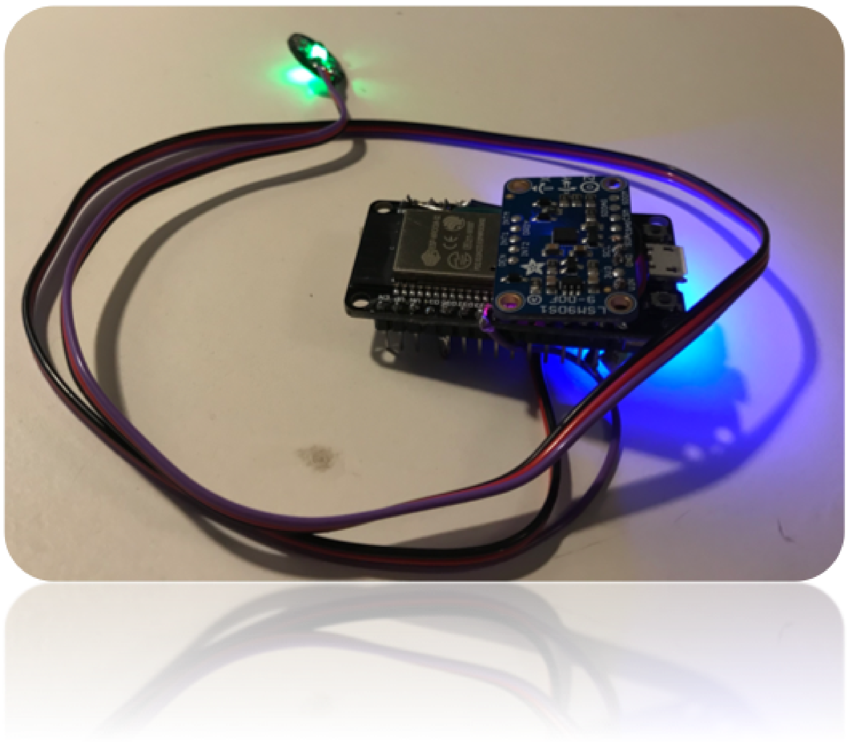
SMEAGOL SS18 Laboratory Report: Slept In - A sleep-tracker

Benedikt
-

Lennart
-



July 27, 2018

I. Introduction

This report outlines the design, implementation and evaluation of our product named 'Slept In' build for the semester project SMEAGOL in the Summer Semester 2018. The sleep-tracker is a small and low-power wearable device which is capable of determining your sleep phases and sleep quality based on sensor measurements performed while you are sleeping. It is so small and ergonomic that the user do not even notice it while sleeping.

II. Goals and Design Constraints

The initial goal of the project was to create a small wearable device which allows home consumers to track and analyze their sleep. Scientifically the best way to measure sleep it to measure brain waves. Unfortunately, this is not suited for everyday usage and consumers. Therefore we focused on the heart rate and physical movement as indicators for sleep states and sleep quality.

Based on our design goals we narrowed down three main requirements for our product:

1) The device needs to be small to allow it to be worn by the user like a smart watch. It should not hamper or interrupt the users sleep.
2) The device needs to be power efficient to allow it to operate for at least 10 hours on a small battery.
3) The device needs to be connected directly or indirectly through another device to the Internet to allow for long term data storage and more flexible data analytics.

III. Hardware Selection

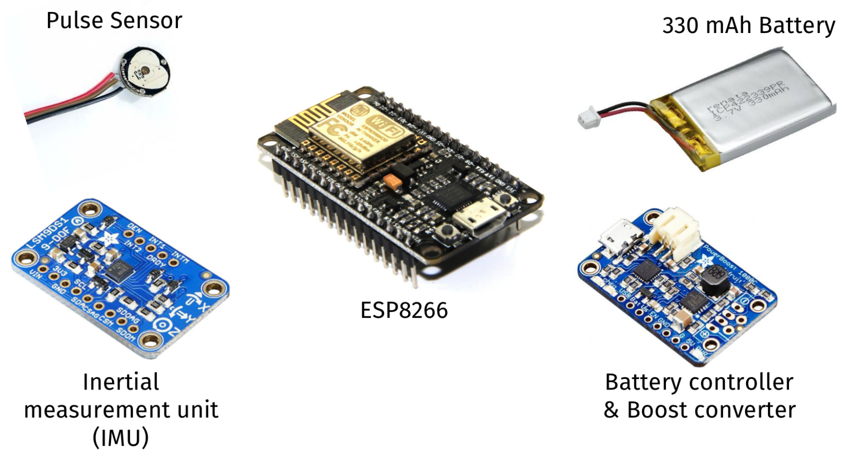

Pulse Sensor
330 mAh Battery
ESP8266
Inertial measurement unit (IMU)
Battery controller & Boost converter

