UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
and
UNIVERSITY OF APPLIED SCIENCES KONSTANZ

MASTER THESIS No. 1536

Enhancement of sensor mesh functionality with application on sleep tracking

Bruno Vunderl

Konstanz, August 2017

Master thesis

Conclusion of Master Degree

Master of Science in Computing (M. Sc.)

at the

University of Zagreb
Faculty of Electrical Engineering and Computing

and

Hohschule Konstanz
Technik, Wirtschaft und Gestaltung

Topic:	**Enhancement of sensor mesh functionality with application on sleep tracking**
Master candidate:	Bruno Vunderl, Havidićeva 22, 10010 Zagreb
First supervisor:	Prof. Ralf Seepold
Second supervisor:	Prof. dr. sc. Mario Kovač
Issue date:	10.3.2017.
Submission date:	30.08.2017.

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING
MASTER THESIS COMMITTEE

Zagreb, 6 March 2017

MASTER THESIS ASSIGNMENT No. 1536

Student: **Bruno Vunderl (0036455534)**
Study: Computing
Profile: Computer Engineering

Title: **Enhancement of Sensor Mesh Functionality with Application on Sleep Tracking**

Description:

The focus of this project is enhancement of functionality, reliability and sensor accuracy of an intelligent bed that monitors human sleep. The scope of the project includes the implementation of an application layer protocol and network communication between the embedded system in the bed and remote server. Furthermore, the implementation possibilities of data preprocessing, filtering, and automatic sleep analysis are explored and tested. The system is tested and evaluated in the Ubiquitous Computing Laboratory at the Hochschule Konstanz University of Applied Sciences.

Issue date: 10 March 2017
Submission date: 30 August 2017

Mentor:

Ralf Seepold, PhD

Full Professor Mario Kovač, PhD
(co-mentor)

Committee Chair:

Full Professor Mario Kovač, PhD

Committee Secretary:

Full Professor Danko Basch, PhD

Author's Declaration

Unless otherwise indicated in the text, references are acknowledged below. This report:

Enhancement of sensor mesh functionality with application on sleep tracking by Bruno Vunderl on 30.08.2017.

is entirely the product of my own scholarly work. This report has not been submitted either in whole or part for a degree at this or any other university or institution. This is to certify that the printed version is equivivalent to the submitted electronic one.

Konstanz, 30.08.2017.

Bruno Vunderl

Contents

1 Introduction

To help readers get better acquainted with the topic, introduction is divided into four sections. First section covers general motivation and relevance of the project. The next section describes state of technology, market and consumer trends of the time this thesis' publication. The following section lays out project goals and defines the scope that they will be tackled on while in the last section, project structure is outlined in such a way that readers can easily navigate through this thesis body.

1.1 Motivation

Sleep is seemingly a trivial thing - from the moment that they are born, all humans have a need to sleep. It is a natural function in the same way breathing and other vital body functions are. Having slept for adequate time and with good quality tends to make people feel good and have more energy performing their daily tasks. When a person does not sleep well or does not sleep enough, it will usually negatively reflect both on their body and behavior. National Sleep Foundation (NSF) along with multi-disciplinary expert panel recommends sleep time for each age group ranging from 14 to 17 hours daily for newborns to between 7 and 8 hours for older adultsNSF. Sleep deprivation effects motor and cognitive abilities as well as mood but these effects can also occur in cases of bad sleep quality regardless of the sleep duration[1]. That same sleep quality is influenced by many factors ranging from physical ones (such as sleeping environment and position) to subjective ones (such as emotional state and dreams). As clear separation of these factors is rarely possible, most of the researches relied on the isolation of influences comparing results between large control and influenced groups. Sleep quality is then usually determined by questionnaires and data analysis which result in quantitative results such as Pittsburgh Sleep Quality Index[2].

To accurately and consistently determine sleep quality, sleep is divided into 4 stages. First stage is called N1 and is a transition between awakeness and sleep. Person is still conscious and aware of their surroundings. Duration of this stage is usually between 5 and 10 minutes. The second stage is called N2 and is categorized by a steady breathing and heart rate as well as with a drop in body temperature. This stage occurs multiple times during sleep and totals for around 50% of the time spent sleeping. Together these categories can be classified as light sleep. Stages N3 and N4 can be categorized as deep sleep as it becomes harder to wake up a person from these stages. They can be recognized by further temperature drop, lower blood pressure and delta waves[1] emitted by the brain. Sleep state can also be categorized as Non-Rapid Eye Movement (NREM) or Rapid Eye Movement (REM). REM phase usually occurs only after an initial stage of deep sleep. It is a phase in which dreams occur whilst eyes move quickly in different

[1]High amplitude brain waves with a frequency of oscillation between 0.5 and 4 Hz.

directions with heart and respiration rate becoming irregular. To prevent a person from waking up, muscles and senses below the neck become inactive. REM phases alternate with light and deep sleep stages and tend to become longer with each alternation. Adults with healthy sleep habits spend 20% of time asleep in a REM phase while this percentage becomes lower with age. Typical alternation of sleep stages during sleep is shown in Figure 1.1.

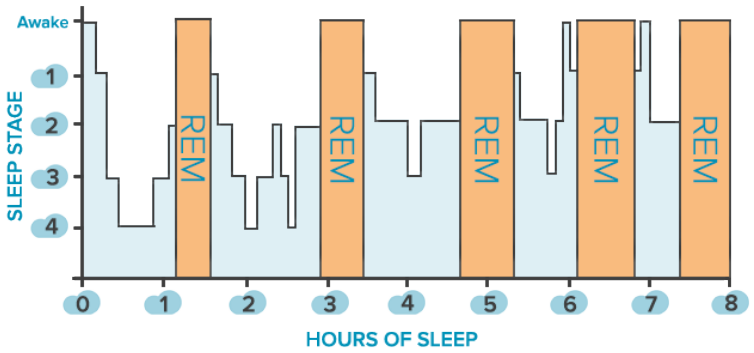


Figure 1.1: Sleep phases during typical 8 hour sleep.

As it is impossible to track these sleep phases personally, a variety of measurement devices is used in a study called Polysomnography (PSG). This study involves continuous or periodical measurement of physical parameters. Electrophysiological measurements are done with the help of Electroencephalography (EEG), Electromyography (EMG) and Electrooculography (EOG). The drawback of using these method is a requirement of complex equipment, necessary knowledge to evaluate the results and a controlled environment which is why these measurements are usually done only for clinical or research purposes. But simple sleep monitoring can be done using much simpler processes - with heart rate, body movement and position tracking. Unlike before mentioned method these measurements can be done unobtrusively and in a home environment. Improving the process and accuracy of these methods and improving correlation of collected results to the real sleep parameters may lead to much easier diagnosis of sleep disorders. Furthermore, availability and accessibility of this technology will encourage larger number of people to monitor their sleep performance which may lead to better mood, efficiency of executing daily tasks and sport results. This thesis will primarily focus on proposing a non-obtrusive way to track both sleep time and quality with a proposal of technology and measuring methods for consumer purposes.

1.2 Technology and current consumer market trends

In recent years, the market for sleep and fitness tracking devices has been expanding with the support of almost all major smartphone manufacturers. Some new brands specializing in the making of such devices have also emerged and have been steadily gaining the market share. Most of the devices that are currently used for consumer sleep tracking are actually multifunctional devices such as smart watches, armbands and rings. Beside sleep, they usually track physical activity, pulse and show time or provide some other information. Smart watches are additionally customizable as they usually support installation of third-party applications. This versatility makes such devices

very attractive to the customer regardless of sleep tracking and monitoring quality built into the devices. To paint a better picture, in 2014 Dr. C. Winter compared a few of the most popular sleep tracking armbands to the PSG[3]. His results are showing that most of the devices, regardless of their cost, were able to distinguish between awakeness and sleep which allowed them to measure the time spent sleeping. Unfortunately, they were not able to separate REM, N1, N2, N3 and N4 sleep phases or estimate the time spent dreaming. Some devices provided an estimate if a person was in a deep or light sleep but the results were mostly inaccurate. In late 2016 J. Yoon tested newer iteration of the consumer devices and the results seen in Figure 1.2 showed improvement of the deep or light sleep phase detection but devices were still not accurate enough to guess the real sleep phase with an acceptable degree of certainty[4].

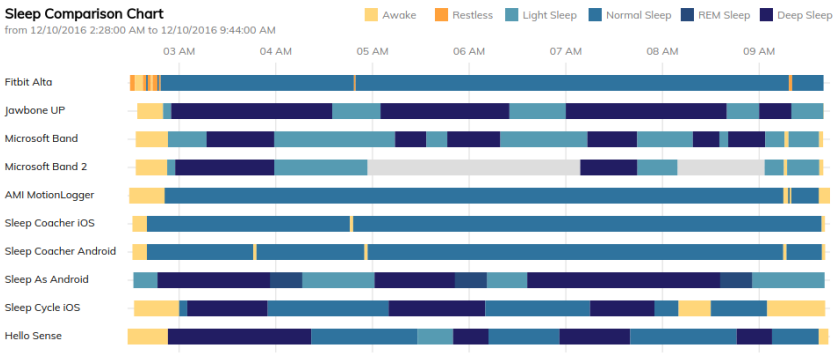


Figure 1.2: Sleep detection comparison between consumer devices

The reason for inaccuracy of consumer on-body sleep tracking devices is the underlying technology. Microelectromechanical systems (MEMS) sensors are used for movement tracking for actigraphy[2] and simple photo sensors are used for Photoplethysmography (PPG) which gives an estimate of pulse frequency. Most of the devices are designed in such a way that they are non-obtrusive, small, easy to use and smart looking. This means that batteries powering sleep monitoring devices must be small and device usage should be minimized to maximize the battery life. This is achieved through the use of low power microcontrollers, through updating movement sensor data in an interrupt routine which wakes the microprocessor up when a movement occurs and through minimizing the number of readings done by the PPG sensor. This, of course results with the inability of devices to categorize sleep phases with acceptable certainty so most manufacturers categorize sleep as just light or deep.

Contactless consumer devices that are specialized for sleep tracking and monitoring are newest to the market. They are using sound to detect breathing and body movement through the night. Depending on the product they can also measure light for easier start of sleep detection. Since smartphones also have microphones and light sensors, multiple applications which analyze the sleep are also present on the market. This method is favorable in some cases because it eliminates the need for a device touching the subject. But what it gains in practicality, this method lacks in accuracy as sound and light sensors are easily disturbed by the events present around the sleep environment. This method has also a problem of distinguishing multiple sound sources eg. multiple people sleeping in the same room.

[2] a non-invasive method of monitoring rest and activity cycles where a device worn by subject records movements

1.3 A poll on perceived influence of sleep and usage of sleep tracking devices

Young adults are an age group with the most early adopters of new technologies. In general and due to the lifestyle, they are likely to have suffered from short term or long term sleep deprivation. Getting the best sleep quality with the minimal time spent sleeping can be a beneficial factor to the outcome of exams and handling of stressful tasks. A poll was conducted between peer students at the University of Zagreb with a goal to analyse the perceived influence of sleep and usage of sleep tracking devices in that group. It should be taken into consideration that the total number of poll participants was 63 and that they are localized both geographically and by social group. Therefore the results will be compared to other polls which include more data in both quantity and diversity. A result from this poll will be used in case that no data on the subject in question was found.

Conducted poll results show that majority of students are on average getting 7.2 hours of sleep daily which is quite close to an average of 7.15 hours determined by a study conducted by B. Wilt for Jawbone[5]. In that study results from a large number of smartband Jawbone Up device worn by students were analysed. For 65% of students in Zagreb this amount of sleep is adequate to their needs which is in accordance with NSF who suggest between 7 and 9 hours for younger adults[6]. To get a perception what influences their sleep, they were asked if sleep duration, sleep environment[3] and external conditions[4] influence their sleep quality. The results show that much bigger percent of participants perceive that sleeping environment and external conditions impact the quality of sleep compared to pure duration of the sleep. Results can be seen in Figure 1.3. 86% of poll participants indicated that they would like to have an insight into their sleep but most of the participants (56%) indicated that they are not certain if that data would improve their sleep quality.

