# Contents

## I Introduction and background

### 1 Introduction

1.1 Motivation ................................................. 3
1.2 Problems caused by sleep disorders ......................... 4
1.3 Non intrusive sensors .................................. 4
1.4 TRIO .................................................. 5
1.5 Problem statement .................................... 5

### 2 Background

2.1 Sleep Apnea Syndrome ................................. 7
   2.1.1 Obstructive Sleep Apnea .......................... 8
   2.1.2 Central Sleep Apnea ............................... 10
   2.1.3 Mixed/Complex Sleep Apnea ..................... 11
2.2 Diagnosis ............................................. 11
   2.2.1 AHI ........................................... 12
   2.2.2 PSG ........................................... 13
   2.2.3 Treatment of OSA ............................... 17
5.2.1 Interface .............................................. 55
5.2.2 Software testing ................................. 55
5.2.3 Testing ................................................ 57
5.3 Creating test data ................................. 58
5.4 Usage of puka ...................................... 60
  5.4.1 Signals used ..................................... 61
  5.4.2 Experiments ..................................... 61
5.5 Adjustments ........................................ 61
  5.5.1 Decimate .......................................... 62
  5.5.2 Pause window size .............................. 62
  5.5.3 Mark pause peak and trough location ...... 63
5.6 Additional software ............................... 64
  5.6.1 Signal processing ............................... 65
5.7 Real time detection ............................... 65
  5.7.1 Data serving .................................. 66
  5.7.2 Data consuming, puka reduced .............. 66
5.8 Modifying puka for online usage ............... 66
  5.8.1 Window sizes .................................. 67
  5.8.2 Looking backwards ............................ 67
5.9 Result/Analysis online ............................ 67
III Discussion and Future Work 68

6 Discussion 69

7 Future Work 70

8 Conclusion 72

8.1 Software . . . . . . . . . . . . . . . . . . . . . . . . . . . ii
8.2 Troubleshooting . . . . . . . . . . . . . . . . . . . . . . . ii
  8.2.1 Script execution freeze . . . . . . . . . . . . . . ii
  8.2.2 Peak detection sample rate assumption . . . . . . iii
  8.2.3 Misc bugs . . . . . . . . . . . . . . . . . . . . . . . . iii
TODO

Cato Danielsen

June 24, 2016
Part I

Introduction and background
Chapter 1

Introduction

A good nights sleep is important in order to stay physically and mentally healthy. Research has shown that the lack of proper sleep can be linked to many health issues.

According to the National Institute of Health (USA) sleep apnea, if left untreated, can lead to different health risks. Among these are increased risk of high blood pressure, heart attack, stroke, obesity, diabetes, heart failure, increased chance of irregular heartbeats and increased chance of having work-related or driving accidents [19]. Other literature has for a long time pointed out the risk of mental health issues related to sleep apnea [21], such as depression.

According to the literature the estimated prevalence of sleep apnea is 2%-4% of the middle aged adult population in USA[59]. One thing we find as a broad consensus in the literature, is that a lot of sleep apnea patients go undiagnosed, as much as 80% to 90%, depending on the criteria for diagnosis.

TODO a bit more on surveys and prevalence

The clinical term sleep apnea was introduced in 1973 by after the first international symposium on "Hypersomnia with Periodic Breathing" in 1972 [15]. The terms sleep apnea syndrome and obstructive sleep apnea was coined in 1976. Over the last 40 years we have seen an increase in interest and concern over the effects
of sleep disorders and it has been discovered to be a more common medical problem than previously assumed.

1.1 Motivation

The most common way and the gold standard of detecting sleep disorders is with a polysomnography (PSG) that requires a patient to sleep with monitoring equipment in a sleep lab. A PSG can also be referred to as a sleep study and it monitors a variety of parameters in order to diagnose sleep disorders. These parameters are described further in Section 2.1.

An important question is: if we already have an accurate and precise way of detecting and diagnosing sleep disorders, why are so many occurrences of sleep related disorders undiagnosed? According to the literature there are several key factors as to why these cases go undiagnosed and untreated. Some of the symptoms associated with sleep disorders, such as excessive daytime sleepiness, daytime irritability, difficulty of concentration and waking with headaches, can be ambiguous and it is difficult for a doctor to identify a sleep disorder based only on symptoms observable in a consultation. The symptoms can be vague and ambiguous and the threshold for recommending a costly, overnight procedure without having clear indications that it is a sleep disorder causing the symptoms can be difficult to justify for the clinical staff. As it is not always clear whether symptoms are caused by sleep disorders other more easily diagnosed alternatives are explored first. The overnight PSG requires technology, personnel, dedication and experience.

This is a recognized problem and attempts have been made to create pre-screening tools in order to detect sleep disorders. We will look into some examples of these solutions in Chapter 3. This can be done by either using mobile devices with their built in sensors such as smart phones, or using custom made home usage device such as home PSGs or other sensors that monitor parameters that can indicate sleep disorders, such as respiration rate, blood oxygen levels, heart rate, body movement or other relevant metrics.
Also, a patient with a sleep disorder will not yield the same result for each PSG recording, as a patients sleep pattern can change from night to night. It would be even more costly to have a patient spend multiple nights in a sleep laboratory for several tests in order to determine the exact extent of the sleep disorder. This also brings us into the problem of sleep quality during the sleep study. A PSG requires multiple electrodes connected to a patient which can cause the patient to not be able to fall asleep or give a false or imprecise impression on the sleeping pattern of the patient.

Even if a PSG is accurate (the current gold standard for sleep related measurements), the threshold for doctors to order a PSG is relatively high due to the cost and effort required to do a complete PSG. This makes the need for non intrusive pre-screening tools in order to clinically diagnose the cases of sleep disorders. If a patient can with minimal effort take a test without the use of intrusive sensors and in their own home, closer to a normal nights sleep it might be easier to justify a more thorough examination.

### 1.2 Problems caused by sleep disorders

TODO: Why are sleep disorders important to detect...

### 1.3 Non intrusive sensors

In order to create a system that can detect sleep disorders without the need for overnight stay at a sleep laboratory or the presence of clinical personnel, we will look into the use of non intrusive sensors.

By sensor we are talking about a device or multiple devices coupled together, able to detect bio markers, such as respiration stops, in order to indicate sleep disorders. Sensor technology will be described in [Section 2.3](#).

The quality of being non intrusive is that the patient is not
hampered of put in physical discomfort by the sensor, as they would have with a sensor that require electrodes or a mask or other probes that might cause discomfort. Whether a sensor is intrusive or not is not well defined, but varies based on different parameters. If we have a sensor that requires the user to sleep with a elastic band around their chest, this might not be seen as an intrusive sensor for a healthy person as they have no problem attaching and wearig the sensor. But for a person with limited mobility, the act of attaching the sensor might prove to be a considerable inconvenience.

1.4 TRIO

TODO: Short description

1.5 Problem statement

TODO: ReWrite!

Using existing respiration analysis tool we can test the logical sensors. For this thesis we will look into how to adapt the software and algorithms found in Physionets application. The data derived from the application will then be changed into events that can be used in apnea detection and respiration monitoring in the TRIO project.

my assignment:

• create atomic events (peak respiration, respiration effort stop etc) that can be passed on to TRIO.

• Detect them in real time.

summary:
• convert existing/simulated data (ECG, airflow, chest/abdomen movement)

• modernize puka

• analyze with puka

• convert puke to realtime application

• feed into logical sensors (adjust)

• compare results, look into potential improvements
Chapter 2

Background

The system described in our problem statement will make use of sensors in order to detect sleep disorders. The sensors captures physical phenomena and converts it into signals, that we in turn process into events for TRIO. This chapter explain some of the underlying concepts for such a system.

2.1 Sleep Apnea Syndrome

Sleep Apnea Syndrome (SAS) is sleep disorder characterized by the disruption of airflow during sleep. SAS is often divided into one of three sub diagnosis, Obstructive Sleep Apnea (OSA), Central Sleep Apnea (CSA), and Mixed Sleep Apnea (MSA), also know as Complex Sleep Apnea.

All diagnosis have in common either total stop or a reduction of respiration with a subsequent decrease in blood oxygen levels. The cause of these respiration reductions is what defines the type of SAS. An apnea event is the name for a complete stop of respiration for at least 10 seconds, while a hypopnea event is defined as an at least 10 seconds reduction in ventilation of at least 50% of normal airflow during sleep\[30\]. When the blood oxygen level is reduced the body is aroused from sleep in order to resume normal breathing. The arousal from normal sleep reduces the sleep
quality.

2.1.1 Obstructive Sleep Apnea

Pathogenesis

OSA is also known as Obstructive Sleep Apnea/Hypopnea Syndrome (OSAHS), due to the occurrence of both apneic and hypopneic events. In OSA the upper airway (UA) passage is either completely or partially blocked. There are multiple structural or anatomic factors that have been discovered to cause UA blockage, and these blockages occur in the pharynx. The pharynx is the area where the nasal and oral cavity meet and it has both the digestive, speech and respiratory functions in human anatomy. The pharynx area consists of muscles and soft tissue and it is necessary to be able to collapse and close the UA for digestive and speech purposes while awake. The negative pressure created by the inspiration process can cause the soft tissue region to collapse, causing blockage.

![Normal Breathing Pathway](image1.png) ![Blocked Breathing Pathway](image2.png)

Figure 2.1: Obstructive Sleep Apnea

There are also genetic factors as some have smaller airways that also can contribute to the lack of airflow. Nasal obstruction can lead to mouth breathing, which predisposes to abnormal airway dynamics that favors not only pharyngeal collapse but also what is called backward displacement of the tongue. The soft tissue of the tongue can cause UA blockage.
In addition to the soft tissue risk factors, the bone structure of the jaw region can be positioned in such a way that the tongue is predisposed to be pulled back into the pharynx during sleep during sleep stages with decreased muscle tone.

The factors that can increase the risk of UA blockage makes OSA difficult to predict and diagnose.

**Epidemiology**

Patients with anatomical vulnerability are considered to be more susceptible to developing OSA\(^{[43, 9]}\). These vulnerabilities can be enlarged tonsils, recessed mandible, small upper airway, impaired retrolingual airway among others. Each of these case is not a clear indication of OSA, but can be a contributing factor. Other factors that increase vulnerability for OSA include age, obesity, menopause, sleep hygiene, and certain health behaviors such as cigarette smoking and alcohol use\(^{[42]}\).

Hypertension, also known as high blood pressure, is an often reported co-morbidity of OSA\(^{[10]}\). During the lowered blood oxygen levels experienced during an apnea or hypopnea event results in increased activity in the autonomic nervous system in order to increase the oxygen level. The literature suggests that as much as 50% of OSA patients suffers hypertension even during wakefulness\(^{[45, 37]}\).

OSA has also been linked as a risk factor for cardiovascular diseases, stroke, abnormal glucose metabolism, insulin resistance, and diabetes mellitus \(^{[42, 49]}\). Cerebrovascular diseases and OSA have been pointed out to have a bi-directional relationship\(^{[15]}\), and as a result of the hypertension and reduced cerebral blood flow the risk for cerebrovascular diseases such as stroke is increased.