Fig. 1. Hardware Components

The requirements stated in Section II necessitate the use of a microcontroller as a small low powered device. We decided to use the ESP8266 as our main development platform. The ESP8266 is a widely used microcontroller with Internet connectivity through an internal WiFi module. It supports the Arduino framework, which is a very popular open source software framework for microcontrollers and offers libraries for a wide variety of peripherals.

We researched low power hardware solutions for measuring the pulse, movement and the sound of the subject. As a movement sensor the LSM9DS1 inertial measurement unit (IMU) was used, since it was already available in the laboratory and met our requirements for low power consumption. It also offers an Arduino library with a simple interface which accelerates the software development. For the pulse sensor we decided to use an open source optical pulse sensor which initially started as a Kickstarter project in 2011 and offers native support for our Arduino platform. Unfortunately, the ESP8266 only offers one Analog-Digital Converter

(ADC) input which is already reserved for the pulse sensor. For this reason, we where not able to use a microphone at the same time.

Additionally a power source was required to allow the device to be carried wireless. We decided to use a small 1.2Wh Li-Po battery and a corresponding charging circuit.

IV. Software

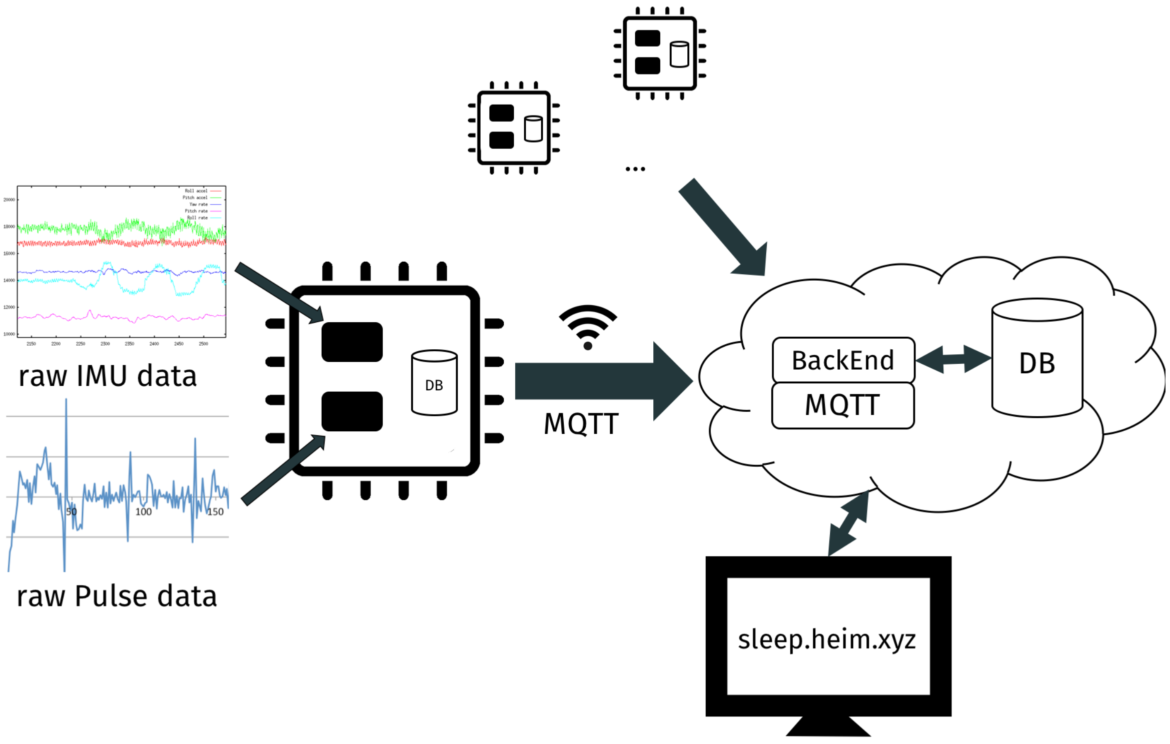


Fig. 2. Architecture Overview

The software was carefully designed to allow the sleep-tracker to be run as efficiently as possible. As seen in Figure IV, the software design consist of two main parts, the micro-controller itself and a cloud server as a central node for all sleep-trackers to connect to. The basic concept is to do the necessary pre-processing and data reduction on the sleep-tracker, while the more complex data analysis is offloaded to the cloud in order to reduce the power consumption. The two components will be described in more detail in the following Sections.