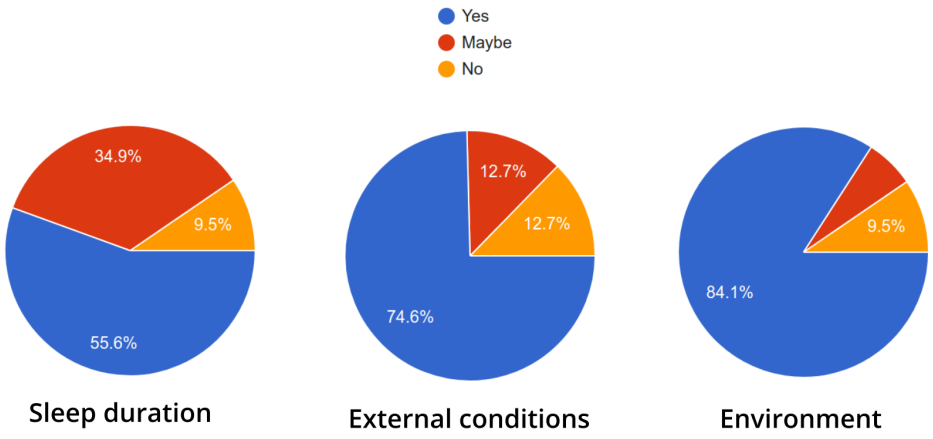


Figure 1.3: Sleep detection comparison between consumer devices

As to what sleep means to them, they were asked how much on a scale of 1 to 10 does the sleep quality correlate with their daily performance. Performance was separated

[3] eg. bed quality, sleeping garments
[4] eg. temperature, humidity, pressure, noise levels, moon phases

in 3 different categories - task performance, mood and physical performance. Task performance is described as perceived efficiency of work and school task completion. Mood is also an important aspect as good mood may affect both professional and private life. Influence of sleep on physical performance may be quite important for students involved in competitive sports activities and a good quality sleep may help them improve their performance. Average, median and standard deviation of these results are displayed in 1.1.

Influence	Average	Standard deviation	Median
Daily task performance	7.51	1.42	8
Mood	7.54	1.49	8
Physical performance	7.40	1.28	8

Table 1.1: Perceived influence of sleep on scale of 1 to 10.

Little more than half (57%) of participants are familiar with sleep tracking devices but only 27% have used a non obtrusive sleep tracking device or devices. Out of all participants, 6 have tested their quality of sleep using clinical methods such as EEG, EMG or EOG. As widely available sleep trackers are still quite new to the market, 76.5% of the participants owning a sleep tracking device have been in a possession of it for less then a year. More than a half (58.7%) of participants that own a sleep tracking device indicate that they check their sleep quality on a weekly basis or more frequently while 29.4% people are checking their devices rarely. A relative majority of 41.2% indicated that data received from the device didn't make them change their sleep habits while 35.3% indicated that they have changed their sleep routines. The remaining 23.5% declared themselves as not certain. Most of the poll participants indicated that for sleep tracking they use smartbands, with the cheapest one (Xiaomi Mi Band) being the most popular. But a considerable amount of 29.4% participants have also used their smartphones for sleep tracking. These are not precise devices and methods. Only 5.9% of group says that they wouldn't like to have a more precise and detailed device to track their sleep.

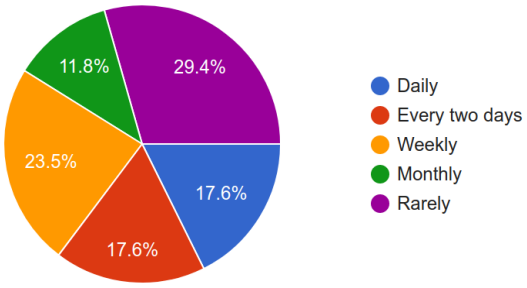


Figure 1.4: Frequency of sleep data review by poll participants.

So what can be concluded from this poll? Well, there is a large market share of younger adults who are not familiar with sleep tracking devices. Even if they are familiar, most of them don't own one despite perceiving that data obtained from those devices could help them sleep better. This means that market for sleep tracking devices has a very large potential growth. It is also visible that poll users would like to have more precise and descriptive devices.

1.4 Goal and scope

This thesis will try to describe a novel implementation which will track sleep for both clinical and consumer purposes improving on the quality of tracking and simplicity of use over the currently available solutions. Proposed solution is based on a pressure sensor mesh network which is placed under the mattress and which tracks the movement and vital signs of the sleepers. It continues on the previous work done at Ubiquitous Computing Laboratory (UC-Lab) at Hochschule Konstanz für Technik, Wirtschaft und Gestaltung (HTWG) and enhances it by providing system architecture, hardware and software required to achieve a goal of precise contactless sleep tracking. Thesis also proposes an interface that can be used to collect and analyse acquired data and will serve as a stepping stone for future research.

Hardware is designed in such a way that it allows an easy installation and so that all of the components are widely available and easily replaceable. Thesis describes design decisions in detail with the proposal of future improvements in terms of features, reliability and precision. Embedded software setup is based on open-source solutions which are not tied to a specific platform which means that an end product may use the same software albeit possible changes in the hardware. Application software provides user interface for an analysis of data but also provides an Application Programming Interface (API) which allows other services to access the data in a standardized way. Together the whole system allows recording, tracking and analysis of sleep data.

In this scope, the reader will be acquainted with the process of design and implementation of such a system. Problems regarding communication between sensor nodes, endpoint data collection and graphical data display will be described in detail. Thesis will also present and give an insight into the results of how system functions. Possibilities for implementation of preprocessing, filtering and automatic sleep analysis will also be presented. What will not be in focus of this thesis however are the medical aspects of sleep recognition and sleep stage classification. They will be considered and reviewed, as they are critical to the functional aspect of the project, but they remain to be described and analysed in a future research to which this thesis will hopefully serve as a technological foundation.

1.5 Thesis outline

Before the new system design is proposed a current one found at UC-Lab will be presented and reviewed. Current system design decisions will be shown along with the implementation based details in chapter 2. Focus in chapter 3 will be on implementation of a mesh network nodes that will serve the purpose of data collection from the sensors. It will present design considerations and decisions that led to the final product. Details on an embedded system serving as an endpoint will be described in the chapter 4. In it a part of application software which takes care of communication between the endpoint and sensor nodes will also be described. In chapter 5 it will be shown how data is stored and displayed and how it can be used by other services. Measurement results will be shown in the chapter 6 after which a conclusion will be drawn in chapter 7.

2 Evolution of system design

To track sleep unobtrusively, multiple techniques and sensor types have been tested and evaluated by other researchers. Placing load cells under the bed supports allowed researchers to determine the precise time when subject fell asleep and when subject woke up[7]. Infrared camera recording of subjects sleeping allowed precise recognition of small movements even under the blanket [8]. Another research group used Plastic optical fiber (POF) sensors to recognize breathing patterns and detect apnea[9]. The same results were also achieved using pneumatic pressure sensors placed in a sealed air-cushion under the mattress[10]. In yet another research, a group of researchers conducted experiment in which they placed two $24GHz$ radars under the bed and found out that it is possible to accurately recognize hearth rate[11].

But, one of the recently most popular methods of unobtrusive sleep tracking is much simpler and more affordable than the others. It is called pressure sensing and it involves continuous measurement of pressure from under the subject. This thesis picks up on the work of Prof. Dr. Ralf Seepold, Raína Kuhn, Daniel Scherz and Maxime Guyot at UC-Lab[12][13] who have successfully used pressure sensors to determine position, detect movement and track vital signs during sleep.

2.1 Devices and technology

To achieve a good sleep tracking from pressure readings under the bed, an appropriate environment has to be selected. Environment consists of an adequate bed frame, a mattress and of base-plates which hold the mattress in the frame of the bed. Bed frame is of a regular size - $90 * 200cm$ which accommodates a vast majority of people. Because of good pressure propagation, mattress should not be too firm or too tight. Therefore, mattress of uniform hardness level 2 has been selected.

To hold the mattress in place, a grid of pressure-disks is used. They are a critical part of the system as they have to absorb the pressure from the mattress. Also, they are the point at which pressure measurement can be done. To provide a better granularity, pressure-disks provided by ErgoProTech and depicted in Figure 2.1 were used. They were placed 3cm from each other under the whole area of the mattress according to the manufacturers usage recommendation. This ensures adequate support across the mattress. For selected mattress size, base-plate grid consists of 12 rows and 5 columns totaling in 60 pressure-disks. Pressure-disks are made of semi-elastic polymer and consist of 4 connected rectangular blades. These blades are supported by 4 double arms anchored at the same point where the whole structure is connected to the bed frame. There are multiple hardness levels of base plates and they are differentiated by the color of the rectangular blades. Firmer base plates are grey while more elastic ones are purple. More about placement of pressure-disks can be found in section 2.2

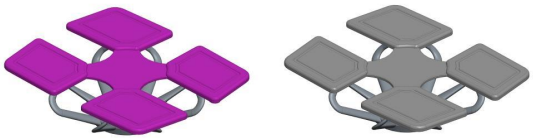


Figure 2.1: Base plates in the bed.

There are multiple types of pressure sensors that could be used for the purpose of this project[14]. Potentiometric pressure sensors are very crude due to their construction and often have reliability and hysteresis issues. Inductive pressure sensors require Alternating Current (AC) excitation of coils and consequentially signal filtering and demodulation. Piezoelectric and piezoresistive pressure sensors measure a change of pressure using piezoelectric effect. Capacitive pressure sensors use a small diaphragm as a capacitor plate. When pressure is applied, diaphragm deflects and capacitance changes. Change may or may not be linear and usually is on the order of a few pF while total capacitance of the sensor is between 50 and $100pF$ which means that it may be hard to precisely measure the values. This method may also suffer from environmental effects and inadequate Printed Circuit Board (PCB) or protoboard design decisions. Force sensing resistor (FSR) is a type of material whose resistance changes when pressure is applied. Advantages of FSR sensors over other pressure sensors are possibility of detecting static pressure, its flexibility, thinness and inexpensiveness. Multiple other researches were made using the same type of sensor and have proven its reliability and accuracy when used in a sensor grid[15][16][17]. Therefore, this type of sensor was selected for use in this specific project.

So how does the FSR sensor work? FSR is a Polymer Thick Film (PTF) device which exhibits a decrease in resistance with an increase of the force applied to the active surface[18]. At 'zero force', conductive ink is separated from the active area by spacer adhesive and in that case FSR sensor has the highest possible resistance. When pressure is applied to sensor, conductive layer is pushed down onto the active area which results in a decrease of resistance. Construction of FSR sensor is shown in Figure 2.2. For a hemispherical sensor with a diameter of 12.7mm (model 402) sensibility starts just below $20g$ and extends to around $10kg$ when saturation occurs. Passing a threshold at around $20g$, resistance changes from greater than $100k\Omega$ to $10k\Omega$. After that resistance falls logarithmically with an increase of force as seen in Figure 2.2. For consistent results application manual[18] suggests using a firm, flat and smooth mounting surface and use of a rubber spring to spread the pressure over the whole sensitive area. Also, an appropriate sensor size and shape is to be used.

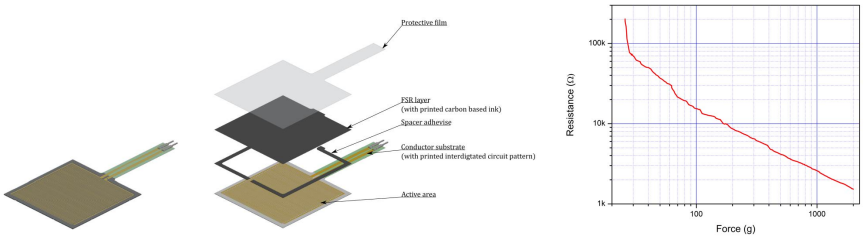


Figure 2.2: FSR sensor construction and resistance characteristics.

There are 4 different types of standard of-the-shelf sensors[18] and their sizes and shapes are shown in Table 2.1. Models 400 and 402 also have short-tailed variants which feature shorter connection between FSR pads and pins found at the end of the lead wires. These same models have a small pressure sensing surface and are not as well suited for this project. Models 406 and 408 have much larger contact surface area, which means that a more consistent distribution of pressure is possible. Model 406 is perfect for use in areas that require a good position resolution such as scapular area. On the other hand, type 408 can be used in crural region.

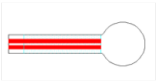
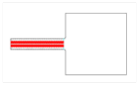
Part number	Description	Part image
Model 400	0,2" circle	
Model 402	0.5" circle	
Model 406	1.5" square	
Model 408	24" strip	

Table 2.1: Standard shapes and sizes of FSR sensors offered by Interlink Electronics.