As Fusetti points out, "the common association of OSAS with hypertension and obesity in general population makes it difficult to separate their respective independent role in the long-term cardiovascular and metabolic consequences associated with OSAS"\(^{[12]}\).
2.1.2 Central Sleep Apnea

Pathogenesis

While obstructive apnea is caused by blockage of the airways, a central apnea is the complete stop of respiratory effort as a consequence of imbalance within the brain's control of the respiratory effort, described as a loss of ventilatory control\[56\]. While instability in the upper airway leads to obstructive sleep apnea, the imbalance of ventilatory control can lead to both obstructive and central sleep apnea.

Epidemiology

CSA can manifest in two broad categories according to the wakefulness CO\(^2\) levels. Hypercapnic and nonhypercapnic. Hypercapnic is defined as elevated CO\(^2\) levels in the blood. Patients often exhibit some degree of daytime hypercapnea and this condition is often worsen during sleep. Two patterns are often used to classify hypercapnic: impaired central drive ("won't breathe") and impaired respiratory motor control ("can't breathe")\[8\].

Impaired central drive can be caused by physiological factors that diminish ventilatory function, but has also been linked to genetic factors without anatomic pathology. Opioid-based medication have for a long time been pointed out to have a respiratory depressant effect\[55\].

Impaired respiratory motor control can experience CSA due to abnormalities in the signaling of the respiratory system. It can be caused by a wide range of neuromuscular disorders that causes some stage of the signaling process to not be able work properly.

Cheyne–Stokes breathing is a nonhypercapnic breathing pattern that is most commonly observed in patients with congestive heart failure and left ventricular systolic dysfunction\[8\]. During Cheyne–Stokes the patient increases the breathing rate gradually in a crescendo/decrescendo pattern broken up by apneic events. Arousal typically occurs mid-cycle at the peak of ventilatory effort.
rather than at the cessation of apnea.

### 2.1.3 Mixed/Complex Sleep Apnea

#### Pathogenesis

As defined by Guilleminault, Tilkian and Dement in 1976, "mixed apnea is defined by cessation of airflow and an absence of respiratory effort early in the episode, followed by resumption of unsuccessful respiratory effort in the latter part of the episode"[14]. This diagnosis is a combination of central and obstructive sleep apnea. In some cases when the respiration effort stops as a result of CSA, the pharynx region is collapsed due to the lack of pressure, so when the body is aroused into resuming breathing efforts it is still completely or partially blocked.

#### Epidemiology

These episodes of central apneas followd by airway collapse and obstructive apneas and hypopneas are considered to be multifactorial. Obesity and/or snoring has been linked as a contributing factor for developing mixed apnea in CSA patients as the increased risk of high passive airways which leads to higher susceptibility for airway collapse[7]. The same article also points out mixed apnea in in patients that are administered chronic doses of opioid medications.

As this diagnosis is a combination of Central and Obstructive sleep apnea, many of the same health effects can be found.

### 2.2 Diagnosis

Hypopneic and apneic events are common symptoms of sleep apnea, and in order to diagnose the different conditions. Respiratory
Disturbance Index (RDI) is often used in sleep studies, but it includes other disturbances other than hypopneic and apneic events. This calls for a more specialized scale to diagnose sleep apnea.

2.2.1 AHI

Apnea–Hypopnea Index (AHI) is a commonly used index for the severity of sleep disturbances during the course of the total sleep time of a patient. The AHI usually refers to the number of events per hour of sleep. The number of events can be used to measure a severity score, where:

<table>
<thead>
<tr>
<th>0-4</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>5-14</td>
<td>Mild</td>
</tr>
<tr>
<td>15-29</td>
<td>Moderate</td>
</tr>
<tr>
<td>30 or more</td>
<td>Severe</td>
</tr>
</tbody>
</table>

Table 2.1: AHI severity scale

In order to calculate the AHI we use the number of apneic and hypopneic events per hour

\[ AHI = \frac{(Hypopneas + apneas) \times 60}{TotalSleepTime(minutes)} \]

The AHI combined with daytime symptoms, such as EDS, dry mouth or headaches when waking up, is the basis of diagnosis for sleep apnea.

The first indication that often warrants the sleep study is the daytime symptoms, but according to the literature there are patients without any associated clinical symptoms (asymptomatic apnea). The literature suggests that the effect of these asymptomatic patients still suffer altered heart rate during daytime without symptoms or co-morbidities[3].

As the name implies, AHI counts both apnea and hypopnea events and is very useful for OSA detection, since a patient suffering from OSA can exhibit both apnea and hypopnea events.
There are several different non intrusive ways of indicating a diagnosis of sleep disorders. Questioners such as the Berlin Questioner, STOP BANG and Epworth Sleepiness Scale (ESS) are used in order to screen for and discover the usual symptoms of sleep disorders. One example of a study using the Berlin Questioner (BQ) and Epworth Sleepiness Scale (ESS) is "A Norwegian population-based study on risk and prevalence of obstructive sleep apnea"[17] where it was used to make an estimate on the prevalence of OSA in the Norwegian population. These questioners help researchers to estimate the prevalence of OSA, but for a clinical diagnosis a physical examination such as a sleep study is needed.

2.2.2 PSG

In order to detect sleep disorders in patients, we need to monitor certain physiological parameters of the patient in order to classify the type of As mentioned in Section 1.1 the gold standard for sleep disorder diagnosis is the polysomography (PSG) or sleep study.

The function of PSG is monitoring of a patient during sleep using an array of medical equipment that is simultaneously recorded. The types of parameters depend on the type of PSG used. As there are at least number of sleep disorders types of sleep disorders diagnosed by sleep studies, variations on what types of signals recorded is classified by different types of PSG. According to AAST (American Association of Sleep Technologists) the standard PSG has the following parameters[36]:

TODO: threshold for hypopneic events
With electrodes:

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEG</td>
<td>Electroencephalogram monitors the electrical activity in the brain.</td>
</tr>
<tr>
<td>EOG</td>
<td>Electrooculogram measures eye movement.</td>
</tr>
<tr>
<td>EMG</td>
<td>Chin Electromyogram monitors level of muscle tone around the chin area.</td>
</tr>
<tr>
<td>ECG</td>
<td>Electrocardiogram monitors the heart rhythm</td>
</tr>
<tr>
<td>Respiration</td>
<td>recorded from the movement of electrodes</td>
</tr>
</tbody>
</table>

Other sensors:

<table>
<thead>
<tr>
<th>Sensor</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Audio</td>
<td>Upper Airway Sound Recording</td>
</tr>
<tr>
<td>Thermistor or Inductive Respiratory Plethysmograph (RIP)</td>
<td>Respiratory effort and flow</td>
</tr>
<tr>
<td>Limb EMG</td>
<td>Limb Movement and Body Position</td>
</tr>
</tbody>
</table>

The EEG documents wakefulness, arousals and sleep stages during the sleep study, which is important in order to know whether symptoms occur while the patient is sleeping and at which sleep stage it occurs. Sleep stages are often classified into five separate stages: 1, 2, 3, 4 and REM (rapid eye movement), or into REM and nonREM stages.

- In stage 1, muscle activity slows down, the eyes move slowly and you drift in and out of sleep.
- In stage 2 the brain waves becomes slower and the eye movement halts.
- In stage 3 the brain waves becomes very slow with occasional smaller, faster waves.
- In stage 4 the brain almost exclusively produces the same slow brain waves as in stage 3.

Stage 3 and 4 are referred to as delta sleep, which is the namesake of the extremely slow brain waves (delta waves) found in these stages. During delta sleep there is no muscle activity or eye movement. During REM sleep breathing becomes more rapid and irregular, eyes move rapidly and limb muscles are temporarily paralyzed. The brainwaves during REM sleep increase to an
activity level which is comparable to a non sleeping person. In order to detect REM sleep, other parameters such as EOG and EMG combined with EEG are usually used. Novel solutions have been proposed in order to be able to monitor all sleep stages with the use of only EEG.\cite{18}

Figure 2.2: Sleep stages\cite{48}

<table>
<thead>
<tr>
<th>Stages</th>
<th>Waking</th>
<th>REM Sleep</th>
<th>NREM Sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stage 0</td>
<td>Stage R</td>
<td>Light Sleep</td>
<td>Deep Sleep</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stage 1</td>
<td>Stage 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Stage 3</td>
<td>Stage 4</td>
</tr>
</tbody>
</table>

Eyes open, responsive to external stimuli, can hold intelligible conversation
Brain waves similar to waking. Most vivid dreams happen in this stage. Body does not move.
Transition between waking and sleep. If awakened, person will claim was never asleep.
Main body of light sleep. Memory consolidation
Transition between waking and sleep. If awakened, person will claim was never asleep.
Transition between waking and sleep. If awakened, person will claim was never asleep.
Transition between waking and sleep. If awakened, person will claim was never asleep.

16 to 18 hours per day | 90 to 120 min/night | 4 to 7 hours per night

The EOG is useful for identifying and studying the REM sleep stages. It uses electrodes positioned near the corner of each eye to measure the existing resting electrical potential between the cornea and Bruch’s membrane in order to determine the position of the eyes.

For sleep studies EMG is used in the mentalis, submentalis muscle, and/or mesater region.\cite{47}. The EMG records the muscle tone and is used as a criterion for staging REM sleep. EMG can also be used on other muscle groups to determine sleep disorders, such as monitoring leg muscles in order to detect restless leg syndrome.

Each time a heart beats it is triggered by an electrical impulse. The ECG (also called EKG) records these impulses as they travel through the heart. The electrical activity is recorded using
electrodes placed on the patient's body. Today the standard ECG consists of 12 leads in order to monitor all three dimensions of the heart \[50\]. Typically there are six limb leads placed on arms and legs and six precordial leads placed across the chest. Each lead has a specific angle from which it observes the heart in order to . . . reread book for better explanation. The limb leads monitor what is called the frontal plane, while the precordial leads monitor the horizontal plane. Each node records the average current flow at any given moment. Each heartbeat is described as an RR interval, also known as a cardiac cycle. Based on which electrode records activity the RR interval can be further segmented into smaller and identifiable intervals of the cardiac cycle and used in diagnosis and evaluation of the heart and breathing of a patient.

There are multiple ways to record the respiration rate during a PSG. Nasal and oral airflow are often recorded either with nasal thermistors or thermocouple, which uses changes in temperature to measure the airflow with prongs or probes placed in or near the mouth or nose. Another way of recording is to measure the physical movement the body during respiration using respiratory inductance plethysmography (RIP), which uses elastic bands around the torso and abdomen to record the movement of the body as a patient inhales and exhales. Based on the inflation and deflation of the chest and abdomen area, the respiration rate can be derived. Both of these methods are used as ground truth in assessing the respiratory rate in sleep studies\[4, 25\].

