UiO : Department of Informatics
University of Oslo

Sleep Movement Analysis using the Microsoft Kinect v1 Depth Sensor

Karine Gran Vifstad

Master's Thesis Spring 2016

Sleep Movement Analysis using the Microsoft Kinect v1 Depth Sensor

Karine Gran Vifstad

May 16, 2016

Abstract

Sleep analysis can help improve sleep quality, and determine the ideal time for waking up a person. Sleep analysis can be performed in different ways, one of them is analysing the movements of the sleeping person, since large movements usually means that the person is about to wake up.

There are multiple tools for sleep movement analysis, but usually they are quite advanced, expensive and not affordable for usage in private homes. Instead, this project explored the usage depth images from Microsoft Kinect v1 to determine sleep movements, which is a cheaper alternative.

This project developed a program for detecting the frequency of movements during sleep using Kinect v1. The process of performing sleep movement analysis was done by taking images with certain intervals, and comparing a pixel in a certain position in one image by the pixel in the same position in the subsequent image.

The project found some challenges which made the sleep movement analysis tricky to do at home, including positioning of the Kinect, noisy images, and that detecting movements of people sleeping is harder than when they are standing up.

Overall, the program was able to distinguish between large sleep movements and smaller ones, and used this information to determine the sleep quality of the patient. Further, the results of the program showed that it is possible to create cheaper alternatives than the ones already developed.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

> "My alarm clock is jealous of the relationship I have with my bed."
>
> *- Unknown*

Sleep is an important part of life, it affects both mood and performance in your everyday life. The quality of sleep is a vital factor in the overall quality of human performance at daytime. It has been studied to great extent to record sleep patterns and sleep disorders in detail. This is very useful to detect for instance sleep apnea (temporarily pauses in breathing during the night) which can be life-threatening when not treated correctly. Analysis of brain waves, the oxygen level in your blood, heart rate and breathing are all important factors to measure sleep behaviour, but this requires quite invasive methods to register. Sleeping in a hospital bed with wires and electrodes attached to the skin while sleeping, is not as comfortable as sleeping in the bed at home, and this may effect the sleep measurement negatively. Body movement can also be a good and descriptive way to analyze sleep, and this can be done at home with just a camera.

Human sleep is characterized by periods of immobility, interrupted by position changes, and research shows that body movement during sleep is closely related to how deep the sleep is. Major movements indicate lighter sleep, which would be the optimal time to wake up. Modern sleep trackers and alarm clocks use motion to find the optimal time for awakening, to avoid interruption of deep sleep which can lead to tiredness during the day. However, sleep is usually registered by a accelerometer, gyroscope or a heart rate monitor, not by video.

The Microsoft Kinect was originally intended as a motion sensor for Xbox gaming, but has been used in many other fields such as computer science, electronic engineering and robotics. The reason why I chose to use Kinect is because it has both depth vision and infrared camera, it is reasonably priced, easy to get hold of, and a lot of people already own one together with their Xbox. This thesis studies sleep movements recorded by the depth sensor of a Microsoft Kinect v1.

1.1 Project goal

This thesis is going to explore the quality of the depth images provided by the Kinect v1 sensor, in order to find major movements during sleep. Further, I will research whether these movements are accurate enough to be used as input to an alarm clock, and see if it provides good enough results to compete with modern applications such as the "Sleep as Android" alarm clock.

The main goal of the thesis is to investigate if the Kinect depth sensor can provide images which are accurate enough to decide whether the patient is asleep or awake at different times during the night. This binary scoring is used to evaluate sleep quality.

Is it possible to achieve precise motion recordings from a home made sleep surveillance system?

1.2 Project structure

In order to achieve the project goal, the project was structured in the following manner:

1. Finding the optimal setup for the sensor. The setup was limited by housing regulations, as discussed in section 1.3, however it was important to minimize blind spots and areas that were outside the range of the depth sensor.

2. Investigating the accuracy of the depth information, in order to detect noise.

3. Comparing consecutive frames by detecting pixels that have changed value compared to the previous frame. Is the amount of motion pixels[1] higher for frames with more movement?

[1]Motion pixels are the pixels that have changed value compared to the last frame.

4. Testing different preprocessing methods in order to improve the result. An improvement of the result could be an decrease of motion pixels in images with low movement, and at the same time equal or increased number of motion pixels in the images with more movement.

5. Registration of motion over a longer period of time. Create motion plots from the data to show the major movements during the night.

1.3 An Overview of Challenges

Normally the Kinect sensor is used to detect standing people, in for instance Xbox games where you control your game character by moving your hands or body. The depth of the people playing is very different than the depth of the wall behind them. In the case of sleep surveillance, the subject is lying in a bed at approximately the same depth, so the Kinect should be placed in the most optimal distance. The sensor is most accurate for the objects within the range 0.8-4.0 meters[11]. Placing the Kinect within this range can be challenging in a normal bedroom without a custom panel placed in the ceiling which the Kinect can be mounted on.

Another challenge is that in order to make the surveillance as non-invasive as possible the subject will be covered by a duvet which makes detecting moving body parts harder. So the motion the kinect will detect is not directly the person moving, but motion of the duvet and visible body parts like the head and arms.

A third challenge was the setup of the Kinect. It was hard to find a good location for the sensor. The optimal setup would be to have the sensor attached in the ceiling directly over the bed. However, because of housing regulations I was not allowed to build a panel in the ceiling which the kinect could be attached to. I also moved during the project, and therefore had to change the setup during the recordings.

The results from the Kinect can either be stored as video or images. Advantages with video is that recordings from one night only results in one video file. However, images are easier to process, and it is easier to look at single occurrences and confirm that there are actual motion in one particular frame. The images from the kinect contain much noise, and this has to filtered out. My setup of the Kinect contained more than just the bed area, so the images also has to be altered to only contain the region of interest.

The last challenge concerns the Kinect itself: I experienced the sensor as unstable. Some nights the program failed to open the device, and thus did not record anything. To avoid this I had to unplug and plug the sensor after

every run. In addition to this, the sensor makes some noise and was blinking during recording, so if I was awake at the time the sensor started, this made it harder to fall asleep. The solution to this was starting the recordings in the middle of the night, so I was already asleep.

1.4 Thesis outline

The first chapter is this introduction, including the problem statement, the project plan and the issues that proved to be challenging. Chapter 2 covers theory and background information necessary for the project. Chapter 3 includes related works in the area of detection of sleep movements. Chapter 4 presents materials and methods used for the implementation of the program. Chapter 5 covers results, discussion and limitations within the thesis. Chapter 6 contains the conclusion of the thesis and possible future work.

Chapter 2

Background

Sleep have always fascinated humans, but the mechanics of sleep remained a mystery until the 20th century. This chapter will start with an overview of important incidents regarding sleep research in section 2.1, and then go into detail about sleep stages in section 2.2. Section 2.3 is dedicated to sleep cycles, while section 2.4 covers sleep monitoring, including polysomnography and alternatives to this. Section 2.5 gives a brief introduction to articles about sleep surveillance. Digital images are covered in section 2.6 and a short introduction to classification is covered in section 2.7. Finally, this chapter ends by giving an introduction to Kinect; section 2.8 covers a general introduction to Microsoft Kinect and the depth sensing technology used, section 2.9 describes the depth space of Kinect, and section 2.10 is dedicated to programming with Kinect.