To get a reading of sensor resistance (R_{fsr}), a sensor is connected in a series with a fixed value reference resistor (R_{ref}). Then, an input voltage (V) is applied to the circuit. Voltage drop (V_{fsr}) is measured on the FSR sensor leads. Pressure applied to sensor is in a reciprocal correlation to the R_{fsr} because FSR has a maximal resistance when there is no external force pressuring its surface as seen in Figure 2.2. The same graph is also sampled for force-resistance pairs which are used for reference resistor selection. R_{fsr} needs to be selected in such a way that it has best resolution for force between $0kg$ and $1.6kg$. These weight values were selected based on R. Kuhns calculation[12]. She took an average weight of a person and mattress and calculated an estimate of how much weight each of the disk springs carry. Equation 2.1 describes the relation between V_{fsr} and R_{fsr} when a R_{ref} has a fixed value. Multiple standard resistor values were put into the equation and at resistance of $10k\Omega$ change gradient was highest. Therefore, $10k\Omega$ resistor was used as R_{ref}.

$$V_{fsr} = \frac{V * R_{ref}}{R_{fsr} + R_{ref}} \tag{2.1}$$

Initial sensitivity tests that were conducted by R. Kuhn and M. Guyot showed that additional layer should be added on top of the FSR sensors to help with pressure absorption. In Figure 2.2 it is clearly visible that adhesive spacer layer creates a non-sensitive frame around the active sensor area. When sensor surface was directly exposed to the mattress, adhesive absorbed most of the pressure as it was not as elastic as active area. This was solved by using felt[1] gliders. This greatly improved the sensitivity and results can be seen in Figure 2.3.

[1]textile material that is produced by matting, condensing and pressing fibers together.

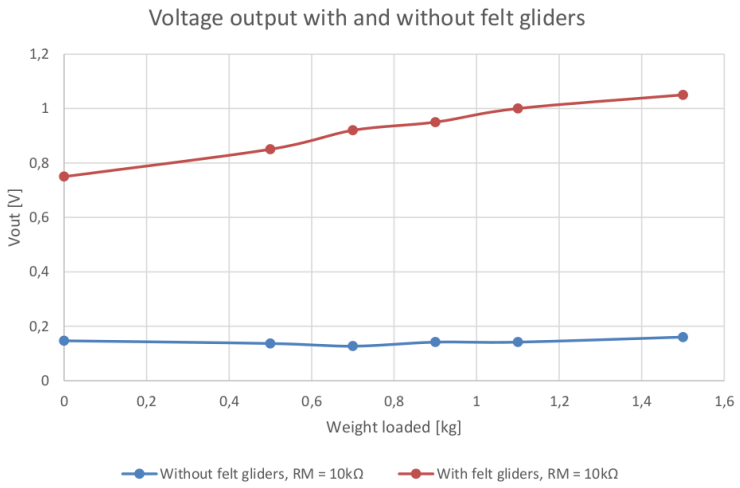


Figure 2.3: Comparison of pressure with and without felt gliders.

To convert voltage to a digital value, Analog to Digital Conversion (ADC) was done with a help of a microcontroller. Sensors were connected to the Trinket Pro 5V microcontroller development board[19]. This board features ATMega328P microcontroller with integrated ADC functionality[20]. From a myriad of different boards this one was chosen because it can be programmed as Arduino Pro Mini but features 8 analog input pins. Unfortunately, two of the analog pins share functionality with Inter-Integrated Circuit (I2C) protocol and 1 was used for board identification. This means that 1 board could support up to 5 sensors. To collect readings from multiple devices, already mentioned I2C protocol was used. A device that takes readings from the pressure sensors and can communicate with the rest of the system will be called a node in the rest of the thesis.

But it would be quite impractical to connect a Personal computer (PC) to each and an every node to collect data. So the system was designed with a new device as an endpoint. This device communicates as I2C master with the nodes and allows easier communication between user and the nodes. For purpose of an endpoint, Intel Edison Computer On a Module (COM) was used. It features Intel Atom Central Processing Unit (CPU) and Intel Quark 32-bit microcontroller[21]. Both have x86 architecture and use x86 instruction set. But what is more important, Intel Edison has $4GB$ Embedded Multimedia Card (EMMC) storage as well as integrated *Bluetooth* and *Wi-Fi*. Using Wi-Fi, sensor readings paired with their position were transmitted to locally situated web server. Web server application saved the incoming data into the database and provided API for the client application. Using client application, user was able to calibrate and review sensor data.

2.2 Test environment

To get a better result recognizing sleep position, a sleeping recognition sensor network has to be designed in such a way that it is able to detect all of the most common sleep positions. Figure 2.4 shows 6 positions people most often sleep in. From left to right those are fetal position, log position, yearner position, supine position, starfish position

and prone position. Fetal, yearner and log position can be left or right depending on the bed side subject is facing. But since there is no difference in weight distribution between left and right log position, it is possible just to classify position as log position. This means that total of 8 different positions can be classified using pressure sensing. Using commercial pressure sensing mat and pattern matching, a research conducted in a clinic has achieved 97% classification accuracy[22] which means that this technique is very reliable. But, putting a FSR sensor pad on every possible blade would require 240 sensors and 48 nodes which is not only expensive in terms of hardware cost but also of time used for setting up the system. The easiest way to minimize the complexity would be reducing the number of sensors but it has to be done in such a way that there is no significant classification accuracy loss compared to fully populated base-plate area. Since vertical movement during sleep is rare and all of the major body regions exceed the surface area of a sensor, the emphasis was placed on horizontal resolution. Pressure sensors were placed on two blades in the same row while the other row was left without sensors. This decision reduced the number of required sensors from 240 to 120.

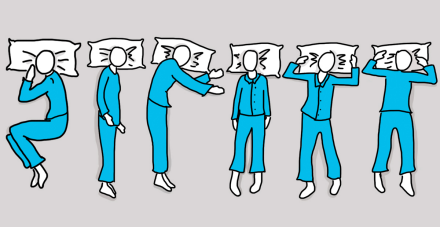


Figure 2.4: Different sleep positions.

To retain the possibility of accurate position recognition but to further minimize the number of sensors used, a look was taken into the weight distribution during supine and log position. In Figure 2.5 we can see that mattress deflects the most in scapular, gluteal and crural regions. Scapular area is especially important because it is the area where respiration and heart rate detection is done. Other areas of the body are not as important to create an accurate picture of the body position and to measure vital signs. Also, at the edge of the bed frame, mattress is transferring a significant amount of pressure to the frame which means that these pads are not very sensitive. As a conclusion, 48 sensors were placed in 6 rows as seen in Figure 2.5 and according to M. Guyots[13] sensor layout proposal.

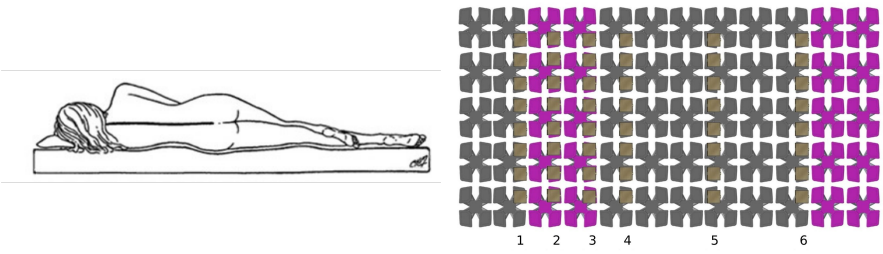


Figure 2.5: Sensor arrangement in bed compared to sleep position.

The described system with 48 sensors required 10 Trinket Pro boards. As all of the boards were communicating using I2C protocol in which each of the slaves has a unique address, addresses had to be distributed in accordance with their physical position. This required either a different firmware for every board or some smarter alternative in which

every board would automatically select different I2C address in an orderly way. The solution that was proposed involved using same-value resistors in a series. Depending on the voltage difference between identification resistor and ground, an I2C address was chosen by the microcontroller. Although this solution was easy to implement, it had used up one ADC pin that could have been used for sensor connection. Since Trinket Pro does not have prototyping holes, 5 reference resistors for sensor reading and 1 for position identification had to be added. For every reference resistor a perforated electronic prototyping board was cut and a resistor and 3 wires were soldered. Also, position identification resistor was soldered onto a piece of perforated board and connected between two adjacent Trinkets. This means that adding a new node with 5 sensors to the system required 6 new small boards to be soldered. Wiring was something that could have been improved with introduction of a signal bus.

2.3 A new architecture

To address this hardware shortcomings, a new hardware architecture was proposed. In it, sensor nodes were implemented as PCBs which featured all previously additionally soldered resistors. A new method of node position identification was developed without the use of identification resistor and connection between nodes was implemented using standardized connectors and cables. This means that FSR sensors can now be directly connected to the nodes. Proposed solution eliminated a need for perforated boards and use of soldering iron for installation. The total number of nodes was also reduced to a third by using microcontrollers with more ADC inputs. This was done to reduce system cost, power consumption and to make installation and modification of development much simpler.

Endpoint was reimagined to integrate data storage and server functionality so no external servers are now needed for a system to serve data to the client. Web based user interface was also implemented on the endpoint so that results could be viewed from a PC or even from a mobile device. But what if there is a need for a service which monitors multiple beds at the same time such as in case of hospital or nursing home? Well, endpoint also serves data over well described API and allows easy readout of data collected from sensor network. A graph depicting a new system architecture is found in Figure 2.6.

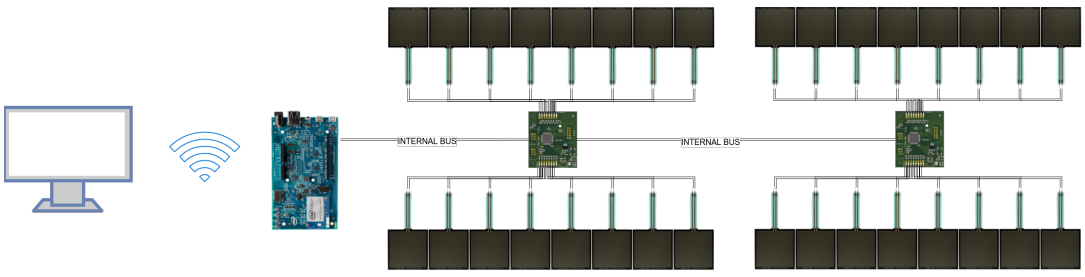


Figure 2.6: A newly implemented system architecture.

3 Sensor nodes

This chapter will describe functionalities of sensor node devices. After that it will describe how electronic parts were chosen. A separate section will include details regarding PCB design while another section will describe software developed for the sensor nodes. Both software and hardware files are available with additional instructions on GitHub page.

3.1 Physical design and connections

Most of the required PCB features were already described in the previous chapter - board should allow direct connection of FSR sensors, it should eliminate need for additional perforated boards and it should provide a more robust solution for physical board position detection. Furthermore, it should feature low power consumption and enable connection of at least two rows of 8 pressure sensors. Also it should feature small dimensions and provide mounting holes so that it doesn't have to be suspended by wires or taped to the bed. The ideal position for the installation of the board is under the bed base slates. This way it can easily be serviced.

There are 4 variants of pin endings found on FSR sensors. A variant with no leads is used for custom pin endings while solder tabs variant is used for direct soldering to the board. Because of the materials used for construction of the sensor, when heat is applied using standard soldering iron, there is a high possibility of sensor leads melting. This is why a female plug connector option was chosen. Distance between leads is 1/10" and they are compatible with standard PCB connector pins. Since sensors will be mounted on pressure disks which are found on the upper side of the bed base, while the board is found under the slates, elongation cables are required. In this case, DuPont 'jumper' cables will be used and a board connectors have been designed in such a way. Two edges of the board, are populated with 8 two-row connectors. Row on the top is connected to the ADC pins of the microcontroller while the bottom pins are grounded. In front of each of the pins, a reference resistor R_{ref} is found.

Connector that was used for internal communication bus features 6 wires - 2 wires are used for I2C, 2 are used for power supply and 2 are used for physical position recognition. Because they are interchangeable with DuPont jumper wires and because of toolless cable connector installation, Insulation-displacement connector (IDC) cables and connectors were used[23]. There are two 3x2 internal communication bus connectors on the board so that boards can easily be 'daisy-chained'. Close to the first connector, two pull-down resistors for I2C bus are situated. For microcontroller programming, the same type of connectors was used. The cortex debug interface consists of 10 pins so a single 5x2 connector was used. Graphic 3.1 shows connectors with positions of pins.

To power the microcontroller and because of debugging possibilities, a type B micro Universal Serial Bus (USB) connector was also added to the board. USB connection

Figure 3.1: Internal communication bus and programming connector layout.

features 2 differential data signals, a $5V$ and ground. Connector additionally has a 'on-the-go' identification pin which is left floating because the requirement does not require a board to become USB host. What board implements is a PRTR5V0U2X ElectroStatic Discharge (ESD) diode[24] which helps protect both the host PC and the PCB from electrical stress in form of surge or overvoltage.