When none of these respiratory signals are recorded, other techniques can be deployed. One such technique is to use the ECG signals to derive the respiration rate. ECG, or electrocardiography, measures the electrical signals generated by the heart. There are different ways of obtaining the respiration rate from an ECG signal and also from the ECG electrodes themselves. One method calculates the respiration rate based on beat to beat variation RR intervals \(\text{Figure 2.3}\). This technique is based on respiratory sinus arrhythmia (RSA) which is a natural variation in the heart rate.

Another technique is ECG Derived Respiration (EDR). When a patient breathes the ECG electrodes on the chest surface move relative to the heart due to the lungs filling and emptying. The transthoracic impedance varies as a result of the expansion and
contraction of the lungs and from the mean cardiac electric axis show variations that correlate with respiration[33]. Another well documented method for deriving respiration from ECG is respiratory sinus arrhythmia (RSA), a method uses instantaneous heart rate variability to derive a respiratory signal.

Oxygen saturation is a useful parameter for detecting OSA, as the SaO² (blood oxygen saturation) drops after the onset of an apneic/hypopneic event. According to Division of Sleep Medicine at Harvard Medical School[44], the SaO2 is usually around 96% - 97% at sea level. A dip to 90% is generally considered mild, while dips to between 80% to 89% are classified as moderate and saturation below 80% are severe.

TODO: staff requirements, cost

2.2.3 Treatment of OSA

In order to effectively treat OSA, physicians have to consider the severity of the disease, co-morbidities and the patients preferences. A non surgical option is lifestyle changes, such as weight loss, avoidance of alcohol and nicotine, position therapy and treatment of co-morbid conditions. Continuous Positive Airway Pressure (CPAP) or therapy is described as a first-line therapy for moderate to severe OSA[15].

CPAP consist of a air pump, tube, and a mask, which provides pressurized air into the patients throat via the mask. The pressurized air helps avoid negative pressure from the inspiration
collapsing the airway.

APAP devices (Autotitrating PAP) detect snoring, airway resistance or impedance in order to only administer positive airway pressure. It also uses diagnostic algorithms in order to adjust the amount of pressure, but are far more complex than a standard CPAP and require calibration by a sleep technician. They do though have the advantage of adapting the pressure to sleep stage and sleep position, reducing the risk of discomfort due to too high pressure during sleep stages with more relaxed muscle tone.

Surgical treatments for OSA is centered around reducing the risk of collapse and removing potential obstructions. Surgical techniques can be to remove some of the soft tissue in the pharynx region, reposition the soft tissue by skeletal mobilization, or bypassing the pharynx region[46]. There is no standard procedure found to eliminate OSA.

Another approach that can be utilized is pharmacological treatment, but the literature suggest that such treatment has not been successful. A review by Hedner, Grote and Zou from 2008 concludes: "Currently, no widely accepted pharmacological treatment alternatives are available for OSA"[16]

2.3 Sensors

The name sensor has according to Webster’s New World College Dictionary its roots in classical Latin "sentire", which means to sense. "A sensor is a device which responds to stimuli, or an input quality, by generating processable outputs"[20]. This is how Kalantar-zadeh defines sensors. He also points out that the outputs of a sensor are always functionally linked to input stimuli of the sensor.

The term sensors refers often to two aspects, i.e. the sensor that quantitatively measures an input quality and the component that converts it to a readable signal for the device or person receiving the recordings. The part of a sensor that is responsible of taking the input signal of the sensory apparatus and converting it
is referred to as the transducer. A transducer converts one type of energy to another and is sometimes used interchangeably with sensors.

An example of a simple sensor is litmus paper, which usually is used for determining whether a solution is basic or acidic. The litmus paper is exposed to the solution and reacts to the stimuli by changing color, allowing an observer to read the results.

The output from sensors is a representation of the measured property and this can be described in different ways depending on the property measured. Over time the output can be used to create a sequence of data points called a time series.

TODO: REWRITE An example with a transducer might be a temperature sensor that records using a thermal sensor which is converted into an appropriate signal used to transmit the value to a separate device for recording or processing. By appropriate we mean a type of signal that is readable and understandable for the observer. END REWRITE

There are different ways a sensor can be constructed in order to record some quality of the real world. Contact and non contact are two broad categories can be used to describe sensors. As we touched in chapter 1 this can also be described as invasive and non invasive sensors.

Signal processing is an umbrella term for operations applied to the signal. J. Moura defines processing as “operations of representing, filtering, coding, transmitting, estimating, detecting, inferring, discovering, recognizing, synthesizing, recording, or reproducing signals”[34].

### 2.3.1 Sensor characteristics

Ideally a sensor should be able to measure a desired quality (input) of the physical world without any other input being registered. This is referred to as sensitivity towards the desired input and an insensitivity towards other potential inputs. It is important that a sensor does not affect the input or the environment it is deploy in.
The accuracy of a sensors recording is the correctness of the output compared with the actual value of the quality it measures. Deviation from the actual value of the quality can be due to rounding error, inaccurate sensor, calibration error, too low resolution etc. The example Kalantar-zadeh uses is a temperature sensor measuring a real temperature of 20.0°C. If the sensor measures 20.1°C it is more accurate than if it had measured 21.0°C. This is not to be confused with precision, which is the capacity to get the same result from repeated measurements of the same quality under the same conditions. The difference between precision and accuracy is illustrated in Figure 2.4.

![Figure 2.4: Precision and accuracy](image)

- sensitivity
- reliability
- accuracy/precision
- stability
- resolution
- life span

McGrath and Scanaill describe "v1.0 sensors" as simple measurement of quantity, such as a mechanical thermometer. For the second generation of sensors we add computational power and communication which allows the sensor to process the data it records and transmit it to other devices. An example of this can be a acidity
sensor, which is connected to an actuator which controls a valve in order to restore the Ph level to a preset value based on the sensors readings. At this stage the cost of production is still so high that it is not commonplace and highly specialized.

"Sensors v3.0" is described as when private consumers adopt the use of sensors. At this point sensors that previously were too expensive for consumers can be found in smart devices and in affordable home-use devices. In addition to the computational power introduced in "v2.0", the connectivity to the Internet opens up for new avenues for communication and pervasive sharing of data in real time. The data recorded by smart devices can be used for location tracking, health applications, consumer habits, and other areas.

"v4.0" is the stage we are currently stepping into. The capabilities of sensor systems have been increased due to increased computing power, smaller sizes, increased connectivity and more affordable prices.

### 2.3.2 Sensor networks

As defined by Phoha, LaPorta and Griffin sensors and sensor networks can be described with the following characteristics: "they monitor changes in the operational environment and collaborate to actuate distributed tasks in dynamic and uncertain environments"[39]. Each sensor has a task, a measurement of the physical world to perform and converts it into a signal. There are two primary approaches to how to process the data recorded: either distributed or centralized.

A human body can be compared to a centralized sensor network. We have different sensing devices such as eyes, touch, smell and hearing among others. The signals from these sensors are processed and coordinated by the central nervous system and the combined information provided from the different sensors gives us information about the world and gives us the ability to detect events around us based on the combined data recorded from the surrounding environment.
A distributed sensor network uses the sensor-nodes themselves to do processing. As the name implies, the sensors do not relay all the information gathered to one centralised storage/processing unit. Each sensor works autonomously but collaboration can be achieved by letting each node share and request information from the network as a whole.

### 2.3.3 Logical Sensors

In order to make sense of data recorded by multiple sensors or a sensor network, they can be grouped together into what is called a logical sensor. By multiplexing signals from multiple sources, be it sensors or external sources, a logical sensor can learn new information and detect complex events based on multiple inputs.

An trivial example can be a system utilizing a temperature sensor and a smoke detector. The logical sensor created from these two physical sensors can use both signals to detect a fire by combining, and decrease the chance of a false positive from a kitchen appliance or other device that generates heat.

### 2.3.4 DSMS

Data Stream Management Systems (DSMS) are used in order to process the information gathered continuously by sensors or sensor networks. A database management system (DBMS) is concerned with persistent storage of data, and is often used in conjunction with DSMS. Instead of sporadic writes and frequent reads as found in DBMS, DSMS have to filter out relevant events as data arrives. Access to the data is done sequentially as it arrives, thus the system has to

TODO: illustrer DSMS Stream –> DMSM –> query –> match –> output

As a data stream can be potentially infinite, the DSMS cannot do aggregation or analysis of data when it has gathered a "complete" set. Many DSMS uses a windowing technique to look at por-
tions of the data as it arrives. These windows can be time or tick-based. Tick-based windows waits for $N$ number of entries to arrive, while time-based windows aggregate on certain intervals. Such aggregations can be averages, sum, count for time-based windows etc.

A DSMS can not make use of a traditional query language, but instead uses what can be described as a Continuous Query Language (CQL). It can also be referred to as StreamSQL, as it shares the declarative nature of SQL-like language. There is no standard language, but several prototypes has been created. A common trait is that all queries has to be one-pass queries, due to the stream-centric nature of a DSMS.

When processing a stream of data we can use two main paradigms. explain event and deviation detection

An event is a match to a Continuous Query (CQ).

Esper, usage in TRIO
Chapter 3

Related Work

We take a look at research that is comparable to our problem statement

Because of the estimated high number of undiagnosed cases of OSA and the high cost of sleep studies, there has been a lot of research into non intrusive methods of detecting and diagnosis of sleep disorders.

3.1 sections: TODO

3.2 Other projects using other approaches

3.3 Other projects using similar approaches

Venkatesh, Anderson, Rivera and Buehrer[54] has developed a system using a Ultra Wideband (UWB) pulse radar to detect respiration and heart rate. The work is based on similar work using Doppler radars. The UWB radar has a much stronger material penetration capability. The article[54] lists possible usecases such as detecting people trapped under debris or snow, through-the-wall health monitoring in hostage rescue scenarios, vital-signs monitoring for
athletes performance, or similar to the TRIO system, as a non invasive method of assessing vital signs of a patient. The team presents a mathematical framework for analysis for analysing the potential of measuring the chest cavity in order to derive respiration and heart rate. The article shows promising results for respiration monitoring even through walls.
Part II

Design and Implementation
Chapter 4

Design

To generate events for the Esper engine in TRIO (see Section 1.4) we need a system that can analyse signals from respiratory sensors. Based on time series generated by sensors such as RIP or thermistor based respiration monitoring we must be able to derive events that are significant to the detection of sleep disorders.

In the analysis system of the signal gathered from sensors we look for two main qualities:

- detect stops in respiration (effort)
- detect these in as close to real time as possible

The signals are discrete-time signals or time series which can be represented as waveforms. This representation makes it easy to illustrate the signal and visually detect events in the signal.

Since detecting a halt in respiratory effort is dependent on time as a factor, it makes the most sense to crate events when we detect changes in the signal instead of looking for complete events, as these can span multiple second. We therefore need a system or algorithms that can provide us with peak and trough detection granting the ability to detect relevant changes in the time series.