A. Microcontroller

The software of the microcontroller needs to be designed in a way, that it can sleep as much as possible to reduce the power consumption. In order to be able to run for at least 10 hours, the device needs to consume less than 24 mA in average. The datasheet of the ESP8266 reports that an active WiFi connection draws an average current of about 80 mA, while consuming about 20 mA when the WiFi module is turned off. The ESP8266 also supports a deep sleep mode at which the device only draws a few μ A. However, for our use case, it is sufficient to turn the WiFi module off and connect every few hours to synchronize the gathered data. This would allow us to run the required minimum of 10 hours. To allow this a small database was implemented on the microcontroller which allows us to accumulate the data locally and send it bundled to the server.

In order to read the data from the sensors we used an open source Arduino library for the LSM9DS1, but decided to not use the library offered with the pulse sensor. We focused on the following sensor values: the Beats per minute (BPM) of the pulse sensor, the linear acceleration and the angular velocity of the IMU.

The pulse sensor library did not provide reliable measurement results and was prone to errors due to movement or ambient light changes. Therefore, we implemented our own open source pulse sensor library and focused on improving the existing pulse sensor library. Thanks to our additions, the library is less prone to errors and also able to determine the accuracy of the measurement. Due to this enhancement we are able to omit inaccurate measurements and reduce the amount of filtering required in the database.

The linear acceleration is calculated as the 2-Norm of the acceleration. The acceleration caused by earths gravitational force needs to be subtracted in order to get the absolute acceleration. This was done by calculating the orientation of the device using Madgwick's AHRS algorithm and subtracting 1 g in the z direction. The angular velocity was calculated by accumulating the angular change of the gyroscope and taking the 2-Norm. These three values where then stored every 10 seconds in the local database and are then synchronized every hour with the cloud server, where further processing is performed.

B. Server

As our wearable sleep-tracker is limited in storage and analyzing capabilities, multiple process were run on a remote server.

MQTT was used as the messaging protocol. The MQTT broker runs on the server and waits for clients, in our case sleep-trackers, to connect and publish their data. MQTT is a wide spread protocol which works on top of TCP/IP and recognizes a wide spread usage in the field of Internet of Things (IoT).

The further processing of the incoming data is conducted by the software *Sleepweb* which we developed in Phoenix. Phoenix is a web development framework in the programming language Elixir. Sleepweb subscribes to the MQTT broker and listens for incoming messages. Incoming messages are parsed and divided into individual datasets, as the data arrives in bundles. Each individual datasets then gets stored in a database. PostgreSQL is used as a database. In our database we store the following values for each timestamp:

- `sensor`: name of the sleep-tracker device
- `time`: timestamp of the datasets
- `bpm`: bpm value
- `lin_acc`: linear acceleration
- `angular_change`: angular change
- `event`: placeholder for specific events
- `misc`: placeholder for miscellaneous data
- `inserted_at`: timestamp when the data got inserted into the DB
- `updated_at`: timestamp when the data got updated

Additionally, a website was developed which plots the data from the past but also receives incoming data live. It is accessible via: https://sleep.heim.xyz. Therefore when new data is received it gets stored in the DB but also broadcasted via. websockets to current website sessions. At the moment the website only plots the raw data which is received from the sleep-tracker device. In the future the target would be to update the web application to enable users to access their analyzed sleep patterns. The source code of the Phoenix back-end and front-end can be found at: https://gitlab.com/SMEAGOL18-sleep-tracker/phoenix-webapp

All of the mentioned components are executed in Docker containers. A docker-compose file was created which allows the deployment within seconds on any systems. This enabled us to update the website within seconds with losing incoming data. Additionally it allows us to serve a website which features security components as SSL/TLS. The docker-compose file can be found at: https://gitlab.com/SMEAGOL18-sleep-tracker/docker-compose

V. Experiment

Once the first hardware and software prototypes were finished, we started a real life experiment and gathered some data. As we designed our prototypes with our design constraints in mind, it was simple to equip the sleep-tracker. We used an sliced sock, which then was worn around the arm, seen in Figure 3