3.2 Component selection and compatibility

A most important part of the node circuit is a microcontroller. Main requirement for it was the possibility of connecting 16 ADC devices without the use of additional ADC Integrated Circuits (IC). Since other members of UC-Lab have experience with programming Atmel microcontrollers, a choice was between ATxmega 8-bit AVR microcontrollers, 32-bit UC3 AVR microcontrollers and Atmel SMART ARM-based Microcontroller Unit (MCU)[1]. So let's take a look at a sample microcontroller from each of the categories. ATxmega64C3 is a 8-bit AVR microcontroller which features 64KB flash program storage and 4KB of Random Access Memory (RAM), 16 pins provide ADC with 12-bit resolution and there is native support for I2C[25]. AT32UC3C164C is a 32-bit AVR microcontroller with the same flash size and number of 12-bit ADC inputs but it features 20KB RAM[26]. ATSAMD21J16 is a 32-bit ARM based microcontroller which has 8KB RAM, 20 ADC channels which have programmable gain stage, automatic offset and gain compensation as well as a possibility to oversample the signal to get 16-bit resolution[27]. Considering that this microcontroller is cheaper than both AVR microcontrollers and that it supported by multiple third-party frameworks such as ARM mbed, Arduino and Simba, Atmel SAMD was chosen as a microcontroller family that will be used. Atmel SAMD MCU family microcontrollers have a naming pattern which makes easy to distinguish chips characteristics. An example will be described on is SAMD21J16B-AU. *SAMD* is product family, *21* is product series, *J* stands for pin count (J = 64pins), *16* is for flash memory density (16 = 64KB), *B* is chip variant, *A* is for package type (A = Thin Quad Flat Package (TQFP)) and finally *U* is for package grade (U = -40 - 85°C). A first prototype of the board was developed using ATSAMD21J18-AU which is the same as one found in Atmels SAM D21 Xplained Pro Evaluation Kit but that processors was out of stock when production boards were to be made. This is why ATSAMD21J16-AU, a pin-compatible variant with 64KB instead of 256KB of RAM, was used in production boards. Pin layout of a TQFP package can be seen in Figure 3.2. This pinout is same for all ATSAMD21 microcontrollers with 64-pin TQFP package so even smaller memory versions can be used for the cost-saving.

To power the selected microcontroller a $3.3V$ source is used. Since the most important feature the microcontroller does is analog to digital conversion, it's very important to have a stable and ripple-free Direct Current (DC) power supply. Oscillations in input

[1](abbrev. SAM)

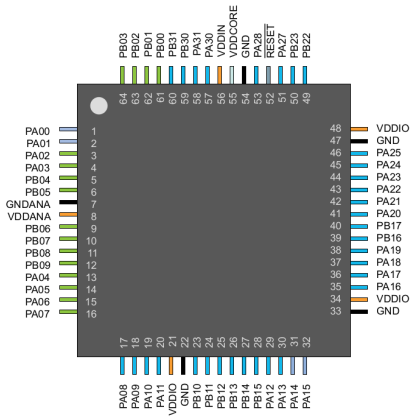


Figure 3.2: AT SAMD 64-pin TQFP package pin layout.

voltage to the microcontroller may lead to inconsistent voltage readings. Switching Mode Power Supply (SMPS) is a very efficient way of DC-DC conversion but it suffers from a constant high and low frequency voltage ripple. Most of high frequency and some of the low frequency ripple can be countered by using ferrite beads and capacitors but ripple still remains. Breakout board for Intel Edison, an endpoint used in this project, uses TPS62133 $5V$ step-down SMPS which can supply a maximum of $3A$ of current[28][29]. Out of available $15W$, Intel Edison consumes on average $0.41W$ with Wi-Fi enabled. This means that it is possible to use the same power supply for powering the nodes. To filter out the noise a Low-Droput (LDO) is found on each of the nodes. This is in accordance to the specification for high-performance ADC power supply design provided by Texas Instruments[30]. LDO that was chosen for this project is LD1117 which provides a stable $3.3V$ power supply with $1.1V$ dropout voltage. Input and output voltage lines of the power supply are bridged to the ground with capacitors to filter out the input noise. For debugging a $3.3V$ source can also be attached to $VREF$ pin of the programming connector.

3.3 PCB design and production

A PCB was designed in KiCad an open-source tool for electronics design automation. First, a microcontroller was placed with decoupling capacitors and ferrite bead according to the manual[27]. This ensures low noise during operation. After already described internal bus, programming, sensor and micro USB connectors were placed, PCB was populated with 2 status Light emitting diode (LED)s. First LED serves as on/off detection and is directly connected to the V_{cc} while the other one is user programmable and can be used for debugging. Board also has a power supply and a reset button. On 3 edges of the board a $3mm$ diameter holes were placed for easy mounting. The layout of the PCB can be seen in Figure 3.3.

To test the initial design, a board was milled at Department of Electronics at HTWG using LPKF ProtoMat S63 circuit board plotter. Since board design features vias, they had to be filled with copper rivets. Prototype production showed that micro USB pin had switched GND and USB OTG ID pins. This error happened due to difference of pin naming between footprint and schematic during the design. After this issue was fixed a board was sent for production to AISLER. The components were ordered from DigiKey

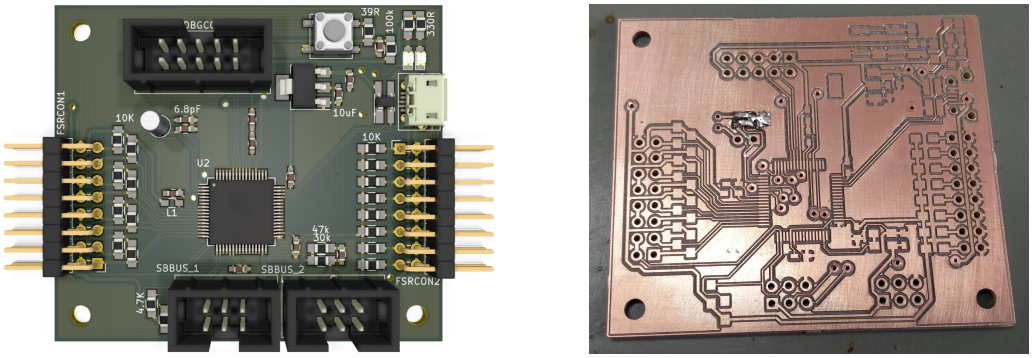


Figure 3.3: 3D view and prototype of printed circuit board developed for the project.

with already described change from ATSAMD21J18 to ATSAMD21J16. 3 boards were produced and were hand soldered. After production, every board was tested with a sample program reading from each of sensors. The final and assembled board can be seen in Figure 3.4

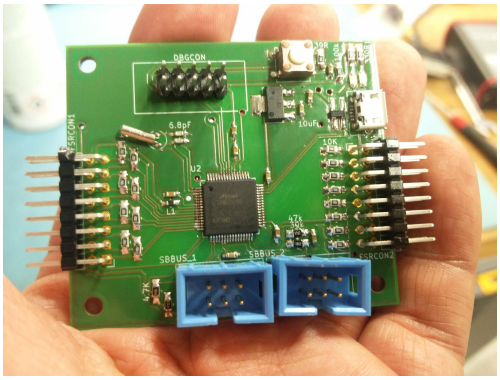


Figure 3.4: Fully populated sensor node circuit board.

3.4 Development environment

Because of the ARM architecture and Thumb2 instruction set, selected microcontroller supports multiple frameworks and development environments. One of most standardized and widely used frameworks is ARM mbed. It is compatible with almost all ARM microcontroller designs from vendors such as STMicroelectronics, NXP, Renesas, Nordic and Atmel[31]. Through abstraction it allows the same code to work on different devices. Depending on the selected device, application code is linked with platform specific libraries and a hex code output file is generated. This hex file can then be uploaded to the microcontroller using "drag and drop" or using Serial Wire Debug (SWD) interface[32]. In case of this project, mbed framework was used through Platformio open source ecosystem for Internet of Things (IoT) development. This environment integrates in Microsoft's VSCode code editor and allows code compiling, upload and debugging along with other features such as intellisense code competition, unit testing and continuous integration. Although both mbed and Platformio support ATSAMD21J18 and Atmels Xplained Pro development board, they provide no support for ATSAMD21J16 variant of the microcontroller. This is why a new configuration for both had to be made.

First, a new board configuration was added to the Platformio configuration folder found in `~/.platformio/boards`. Then a JavaScript Object Notation (JSON) structured file was created based upon Atmel Xplained Pro board file but maximum ram size and maximum size were changed because of the smaller RAM and flash size. Then a support for a new processor had to be added to the mbed Software development kit (SDK). First a new microcontroller and variant were declared an then microcontroller features were described in `.platformio/packages/framework-mbed/targets/targets.json`. They can be seen in Listing 3.1. Then, General Purpose Input Output (GPIO) port mapping of this chip variant was declared in `port_api.c`. Next required change included modification of load script. Read Only Memory (ROM) was set as rx memory from address 0x00000000 until address 0x00010000 while RAM was of rwx type starting from 0x20000000 and with size of 0x2000. Stack size was defined to be 0x500. After these changes were introduced, Board Support Package (BSP) was generated using tool 'tox'. After that, a board became visible in Platformio and programs could be compiled for it.

Listing 3.1: Description of mbed features implemented in ATSAMD21J16

```
"SAMD21J16A": {
    "inherits": ["Target"],
    "core": "Cortex-M0+",
    "macros": ["__SAMD21J16A__", "I2C_MASTER_CALLBACK_MODE=true",
        "EXTINT_CALLBACK_MODE=true", "USART_CALLBACK_MODE=true",
       "TC_ASYNC=true"],
    "extra_labels": ["Atmel", "SAM_CortexM0P", "SAMD21"],
    "supported_toolchains": ["GCC_ARM", "ARM", "uARM"],
    "device_has": ["ANALOGIN", "ANALOGOUT", "I2C", "I2CSLAVE", "
       I2C_ASYNCH", "INTERRUPTIN", "PORTIN", "PORTINOUT", "
       PORTOUT", "PWMOUT", "RTC", "SERIAL", "SERIAL_ASYNCH", "
       SERIAL_FC", "SLEEP", "SPI", "SPISLAVE", "SPI_ASYNCH"],
    "release_versions": ["2"],
    "device_name": "ATSAMD21J16A"
}
```

After program has been compiled and linked for the specific microcontroller, the hex file needs to be uploaded to the microcontroller flash storage. Segger JLink programmer is used for this purpose. To automate the upload a custom script and configuration file were written and integrated into the Platformio. Debugger interface using OpenOCD was also setup so that microcontroller could be debugged from the same development environment.

3.5 Software implementation

A software for the sensor node was written in C++ and compiled using gcc-arm-none-eabi. The main functionalities that were implemented are data readout from sensors and communication with the rest of the system. Each node can be described by the state which is defined by its I2C address, sensor state, number of active sensors, with value of each sensor and by currently executing function. To abstract this, a library containing classes used in the project was created. Classes describe data and structures that are used and implement some of the functionality. The rest of the functionality

is implemented in main file. It's worth noting that node does not save a longer sensor history but only upon requests serves most recent data.

Class Smartbed is a singleton class which holds state information and provides an easy access to functions regarding the system functionality. Address is a private property of the class and it holds the 7-bit I2C address. The address can safely be changed and accessed using mutator methods. Private property `active_sensors` saves information which sensors are active in an uint32 data type. For example, if sensor 14 is active, bit 14 is set to 1 while for inactive sensors bits are set to 0. When a change occurs it's considered that sensor is active while an inactive sensor reads the maximum voltage the hole time. Sensor values and pin positions are described in class Sensors which is referenced by Smartbed class. It is also a singleton type and provides methods which access one or all of the sensors and save data in the private property array. Since respiration and hearth pulse are low intensity signals, it is important to have the best possible resolution. Standard resolution of ATSAMD21 ADC unit is 12-bit but through the use of subsampling it is possible to get 16-bit resolution which greatly helps in signal recognition.

Two other classes are used for storing temporary communication information. I2C packets consist of 7-bit address, read/write flag and data byte or bytes. When a packet is received, an object is created with separated data as a class property. I2C also specifies that after every received byte an acknowledgement bit should be set by receiving party. On node side, this is handled by the I2C unit of the microcontroller. An endpoint can as a bus master broadcast data using general call address 0, write data to a specific register of the slave or request a read from the device[33]. To distinguish between access types, a class called `I2C_RESULT` stores the access type as an enum. This property is set when object is created. Constructor method is overloaded and saves access type depending on the method parameters. If no parameters were supplied, package is invalid and type set to None. If only register address is supplied, master is requesting to read a data from the specified register. If register address and value are supplied master is writing to a register. All of the described classes are shown in Figure 3.5.