2.1 Sleep Research Timeline

Sleep research has been an increasing field the past century. The webpage www.howsleepworks.com has collected sleep research and provided a timeline for sleep research through the years[27]. Following is an excerpt from the timeline:

- 1924: Hans Berger, a German psychiatrist, records the first human electroencephalogram (EEG) - a graphical representation of brain waves.

- 1925: Nathaniel Kleitman opens the world's first sleep laboratory at the University of Chicago, where he researched circadian rhythms, sleep and wakefulness regulation and sleep deprivation.

- 1929: Hans Berger develops an electroencephalograph device to record

brain waves, and notes differences in brain activity during sleep and wakefulness.

- 1937: Alfred Loomis, E. Newton Harvey and Garret Hobart identify five distinct stages of sleep, with use of EEG traces.

- 1953: Nathaniel Kleitman and Eugene Aserinsky discover rapid eye movement (REM) sleep, separating it from non-REM sleep.

- 1954: William C. Dement shows that sleep consists of cycles of different stages of sleep, repeated four or five times a night.

- 1959: Michel Jouvet, a French physician, finds that REM sleep is a totally distinct phase of sleep, where the brain activity is similar to that during wakefulness.

- 1968: Rechtschaffen and Kales make a standardized terminology and scoring system to classify the different sleep stages.

- 1975: The Association of Sleep Disorders Centers (changed name to American Academy of Sleep Medicine in 1999) is established, with a goal of bringing sleep medicine into the scientific mainstream.

- 2007: The American Academy of Sleep Medicine changes the model of sleep stages, reducing the amount of non-REM sleep stages from four to three.

2.2 Sleep Stages - REM and NREM

Human sleep is divided into two states: REM and NREM. The first state is characterized by rapid eye movement (REM), small muscle twitches, changes in autonomic activity and the absence of other body movements.

The other state is known as non-rapid eye movement sleep (NREM). There are usually little or no eye movement during NREM, and the muscles are not paralyzed as in REM sleep. NREM is divided into 3 substages[20]:

2.2.1 Subdivision of NREM

- **Stage 1** is often referred to as a transition phase between awake and asleep. This stage occurs mostly in the beginning of sleep. People aroused from this stage often believe that they have been fully awake.

- **Stage 2** is considered to be a more stable sleep stage. No eye movement occurs, and dreaming is very rare. The person sleeping is quite easily awakened.

- **Stage 3** - previously divided into stages 3 and 4 - is considered deep sleep, or slow-wave sleep (SWS). Dreaming is more common in this stage than in other stages of NREM sleep, although not as common as in REM sleep. The content of SWS dreams tends to be disconnected, less vivid, and less memorable than those that occur during REM sleep. This is also the stage during which parasomnias[1] most commonly occur.

NREM separation into 3 or 4 substages

NREM sleep was divided into four substages in the Rechtschaffen and Kales standardization of 1968. This has been reduced to three substages in the 2007 update by The American Academy of Sleep Medicine (AASM)[24]. The articles I have studied regarding sleep movement (section 2.5.2 and 2.5.3) are written before 2007, and will therefore refer to 4 substages.

2.3 Sleep Cycles

During the night, humans cycle through these stages approximately four or five times, with a duration of 90 to 110 minutes each. A typical sleep cycle begins with light NREM sleep, progresses through the stages to deep sleep, and then back again, followed by a very short period of REM, which is generally when dreams occur.

Figure 2.1 shows an example of normal sleep cycles during the night. The graph shows that most of the deep sleep (stage 3 and 4) happens early in the sleeping period, and that the duration of REM sleep increases later in the night. People who do not go through the sleeping stages properly get stuck in NREM sleep, and because muscles are not paralyzed a person may be able to sleepwalk.

[1]Parasomnias refer to all the abnormal things that can happen to people while they sleep, apart from sleep apnea. Some examples are sleep-related eating disorder, sleepwalking, nightmares, sleep paralysis, REM sleep behavior disorder, and sleep aggression. Parasomnias can have negative effects on people during the daytime, including sleepiness[20].

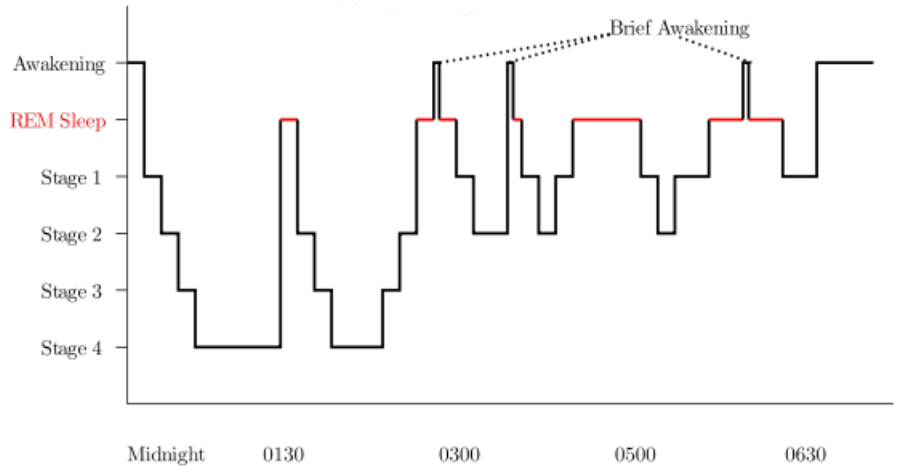


Figure 2.1: Graph showing sleep cycles from midnight to 6.30 am. Stage 1-4 is NREM[25].

2.4 Sleep monitoring

Sleep monitoring is usually done to diagnose sleep disorders, including narcolepsy and sleep apnea. The sleep stages are monitored to determine the pattern and duration. Generally there are two ways to detect sleep quality. The first is directly attaching sensors to the human body. The second is recording of movements during sleep.

2.4.1 Polysomnography

By measuring electrical signals produced by brain and muscle activity, doctors can evaluate the quality and quantity of sleep. The gold standard of sleep monitoring today is polysomnography (PSG), which involves many input channels such as brain activity, eye movement, muscle activity and heart rhythm during sleep. This procedure is very invasive, it is usually performed in a sleep lab, and it typically requires at least 22 wires attached to the patient[22]. Figure 2.2 shows polysomnography wires on a patient. It goes without saying that these wires will disturb the sleep quality.