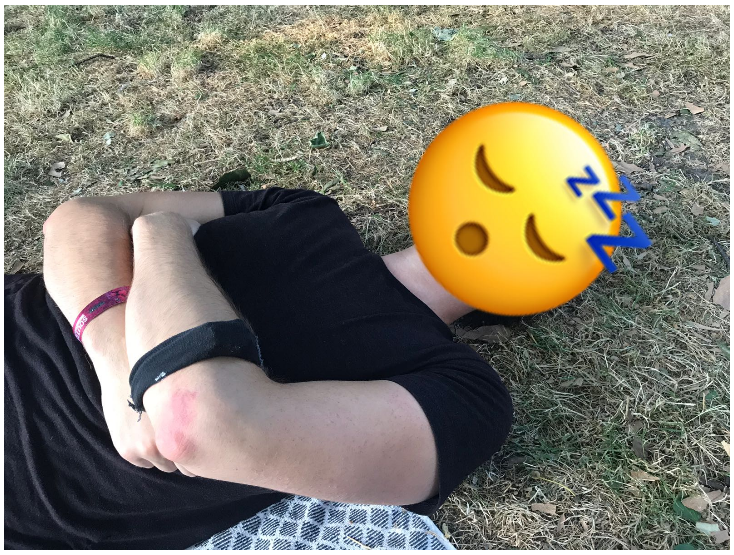


Fig. 3. Outdoor experiment with the wristband and sleep-tracker

We conducted multiple experiments during our sleep. For comparison purposes we also used a sleep tracking application on our smartphone. Our application used the IMU to track our movement on the mattress to make assumption about the sleep state. Unfortunately, the smartphone app did no give detailed insights, only a plot of the current sleep state over time. It does not provide us with additional information like the measurement error or the uncertainty. Because of this a direct comparison does not give us any insight in how good our system performs.

VI. Analysis

The first step of the analysis was to do some post processing on the data. One night usually consisted of up to 3,000 data points. As seen in Figure 4 the signal is noisy and includes some artifacts.

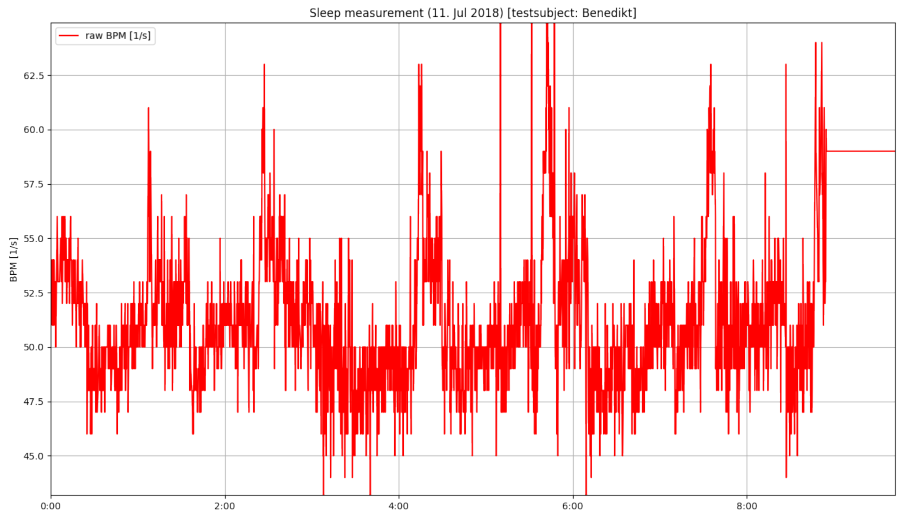


Fig. 4. Unprocessed BPM Signal from the Database

Therefore some filters were applied to achieve a smother signal and get rid of outliers. The processed signal is seen in Figure 5.