When we detect that a waveform flattens based on a threshold we create an event, indicating that we have a halt in respiration
effort. A new event will be created when we detect changes that the respiration effort appear to have resumed. Based on further information at a later stage, the system (TRIO) that receives these events will then have to interpret events. That is, to turn the data into information.

By using an existing system for detecting these types of events, we gain the prior testing and verification of the algorithms used in such a solution. If we also can find an open source solution we can in the future make additions to the existing as we might come across new solutions that either give us capabilities beyond the existing system, or that work better than the existing ones.

4.1 Puka

The application we use for the respiration analysis is called puka. The decision to use this particular solution was based on the fact that even though a lot of other more recent and novel approaches exist, none of the could be found fully implemented. Since puka is not only implemented, but also open source we are able to tailor it to our needs and make modifications where we see fit. The source code for the application can be found on PhysioNets websites[40].

PhysioNet Resource is a public service funded by funded by the National Institute of Biomedical Imaging and Bioengineering (NIBIB) and the National Institute of General Medical Sciences (NIGMS) at the National Institutes of Health. PhysioNet can be divided in three parts:

1. **PhysioBank**: a collection of digital recordings of physiologic signals, time series, and related data

2. **PhysioToolkit**: a library of software for physiologic signal processing and analysis

3. **PhysioNetWorks**: a virtual laboratory for collaboration
4.1.1 History/overview

Puka is an open source application written by Joset A. Etzel, Erica L. Johnsen, Julie A. Dickerson and Ralph Adolphs in 2004 to analyse data collected from equipment and software from BIOPAC Systems, INC. BIOPAC is a company founded in 1985 that makes physiological measurement tools. The authors of the software found that puka was able to analyse other physiological signals as well. The latest implementation (2004) contains ECG and respiration analysis tools.

Puka uses MATLAB to calculate descriptive statistics such as heart rate variability, peak-trough respiration sinus arrhythmia and respiratory variables from ECG and strain gauge respiration data[41].

Strain gauge respiration data are time series that show the conductivity of the strain gauge allowing the variation of amplitude in the signal to reflect inhalation and exhalation. The respiratory analysis was designed to use signals collected with a TSD201 Respiratory Effort Transducer, a single strain gauge recorder.

The same analysis can also be applied to other signals that share the same characteristics as strain gauge respiration data, such as thermistor sensors and RIP (described in Subsection 2.2.2). These types of signals fluctuate around a base value, and give us a respiratory waveform when plotted against time. The rise and fall in amplitude is representing different physical attributes, such as conductivity in the case of RIP, or temperature in the case of respiratory thermistor.

describe RSA and articles

4.1.2 Program structure

The structural code of Puka is written in Java. This part of the code is responsible for I/O operation, interactions with the user and data persistence. Interactions with MATLAB are synchronous operations, originally via the library JMatlink, initiated by the user
using a GUI. The GUI has been created using the Netbean module Form, but since the intent is finally to strip away the GUI, no modification or upgrades are necessary for this project.

The application is initialized via the \texttt{frmMain} main class which instantiates the GUI and prompts the user as to what data input to use. Data can either be read from a database or from a file that adheres to the format allowed, described in Subsection 4.1.5. A simple program must be written to change the raw text output of the PhysioNet data to the format specified by puka.

\textbf{TODO:} \texttt{frmmain} -> creates an instance of preferences and loadData classes  
\texttt{frmpreferences} -> persistent storage of... well... preferences  TODO describe  
\texttt{frmloadData} -> opens the connection to MATLAB, contains the methods for reading data  
\texttt{frmrespiration} -> does the init, calls to MATLAB and collection of results

\textbf{Illustration of the program flow and functionality}  
\texttt{signal} \rightarrow \texttt{puka} \rightarrow \texttt{data storage (db, calculations, subject info)} \rightarrow \texttt{MATLAB (plot, calculation)} \rightarrow \texttt{java (present options and calculated data)} \rightarrow

\texttt{/TODO}

\subsection*{4.1.3 Runtime requirements}

In this thesis a 64-bit Windows 7 machine is used to compile and run all software. As stated in the manual for the software, operating systems other than Windows XP and 2000 has not been tested.

The two main dependencies for puka to run is Java and MATLAB. On the development machine the Java code has been compiled with Java JDK 1.8.0_60, with the exception for the attempt to compile JMatLink when Java 1.4 was used. MATLAB R2012b has been used to run all scripts.

Puka is dependent on parts of the WaveForm DataBase (WFDB)
Software Package. The applications found in this package are implementations of well-validated algorithms and can be used as a standard when testing the respiration analysis on real world data on recordings with ECG data as well as other sensors.

To be able to run puka on a windows machine, we have to convince puka that we have certain of the WFDB software present on the host machine. `ecgpuwave.exe` and `convertecg.exe` can be compiled using the compilers `gfortran` and `gcc` respectively, but we can also use dummies, empty files with their names as they are only used in the ECG analysis.

WFDB Software Package is available both as command line tools and as a MATLAB toolbox. This software package contains tools and libraries for working with data from the physiobank project. The package is available both for command line usage and for MATLAB. puka makes use of a small subset of the package (Figure 4.2), but other components can be useful for reading, retrieving and manipulating the recordings found in physiobank. The package requires Cygwin and certain libraries within the environment. Cygwin replicates significant parts of the POSIX system call API for a Windows environment, which WFDB package applications are dependent upon.

Cygwin also allows us to utilize the `gcc` and `gfortran` compilers, which are necessary to compile the support applications `convertECG` and `ecgpuwave` respectively. To read the file format used by physiobank WFDB gives us `rdann`, a program that can read local files or download the files from physiobank.

TODO: map all functions from WFDB used

- compile time dependencies:
  - javac or the puka jar
  - cygwin (runtime? nope)
  - install wfdb library
  - ecgpuwave: compile with gfortran
  - eegconvert: compile with gcc--wfd
  - WAVE?
Figure 4.1: Normal Respiration Rate

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdann</td>
<td>read the annotation created by ecgpuwave into an external file</td>
</tr>
<tr>
<td>ann2rr</td>
<td>create a RR interval series</td>
</tr>
<tr>
<td>ihr</td>
<td>create a instantaneous heart rate series</td>
</tr>
<tr>
<td>ECGPUWave</td>
<td>marks ECG waveforms</td>
</tr>
<tr>
<td>convertECG</td>
<td>converts ecg.txt into wfdb format</td>
</tr>
</tbody>
</table>

Figure 4.2: WFDB programs used by puka

- WAVE is an extensible interactive graphical environment for manipulating sets of digitized signals with optional annotations.
  - MATLAB : used for graphics (plot) and computations

- Run time dependencies:
  - java 1.4.2 ++
  - with JMatLink: certain 32-bit libraries

- Which are critical, which are for ECG / respiration analysis

4.1.4 Preferences

Puka looks for the preferences.txt file to working directory. The preferences consist of the absolute path to helper programs such as the WFDB applications and convertECG. In addition to keeping a track of helper application the preferences also keeps track of certain meta-data about signal clips to be analysed by puka.

The preferences window contains five tabs with different values [Figure 4.1.4]. These preferences has to be set for each system.

4.1.5 Data format

The program can either read data from a database or from a raw text file. Each line in a text file represents a sample and if we have
### Paths
- WFDB tools
- Installation directory (eccgpuwave.exe and puka.jar)
- WFDB data file directory (download and signals)
- ConvertECG.exe directory

### ECG
- Signal Frequency (Hz) (even though under ecg spec, used in resp)
- Signal unit (mV)
- Signal Gain (adu/mv)
- ADC resolution (bit)
- Zero-level (adu)
- Length of Record H:M:S

### Data columns
- Column for ECG and respiratory signal
- Onset trigger

### Clips
- Clip name and length (num samples)

### Database
- List of database connections

Figure 4.3: Preferences tabs
multiple channels they are separated by a white space character. The column number of the signal used by ECG and respiratory analysis is indicated in pukas preferences file which can be edited in the GUI.

TODO: flytt til PhysioNet data-description section??

The data found in PhysioNet is stored as .dat files and has to be converted in order to be used in puka. The PhysioNet toolbox offers tools for converting signals into text. The rdsamp program is used to read a specified record, either from a local .dat file or from the online database. The output is the decimal number on standard output. If a record contains more than one channel it will write output from each channel on the same line separated by tabs. The function also takes parameters allowing us to for example create CSV (comma separated value) files, read certain intervals, add time data based on the information found in the header file for the record. /TODO

4.2 Test data

In order to evaluate the accuracy and precision of pukas algorithms used for detecting respiration, we use both real world recordings from PhysioNet to derive respiration events from different types of sensors as well as simulated data. The respiration events derived with puka can then be used as input for the logical sensors found in the TRIO project we can compare the AHI and apnea annotation.

4.2.1 Simulated data

In the initial testing of puka we use a smaller sample sizes of the recordings representing different challenges as well as using simulated signals in order to reproduce different potential errors that can occur in recordings. The experiments conducted to map potential weak points and errors will be described in Subsection 5.4.2.

BIG TODO: describe normal respiration here or in background

A normal respiration rate, eupnea, varies with age, activity, illness,
Figure 4.4: Normal Respiration Rate[26]

<table>
<thead>
<tr>
<th>Age</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newborns</td>
<td>44 RPM</td>
</tr>
<tr>
<td>Infants</td>
<td>20-40 RPM</td>
</tr>
<tr>
<td>Children (1-7 years)</td>
<td>18-30 RPM</td>
</tr>
<tr>
<td>Adults</td>
<td>12-20 RPM</td>
</tr>
</tbody>
</table>

emotion and pharmaceutical influence[26]. In "Delmar’s Comprehensive Medical Assisting", normal respiration is defined as described in Figure ??.

The signals can be generated using a sine function. The sine of an angle $\omega$ in a right triangle is as the ratio of the lengths of the side of the triangle opposite the angle of the hypotenuse[2].

TODO: illustration of sine functions

This formula gives us a smooth sine curve. Actual respiratory waveforms for signals such as RIP are not smooth, and the during recording several types of noise is introduced. The respiration rhythm has to be taken into account. As described in "Evaluation of respiratory inductive plethysmography" [5], waveforms from respiratory sensors such as RIP tend to show pauses, especially at the end of expiration. TODO: !!! Finn kilde på ”normal” respiration

![Respiration signal from 1 minute of stage 2 sleep](image)

By manipulating these simulated signals we can create different scenarios such as respiration stops, errors in sensors and so on. Some of the potential hurdles we want to control for are listed in Figure ?? Using simulated data allows us to more easily test different challenges and aspects before moving on to more complex signals. We can also experiment to detect weakness in the puka implementation by adding and removing the different hurdles.
• amplitude fluctuation (threshold variability due to signal strength)
• noise (unexpected fluctuations)
• offset (we know puka relies on a baseline amplitude of 0)
• baseline shift (check out detrend)

For every type of hurdle, we can implement a script that generates that specific hurdle, and by combining these scripts we can compose the signals we need to perform the experiments to test pukas ability to handle the hurdles.