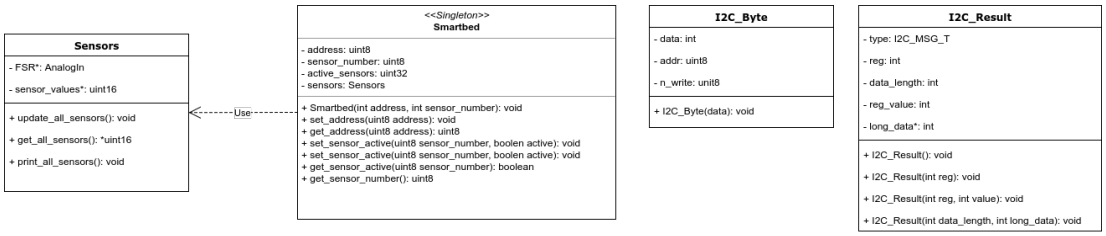


Figure 3.5: Helper classes used in firmware.

The main program starts with the initialization of peripherals - position sensing input pin is set to digital pull down input while and led and position sensing output pin are set as output pins. Then, I2C communication is initialized to operate on $100KHz$ frequency and takes default address 0x10. Since reading subsampled data from ADC takes considerable time, reding sensor value after request for its value was requested would result in a timeout. This is why a periodical sensor update is executed roughly every 100ms. Framework's inbuilt Ticker class takes care that function `update_all_sensors()` is executed regularly. Unfortunately, mbed does not provide API functionality to add custom function as software interrupt when I2C package is received. This is why a main program loop is executing `i2c_slave_worker()` function. This function contains main

communication logic. It listens to I2C bus and depending on the received data initializes `I2C_RESULT` object which is then returned to main function. This object is then transferred to `process_i2c_call()` which depending on the message type calls read, write and broadcast (general call) routines. Program logic is visualized in Figure 3.6.

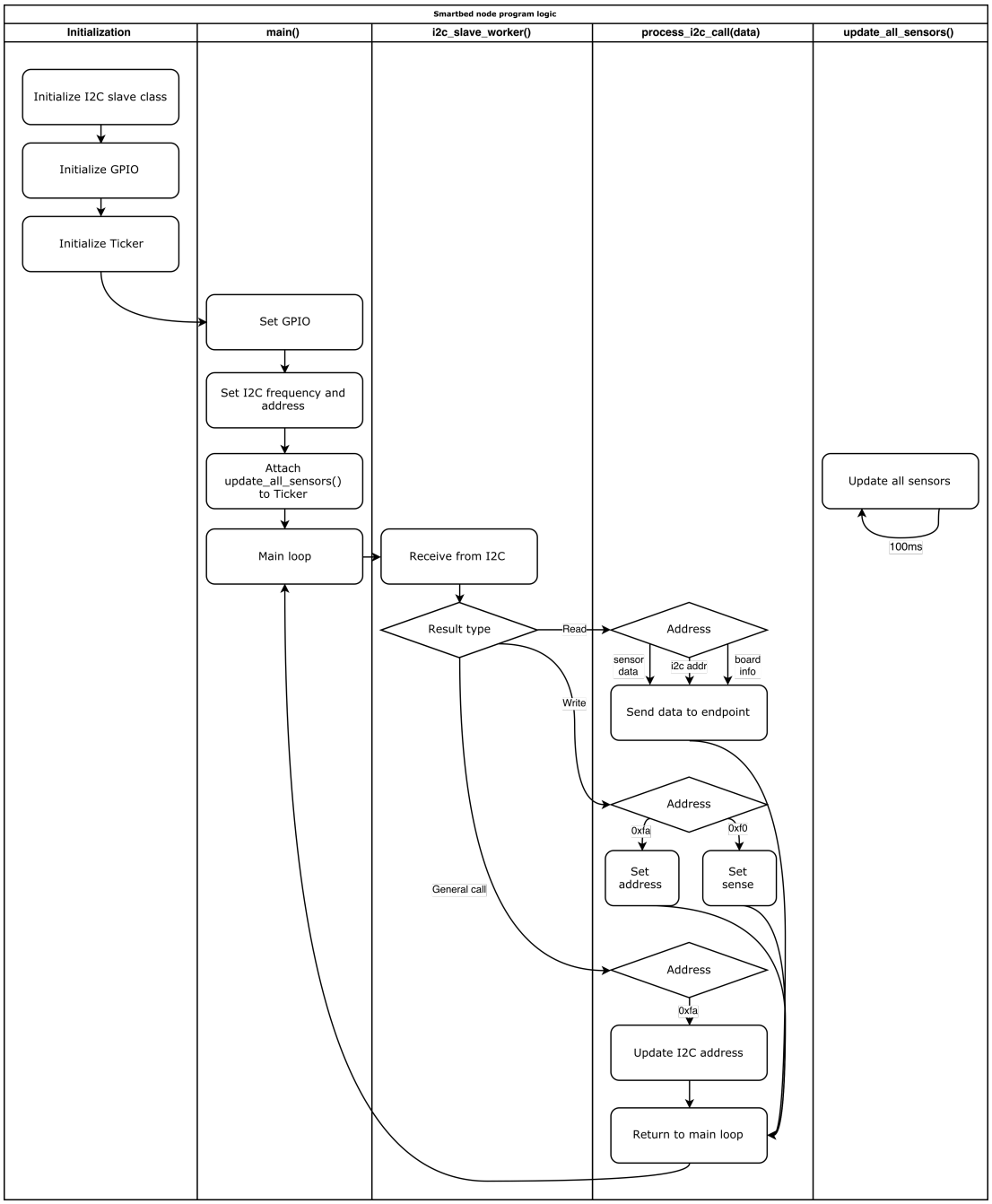


Figure 3.6: Sensor node program flow.

When node is accessed in write mode, two register addresses are writable. Writing a byte to 0xfa will change the address of the node while writing to 0xf0 will set the position sensing output pin high or low depending on the data byte received in the write request. Read requests for registers 0x00 to 0x0f address will return 16-bit value of specified sensor that is stored in `Sensors` class. Reading from 0xa0 will return values

of all sensors. It's also possible to read from register 0xfa which returns I2C address, from 0xfb which returns number of active sensors, 0xfc returns the board revision which is defined in the header file and 0xfd returns firmware version also found in header file. A read request to all other addresses returns 0xab. When a general call is issued, if register 0xfa is addressed, node will check if position sensing pin is high and change its address to the one in I2C package. This enables dynamic configuration of the sensor network and eliminates the need for a different firmware for each board.

	Register	Description
Read	0x00 .. 0x0f	Data for specified sensor (2B)
	0xa0	All sensor data (32B)
	0xfa	I2C address (1B)
	0xfb	Number of active sensors (1B)
	0xfc	Board revision (1B)
	0xfd	Firmware version (1B)
	Default	Returns 0xab (1B)
Write	0xfa	Changes node address (1B)
	0xf0	Sets position sense sensor value (1B)
General call	0x55	All nodes reset address to 0x55 (0B)
	0xfa	If position sense is high, this sensor node takes the supplied address (1B)

Table 3.1: I2C communication interface.

The first node is connected to the endpoint using the system bus cable. The cable attaches to the 6-pin IDC connector labeled as `SBBUS_1`. To add the second node to the system, a cable needs to connect connector `SBBUS_2` on the first node and connector `SBBUS_1` on the second node. Difference between this connectors is that pins for `SENSE_IN` and `SENSE_OUT` are swapped on the PCB. This eliminates the need for custom cables or additional resistors. Figure 3.7 demonstrates proper connection of a system containing of an endpoint and 2 nodes. Power can be supplied to the endpoint using microUSB cable.

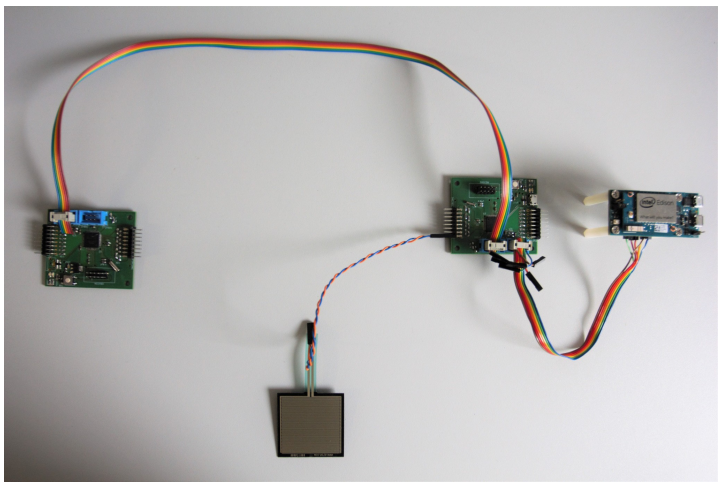


Figure 3.7: Proper connection of endpoint, two nodes and a FSR sensor.

4 Endpoint node

Endpoint is a device that acts as an interface between sensor node network and external clients and services. This chapter focuses on the embedded system implementation, on how communication with nodes has been implemented and then on data acquisition process.

4.1 System setup

For an endpoint Intel Edison COM was used. It runs x86 instruction set[21] which should allow for compatibility with most of the embedded operating systems. Unfortunately, it has a custom architecture with integrated Intel Quark microcontroller as a coprocessor which most of the standard operating systems don't support. Intel in its BSP provides a Yocto Linux image and compilation tools[34]. For an easy setup, already compiled system images are available. Although Yocto is relatively easy to use, it requires an image to be compiled and uploaded for every change. This takes a lot of time in development process. Prebuilt system image provides most of the features needed by this project but it doesn't have a Hypertext Transfer Protocol (HTTP) server such as Nginx or Apache2. Instead, most of Edison projects are running HTTP server inbuilt into NodeJS. This usually works just fine with simple applications providing data to 1 client. But if data is to be accessed by multiple clients, client requests are queued and served by a single core. Also, if the server crashes, there is no automatic restart or repair features. That's why port of Debian Linux distribution called Ubilinux was used in this project[35]. For an HTTP server Nginx was used[36]. It was configured to monitor and automatically restart the application if it crashes. Another feature that nginx provides is automatic serving and caching of static files. Server was instructed to serve all javascript, css files and images directly from file system. Another feature that Nginx allows is logging so access and server log hold the data on the server usage.

One of currently most popular programming languages is Python. It is a scripting language which supports object oriented programming and has some functional programming features. There are multiple frameworks designed to allow Python users to serve web pages. Most popular ones are Django, Pyramid and Flask. Django is a big framework which is very structured and pragmatical which is a very good feature for big projects but doesn't work well for smaller ones. Pyramid on the other hand is very customizable but requires a lot of configuration to work as intended. Flask is actually a microframework which urges users to follow good design practices but does not require a lot of configuration or boiler-plate tasks to be done. Also, it is very easy to learn and use so it was a framework of choice for this project[37]. Web pages and services programmed in Flask are usually served by uwsgi server application. uwsgi was configured in such a way that it serves data to Nginx through Linux socket which then serves web pages and content to the end user.

Since endpoint is also used for data accumulation, a data is stored in database. Database of choice for this project is PostgreSQL[38]. To allow easier communication with database a SQLAlchemy Object-Relation Mapping (ORM) toolkit was used. It allowed direct mapping of Python classes to ER modelled database tables. It removed the need to write SQL code in the application. Instead of that, a session is created which is then queried for data which allows easy adding, update and delete of database rows. When a database model is developed for the first time, a table can be created. But when a new column needs to be added to the table, it's quite impractical to delete table and create a new one manually. This is why Flask package flask-migrate is providing possibility of automatic database initialization and when changes to the model are done it can be run to generate migration scripts and upgrade the database. To start nginx, postgres and uwsgi systemd scripts are used. Postgres and nginx scripts were provided with the respected packages but a custom script had to be written for uwsgi. The script creates the directory for socket file and starts the application. A detailed description how to set the project up is available on project GitHub page.

4.2 Communication with sensor network

From endpoints perspective, communication with sensor network relies on Intel's mraa library[39]. Library allows for an easy access to hardware features such as GPIO, I2C, Universal Synchronous/Asynchronous Receiver/Transmitter (USART) and Serial Peripheral interface (SPI) on GNU/Linux systems. Library detects the platform on which it is deployed in runtime and by doing so makes the same code run on all supported platforms. Some of the supported boards are Intel Galileo, Intel Edison, Intel Joule, Raspberry Pi, Beaglebone Black and even Terasic DE10-Nano Field Programmable Gate Array (FPGA). Library also provides APIs for multiple programming languages - C++, Python, Java and NodeJS. Intel Edison has 2 available I2C interfaces - I2C1 and I2C6. To communicate with the nodes an IO expansion shield was used which exposed I2C6 pins as only I2C interface[40]. First, a board was connected to node and a script was run to test I2C functionality. This didn't work and node was unable to read any packages sent from the endpoint. A logic analyzer was connected to the bus and what was seen was that data signals were raised before the appropriate clock signals. This caused the packets to be unreadable. To fix this, another expansion board which provided access to I2C port 1 was used. After plugging in the Edison module and running the script everything was working. The difference between I2C1 and I2C6 can be seen in Figure 4.1.