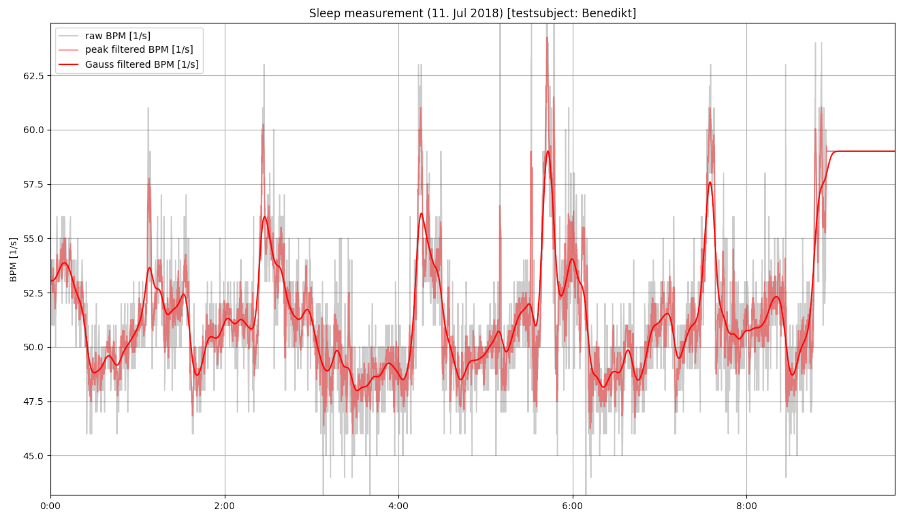


Fig. 5. Processed BPM Signal

As BPM is the best indicator for a sleep phase, we started analyzing the data. A light sleep and deep sleep threshold was calculated. This calculation was done the following: First all local minimums and maximums of the BPM curve where located. The light sleep threshold was then calculated by averaging over all maxima of the data curve. All values above the average maximum peaks indicate that the test person is sleeping. The deep sleep threshold was then calculated by averaging over all minima which are below the sleep curve.

Based on those thresholds a probability of being awake or asleep was calculated. This is seen in the second sub-figure of Figure 6.

This pattern includes multiple awake stages which is usual for a healthy sleep. As multiple research experiments pointed out, the typical human sleep consist of 5 sleep cycles. Furthermore, our test subject evaluated this sleep as a good sleep, and so does our analysis.

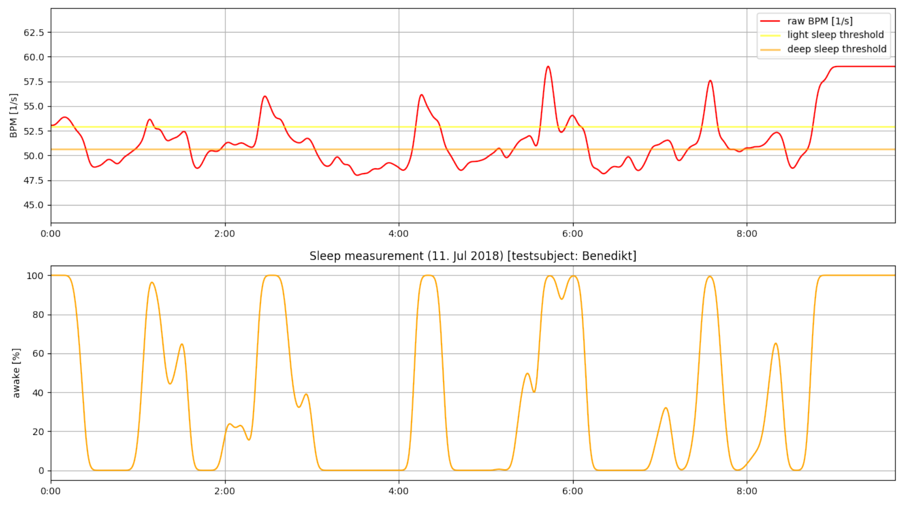


Fig. 6. Processed BPM Signal with Thresholds & Sleep Analysis

VII. Conclusions

We have build a first prototyping device which allows to monitor our sleep. As discussed, we focused on low power consumption and a small form factor. We were aware of the development issues under those constraints, but achieve our goal of having the device sustain a night. Our first analysis worked surprisingly well and the sleep pattern was analyzed. The analyzed sleep pattern matches our expectations and fits the human sleep behavior.

In the future we would have liked to include further sensors into our analysis. In our current analysis only the BPM sensors was used, as it is the most reliable and the best indicator for sleep. Using an additional sensor, such as the IMU or a microphone, might increase our accuracy.

All of our code can be found at https://gitlab.com/SMEAGOL18-sleep-tracker. We might continue working on it in the future. Otherwise feel free to fork and enhance it.

VIII. Abbreviations

ADC	Analog-Digital Converter
BPM	Beats per minute
IMU	inertial measurement unit
IoT	Internet of Things