Figure 2.2: Image of a polysomnography wires on a patient[23]

2.4.2 Alternatives to polysomnography

Invasive lab-based monitoring may be replaced, at least partly, by actigraphy as a basic method to measure sleep quality. Actigrapy (accelerometric movement measurement) is a non-invasive method of monitoring human activity cycles. A small actigraph unit is worn on the wrist to measure large motor activity. The movements are continually recorded, and the data can later be analyzed on a computer. Examples of consumer electronics relying on frequency of wrist movement over time or similar information are Fitbit or NikePlus. While these devices are popular tools to monitor sports and other everyday motion during daytime, there is a common phenomenon to have reservations against using them at night since they impair the sleep quality[14]. The focus of this work will be on using consumer electronics that do not have immediate physical contact with the subjects.

2.4.3 Sleep as Android

Sleep as Android[26] uses actigraphy to recognize the sleep phases. The method is not as precise as PSG, however it still provides comparable results. The biggest advantage with actigrapy is the simple setup which makes it easy to use at home.

Modern Android phones have a built-in accelerometer sensor which is very sensitive. When placed on the bed, the app receives a record of the movement over night. In deep sleep the muscular movements are suppressed, so in this phase the sleep graph gets nearly flat. In contrast, during light sleep is characterized by a tendency to turn around which exhibits as significant peaks in the sleep graph.

The body movements are directly related to the current sleep phase. In general, the more movement the lighter the sleep. The sensors on current Android smart phones are so sensitive, that even when the phone is placed on the mattress near your body Sleep as Android is able to track significant differences in the sleep patterns. Measuring the sleep cycles allows Sleep as Android to do two important things:

- Finding the optimal wakeup time for the alarm clock
- Calculating light and deep sleep

2.5 Articles about sleep surveillance

Sleep research and surveillance is not a new invention. Research and surveillance of sleep started in the beginning of the 20th century (see timeline in section 2.1), and the way we sleep have not changed since then. For this reason the basic articles regarding sleep are relatively old. Newer articles about sleep is generally focused on new methods to observe sleep by using more advanced technical equipment.

The first article in section 2.5.2 describes basic movements during sleep, so even though it is from 1957, it is still relevant. Among others they found a connection between body movements and EEG changes. EEG is explained in section 2.5.1. Section 2.5.3 describes an article from 1982, when they started using video as a means of monitoring sleep. The video equipment has definitely improved since then, but the survey got good results. In spite of the dramatic changes that have taken place in recording and storing techniques, sleep staging has undergone surprisingly few changes.

2.5.1 Electroencephalography

Electroencephalography – or EEG – is a monitoring method to record electrical activity in the brain. EEG measures voltage fluctuations resulting from ionic current within the neurons of the brain. In clinical contexts, EEG refers to the recording of the brain's spontaneous electrical activity over a period of time, as recorded from multiple electrodes placed on the scalp. EEG is used to diagnose epilepsy, sleep disorders, coma, and brain death[19].

2.5.2 Sleep Movement Measured with EEG

The department of Physiology at the University in Chicago did a survey in 1957 to find a connection between EEG patterns and its connection to dream-

ing. In addition to eye motility, they observed body movement, because of its possible relation to dreaming and its close association with EEG changes during sleep.

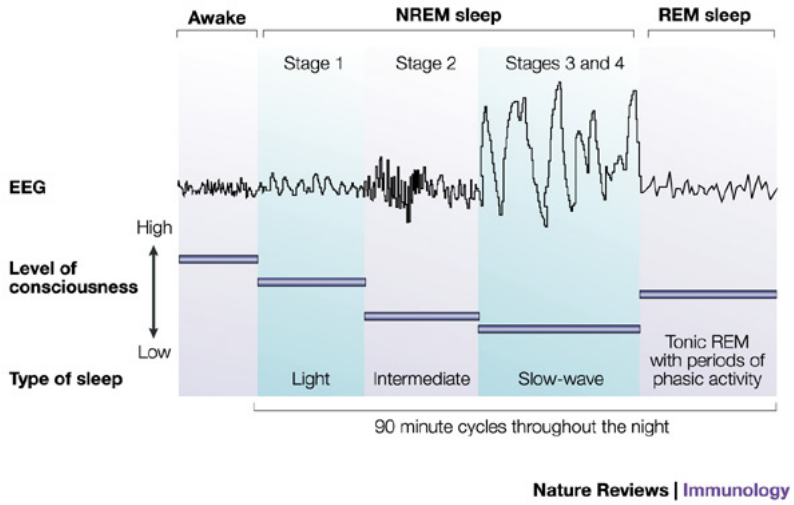


Figure 2.3: EEG patterns for different sleep stages[4]

The survey describes two different kinds of body movement: major and minor body movement. Major movement is when the whole body changes position, while minor movement is stirring and limb movement. Body immobility generally begins in descending stage 2, and ends in stage 3 or 4, just prior to the ascending leap to REM. When major movements occur in the deeper stages, the movements were always accompanied by an upward EEG change, and this often represented the upward swing of a sleep cycle. If the subject was in stage 2, a major movement would usually result in a fleeting appearance of stage 1 or waking EEG, while minor movements caused little apparent change.

In addition to finding a connection between body movements and EEG changes, they also found that REM periods often started immediately after a series of body movements, that the body was relatively motionless during REM, and that movement often reoccurred at the end of REM. They concluded that EEG, eye movement and body movement have cyclic variations throughout the night, and that the peaks of eye and body movements coincide with the lightest phase (stage 1) of the EEG cycles[7].

2.5.3 Sleep Movement Measured with EEG and Video

Another survey from 1982 explored the connection between brain state and body position with two assumptions:

- Major body movements usually occur right before and after REM sleep.
- The longest periods of immobility are associated with deep NREM sleep.

In addition to EEG, they used video surveillance together with the illumination provided by a night-light to monitor four subjects, in hopes of determining the relationship between movements and sleep cycle phase.

They performed an independent frame-by-frame video analysis with a sampling rate of one frame per minute, and used the EEG recordings as a reference. Various degrees of motion were detected, ranging from limb twitches to major posture shifts. A major posture shift was defined as at least a 45 degrees rotation of the body, or more than three limbs moving. Immobility was defined from absolute no motion to maximum two limbs changing place between frames. On average they counted 12.1 major position shifts during the first 6 hours of sleep.

In the comparison of the observed posture shifts in the video frames versus the polygraph, they saw that when there was no motion in the video, the polygraph showed a descending NREM phase of the first sleep cycle. The last movement observed in the video always happened before the EEG showed the first sleep spindles. A spindle is a burst in the brain activity that occurs during stage 2 sleep. In short, this means that they could measure the transition between stage 1 (almost awake) to stage 2 (stable light sleep) from the movement alone. The timing of the major body movement was connected with specific stages of sleep. Most of the movement happened at the end of REM sleep, or at the end of descending NREM sleep. Put differently; the movements happened at the same time as the transition between NREM and REM[1].

This means that a sufficiently detailed video record of nightly movements can document sleep stage changes without using invasive monitoring devices such as the polysomnography electrodes that they place on the skin.

