[9] X. Sun, L. Qiu, Y. Wu, Y. Tang, and G. Cao, "Sleepmonitor: Monitoring respiratory rate and body position during sleep using smartwatch," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 1, no. 3, Sep. 2017, https://doi.org/10.1145/3130969.

[10] Fédération Internationale de l'Automobile, *F1 - safety measures*, https://www.fia.com/news/f1-safety-measures, Last accessed: May. 16, 2024, Jun. 2016.

[11] L. F. Gabler, N. Z. Dau, G. Park, A. Miles, K. B. Arbogast, and J. R. Crandall, "Development of a low-power instrumented mouthpiece for directly measuring head acceleration in american football," *Annals of Biomedical Engineering*, vol. 49, no. 10, pp. 2760–2776, 2021, https://doi.org/10.1007/s10439-021-02826-8.

[12] L. F. Gabler, S. H. Huddleston, N. Z. Dau, *et al.*, "On-field performance of an instrumented mouthguard for detecting head impacts in american football," *Annals of Biomedical Engineering*, vol. 48, no. 11, pp. 2599–2612, 2020, https://doi.org/10.1007/s10439-020-02654-2.

[13] M. Brandl, J. Grabner, K. Kellner, *et al.*, "A low-cost wireless sensor system and its application in dental retainers," *IEEE Sensors Journal*, vol. 9, no. 3, pp. 255–262, 2009, https://doi.org/10.1109/JSEN.2008.2012205.

[14] Z. Hu, A. Radmehr, Y. Zhang, S. Pan, and P. Nguyen, "Ioteeth: Intra-oral teeth sensing system for dental occlusal diseases recognition," vol. 8, no. 1, Mar. 2024, https://doi.org/10.1145/3643516.

[15] S. Pan, D. Y. Lee, J. H. Lee, and P. Nguyen, "Teethvib: Monitoring teeth functional occlusion through retainer vibration sensing," in *2021 IEEE/ACM Conference on Connected Health: Applications, Systems and Engineering Technologies (CHASE)*, https://doi.org/10.1109/CHASE52844.2021.00018, 2021, pp. 92–96.

[16] Z. Cheng, P. Li, J. Teng, *et al.*, "A dual-layer orthogonal-layout thin-film force sensor for digital orthodontic functional appliances," *IEEE Sensors J.*, vol. 23, no. 15, pp. 17 524–17 530, 2023, https://doi.org/10.1109/jsen.2023.3275943.

[17] M. Kohli, L.-F. Hsu, C.-C. Chen, C.-C. J. Yao, and T.-K. Chung, "Wearable wireless magnetic-reluctance-based pressure sensing module for intraoral pressure monitoring," *IEEE Transactions on Magnetics*, vol. 59, no. 11, pp. 1–9, 2023, https://doi.org/10.1109/TMAG.2023.3290768.

[18] L. de Almeida e Bueno, W. Milnthorpe, and J. H. M. Bergmann, "Determining the performance of a temperature sensor embedded into a mouthguard," *BDJ Open*, vol. 8, no. 1, p. 23, Aug. 2022, https://doi.org/10.1038/s41405-022-00114-8, ISSN: 2056-807X.

[19] J. Leube, J. Zschocke, M. Kluge, *et al.*, "Reconstruction of the respiratory signal through ECG and wrist accelerometer data," *Sci Rep*, vol. 10, no. 1, p. 14 530, Sep. 2020, https://doi.org/10.1038/s41598-020-71539-0.

[20] M. S. Mannoor, H. Tao, J. D. Clayton, *et al.*, "Graphene-based wireless bacteria detection on tooth enamel," *Nature Communications*, vol. 3, no. 1, p. 763, Mar. 2012, https://doi.org/10.1038/ncomms1767, ISSN: 2041-1723.

[21] J. Kim, G. Valdés-Ramírez, A. J. Bandodkar, *et al.*, "Non-invasive mouthguard biosensor for continuous salivary monitoring of metabolites," *Analyst*, vol. 139, pp. 1632–1636, 7 2014, https://doi.org/10.1039/C3AN02359A.

[22] J. Kim, S. Imani, W. R. de Araujo, *et al.*, "Wearable salivary uric acid mouthguard biosensor with integrated wireless electronics," *Biosensors and Bioelectronics*, vol. 74, pp. 1061–1068, 2015, https://doi.org/10.1016/j.bios.2015.07.039, ISSN: 0956-5663.

[23] Y. Lee, C. Howe, S. Mishra, *et al.*, "Wireless, intraoral hybrid electronics for real-time quantification of sodium intake toward hypertension management," *Proceedings of the National Academy of Sciences*, vol. 115, no. 21, pp. 5377–5382, 2018, https://doi.org/10.1073/pnas.1719573115.