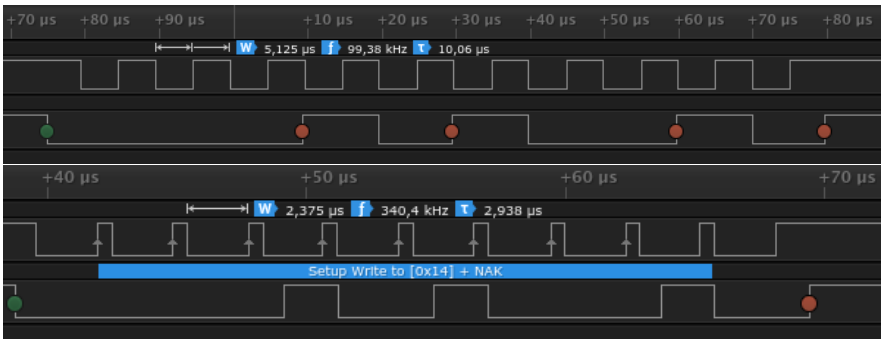


Figure 4.1: I2C port 6 exposed on custom expansion board not working.

After I2C communication was working, additional wire for position sensing was connected to the pin 31. This pin is labeled as GPIO-44 on Intel Edison. To distribute I2C addresses to the sensor node network first a general call is issued by addressing 0x00. To reset node addresses, 0x55 is sent. All of the listening nodes switch their address to a default value of 0x55 and set their position sensing pin output to low. After that, endpoint rises its position sensing pin to high and writes 0x11 into register 0xfa of address 0x55. Node which is directly connected to the endpoint will have its position sensing pin input set to high and will set its address to 0x11. To confirm that address was successfully set, endpoint reads from register 0xfa of address 0x11. If it receives 0x11 as an answer, address was successfully set and endpoint will issue a command that will pull the position sensing pin down. Then, it will write 0x01 to register 0xf0 of the node 0x11 which means that addressed node has to set its position sensing output pin high. After that endpoint does the same procedure for the next node increasing the number of the issued address. An edge condition is recognized by node not responding to read of its address. For example, if node 7 doesn't exist, reading from register 0xfa of address 0x17 won't return anything. Endpoint has a timeout and will detect that and set the position sensing output of the last addressed node back to low. The whole process can be seen in Figure 4.2.

4.3 Data acquisition routine

For the system to function as described, both data acquisition process and web server have to run in parallel and without concurrency issues. Sensor network data is shared through the database and this doesn't cause concurrency issues because data acquisition routine is constantly adding new data while web server is only reading or modifying already stored data. But some things like acquisition start and stop or I2C address redistribution commanding are a part of control logic which doesn't have to be stored in the database. In fact, implementation of communication between two separate processes using database would be quite inefficient. This is why a system pipe is used for message communication. Starting and stopping the data acquisition routine, changing between simulation and sensor network data collection and I2C address redistribution are implemented using pipe communication. Pipe has two ends - data acquisition has one end and web server has the other. When a process is reading or writing data to a pipe, it uses a pipe end provided to it. This makes communication between processes easy and robust. Data acquisition process also uses an activity flag which is set when process is collecting data. This flag is used by web server to check if there is a need to read new data from the database and to decide if it is possible to start or stop the routine. This flag is implemented as a shared memory object. Both pipe and shared memory are created when uwsgi starts the application.

First thing data acquisition does when it's instantiated is read the system configuration. Configuration file defines 3 different scenarios - production environment on Intel Edison, testing environment on Intel Edison and testing environment on development PC. Configurations are implemented as classes extending the same parent class which is holding shared configuration settings such as web application secret and update frequency. Extending configurations hold information on database Uniform Resource Identifier (URI), debug properties and deployment environment. Then, according to the configuration URI, data acquisition routine acquires connection to the database and depending on the configuration sets the data read function to point to read function from nodes or

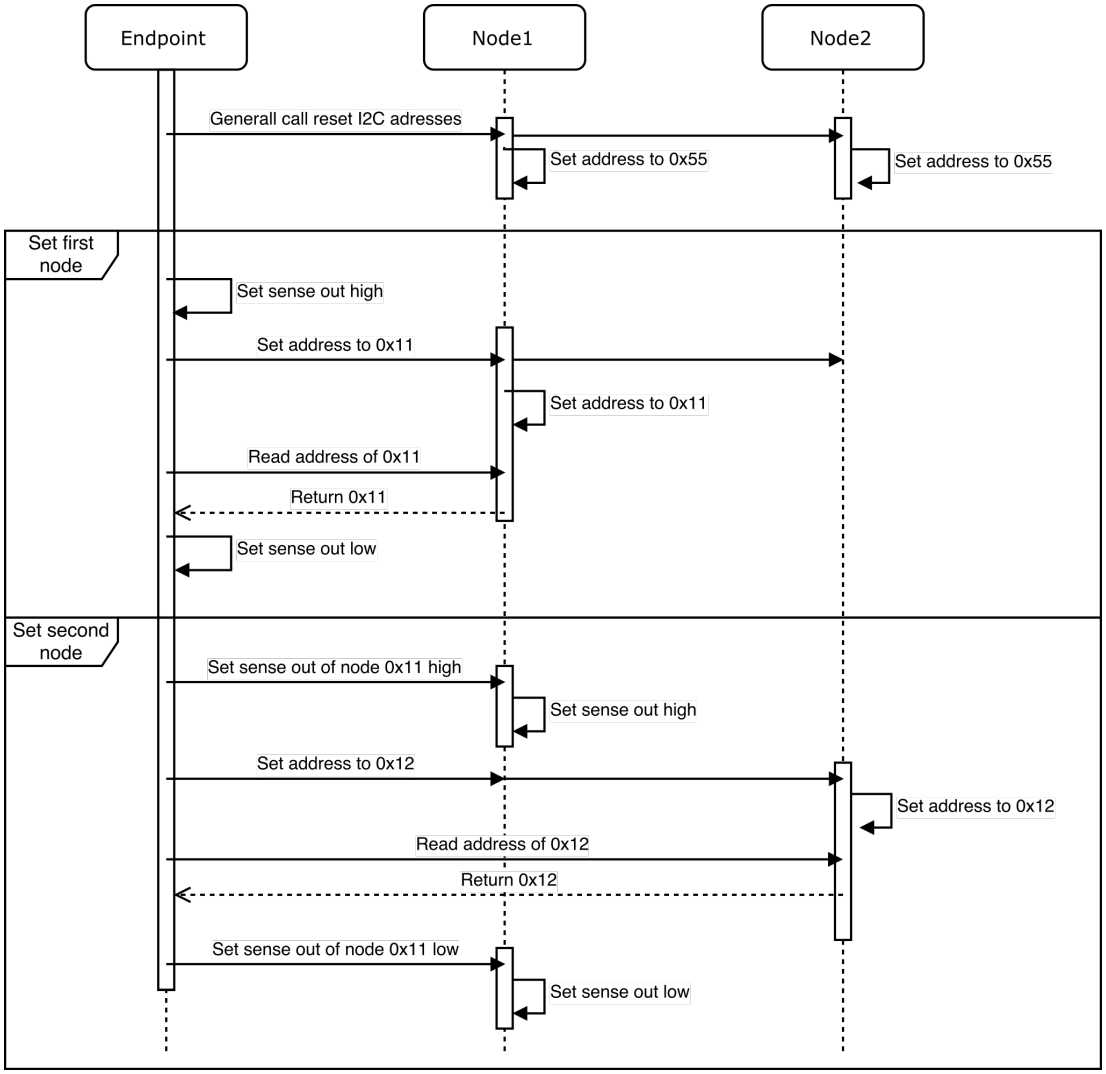


Figure 4.2: Sequence diagram showing process of address renewal for 2 nodes.

from a simulator. It isn't possible to always get the data from the sensor nodes. Such case would be development of frontend or backend software on an external PC. This is why simulator was implemented to randomly generate sensor node data.

The database has 3 tables for storing the sensor network data. First there is a `Node` class. Each node has its unique identifier (id) but is also described by its position (eg. 1 for node covering pillow and shoulder region, 2 for node covering chest). Second model is `Sensor` and it holds information on each of the FSR sensors. Each sensor has a single node which it references but node can also access aggregated sensors as an array with the help of SQLAlchemy's backreference feature. To know which sensor is which, sensor table also holds information on sensors position (eg. sensor 1 is the one closest to the left bed frame side). When a reading from sensor is taken, it is saved as an instance of `SensorValue` model. This table holds the timestamp and value as well as a reference to which sensor this sensor reading belongs to. For each of the models, two additional methods are implemented. `__repr__()` method is used for easier debug reading and `serialize()` is used for object representation as JSON. An Entity Relationship (ER) model can be seen in Figure 4.3.

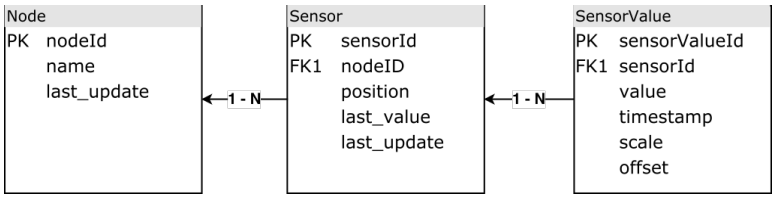


Figure 4.3: Database ER model for storing sensor values and system information.

So how does the system acquire sensor data from the nodes? Process first checks for data in the pipe. If start, stop or address redistribution command is received it will accordingly change its activity state or execute the redistribution procedure. Then, depending on the specified platform - edison or PC it will execute a sensor readout method. Simulation method queries all available sensors and generates new sensor value. After that sensor value object, sensor and node are added to session and session is synchronized so that data is written to the database. For actual sensor readout on Edison, a method involves reading from every sensor on every available node. Problem with mraa implementation on Edison is that it doesn't set the acknowledge bit after it receives data from the node. This locks the node and it can't process another read request if a new byte is sent by master. This is why before the read request, a write request is issued. After that, a request for specified sensor is issued. Since sensor reads maximum voltage when there is no pressure, value is inverted by subtracting current sensor value from maximal value. As read error encountered on the I2C bus is the same as maximum voltage, reading procedure is retried multiple times until value is over threshold. To prevent program locking in this state, number of retries is limited. After reading was taken it's saved to the database. After all sensors were updated, process sleeps for a period defined in configuration file. Program flow is shown as a graph in Figure 4.4.