The data has to be converted into a format described in Subsection 4.1.5. These are trivial IO operations that require us to manipulate raw text files, delimiting the signal values with newlines, as is the delimiter puka uses when reading data.

4.2.2 Real world data

PhysioNet is a collections of recorded physiologic signals and related open-source software for signal processing and analysis. The data available is from different institutions around the world and it contains a variety of digital recordings of physiologic signals and related data for use by the biomedical research community[35].

To identify suitable datasets we will look for two main criteria: ground truth (AH1 and apnea annotations) in order to evaluate the results, and signal types allowing us to use puka to calculate respiration events in a format equivalent to the supported types.

An ideal dataset will contain different types of signals to enable us to compare the result on different sources. Annotations of respiratory events would be ideal and annotations for apneic events facilitates verification of the logical sensor using the events generated by puka.

MIT-BIH Polysomnographic Database contains both a respi-
ratory signal from a nasal thermistor and a respiratory effort signal from inductance plethysmography in some cases both chest and abdomen and others either one of them. Each record includes a header (.hea) file, a short text file that contains information about the types of signals, calibration constants, the length of the recording. It also contains AHI and sleep stage and apnea annotations which makes it a good candidate for usage in our tests.

The St. Vincent’s University Hospital / University College Dublin Sleep Apnea Database also has oro-nasal airflow (thermistor), ribcage movements, abdomen movements (uncalibrated strain gauges). As with the MIT-BIH database we have sleep stage and apnea annotations. In this database the annotation distinguishes between types of apnea, meaning we can control for the different types of apnea. It also contains annotations for other types of respiratory disturbance, meaning it can be a good candidate for future work. The database has both ribcage and abdomen movement recorded with uncalibrated strain gauges.

Another good candidate for our purpose is the apnea test database "Data for development and evaluation of ECG-based apnea detectors" which was used in the CINC challenge in 2000. Some of the recordings contains thermistor and or RIP signals in addition to the ECG signal. In the cases where we have all three signals, we will be able to compare the results from the different respiration rate estimation techniques allowing for comparative studies of the techniques. These records also contain the necessary AHI in order to determine the accuracy and precision of TRIOs logical sensors. The database also contains annotations for apneic events, which will allow us to see if there is correspondence between the same type of events found by the logical sensor and the data in the database.

All of the mentioned databases contains signals that can be used for different experiments.
4.3 Algorithms used in puka

The algorithms that are used in puka for respiration analysis are all implemented in MATLAB, found in the `matlabscripts` folder that accompanies the puka source code.

4.3.1 ECG algorithm

We will not spend too much time looking into the specifics of the ECG algorithm, but seeing how we can do a comparison between the different algorithms, we will briefly look into how puka derives respiration and heart rate data from ECG signals.

In order to compile the necessary external programs for the ECG analysis, we need to install a Fortran compiler in cygwin to create the Windows executable. This has to be included in the project and referenced in the preferences alongside.

4.3.2 Respiration Analysis Algorithm

The project site describes puka in the following terms: “puka incorporates a new method of identifying the breaths and pauses in strain gauge belt recordings. This technique locates the points of maximum inspiration and expiration for each breath as well as post-inspiratory and post-expiratory pauses”[41].

The algorithms used in the respiration analysis identifies critical parts of a recording. The critical parts are peak, trough, post-inspiratory pause (PI) and post-expiratory pause (PE). These four parts are useful metrics for detecting sleep apnea, and will have to be converted into events for the TRIO system.

The program uses a four step algorithm to analyse the respiration signal:

1. Identify peaks and troughs

38
frmLoadData.engMatLab.engEvalString("[P,T,th,Qd] = newPT(y, .1, onsetTime, endTime)");
...
frmLoadData.engMatLab.engEvalString("[peakLabels,troughLabels] = classifyPeaks(Qd,P,T,th);");
...
frmLoadData.engMatLab.engEvalString("[validPeaks, validTroughs] = makeValidArrays(P,T,peakLabels, troughLabels);");
frmLoadData.engMatLab.engEvalString("[newP] = markPeakPauses(Qd, validPeaks, validTroughs, th);");
frmLoadData.engMatLab.engEvalString("[newT] = markTroughPauses(Qd, validPeaks, validTroughs, th);");
frmLoadData.engMatLab.engEvalString("plotPauses(Qd, validPeaks, validTroughs, th, newP, newT);");
...
CalculateResp();

Figure 4.6: Main programmatic flow of respiration analysis

2. Check Validity of Peaks and Troughs
3. Mark Pauses at each Peak and Trough
4. Centre peaks and Trough

When this has been done puka does the following statistical computations: number of breaths, shortest breath, longest breath, average breath length, standard deviation of breath length. For post-inspiratory and expiratory (PI and PE) pause calculations the system calculates the average PI and PE pause of the clip as well as reporting the longest and shortest. The statistical computations gives us an indication of what kind of events puka is able to detect.

TODO: move this to impl? /TODO

Firstly the algorithm makes a pass over the whole clip, marking peaks and troughs using a peak detection algorithm. When we use the term peak in this section, we refer to both peak and troughs. Both are in principle the same.

Classification of peaks in the puka respiration analysis:
1. valid
2. invalid
3. questionable

After the classification the user is prompted to evaluate the marked peaks and make the final call on which peaks to accept and which to discard. When the peaks and troughs are identified and validated, the algorithm calculates the pause, if any, surrounding the peak or trough. This is done checking each direction from the current peak location, and based on the threshold the algorithm looks for neighbour sections with a slope higher than the threshold.

- (Step 2 has to be automatic (in puka this is done manually by end user) or disregarded for a real-time)
- split input-signal into smaller chunks (What size?)
- recording of statistical information for real time analysis
- normalization of signal (based on experience with getting the program to run)
- try to make it into a one pass algorithm for realtimeness

**for threshold (step2):**

- Keep track of an average amplitude, use this as a threshold for detecting peaks and troughs.
- Implement threshold variable into existing program.

Create `frmRespiration`, `loadData`. `y` in MATLAB contains the signal (start to stop only). ->

must be adjusted - discussed in implementation? The respiration analysis analyses each peak and through, and for this to work it first has to define a window to validate the peak or through. The `classifyPeaks` function found in the MATLAB scripts contains hard coded values that indicate the size of the window.
based upon the sampling rate. This should be parametrized and consistent throughout the program by passing the value stored in preferences in the control part of the application.

4.4 Modernization of puka

If we are to make a version where we won’t have to recompile the JMatLink library for each target, it might be of interest to look at different approaches than the use of JMatLink. There are different approaches to achieve this modernization. We will take a brief look at a few of the different options that can be considered for such a process.

4.4.1 Puka’s use of JMatLink

Puka uses the JMatLink library to communicate with the MATLAB engine. The library was created by Stefan Müller in 1999 [27] to allow users to interact with MATLAB via a web server, running a Java program. The library was re-released in 2005, but according to the change log it has not seen much development since then. Since the library is, at the time of writing, over 15 years old it is difficult to make it run on a modern system. It can be described as legacy software, in the sense that it can not be easily installed and executed on a modern system.

By running the jmatlink.dll through "Dependency Walker" [32], we are able to map the dependencies of the library. It seems that some of the 32-bit libraries JMatLink is dependent upon are only found as 64-bit versions on our version of Windows, and others are not found at all.

Since the library has some limitations considering modern systems we need to make modifications to be able to test puka for use in TRIO. The source code for both puka and JMatLink is publicly available so either one or both can be altered in order to make puka compatible with modern system and modern versions of Matlab.
Different possible approaches, pros and cons

4.4.2 Virtual machine

By running an old version of MATLAB and an operating system such as Windows 98, 2000 or Windows XP 32-bit, we should be able to run JMatLab library without having to recompiling the library. These are the platform and software puka was designed to run with, so this will allow us to start testing the application quickly.

This is not a long term solution and even if it would work we want to make changes to puka and integrate it into TRIO. This is also not in line with the overall goal of actually modernizing and utilizing puka in new ways. A practical consideration is how to get a hold of licenses for discontinued software such as Windows 98 and older versions of MATLAB. This solution is only considered as a last resort or for experimenting with the original source code and library if necessary.

4.4.3 Update JMatLink or write a new library

The source code for JMatLink is hosted on sourceforge.net [29], and the source code for puka is available on PhysioNet[40]. We can potentially update JMatLink to make it compatible with modern systems and software.

This solution will probably require a lot more work than other solutions. As described in ??, the library depends on decrepit systems, and a modernization of the library will require deep understanding of MATLAB internals, which is not in the scope of this thesis.
4.4.4 Writing a MATLAB wrapper

By removing the Java code, we strip away the need for the decrepit library. We can utilize the MATLAB scripts containing the algorithms by calling them from a new controller script, removing the need for calls from Java to the MATLAB engine.

We get to remove parts of the software that is redundant to the respiration analysis. There might be better ways to test puke before completely rewriting the control layer, but for the real time version of the respiration analysis this approach has to be considered, as the control layer has to be redesigned. The steps that are manual in the original version of puka will be altered into automatic versions or ignore the steps altogether.

4.4.5 Create Adapter for JMatLink

We can modernize the calls to the MATLAB engine by utilizing a modern Java - MATLAB interface and wrapping the calls that are intended for JMatLink in the new interface. This allows us to avoid much changes to the original source code.

Instead of using JMatLink we will look at the potential of using MatlabControl\cite{22} as a wrapper around the JMatLink interface to intercept and reroute calls to MATLAB to the new interface, that has none of the tightly coupled dependencies on specific system libraries and also supports calls to a 64 bit version of MATLAB. The adapter is based on pukas usage of the methods in JMatLink.

4.5 Adapter

The adapter has to capture calls that are intended for JMatLink, and execute the same operation as the library would have. Based on the JavaDocs for the library we can map what types and methods we need for the adapter.

There are exist a few solutions for executing MATLAB scripts
from a running Java application. There is support for calling Java classes from MATLAB, but no official way for the other way around. The solutions range from corporate [6] to hobby projects[24].

A promising project is matlabcontrol[22], as it seems to be frequently cited on MathWorks forums, it had an active development and has been forked and continued after the closing of the google code [] service where the project was initially hosted.

4.5.1 matlabcontrol

MatLabControl was originally created as a Remote Method Invocation (RMI) wrapper around an existing implementation made by Kamin Whitehouse at University of Virginia[58]. MATLAB has had the ability to make use of Java code since version 5.3 (R11) with the Java MATLAB Interface(JMI). Whitehouse sought to provide techniques[57] to call MATLAB commands from Java with a program written in 2001 using undocumented parts of the JMI library. The work on this Java class has been continued by Joshua Kaplan and the project matlabcontrol[22].

• More modern interface
• More recently maintained
• Not dependent on system libraries
• Compatible functionality
• Reasonably well documented

There are a few projects that have kept the MatLabControl project available even though the Google Code-service has been . The source code has been hosted on github and been made available through the Maven software project management and comprehension tool[1] allowing us to easily set up puka and a version of MatLabControl together.