2.6 Digital images

A digital image can be expressed as a 2-dimensional matrix, where each element in the matrix is a sample of the original image. This sample is called a pixel, or picture element, and more pixels gives higher accuracy. The pixels

are arranged in rows and columns, and the number of columns by the number of rows is called the resolution of the image. Images from the Kinect has a resolution of 640x480, which means there are 640 columns and 480 rows of pixels. In total there are 307 200 pixels in each image from the Kinect. The Kinect has a color camera which provides color images, and a depth camera which gives grey scale images. Color images consist of pixels with typically three intensities, such as red, green and blue (RGB). Grey images consist of pixels with just a single value representing the shade of gray between the two extremes black and white.

2.6.1 Storing images

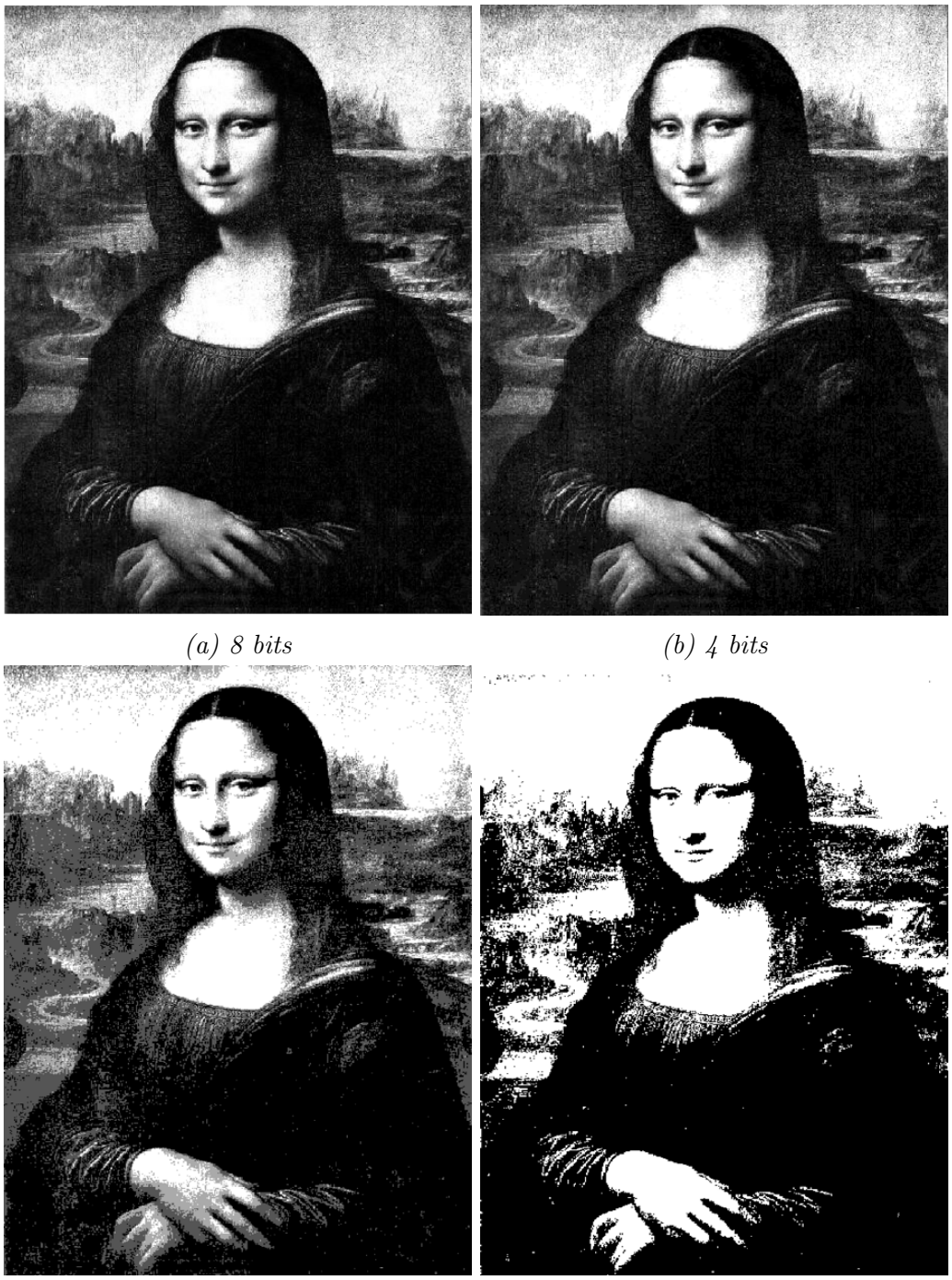
Analog motives consists of continuous values, so when an image is stored digitally, the values have to be quantized[2] to a certain amount of levels. Each pixel in the image is stored using n bits, which means that every pixel value can be an integer ranging from 0 to $2^n - 1$. It is normal to use 8 bit for grey images, and 3*8 bit for color images (8 bits per color channel). The human eye is capable of detecting around 10 million unique colours, but it is only able to detect approximately 30 shades of grey, depending on the lighting. Figure 2.4 shows the famous Mona Lisa painting as a grey image with various amount of bits.

2.6.2 Detecting humans in images

In the past few years, there has been much research in the area of human detection. Much research is based on detection in color images. Common methods involve statistical training based on local features, e.g. gradient-based features such as HOG[6], and some involve extracting *interest points* in the image, such as scale-invariant feature transform (SIFT)[17]. These methods can provide highly accurate human detection.

However, color image based methods have difficulties finding the human shapes when the background is complex, or the human is standing in a particular challenging pose. Using depth camera instead of color camera is more robust to inconsistent colors and changing light because the objects must occupy an integrated region in space[29].

[2]Quantization is the process of transforming a continuous sample to a discrete sample.



(a) 8 bits *(b) 4 bits*

(c) 2 bits *(d) 1 bit*

Figure 2.4: Mona Lisa with varying amount of bits

2.7 Classification

In machine learning and statistics, classification is identifying to which set of categories a new observation belongs, based on a training set of data containing observations whose category membership is known.

In the terminology of machine learning[2], classification is considered supervised learning, where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of characteristic similarity or distance.

This section contains a short description of Support Vector Machines, a classification method used in chapter 3.

2.7.1 Support Vector Machines

Support vector machines (SVMs) are supervised learning models that analyze data and recognize patterns. Given a set of training samples, each labeled with one of two categories, an SVM training algorithm builds a model that assigns new samples into one category or the other. This model is a representation of the samples as points in space, mapped so that the samples from the separate categories are divided by a clear gap that is as wide as possible. New samples are then mapped into that same space and predicted to belong to a category based on which side of the gap they belong to [28].

Figure 2.5 shows a maximum margin hyperplane (a line in the 2D case) that separates the samples in two classes. The hyperplane can be written as the set of points x satisfying $\mathbf{w}\cdot\mathbf{x}-b = 0$, where $\cdot$ denotes the dot product and $\mathbf{w}$ the normal vector to the hyperplane. The parameter $\frac{b}{\|\mathbf{w}\|}$ determines the offset of the hyperplane from the origin along the normal vector $\mathbf{w}$. When the training data are linearly separable, two hyperplanes can be selected in such a way that they separate the data and there are no points between them, and then maximize the distance between the hyperplanes. The region bounded by them is called "the margin". These hyperplanes can be described by the equations $\mathbf{w}\cdot\mathbf{x} - b = 1$, and $\mathbf{w}\cdot\mathbf{x} - b = -1$.