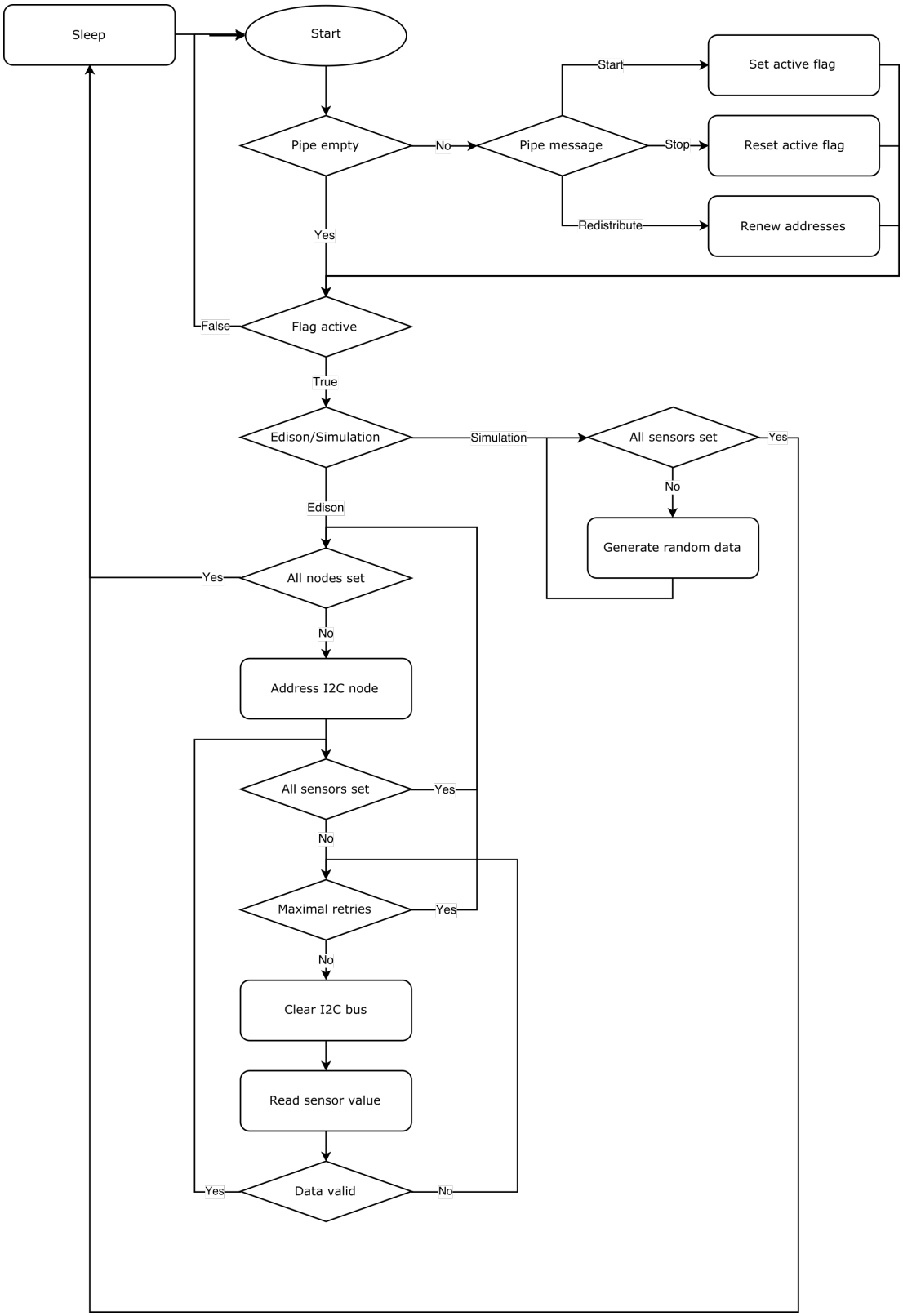


Figure 4.4: Flow diagram of data acquisition process.

5 Data distribution and visualization

This chapter focuses on user interface and API provided by the project. In first section, API is described which serves both user interface and machine-to-machine communication allowing integration of external services. The next section covers user interface, graphs and features available to the end user.

5.1 Application programming interface

Uniform Resource Locator (URL) for API routes is distinguished by `/api`. Data served by API is in JSON format and is structured to be both human and machine readable. Accessing `/api/system` provides the list of all of the nodes with their names, ids, timestamps of latest update and with list of attached sensor ids. This allows external services to navigate though the system and query specific nodes and sensors for values. `/api/node` accepts node id as a parameter and serves node id, name, list of attached sensor ids and last update timestamp. An example URL for accessing node with id 1 would be `/api/node?id=1`. When sensor is accessed using `/api/sensor?id=1`, last update, last value, parent node id and sensor position on that node are displayed. If external service wants to access value history for a specific sensor this can be done via route `/api/sensor_value` with supplied sensor id as a parameter. If no other parameter is received, server will provide array with 50 latest readings. Result is structured in such a way that it timestamp is a key for value at that given time. On that same route simple filtering can be introduced using start and end timestamps. By default, 50 latest readings are provided but more or less can be accessed using `readings` as a parameter. An example of data provided by accessing api route with start, end and number of readings parameters is seen in Listing 5.1 while an overview of all of the model routes can be found in Table 5.1.

Listing 5.1: API result for first 5 readings done in a set period of time

```
{
  "result": {
    "2017-08-19-19-28-47": 17181,
    "2017-08-19-19-28-50": 8522,
    "2017-08-19-19-28-53": 1275,
    "2017-08-19-19-28-57": 41146,
    "2017-08-19-19-29-00": 45302
  }
}
```

The next part of the API concerns the charts. Most of the charts are implemented using Chart.js javascript library while plotly is used for heath map. All of the charts are updated in real time but can also show history so API was designed to support these

Route	Parameters	Description
/api/node	id	Returns the timestamp of last node update, it's name and list of all available sensors
/api/sensor	id	Returns the position of the sensor, last value and its timestamp
/api/sensor_value	id, readings, start, stop	By default returns last 50 values read by the sensor specified by the supplied id. Time period of readings can be filtered by start and stop timestamps. Readings parameter filters the number of readings.

Table 5.1: System model description API routes.

features. The first chart user is greeted with is histogram of all sensor values. This graph helps live setup of the sensors. Users can easily indicate issues with connection or positioning of the sensors. The next graph is node graph - for each sensor attached to the selected node it shows the history how the values changed through time. It's also possible to view a single node history graph. The last graph is heath map of the bed which shows the topological preview of the sensor grid. With the increase of weight detected by the sensor, fields become red while with the decrease they become blue. API allows user interface to specify the time for which sensor data should be shown. Graph data can be accessed using path `/api/chart` and is also served in JSON format. Systematical overview of all chart routes is found in Table 5.2.

Route	Parameters	Description
/api/chart/heatmap		Returns 0xab (1B)
/api/chart/histogram		Returns 0xab (1B)
/api/chart/node		Returns 0xab (1B)
/api/system		Returns the system overview with node and sensor ids

Table 5.2: API routes used providing the chart data.

It's also possible to see the status of the data acquisition service using path `/api/service`. This route will return true if the service is running and false if service is inactive. It's also possible to start and stop service accessing start or stop API routes. These routes will execute the procedure for either starting or stopping the service and then return if the action was successful. To redistribute addresses to I2C nodes, a route `/api/distribute` is used. Accessing it will execute the redistribution routine. Since user might want to save different sleep sessions it's also possible to save the start or end of the active session using route `/api/session`. All of these routes can be used by external services but are also used by web user interface. To avoid the whole page reloading for every new chart update, Asynchronous Javascript and XML (AJAX) technique was used but with JSON replacing Extensible Markup Language (XML). A list of all API routes with their descriptions can be seen in Table 5.3. Routes are serving GET requests but it's easily possible to transfer them to POST requests in which user session info or external

token info would be saved. This would protect the data from unauthorized access but this is not needed for this stage of the project.

Route	Parameters	Description
/api/distribute		Redistributes the node addresses based on their physical position
/api/service		Returns if the data acquisition service is running
/api/service/start		Starts the data acquisition service, returns the operation outcome
/api/service/stop		Stops the data acquisition service, returns the operation outcome
/api/session		Returns if session is in progress and its id
/api/session/start		Starts a new session if another session is not active
/api/session/stop		Stops the session if there is an active session in progress

Table 5.3: API routes for controlling the system with described functions.

5.2 User interface

Python Flask framework follows Model View Controller (MVC) rules regarding the separation of logic. Model and a part of controller were already presented but this section will focus mainly on view and its integration. Flask uses Jinja2 template engine which allows specific web pages to easily implement and extend template design. It also allows injection of data and control logic into the view. This means that data objects can be passed directly from the controller while control logic like loops or if clauses may be used to construct the page that will in the end be rendered. For each of the pages a separate template is created in `app/templates/` folder. `base.html` is a template which all other pages extending. It implements Hypertext Markup Language (HTML) headers, user interface parts such as sidebar, header and footer. It links all of Cascading Style Sheets (CSS), javascript pages and allows insertion of page specific javascript files in separate block. File `/app/views.py` holds the request routing logic. Based on the received HTTP requests, program renders different templates by inserting required data objects. A bigger part of user interface design is based on CoreUI Bootstrap admin interface template. The template was modified and structured to work as a Jinja2 template. CoreUI is licensed under MIT license[41] and can be used and distributed free but with the preservation of license.

As it was already mentioned, website was designed to allow live updating and graphics rich design. For that, multiple javascript libraries were used. Some of them are jQuery, Bootstrap, chart.js and plotly. To take care of these dependencies, bower was used. On CSS side, Bootstrap was extended by a custom script.

So how do these template generated, javascript and css infused pages look and what do they do? First thing a user is greeted with is a dashboard. It shows the number of

currently present nodes and sensors and state of the data acquisition service. It also shows the histogram which helps the user see if system is working properly. In a case when there is no pressure on the bed, all of the sensors should be at the left of the chart where 0 is located. When person lies on the bed, the chart changes and values get redistributed throughout the chart. The dashboard can be seen in Figure 5.1.

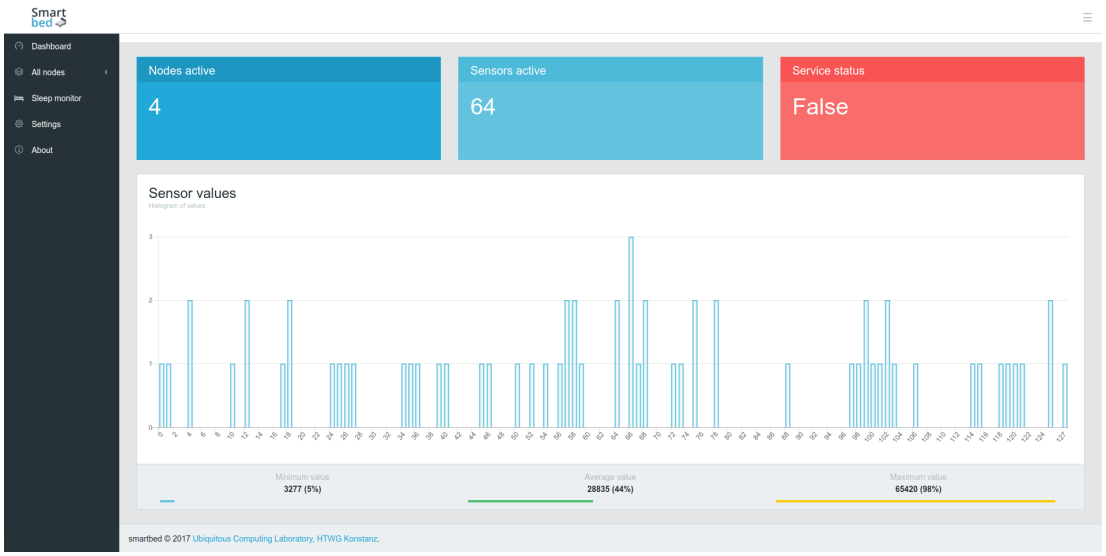


Figure 5.1: Flow diagram of data acquisition process.

To observe the nodes and adjust the values, node view displays line charts of all attached sensor values. Graphs are updated periodically and latest value is displayed in the appropriate card. If sensor needs to be adjusted, this can be done though modal window for scaling and offsetting. For example, if the weight of the pillow or some other object is affecting the zero-weight reading, it's possible to offset the future sensors readings. Described view is shown in Figure 5.2.

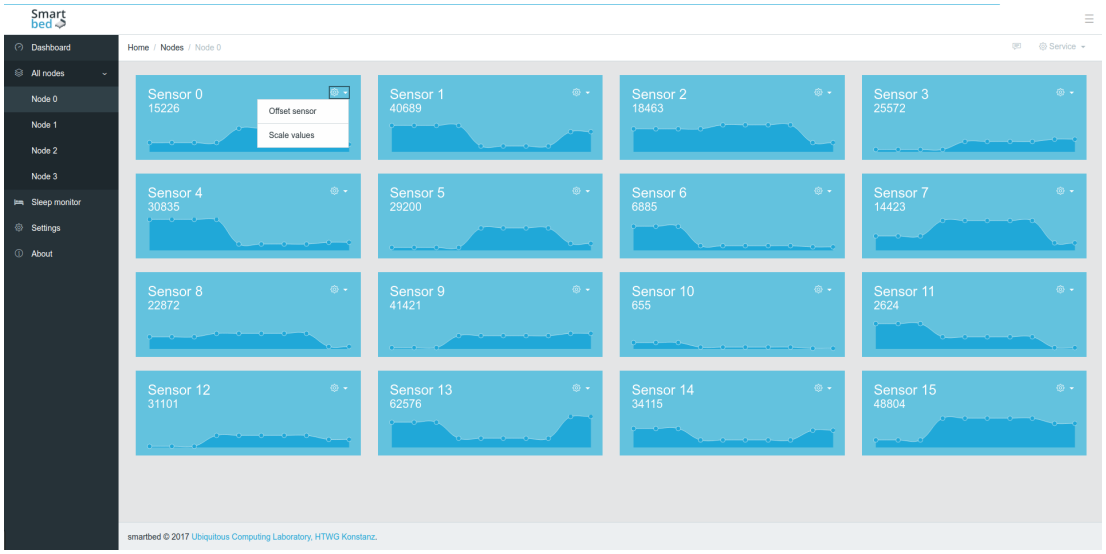


Figure 5.2: Flow diagram of data acquisition process.