The two versions that we found to be interesting are:
• matlabcontrol[38]
• MatConsoleCtl[53]

Both projects are based on the code hosted on Google code, and now hosted on GitHub. Both have been made into Maven artefacts. The project called matlabcontrol contains the pre packaged jar file of matlabcontrol 4.1.0, while MatConsoleCtl has seen changes made to it since it was forked.

Since MatConsoleCtl contains the source code and has been maintained since the last release in 2013, this seems like a good candidate for this thesis. The changes since 4.1.0 seem to be mostly minor bug-fixes such as error handling and a demo project for tutorial purposes.

4.5.2 JMatlink analysis

table of return value, name and parameters used in puka

We need to assess what types are being set and used Java, in order to make sure the conversion of these types is done properly. We know the return type of all the methods used in JMatlink[28] and will have to make sure the conversion between system is correct.

<table>
<thead>
<tr>
<th>JMatlink method</th>
<th>return type and description</th>
</tr>
</thead>
<tbody>
<tr>
<td>engGetArray</td>
<td>double[][][]. Used for both 1 and 2 dim array</td>
</tr>
<tr>
<td>engGetScalar</td>
<td>double</td>
</tr>
</tbody>
</table>

Table 4.1: JMatlink methods used by puka

1. Map what functions has been used in puka
2. Create interface for the adapter according to JMatLink javadoc
3. Prioritize methods that have been used in puka
4. Find equivalent methods in matlabcontrol and the necessary extensions found in the library
• MatlabTypeConverter and
• MatlabNumericArray for multidimensional arrays -describe in implementation. We get stuff for free since Java arrays are 0 indexed while MATLAB uses 1 indexed arrays
• double[x][y] -> x[1-n] y[nada] when using engGetArray to get single dim arrays?

5. Create unit tests for the prioritized methods

For our implementation of the adapter we will have to prioritize the methods that have been used in the implementation of puka. The software adheres to a strict naming convention which allows us to find all instances of the JMatLink class and what methods are called via text searches in the source code.

These searches show that all get calls to MATLAB returns a scalar, single dimensional array, or two dimensional array of Java primitive double.

4.6 Changes to puka

Two major tasks:

1. modernizing -> impl done

2. real timify -> need more theory and implementation after verification of step 1

4.7 Execution on modern system

Once the program is adapted for modern system we use the existing GUI while testing purposes, but ultimately we only make use of the respiration analysis part of the application. To create the adapter for JMatLink we set the project up as a Maven project to handle build and dependencies. We create a package called
\texttt{JMatLinkAdapter} which we then import into the puka source code-project.

The source code changes which has to be made to puka in order to swap out the original JMatLink library can be found by a text search through the source code for instances of the \texttt{JMatLink}. For clarities sake we rename the occurrences of JMatLink as JMatLinkAdapter, to make it clear that we are no longer using the actual JMatLink library.

The instances of JMatLink have been given the same name throughout the entire project, meaning we only replace the initialization of the class, since we keep the method names as described in the JavaDoc. The \texttt{JMatLinkAdapter} implements an interface based on the JavaDoc to make sure we maintain the same return types and arguments as expected by the existing code.

4.8 Real time analysis

Since the algorithm has been created to be run on a pre recorded signal we will have to look into what changes has to be done in order to provide a data stream with data that TRIO can use to derive useful information about the sleep quality and detect deviations from normal sleep. Looking into what makes the algorithm work, we identify the parts that can be removed, modified and added in order to be able to supply TRIO with useful events.

The overall goal of doing the analysis real time is to be able to detect sleep disorders report it to a different system that can act upon this information. Puka has algorithms for finding peaks and troughs based upon the surrounding signal, so we can not classify a peak the instant it occurs due to the definition of a peak. A peak is the highest point in a given section of a signal, surrounded by increasing and decreasing amplitude.

The statistical calculations done in the end of the analysis of a given clip gives us different insights to the qualities of the signal. But for a real time system, these calculations are not required, as they are aggregations on a static signal. We need to identify the
changes in the signal in order to convert them into events that can be sent to TRIO.

4.8.1 Approaches

Since the algorithm for detecting the pause around a peak or trough originally tries to find the entire pause, we need to modify it in order to be able to detect events as data arrives from sensors.

We use chunks of data. The smaller the chunks are, the better since we get closer to real time. A chunk is a finite section of a time series. We will get some lag in the system since the time series is generated while the program is running, but as long as we can report accurately the time of each event we will be able to detect the types of complex events we are interested in.

The GUI that accompanies puka can fairly easily be removed, making a system that takes a stream input and then applies the respiration analysis to it. For our purposes we create a program that feeds data into the stripped down version of puka.

Feeder -> puka -> TRIO input output illustrasjon med PI, PE samt start punkter

4.8.2 Data Feeder

We create an application which reads the data files and serves them to the puka reduced application, simulating a sensor. Each discrete data point is accompanied with a time stamp. This applications should be kept very simple.

The data serving application is a simple server that serves the data via a non blocking socket to any application that connects in the sample rate defined by the signal used.

The receiving application stores the data in a buffer until we have enough data to fill a window, and then applies the algorithms that have been adapted from the ones found in puka. These meth-
ods will have to be timed in order to discover how much time that can reasonably be used on the signal processing.

We will also need monitoring of the buffers in order to detect any overflows, indicating that the processing is slower than data entry rate, suggesting we need to downsample the signal.

### 4.8.3 Changes to puka

"Looking back". How large must the window size be to be able to identify a peak or trough? We need to experiment using different window sizes in order to find a balance between lag (the time we detect the pt) and accuracy (the number of false pt’s).

We identify which parts of the application define the window size (which scripts, and if there are anything in the Java code that needs to be changed). What variables can be parametrized in order to make the experiment easier to conduct?

The classification of peaks and troughs as it is done in the implementation of puka will have to replaced with an automatic alternative or removed. An automatic implementation can use statistical data to generate a threshold or a score criteria that each event has to fulfill in order to be classified as an actual event.

### 4.9 NOTES to be removed

Two main approaches:

Either create events for each PI/PE start and stop, or calculate the pause (most similar to the existing implementation) by increasing the size of the clip! (makes more sense?)

**WIP:**

- Windows - timed, find useful timeframe
- types of events, trough, peak, respiration stops...
• what is required from TRIO?
• update frequency?
• increase decrease in frequency = sacrifice precision?

• main experiments
• window size
• results compared to manual reading
• results compared to ECG algorithm???

other events that are noteworthy?

• matlabscripts:
• plotECGwithR.m
• checkPT.m
• testScript.m
• vssver.scc
• newPT.m
• markTroughPauses.m
• redrawResp.m
• calculateTtotal.m
• calculatePauses.m
• calculateInsExp.m
• TESTING.m
• calculateInsExpNoPauses.m
• adjustFlatPT.m
• plotRSA.m
• markPeakPauses.m
• findOnset.m
• classifyPeaks.m
• generatePT.m
• calculateRSA.m
• makeValidArrays.m
• plotPauses.m

/NOTES
Chapter 5

Implementation

5.1 Building or altering JMatLink

To execute puke it has to have some way of communicating with MATLAB, and it was written to use JMatLink in order to achieve this. We try to use the intended library, but we also look into the option of wrapping the calls in a more modern interface.

changes this to discuss why we don’t refactor JMatLink in a separate subsection

5.1.1 Compiling JMatLink

Since the JMatLink-dll that can be found precompiled by the author cannot be used as this version apparently does not support the 64-bit version of MATLAB, we attempt to compile a new version. The source code of the library is accompanied by a build file for Ant, a Java-based build tool [11].

The ant build file used to compile the project contains hard coded values that has to be changed in order to compile the library locally. These includes the path to Java Software Development Kit (SDK), Borland C++ Compiler[52] and MATLAB compiler support libraries (Listing 5.1). All of these components need to be installed

52
on the host machine in order to attempt a recompilation of the library.

**Listing 5.1: Hardcoded includes in build script**

```
-Ic:\j2sdk1.4.2_06\include
-Ic:\j2sdk1.4.2_06\include\Win32
-Ic:\bcc\INCLUDE
-IC:\MATLAB6p5\extern\include
-IC:\MATLAB6p5\simulink\include
```

As evident from the parameters, we need to update the arguments passed to the compiler based on the system we are using. We also have to make sure the targets within the different parameters actually exist.

**Listing 5.2: From JMatLink build script**

```
<target name="compile" depends="env">
<!-- compile object file -->
<exec executable="bcc32" dir="${build.src}/jmatlink/" >
  <arg line="-Ic:\j2sdk1.4.2_06\include
  -Ic:\j2sdk1.4.2_06\include\Win32 -c -3 -a8 -w- -b
  -g30 -Ic:\bcc\INCLUDE -oJMatLink.obj
  -IC:\MATLAB6p5\extern\include
  -IC:\MATLAB6p5\simulink\include -O1 -DNDEBUG
  JMatLink.c"/>
</exec>

<!-- link object file to DLL -->
<exec executable="bcc32" dir="${build.src}/jmatlink/" >
  <arg line="-DLL -eJMatLink.dll -tWD
  -Lc:\bcc\lib\32bit -Lc:\bcc\lib
  -LC:\MATLAB6p5\extern\lib\win32\borland\bc50
  libmx.lib libmat.lib libeng.lib JMatLink.obj" />
</exec>

<move file="${build.src}/jmatlink/JMatLink.dll"
todir="${build.dir}" />
<delete file="${build.src}/jmatlink/JMatLink.obj" />
<delete file="${build.src}/jmatlink/JMatLink.tds" />
</target>
```

When trying to build JMatLink with Java 1.4 JDK, Borland 5.x using ant 1.8.2 we get the following, self describing errors. As
we will discover later there is no support for the Borland compiler in the version of Matlab we have available. It is also more relevant to find a solution that can support modern versions of Matlab instead of relying on an older and specific version.

Listing 5.3: Errors when attempting to compile

```
[exec] Error E2194: Could not find file
   'Files\MATLAB\R2012b\extern\include.cpp'
[exec] Error E2194: Could not find file
   'Files\MATLAB\R2012b\simulink\include.cpp'
```

Digging into the documentation of MATLAB, it is clear that the compiler support needed to build the JMatLink library is not present in MATLAB 2012b (the version available to UiO students). In the original build-file we can see it includes a references to MATLAB6.5 specifically, which did have support for the Borland5.x compiler, but has not been present in subsequent releases.

Without any documentation as to what the missing components do, the compilation of the library is difficult to complete. Based on the error messages from the compiler, we can track down what components found in MATLAB 6.5, the bc50.cpp, that will have to be included in the project. A dependency on an obsolete version of MATLAB is not desirable, so other approaches to running puka might be more future proof.

According to MathWorks, MATLAB had support for the Borland compiler up until Release 2007b. The versions of MATLAB that have support for the Borland compiler are 32 bit versions, so the necessary libraries for compiling for MATLAB 2012b 64-bit does not exist. So even if the JMatLink source code is available, the hurdles of compiling and linking the library are far greater than writing a wrapper to reproduce the functionality of the library.