[24] T. Arakawa, K. Tomoto, H. Nitta, *et al.*, "A wearable cellulose acetate-coated mouthguard biosensor for in vivo salivary glucose measurement," *Analytical Chemistry*, vol. 92, no. 18, pp. 12 201–12 207, Sep. 2020, https://doi.org/10.1021/acs.analchem.0c01201, ISSN: 0003-2700.

[25] C. Peppelman, "New advancements in head-worn sensors," http://resolver.tudelft.nl/uuid:16839b01-4e63-42da-86f4-02d5ed200739, M.S. thesis, TU Delft, Sep. 2023.

[26] MATRIX Industries, *Nanopower energy harvesting synchronous boost converter with microwatt cold-start, input impedance matching and regulated output*, https://www.mouser.com/datasheet/2/1000/MATRIX_Industries_09062019_MCRY12-125Q-42DI-1634489.pdf, Last accessed: Mar. 3, 2024, Jun. 2019.

[27] V. Yi-Fong Su, K.-T. Chou, C.-H. Tseng, *et al.*, "Mouth opening/breathing is common in sleep apnea and linked to more nocturnal water loss," en, *Biomed J*, vol. 4, p. 100 536, ma 202, https://doi.org/10.1016/j.bj.2022.05.001.

[28] F. Safiruddin, I. Koutsourelakis, and N. de Vries, "Analysis of the influence of head rotation during drug-induced sleep endoscopy in obstructive sleep apnea," *The Laryngoscope*, vol. 124, no. 9, pp. 2195–2199, 2014, https://doi.org/10.1002/lary.24598.

[29] STM32World Wiki, *Stm32 internal temperature and voltage reference*, https://stm32world.com/wiki/STM32_internal_temperature_and_voltage_reference, Sep. 2022.

[30] aaaTU Delft Sustainable Systems Lab, *Densor Source code repository: Hardware and software artifacts*, https://github.com/TUDSSL/densor/tree/master, Last accessed: Jan. 10, 2024, Jan. 2024.

[31] Scikit-learn developers, *scikit-learn, machine learning in python*, https://scikit-learn.org/stable/index.html, Last accessed: Jun. 3th, 2024.

[32] Nexperia B.V., *BAS40 series; 1PSxxSB4x series general-purpose schottky diodes*, https://docs.rs-online.com/6796/0900766b81270f45.pdf, Last accessed: Feb. 19, 2024, Mar. 2015.

[33] Texas Instruments, *TS3A44159 quad spdt analog switch*, https://www.ti.com/lit/ds/symlink/ts3a44159.pdf?ts=1708948686558, Last accessed: Feb. 26, 2024, Feb. 2024.

[34] STMicroelectronics N.V., *Ultra-low-power Arm Cortex-M0+ MCU with 16-Kbytes of flash memory, 32 mhz CPU, AES*, https://www.st.com/resource/en/datasheet/stm32l021f4.pdf, Last accessed: Feb. 16, 2024, 2017.

[35] STMicroelectronics N.V., *LIS2DW12 3-axis MEMS accelerometer, ultra low power, configurable single/double-tap recognition, free-fall, wakeup, portrait/landscape, 6D/4D orientation detections*, https://www.st.com/resource/en/datasheet/lis2dw12.pdf, Last accessed: Feb. 16, 2024, 2019.

[36] Abracon Corp., *AB18X5 real-time clock*, https://abracon.com/Precisiontiming/AB18X5-RTC.pdf, Last accessed: Feb. 16, 2024, 2014.

[37] STMicroelectronics N.V., *AN4509 tilt measurement using a low-g 3-axis accelerometer*, https://www.st.com/resource/en/application_note/dm00119046-tilt-measurement-using-a-low-g-3-axis-accelerometer-stmicroelectronics.pdf, Last accessed: Feb. 26, 2024.

[38] I. S. Fathi, M. A. A. Makhlouf, E. Osman, and M. A. Ahmed, "An energy-efficient compression algorithm of ecg signals in remote healthcare monitoring systems," *IEEE Access*, vol. 10, pp. 39 129–39 144, 2022, https://doi.org/10.1109/ACCESS.2022.3166476.

[39] M. Hooshmand, D. Zordan, T. Melodia, and M. Rossi, "Surf: Subject-adaptive unsupervised ecg signal compression for wearable fitness monitors," *IEEE Access*, vol. 5, pp. 19 517–19 535, 2017, https://doi.org/10.1109/ACCESS.2017.2749758.

[40] J. De Roose, H. Xin, M. Andraud, P. J. Harpe, and M. Verhelst, "Flexible and self-adaptive sense-and-compress for sub-microwatt always-on sensory recording," in *ESSCIRC 2018 - IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, https://doi.org/10.1109/ESSCIRC.2018.8494270, 2018, pp. 282–285.