The samples lying on the margin are called the support vectors. In figure 2.5 there are no samples lying inside the margin, but this is not always the case. Sometimes it is not possible to find a hyperplane that can split the classes. In this case a *soft margin* can be applied to find a hyperplane that splits the samples as cleanly as possible, while still maximizing the distance to the nearest cleanly split sample. This method introduces slack variables in the equation which means that a certain amount of samples are allowed inside the margin.

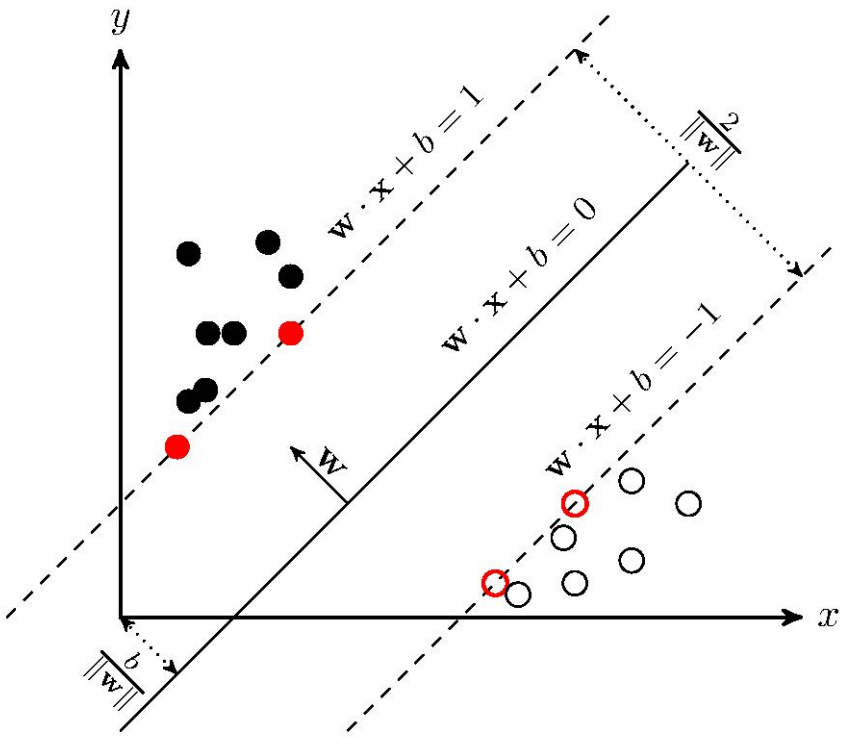


Figure 2.5: SVM trained with samples from two classes[28]

2.8 Microsoft Kinect

The Microsoft Kinect was originally developed for the Xbox video game console. It is a motion sensing input device that enables users to interact with their console through speech and gestures, which means they don't need an external game controller. Human-computer interaction has always been an active research field in computer vision, but it is difficult to achieve with normal color cameras. By using the third dimension (depth), Kinect has made the task much easier.

Because of its low cost and wide availability, Kinect has extended far beyond the gaming industry to computer science, electronic engineering and robotics. In the first three months after the launch in November 2010, Kinect sold 10 million devices, thus setting a new Guinness World Record for the fastest-selling consumer electronics device[30].

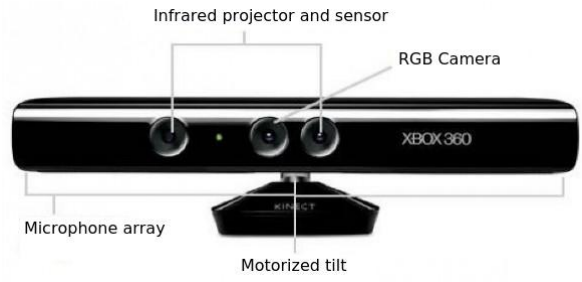


Figure 2.6: The Kinect sensor

2.8.1 Depth sensing technology

Figure 2.6 shows the components of the Kinect. It consists of a color (RGB) camera, an infrared (IR) projector and sensor, multiple microphones, and a motorized tilt that enables you to adjust the camera up or down 27°in order to find the best possible view. This thesis focuses on the IR projector and camera which combined gives depth vision.

Structured Light Principle

The depth-sensing technology in the first version of Kinect (v1) is licenced from the Israeli 3D sensing company PrimeSense (bought by Apple in 2013), so the complete technical description is not available to the public, but the technology is based on structured light. To determine the depth, PrimeSense uses something they call "light coding". Light coding is structured light emitted from the IR projector, and received by the sensor[30].

The Kinect's infrared projector is an IR laser that moves through a diffraction grating, a kind of super prism, which splits the light into several beams travelling in different directions. This gives us a set of infrared dots in a pseudo random pattern. The set of dots have locally different neighborhoods since the pattern is random. The original dot pattern is hardcoded into the chip logic, and the relative geometry between the IR projector and IR camera is known, which means a dot observed in an image can be matched with a dot in the projector pattern.

The position of the object can be reconstructed using triangulation. Triangulation is simple geometry where you can determine the location of a point (or in this case a unique small neighborhood) by measuring angles to it from known points at either end of a fixed baseline. The downside to the structured light method is that the camera only works inside because sunlight will wash of the light pattern, and multiple Kinects can confuse each other[9].

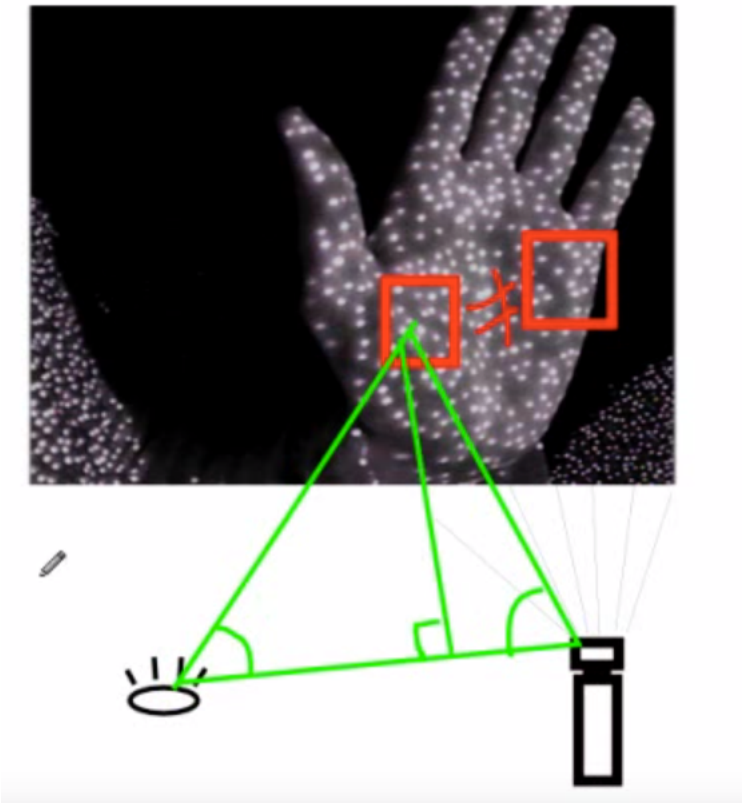


Figure 2.7: Triangulation with Kinect[9]

The matching gives us a depth map encoded with gray values; the darker the pixel, the closer it is to the sensor. If the object is too close, too distant or too small to create a unique neighborhood, the pixels are set to zero, and will appear completely black[30].