The process of compiling the original version of JMatLink made extra convoluted because of the lack of documentation and the fact that the dependencies are deprecated and no longer maintained. The necessity for a system library and specific MATLAB versions to run puka is not ideal. A more portable solution will make it more useful. Therefore the focus will be shifted to modernising the software.
5.2 Creating the adapter

From design:

- Virtual machine
- Update JMatLink or write a new library
- Writing a MATLAB wrapper
- **Wrapping the Java-MATLAB interface -adapter**

There are multiple reason for making an adapter instead of any of the other solutions put forward in [chapter 4](#). TODO: why? or is this already described in design?

5.2.1 Interface

Based on the javadoc and the source code, we can create a list of methods that the application makes use of. Ideally we want to create an adapter that replicates the entire functionality of the JMatLink library. But the prioritized methods are the ones in use by puka. These methods are listed in ??

The following methods are the complete list of methods found in the JMatLink library, but only the ones used by puka will be described further.

5.2.2 Software testing

When writing the adapter we first write the unit tests to validate the communication between Java and MATLAB. Matlabcontrol has been fairly well documented, but we need to create tests to validate that the results are correct. The type conversion between the two systems (Java and MATLAB) has to be correct, and the return types has to match the expected types found in puka.
Listing 5.4: Used methods in puka from JMatLink

```java
//matlab session:
engOpen() : void
engClose() : void
setDebug(boolean debugB) : void
kill() : void

//matlab commands:
engEvalString(String evalS) : void
engGetScalar(String arrayS) : double
engGetVariable(String arrayS) : double
engGetArray(String arrayS) : double[][]

engPutVariable(String arrayS, double[][] valuesDD) : void
engPutArray(String arrayS, double valueD) : void
engPutArray(String arrayS, double[] valuesD) : void
engPutArray(String arrayS, double[][] valuesDD) : void
```

Even though the API is similar there are certain differences we needed to take into account when writing the adapter. The major concern is to make sure we convert types and arrays correctly. As the assessment of the puka source code and analysis in Section 4.5, we have the following Java types to convert between Java and MATLAB, and back again:

<table>
<thead>
<tr>
<th>Java Type</th>
<th>MATLAB Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>double scalar</td>
</tr>
<tr>
<td>double[]</td>
<td>Numeric array</td>
</tr>
<tr>
<td>double[][]</td>
<td>Numeric array</td>
</tr>
</tbody>
</table>

All of the Java types passed to MATLAB can and should be represented as an numeric array. This allows the MATLAB scripts to do calculations without the risk of receiving a non numeric type which will be incompatible with the scripts that implement the peak detection.

Before we implement the adapter we create unit tests for all
calls puka does through the MATLAB interface [Listing 5.4]. We control that values passed between each system is appropriately converted and retain the correct value. We must also make sure that operations and conversions return the expected result and type.

The interface does type conversion between MATLAB and Java for primitive types, but there are certain differences to be aware of. As stated in the documentation for the library [23], it is not possible to send Java primitives directly to MATLAB. All variables are treated as arrays in MATLAB, even scalar variables. This means the programmer has to keep track of the types used in the MATLAB scripts and decide what method to use when retrieving the variable. When getting scalar variables we need to cast the first (and only) member of the array to a Java double primitive. We also make sure that MATLAB considers it a scalar by using the built in function `isscalar()` after setting a scalar.

For conversion of single dimensional arrays `matlabcontrol` automatically converts between Java and MATLAB arrays. Most Java primitives have a corresponding MATLAB array type, with two exceptions. These two are `char[]` and `long[]` and will cause a MATLAB to throw an exception if used in MATLAB version R2009b or higher.

When converting multi dimensional arrays, we need to make use of the `MatlabTypeConverter` class, which can be found in extensions. This class converts between Java array to the type MatlabNumericArray. If the conversion is not done, the resulting array in MATLAB will be a cell array. The scripts used in puka expect arrays of the type `double`, and will throw an exception if they receive cell arrays due to the inability to do double precision mathematical operations on cell arrays.

### 5.2.3 Testing

To test that arrays are imported correctly we perform matrix addition and multiplication to verify that both Java and MATLAB gives us the same result.
Java uses zero-indexing for the arrays. The first element in a given array is given position zero, as opposed to MATLAB whose index starts with one. The conversion of indexing is handled by the matlabcontrol library, so no consideration has to be made to this potential problem, but it can be important to make note of the difference. To prove that this is handled correctly, we write a unit test to verify that values at the first and last position of an array are the same.

JUnit is used for testing the methods in JMatLinkAdapter. The tests has to cover each method used by puka [Listing 5.4]. They also have to verify that results are as expected in order to be certain that the type conversion works as expected.

Some of these tests has to rely on other parts of the interface to be able to automatically assert the result. In addition we add print-statements to both the MATLAB window and the standard output. The only test for which we can not find a JUnit assert solution for is the debug print from matlabcontrol.

In order to be able to test for exceptions we let all methods throw exceptions. This leads to changes in the puka source code. For example for engGetScalar we need to add throws MatlabInvocationException statements to the methods that makes use of the method and surround the callee with a try/catch statement.

5.3 Creating test data

A simple implementation of this formula in MATLAB gives us a perfect sine curve and a time series that can be used as an ideal signal.

TODO: update

The noise added to the signal is created by adding random deviations from the mean value of the signal, but maintaining the normal distribution. Ideally, based on the sample rate found in the RIP recordings in the CINC data set, we would use 100 samples per second. Due to quirks in the source code we start out using 1000
ns = 0:2999; % number of samples
am = 1;    % amplitude
base = 0;  % offset
dur = 500; % duration of each "respiration"
    % dur:500/100 samples per sec = 5 second
h = 100;  % samples per second

sinewave = base + am * sin(2*pi*ns/dur);

figure();
plot(f/h, sinewave); % y-axis: 1 second = 100 samples per second and change the scripts to not assume sample frequency. This is described in detail in Section 5.5.

Listing 5.5: Adding noise
% if we want to add noise:
deviation = 0.2;
noiseAmplitude = deviation; % should be less than am
noisy_x = x + noiseAmplitude * rand(1, length(x));

Since the normal respiration of a human being is not the same as a perfect sinusoid curve, we create a couple of functions to generate more lifelike signals. One such approach is to clip the signal both on inspiration and expiration end, to more easily be able to validate detection of peaks and troughs and their beginning and end, in order to detect pauses in respiration.

Listing 5.6: Clipping sine wave
% for creating clipped time series
clipped = sinewave;
cutoftop = 1.0; % am has to be set to > cutoftop
cutoffbottom = -1.0;

clipped(find(clipped> cutoftop)) = cutoftop;
clipped(find(clipped < cutoffbottom)) = cutoffbottom;

figure();
plot(ns/h, clipped);
\[ o + a \cdot \cos \left( \sin \left( \frac{x}{n} \right) \cdot \pi \right) \] (5.1)

Figure 5.1: o: offset, a: amplitude, n: number of samples between each peak, x: number of samples

A different approach is to create the signal based on Figure 5.1.

### 5.4 Usage of puka

In order to run puka using the "JMatlinkAdapter" we have to change the instantiations of JMatlink in the source code the adapter class. Calls to the loading of the JMatLink library can be removed, as it is no longer a system library we are dependent upon. The creation of a new instance of the library is done in `frmLoadData`. The library is also loaded in `frmConvert.java`, but this class is only instantiated by itself, and is probably separate from puka and can safely be ignored.

```java
engMatLab = new JMatLink(); // initiate connection
try {
    System.loadLibrary("JMatlink"); // no longer in use
    engMatLab.setDebug( true );
    int intC = engMatLab.engOpen(); // open connection to MATLAB
} catch (Exception e) { e.printStackTrace(); }
//===========================================================
// Replaced with:
engMatLab = new JMatLinkAdapter();
```

Calls to JMatLink are now intercepted by the JMatLinkAdapter class which replicates the expected behaviour by using matlabcontrol.

In `preferences.txt` we need to define which column in a raw text file we will find the signal used in the respiration analysis, as described in Subsection 4.1.4. When creating the simulated data we only have one column, meaning this parameter will be set to
1. As we do not simulate ECG signals the ECG column will be set to -1, as per documentation. The onset trigger will be set to 1 as well. This value will be updated by the execution of the respiration analysis.

The length of the record can be set longer than the clip length, but this parameter is only for the GUI part of the application.

When the pause analysis has been completed, we have two new arrays that are based on the previous P and T: newP and newT. For each peak and trough we have twin tuples marking the beginning and end of the pause around a peak.

5.4.1 Signals used

5.4.2 Experiments

Figure ?? gives us scenarios we need to tackle through our experiment design. The signals used in these experiments are created using the techniques described in Subsection 5.4.1 so that we can see how puka and its respiratory analysis handles the different scenarios.

Firstly we create experiments to validate the claims of the puka documentation. That is peak detection. As stated in the peak detection finds the extreme points in a clip. The ends of expiration and inspiration.

5.5 Adjustments

When we run pukas respiration analysis on the simulated data found in Section 5.3 we get different errors and exceptions based on what signal is used and the clip length.

Her vil jeg beskrive gjennomføringen av to, tre eksperimenter og vise til feilmeldinger og konsekvenser av typer signaler, for så å
5.5.1 Decimate

The first MATLAB script to be called is \textit{newPT.m}. In this script, which is based on The Identification of Peaks in Physiological Signals\cite{51}, the programmer has assumed a fixed sampling rate, based on the fact that the raw signal ($Q_{raw}$) is down sampled with a factor of 5 using the MATLAB toolbox decimate function.

A more explicit mention of an assumed sample rate by the script creator is found in the comments of \textit{classifyPeaks.m} (Listing 5.7). Here it is stated that the sample rate has been decimated by five, and that the original sample rate was 1000 Hz.

This has to at least be considered when creating sample data, but should ideally rewritten into using the parameter in \textit{preferences}, and only down sample if it is needed based on performance.

5.5.2 Pause window size

The respiration algorithm classifies the peaks with the MATLAB script \textit{classifyPeaks.m}. The main purpose of this script is to evaluate the peaks based on the threshold set in step 1, and indicate which peaks and through the system deems valid or invalid, and which peaks it will prompt the user to consider. The documentation and implementation is at odds, and requires extra investigation as this script often creates difficulties for the execution.

\textbf{Listing 5.7: Classify Peaks comments}

\begin{verbatim}
% go across entire signal, looking at narrow window around each peak
% try 1 second windows around each peak/trough, centered on found peak
% 1000 Hz signal decimated by 5, so now 200 Hz; 200 data pt window either side
\end{verbatim}
In the files comments the programmers have written. It does not use the sample rate set in preferences, and there are two more discrepancy in the source code. Firstly, the hard coded value it uses to check how many samples we are into the time series is 156, which does not reflect a 1 second window even with 200 Hz. Also, in the case that the first peak detected is not after 156 samples into the time series, a comment states that we should "start at first peak higher than 101". This refers to peaks/troughs at 100 samples or more, but when the signal has been decimated, this is just an arbitrary number, and no longer reflects the intention of making sure we have enough room to check for pauses.