To help future research which will focus on sleep position recognition, a heat map was implemented. It shows the topological view of the bed with all of the attached sensors.

Each sensor is represented by a colored rectangle which changes it's color depending on the pressure applied to it. There is also a possibility to view previous values so a pose of person sleeping in the bed can be observed through the night. This chart is presented in Figure 5.3.

Figure 5.3: Flow diagram of data acquisition process.

The whole interface was also adapted to work with the mobile devices. In case of the lower resolutions, page elements are redistributed and resized to fit the smaller screen. Navigation transfers to the 'hamburger' menu implemented sidebar while the rest of the functionality remains unchanged.

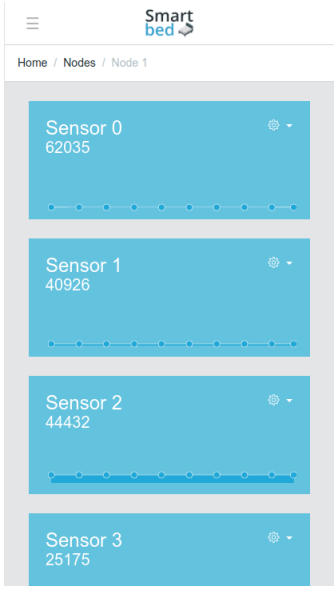


Figure 5.4: Flow diagram of data acquisition process.

6 Testing and results

To prove that system is working a test was conducted at the Ubiquitous Computing Laboratory at HTWG. Four rows of six FSR sensors were placed on the pressure-disks. Sensors were placed in shoulder and lower torso area where it's easiest to detect vital signs. Two nodes and Intel Edison were attached to the bottom of the bed frame and connected to the sensors. To power the whole system an external 2.500mAh battery was used. Assembled system can be seen in Figure 6.1. Previously used system required 2 nodes, 10 hand soldered boards and over 35 wires to function while the new one was set up in less than 15 minutes when all sensors were installed on the pressure-disk blades.

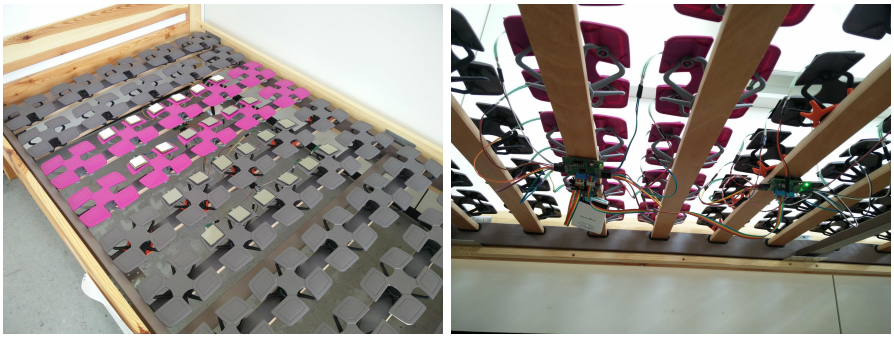


Figure 6.1: Sensors and nodes installed into the bed.

First test was conducted to see if all of the sensors work. Sensors were excited and results were tracked in real time using laptop connected to the same wireless network as an endpoint. It was observed that sensors 3 and 6 were not reacting to the applied pressure. After reinsertion of wires, sensor number 6 started working while it was found that the cause of sensor 3 malfunction was ADC pin unresponsiveness. This issue was fixed by connecting another pin to the sensor but this remains to be fixed on the PCB.

Figure 6.2: Testing of the responsiveness and sensitivity.

In the next phase, a mattress was placed onto the bed. In this position, a pressure detected by the sensors was lower than threshold set in the firmware so they were registering 0. A subject lied on the bed and rested in log position for 2 minutes after which he changed position to supine and rested for further 4 minutes. After that, data was exported to the CSV file and analysed. Charts found in Figure 6.3 and Figure 6.4 represent plotted sensor data on both nodes.

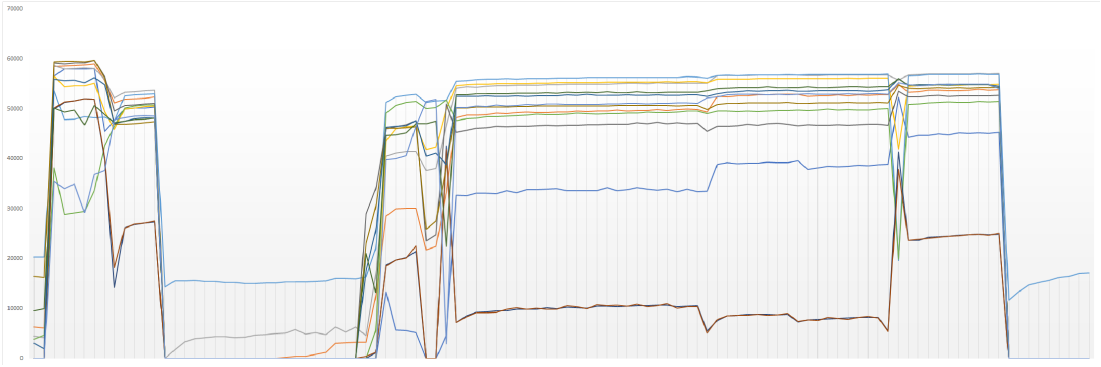


Figure 6.3: Pressure measured on sensors in shoulder area.

When subject sat on the bed, sensors were excited with the peak value reaching 57118 while saturation values is 65535. Average value during the test for all of the sensors was 44565 which is 68% of maximum value. While subject was still, values for each sensor were constantly fluctuating with an average deviation of 284 points. This means that 9-bit resolution for vital sign recognition was achieved.

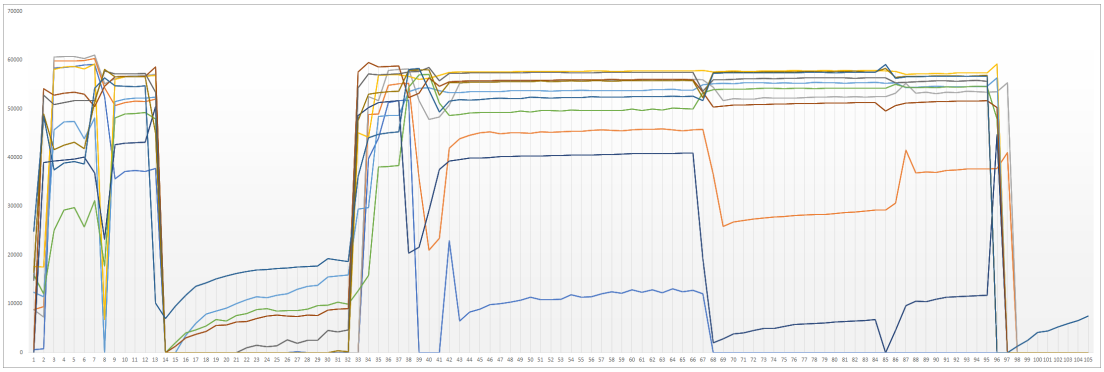


Figure 6.4: Pressure measured on sensors in upper torso area.

A data from the single sensor, located in the middle of chest area was further analysed. The goal was to see if it is possible to detect breathing. A reoccurring pattern with an interval of 20 seconds was observed. Regular respiratory rate is between 12 and 18 times per minute which should roll out this pattern being breathing. To be able to detect breathing a system sample frequency should be at least double the respiration rate. In case of respiration this means that single sensor sample period should be at least 1.6 seconds. Conducted measurement had sample rate of 4 seconds which means that sample rate should be greatly increased for the future tests.

7 Conclusion

Based on previous work and research it is concluded that it is possible to track subject position during sleep and to monitor vital signs using a grid of FSR pressure sensors underneath the bed. Based on these unobtrusive measurements it is possible to classify sleep stages and measure sleep quality. This information is not only to useful for medical and research purposes but there is also a growing market share of people interested in sleep tracking who would like to self-improve their sleep quality. System architecture and implementation proposed in this thesis provide means to develop a robust and scalable product for sleep tracking. Hardware architecture was improved on the old one by reducing the number of nodes and by introduction of custom PCBs that eliminate the need for additional hand-soldered boards. A communication inside of the system was also improved by introduction of system-wide bus. Endpoint node was reimagined to implement data storage and display features so that the system doesn't require additional servers and processing units. But, if multiple sensor grid systems are to be integrated into a larger one such as in case of a medical facility, where multiple patient vitals signs are tracked at the same time, an API was developed to allow for this possibility. On the other hand, the developed webpage based user interface allows single users to interface the system and review their sleep without the need of external devices. As all of the technology used in this project is based on open source it is easy to grow and improve the system further.

The current system acquires, distributes and displays data but does not provide an insight into the sleep quality. Next steps to improve the system would be integration of position classification and body movement during sleep. This data could be paired with an output of the algorithm for vital sign detection that also gathers data from the pressure sensors. Together, these algorithms could possibly provide a quite accurate estimation of sleep stage and sleep quality. Moreover, they could be used in clinical purposes to monitor vital signs and help medical personnel prevent bed sores in bed-bound patients. For home use, implementation of sleep classification and automatic analysis algorithms could help end users sleep better. It would also be possible to integrate the system as a part of IoT infrastructure for home automation.

Symbols, Units and Abbreviations

Symbol	Unit	Name	Description
R	$\Omega = V/A$	Ohm	Resistance
V	$V = J/C$	Volt	Voltage
I	$A = C/S$	Ampere	Current

AC Alternating Current

ADC Analog to Digital Conversion

AJAX Asynchronous Javascript and XML

API Application Programming Interface

BSP Board Support Package

COM Computer On a Module

CPU Central Processing Unit

CSS Cascading Style Sheets

DC Direct Current

EEG Electroencephalography

EMG Electromyography

EMMC Embedded Multimedia Card

EOG Electrooculography

ER Entity Relationship

ESD ElectroStatic Discharge

FPGA Field Programmable Gate Array

FSR Force sensing resistor

GPIO General Purpose Input Output

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTWG Hochschule Konstanz für Technik, Wirtschaft und Gestaltung

I2C Inter-Integrated Circuit

IC Integrated Circuits

id identifier

IDC Insulation-displacement connector

IoT Internet of Things

JSON JavaScript Object Notation

LAN Local Area Network

LDO Low-Droput

LED Light emitting diode

MCU Microcontroller Unit

MEMS Microelectromechanical systems

MVC Model View Controller

NREM Non-Rapid Eye Movement

NSF National Sleep Foundation

ORM Object-Relation Mapping

PC Personal computer

PCB Printed Circuit Board

POF Plastic optical fiber

PPG Photoplethysmography

PSG Polysomnography

PTF Polymer Thick Film

PWM Pulse Width Modulation

RAM Random Access Memory

REM Rapid Eye Movement

ROM Read Only Memory

SDK Software development kit

SMPS Switching Mode Power Supply

SPI Serial Peripheral interface

SWD Serial Wire Debug

TQFP Thin Quad Flat Package

UC-Lab Ubiquitous Computing Laboratory

URI Uniform Resource Identifier

URL Uniform Resource Locator

USART Universal Synchronous/Asynchronous Receiver/Transmitter

USB Universal Serial Bus

XML Extensible Markup Language

List of Figures

List of Tables

Enhancement of sensor mesh functionality with application on sleep tracking

Abstract

The focus of this project is enhancement of functionality, reliability and sensor accuracy of an intelligent bed that monitors human sleep. The scope of the project includes the implementation of an application layer protocol and network communication between the embedded system in the bed and remote server. Furthermore, the implementation possibilities of data preprocessing, filtering, and automatic sleep analysis are explored and tested. The system is tested and evaluated in the Ubiquitous Computing Laboratory at the Hochschule Konstanz University of Applied Sciences.

Keywords: sleep tracking, embedded systems, sensor meshes, sleep analysis

Primjena senzorskih mreža na praćenje ljudskog sna

Sažetak

Tema projekta je unaprjeđenje funkcionalnosti, pouzdanosti i preciznosti rada inteligentnog kreveta koji prati ljudski san. U sklopu projekta implementira se aplikacijski sloj te ostvaruje mrežna komunikacija između ugradbenog sustava u krevetu i udaljenog računalnog servera. Nadalje, rad istražuje i testira implementaciju preprocesiranja podataka, izrade podatkovnih filtera i automatske obrade i analize podataka o snu. Sustav se testira i evaluira u Laboratoriju za sveprisutno računarstvo pri Hochschule Konstanz University of Applied Sciences.

Ključne riječi: praćenje sna, ugradbeni sustavi, mreže senzora, analiza sna