Time-of-flight

Another method to measure depth is time-of-flight (TOF). This method is used in the second version (v2) of Kinect. TOF measure the time delay from the light leaves the emitter until it returns to the sensor. The distance can be calculated from this simple formula:

$$D = \frac{c * \Delta t}{2} \tag{2.1}$$

where c is the speed of light, t is the time, and you have to divide the answer by 2 because the light travels twice the distance; from the emitter to the scene, and back again to the sensor[10]. The CMOS sensor in Kinect v1 is not capable of extracting the time of return from the modulated light.

2.9 Depth space of Kinect

The depth sensor captures gray scale images of everything visible in the field of view or the sensor. Each pixel in the image contains the Cartesian distance in millimeters from the camera plane to the nearest object at that particular `(x,y)` coordinate, as shown in figure 2.8.

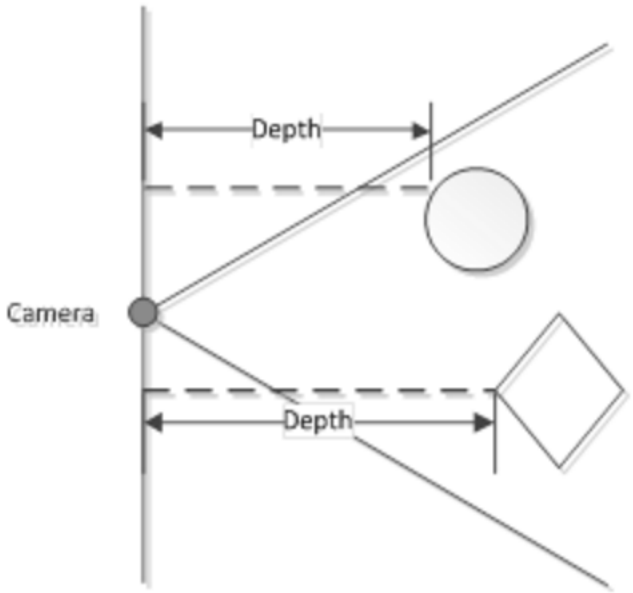


Figure 2.8: Depth stream values[11]

There are three values that indicate the depth that can not be reliably measured at a location; too near, too far and unknown. The two first means an object was detected, but was either too near or too far for the sensor to get a reliable measure. The unknown value means no object was detected. The default range of the Kinect sensor is 0.8 to 4.0 meters[11], so if an object is outside this range it will get the value 0.

The Windows edition of the Kinect also has an available "near range" option which gives reliable measures within 0.4m to 3.0m. Figure 2.9 illustrates the ranges.

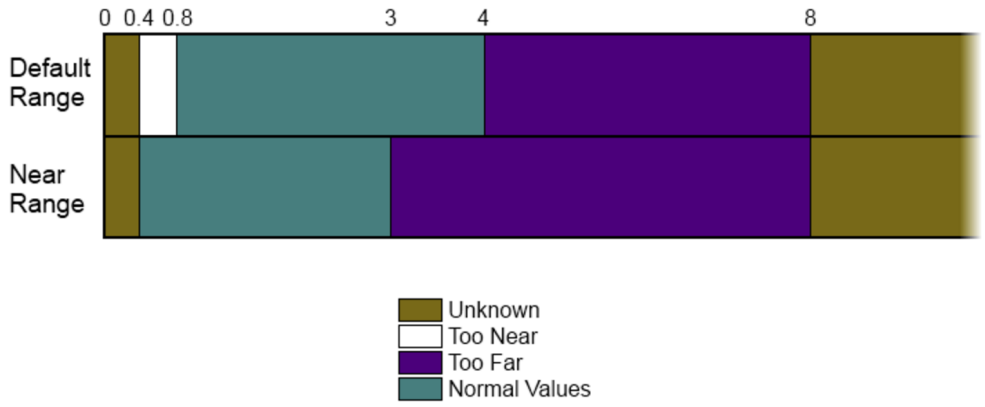


Figure 2.9: Distance from sensor in meters[11]

2.10 Programming With Kinect

In order to process the data from the sensor, there is need of a driver to transfer the data to a computer, and an image analysis library together with a programming language to manipulate the data. This section covers the libraries and programming languages that were considered for the project.

2.10.1 Kinect Drivers

For a computer to be able to read the input from Kinect, it requires a driver for the USB connection between the computer and the Kinect. Microsoft chose by design to not protect this connection, so it is possible to use the sensor for other purposes than Xbox gaming. There are three main drivers[12]:

- **Microsoft:** The first option is the official driver from Microsoft. This was released together with a software development kit for Kinect, but it only works on Windows with Visual Studio with C++ or C#.

- **OpenKinect:** The open source community OpenKinect has enabled use of the Kinect with PCs through the library "libfreenect". This is available for both Windows, Linux and OS X, and it has wrappers for a lot of different programming languages[21].

- **OpenNI:** An industry-led, non-profit organization called OpenNI has provided middleware and a basic driver to access the Kinect. One of the founding members of OpenNI is PrimeSense, the company behind the 3D technology behind Kinect. OpenNI supports the same operative systems as OpenKinect.

- **SimpleOpenNI:** Simple version of OpenNI. Among others does not contain skeleton tracking and gesture recognition.

2.10.2 Image processing

Image processing is the study of any algorithm that takes an image as input, and returns an image as output. The reason we want to process an image can be to detect and enhance a specific feature, to sharpen or correct colors, to improve the image so it is "better" to look at, or image compression. The following two subsections describes two image processing libraries considered for this project.

MATLAB

MATLAB (matrix laboratory) is a proprietary (not open source) programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java and Python[18].

OpenCV

OpenCV (http://opencv.org/) is an open source computer vision library, so it is free to use. OpenCV is written in optimized C/C++. It is designed for computational efficiency, and it focuses on real-time applications. The library contains over 500 functions, including image processing, motion and tracking, pattern recognition, classification and data analysis functions. The library comes with an interface for C, C++, Python and Java, and supports Windows, Linux, Mac OS, iOS and Android[3].

2.10.3 Programming languages

There are multiple programming languages that can be used to process the data from the Kinect. This section provides an overview of the programming languages that were considered for the system.

C++

The first option is to use C++ (pronounced "C-Plus-Plus"). C++ is an object oriented programming language, developed by Bjarne Stroustrup as an extension of the C language. C++ is considered to be an intermediate level language, as it encapsulates both high and low level language features.