The downsampling happens in the first function that is called, "newPT" and we assume it is done for performance. The major problem with this aspect of the scripts used by puka is that it does not accommodate different sample rates. If we use a signal with 100 samples per second, the only have 20 samples per second after the decimate, resulting in inaccurate signals and huge windows when looking for pauses around peaks and troughs in classifyPeaks.m.

Listing 5.8: Downsample of the signal in "newPT"

\[ Qd = \text{decimate}(Qraw, 5); \% \text{downsample the signal} \]

These inconsistencies suggest we need to modify the scripts to be able to dynamically adapt to the sampling rate. By using the variable sampling rate in the "ECG"section of the preferences and making it a global variable within the MATLAB instance, we can calculate a one second window for use in the classifyPeaks.m.

5.5.3 Mark pause peak and trough location

When running markPeakPauses.m with a 10 second clip on a 30 second record using the clipped sine generated signal we end up with a deadlock in the MATLAB script. The amplitude of the initial clipped sine signal was 0.8.

It checks the number of valid peaks created classifyPeaks.m
"newP" and "newT" are variables in the script that both contain the new, centred peak or trough based on the pause one each side. During the execution of this script we run into a problem when checking each side of the original peak/trough. The signal passed to the script is called "Qd" and is the signal in which the peaks and troughs have been found.

When peaks or troughs are too close to the beginning or end of the clip, the MATLAB script freezes during markPeakPauses or markTroughPauses. These functions iterates through the peaks and troughs analysing a windows around each event. There is no fail safe implemented, but in puka a user can choose what peaks to analyse. Here we can de-select detected events that border too close to the beginning or end at the cost of the accuracy of the analysis.

This can also be avoided by adding an automatic discarding of these events by checking the index (sample number) of the peak/trough, making sure it is not positioned as to make

\[
\text{My very own formula} : f(x) = \cos \left( \sin(x \cdot \pi \cdot \text{samples}/) \cdot \pi \right)
\]

if \( o = 1 \) or \( o = -1 \) we get a respiration like graph with sharper peaks and a pause in the trough.

### 5.6 Additional software

- **SignalPatcher**: Rydde opp i signaler fra physionet ettersom de mangler enkeltmålinger. Se Cleaner.java for enkel løsning
- **SignalGenerator**: lager simulerte greier. Sinegenerator.m lager ideelle og problematiske signals
- **DataFeeder**: leser inn og sender sender data til en klient via socket
5.6.1 Signal processing

The real world data that is found in the different Physionet databases prepares us for what kind of challenges can be found when using these types of sensors. Certain signal processing has to be done to prepare the signal for usage in puka.

Post/pre processing test data:

- fft? high pass/low pass filter?
- smooth signal:
  - moving average - really slow, removes more noise
  - SG - much faster, keeps more of the features
  - initial
- normalization:
  - 

resultater etter endringer

5.7 Real time detection

As described in chapter 4, the real time version of puka will have to be able to detect changes and report these events to TRIO using some form of change point detection. Both the algorithm for detecting peaks and troughs and the algorithm for calculating pauses in respiration has to be modified in order to detect events in as close to real time as possible.

We create a simple program that reads files containing time series which can be served to a the modified version of puka. This version of puka is stripped of the GUI components, and is made to work in conjunction with the data serving program. The events detected by the modified version of puka is then compared with the results of a respiration analysis done with the original version of puka.
5.7.1 Data serving

The application that serves the data is implemented in Java and uses Java NIO socket channels to send data to a connecting application. In order to simulate a fixed sample rate we will add a time stamp to each discrete data package. We will also need to make sure that the program is implemented efficiently enough as to be able to send at a realistic rate.

A very simple text based protocol is implemented in order to control the flow of data. The connection phase consist of a simple handshake between the client and the server, where the client sends the server the name of the signal. The server then reads the signal file and stores it in memory to reduce the number of disk IO operations. The size of a given signal file will vary based on the length and sampling rate of the time series.

Since both the data serving application and the data consuming application is running on the same system, use Java System.currentTimeMillis. By using milliseconds we can send data at a rate up to 1000hz, or 1 per millisecond.

5.7.2 Data consuming, puka reduced

Extracted from the puka application, the respiration analysis can be summarized as shown in ?? in Part 8.

By receiving and processing in parallel, we can batch the signal and keep receiving data from the simulated sensor.

5.8 Modifying puka for online usage

1. create window
   - try different window sizes
   - window: overlapping, sliding?

2. analyse while filling the next window – parallel or
5.8.1 Window sizes

As puka uses a *clip* as a measurement as how many discrete points it analyses at a time, we use this clip parameter in order to express the window size in puka. The main consideration when choosing window size is the *responsiveness* of the application. The smaller the window, the closer to real time, but the number of false positives and negatives will necessarily increase as each pass of the algorithm will have less data to base its analysis on.

5.8.2 Looking backwards

5.9 Result/Analysis online

Hva fungerte av online biten?
Part III

Discussion and Future Work
Chapter 6

Discussion

...
Other non invasive approaches

• Respiration rate using Novelda radar
• wake/sleep state
• oxygen saturation
• other bio markers?

• Calculate respiratory flow in addition to bpm


• Other sensors, $so^2$ $sco^2$.

• Combine the events generated by pukaReduced (middertidig navn) with other sensors.

• Recreate the respiration analysis as an CQL or some other stream favourable system.

• Now that we have a framework for applying algorithms to a respiratory signal, novel algorithms can be implemented in order to detect new types of events or improvements to the existing ones.
• Test on flow as well as respiration effort. Focus here have been on RIP, but this should be applicable to other sensors such as thermistors, which are able to detect changes in flow, not only respiration effort.

• Use the system for detecting other respiratory abnormalities, such as tachypnea, bradypnea, Cheyne-Stokes, hypoventilation, hyperpnea or hyperventilation[26].
Chapter 8

Conclusion

...
8.1 Software

Software delivered:

- Written:
  - JMatLinkAdapter
  - pukaReduced
  - sineGenerator.m

- Modified:
  - puka
  - matlabscripts

8.2 Troubleshooting

The following section contains some of the bugs found and corrected.

8.2.1 Script execution freeze

When running puka it sometimes freezes at after entry to newPT.m:

```
Listing 8.1: Last log entry

mar 27, 2016 8:03:00 PM
matlabcontrol.LoggingMatlabProxy eval(String)
FINER: ENTRY [P,T,th,Qd] = newPT(y, .1, onsetTime, endTime)
```

Does not return from engEvalString(), but when the Java process is stopped, the execution of the MATLAB script finishes. The instance of MATLAB does not respond during this time, so no information about where in the execution it might stop.
Listing 8.2: Expected log entry

mar 27, 2016 8:12:48 PM
matlabcontrol.LoggingMatlabProxy eval(String)
FINER: ENTRY [P,T,th,Qd] = newPT(y, .1, onsetTime, endTime)
mar 27, 2016 8:12:48 PM
matlabcontrol.LoggingMatlabProxy eval(String)
FINER: RETURN

Does not work with proxy: setHidden() set to true. Something in the implementation creates problems. Without any debug information from either MATLAB or Java, we do not know exactly what causes this bug. But when run with setHidden to false, the call to initiate newPT returns and the execution continues.

8.2.2 Peak detection sample rate assumption

In the MATLAB scripts containing the respiration analysis, the authors have assumed a given sample rate of 1000 Hz. This is evident in comments and code in newPT.m and classifyPeaks.m. More details can be found in Section 5.5.

To avoid future confusion, we introduce a sample rate variable set in each script which is in the case of the GUI application set in preferences and in the case of the command line application in the Settings class.

8.2.3 Misc bugs

frmRespiration.java:43 – ‘,’ instead of ‘:’, might be change in MATLAB, but this results in y being set to a scalar?
Listing 8.3: Respiration analysis programatical flow

// :745 prepare and clear data
frmLoadData.engMatLab.engEvalString("[P,T,th,Qd] = newPT(y, .1, onsetTime, endTime)"); //run the matlab script

// :536 peak analysis
frmLoadData.engMatLab.engEvalString("[peakLabels,troughLabels] = classifyPeaks(Qd,P,T,th);");
//get the peaks/troughs and labels back into the table
dblP = frmLoadData.engMatLab.engGetArray("P");
dblT = frmLoadData.engMatLab.engGetArray("T");
db1Plabels = frmLoadData.engMatLab.engGetArray("peakLabels");
db1Tlabels = frmLoadData.engMatLab.engGetArray("troughLabels");
FillPeaksTable(0); FillTroughsTable(0); //fill both with all peaks/troughs

// :575 Do apply has to be described later...
DoApply(); //call cmdApply first
frmLoadData.engMatLab.engEvalString("[validPeaks, validTroughs] = makeValidArrays(P,T,peakLabels, troughLabels);");
frmLoadData.engMatLab.engEvalString("[newP] = markPeakPauses(Qd, validPeaks, validTroughs, th);");
frmLoadData.engMatLab.engEvalString("[newT] = markTroughPauses(Qd, validPeaks, validTroughs, th);");
frmLoadData.engMatLab.engEvalString("plotPauses(Qd, validPeaks, validTroughs, th, newP, newT);");

// :481 //control the computation of breathing statistics
ArrayList jcTempList = new ArrayList(); double[][]
dblTemp; int intTemp = 0; int intC = 0;
double[][] dblTroughs; double[][] dblNewP; double[][]
dblNewT;

//need the peaks and troughs array regardless of using pauses or not
frmLoadData.engMatLab.engEvalString("[peaks,troughs] = generatePT(P,T,peakLabels, troughLabels);");
//Ttotal is calculated off of the troughs array - pauses don’t matter
frmLoadData.engMatLab.engEvalString("[avgTtot,stdTtot] = calculateTtotal(troughs);");

//call the matlab scripts to do the calculations either with or without pauses

List of Figures

2.1 Obstructive Sleep Apnea ............................... 8
2.2 Sleep stages[48] ................................. 15
2.3 RR Intervals ........................................ 17
2.4 Precision and accuracy[20] ......................... 20

4.1 Normal Respiration Rate[26] ........................... 32
4.2 WFDB programs used by puka ......................... 32
4.3 Preferences tabs ....................................... 33
4.4 Normal Respiration Rate[26] ........................... 35
4.5 Respiration signal from 1 minute of stage 2 sleep [13] 35
4.6 Main programmatic flow of respiration analysis .... 39

5.1 o: offset, a: amplitude, n: number of samples between each peak, x: number of samples ....... 60
5.2 Window sizes used in experiment ...................... 67
List of Tables

2.1 AHI severity scale ........................................ 12

4.1 JMatlink methods used by puka ...................... 45


[52] TODO. Borland compiler. [http://TODO.org/], TODO.