The language is compiled before runtime, and has focus on efficiency and performance. C++ is one of the most popular programming language for graphical applications, and works well with OpenCV[5].

Processing

Another option is to use Processing, which is a Java-based sketching language. Processing uses the SimpleOpenNI library which makes it easy to install the required software and start programming with Kinect. Processing has its own image processing library.

Python

The final alternative that I considered is Python. Python is a high level programming language, easy to read, and you can usually express concepts in fewer lines of code than in Java or C++. Python works well with OpenCV, and it supports plotting with the library called Matplotlib.

One disadvantage with Python is that is does not compile before runtime, so for instance running nested for-loops are slow. But by using a Python extension called NumPy, it is possible to use vectorization instead of loops. Rewriting loops to arrays with a fixed size where all the elements are the same datatype, allows aggregations such as summing to be performed by pre-compiled C code.

Chapter 3

Related Works

Sleep analysis and sleep movement analysis is not a new topic, nor is using Kinect for other purposes than Xbox. There are many papers about the topic, and I have researched how other projects have performed sleep movement analysis with a near-infrared camera, how they used the depth information from Kinect for human detection, and how they used Kinect v1 and v2 for sleep monitoring.

This chapter focuses on four projects that used relevant or similar technologies and scientific methods. Section 3.1 describes a video-based alarm clock from 2010. Section 3.2 is dedicated to a method for human detection using Kinect from 2011. Section 3.3 is a project from 2014 about sleep monitoring using the first version of Kinect, and section 3.4 describes a sleep monitoring system from 2015 which uses the second version of Kinect. Finally, this chapter ends by comparing these projects with my project in section 3.5.

3.1 The iWakeUp System

The Journal of the Chinese Institute of Engineers published an article about a video-based alarm clock called iWakeUp in 2010. It is written by Wen-Hung Liao and Jen-Ho Kuo from the Department of Computer Science, and Chien-Ming Yang and Ivy Y. Chen from the Department of Psychology. The authors envisioned a smart living space where a data collection and processing module named iWakeUp was installed in the bedroom to monitor sleep in a non-invasive way. They refer to documentation that says that waking up from light sleep will result in a better mental status than waking up from deep sleep[16].

3.1.1 How iWakeUp works

The iWakeUp system works by setting an alarm time for when the user want to wake up. At a certain time (for instance 30 minutes) prior to this pre-set time, the monitoring system is activated. The system analyzes the incoming video continuously to estimate the sleep status, and wakes the user if the wakeup criteria is met. Figure 3.1 shows the overall setup for the system.

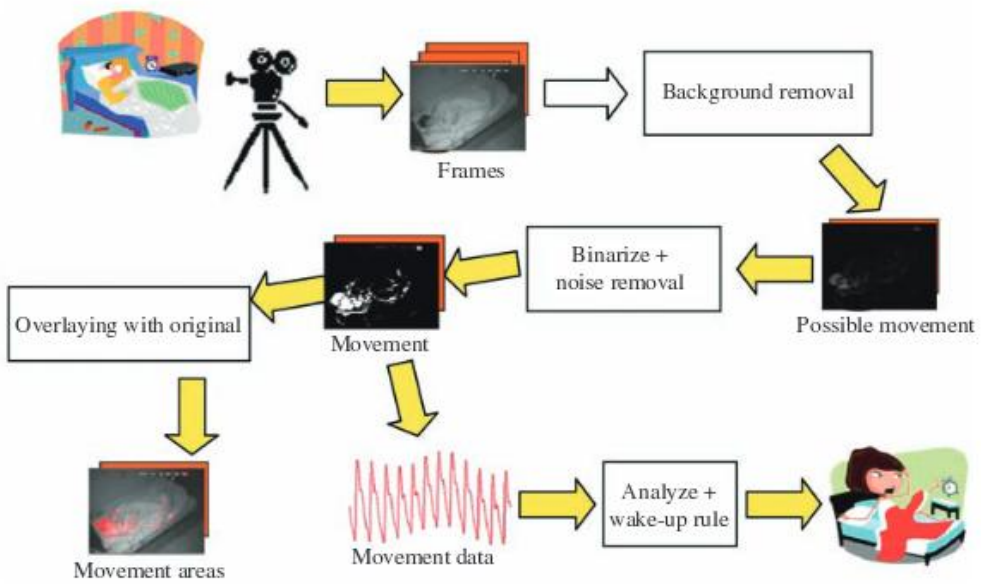


Figure 3.1: Architecture for iWakeUp

3.1.2 Background modeling and noise removal

If a robust background model is available, then it is easy to detect the movements of the human subject by using frame differencing. The simplest way to model the background is by using frame averaging, but this method was not reliable for iWakeUp because they used an near-infrared camera. The images from this camera were very noisy, and also sensitive to the bedroom lighting changing gradually at dawn.

Instead of frame averaging, they used a method where they classified the neighboring pixels in a region into three sets which was greater than, less than or approximately equal to the previous pixel value. This means that a slight change in a pixel's intensity value did not result in a change in the corresponding representation[8].

The background model only dealt with noise at the individual pixel level. All the motion pixels they found after removing the background noise, could therefore not be interpreted as movement of the subject. There were still some motion areas caused by noisy pixels. This was resolved using morphological filters to remove small and isolated blocks. There was also motion outside the region of interest, which was resolved by only using the motion data within the bed area.

3.1.3 Deriving wake-up rules

The outcome of the noise removal was a binary image that contained the position of motion pixels. The number of motion pixels was called *the global motion amount*. It is possible to compute the history of motion, and using this together with the associated motion gradient to acquire the direction of global movement. They found two types of motion pattern that indicated wakefulness: Long term minor movements, and short term major movement. These two types of motion resulted in two wakeup rules:

- The global motion exceeds M_L for at least T_L seconds

- The sum of the motion exceeds M_S in T_S seconds

Determining these parameters was done by using the support vector machine (SVM) model which is a supervised learning model used for data analysis and pattern recognition[1]. The time interval for motion computation was decided to be 1 second. At each time interval, they collected the amount of motion of the previous and following 7 seconds to create a 15-dimensional feature vector. This vector became the input to an SVM to perform classification of awake/asleep status.

On average their system registered three "awake" cases in the 30 minutes period before wakeup time, which should be enough for their system to function as a reliable alarm clock.

3.2 Human detection using depth information by Kinect

Another relevant project is the article "Human detection using depth information by Kinect" written by Lu Xia, Chia-Chih Chen, J. K. Aggarwal from the Department of Electrical and Computer at the University of Texas[29].

[1]More information about SVM is found in section 2.7.1 in the Background chapter

This article describes a method for detecting humans by using depth information from the Kinect. The method is hereinafter referred to as Xia's method[2].

The first step of Xia's method uses 2D chamfer distance matching to locate regions with possible heads in the image. These regions are then examined using a 3D head model which exploits the relational depth information for verification. They also have a region growing algorithm to detect the entire body belonging to the located head, and extracting the body contour.

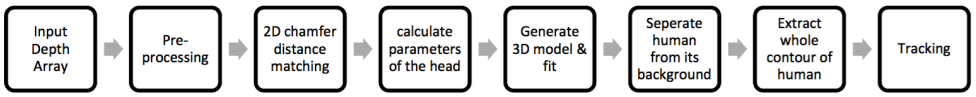


Figure 3.2: Overview of Xia's method[29]

3.2.1 2D Chamfer Distance Matching

2D chamfer distance matching is a scale invariant method that gives a rough detection result with few false negatives[3]. The method uses Canny edge detector to find all the edges in the image: If the amount of pixels in an edge is below a certain threshold, the edge is eliminated.

Based on the edge image, they calculated a distance map where each pixel contained the distance to the closest edge. A binary head template was positioned around on the distance map, and a match was found when the sum of the pixel values inside the template was below a threshold. The process was repeated with the head template in different scales and rotations.

3.2.2 3D model fitting

The detected regions from 2D chamfer distance matching had to be verified to see if the region actually was a head. Based on the depth of the match, it could be calculate at roughly which height the head should be. Acceptable location of the head was based on a regression result for the depth of the head and its height which Xia presented in their article.

If the head had a radius within the range $R = 1.33 * height/2$, then it was verified as a real match. The actual radius of the head was already

[2]Xia's method is not the official name, but the article did not give the method a name.

[3]A false negative is a result that is registered as negative, but really should have been positive.

calculated in the distance map; the pixel value at the center of the head contour contained the distance to its nearest edge. A real head looks different from different angles. In order to get a model which was invariant to different views, they used a hemisphere as the 3D model for the head.

3.2.3 Extract contours

Based on the detected head in the last section, they wanted to extract the overall contour of the person, so they could track hands and feet and recognize the motion. In a depth array, the depth values of the person's feet is the same as the local ground, and when the person touches something, that object will also be at the same depth.

To distinguish between feet and ground, Xia took advantage of the fact that the feet generally appear upright regardless of posture. They used a region growing algorithm to extract the whole body contour from the depth array, assuming that the depth values on the surface of a human were continuous and varied only within a specific range.

3.2.4 Tracking

Tracking objects in depth images was based on two assumptions:

- the movements of the object

- that the neighboring frames should change smoothly

Tracking in this article started with finding the *center* of an detected blob (contour of a person). Then the *coordinates* in the depth array of this center was found. The speed of the blob was calculated from the coordinates of the center in the neighboring frames. The following energy score (E) was used to label the matches so they would get the smallest energy available:

$$E = (c - c_0)^2 + (v - v_0)^2 \tag{3.1}$$

The coordinates of the person in the current frame is represented by c, and c_0 is the coordinates in the last frame. The speed in the current frame is represented by v, and v_0 is the speed in the last frame.

3.2.5 Results from Xia's method

Xia's method was evaluated using a sequence of depth images from the Kinect in an indoor environment with at most two people in the images. Figure 3.3

shows examples from the detection result. The detection worked well in most cases; There were no false positives[4], and only a few false negatives. Advantages with Xia's method is that it easily adjusts to new datasets, and that the 2D chamfer matching largely reduces computational costs. One disadvantage with the method is that the tracking is dependant on accurate head detection.

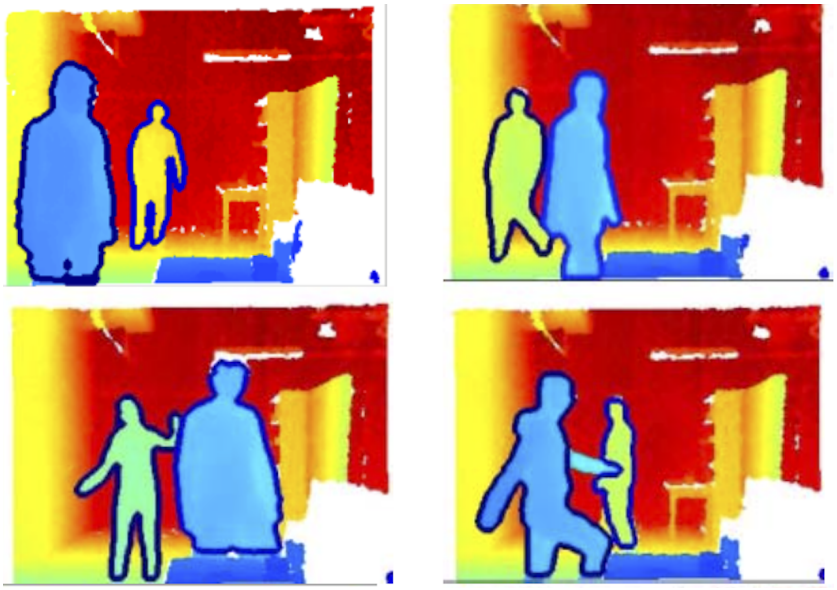


Figure 3.3: Results from Xia's method

3.3 Sleep Monitoring with Kinect V1

The institute for Computer Science in Germany published an article called "Sleep Detection Using a Depth Camera" in 2014, written by Krüger et al.[14]. They presented a way to assess the quality of sleep within a non-laboratory environment. They monitored their patients with a Kinect v1 device, and compared their results with the polysomnography-based gold standard of sleep analysis.

3.3.1 Kinect-based setup

The project mounted the Kinect to a panel in the ceiling, in such a way that the bed and the surrounding area was within the sensors visible range. There

[4]A false positive is a result that is registered as positive, but really should have been negative.

was much noise at the edges of the bed and other furniture. The floor was also a source of noise because of the high distance from the Kinect to the floor. In order to find the relevant motion – as opposed to noise – in the data, two measures of noise was computed:

- The first measure was a visual motion summary – an image – where they averaged the sum of the pixel differences and divided with the amount of frames. Equation (3.2) gives the average spatial distribution D:

$$D(p_{i,j}) = \frac{1}{T}\sum_{i=1}^{T-1} |p_{i,j}(t) - p_{i,j}(t+1)| \tag{3.2}$$

- The second measure was the average change N per frame, in the form of a plot. This is an average over the pixel differences divided by the amount of pixels for each frame. Equation (3.3) gives an overview in the temporal (time) domain:

$$N(t) = \frac{1}{nm}\sum_{i=1}^{n}\sum_{j=1}^{m} |p_{i,j}(t) - p_{i,j}(t+1)| \tag{3.3}$$

These two measures where used on a 5 minutes test sequence with no motion at all. In order to reduce noise, they had two strategies: First, they smoothed the data by using a simple temporal box filter on the time series of each pixel. The size of the filter window was 3 frames. Secondly, they only considered the relevant part of the images by cropping the image to only containing the bed itself. For all experiments described in the article they used filtered and cropped images.

3.3.2 Kripke's algorithm and Krüger's adaption of it

Krüger's experiment was inspired by an algorithm presented in an article regarding sleep stage scoring using actigraph readings written by Kripke et al[13]. Kripke measured actigraph activity once every 30 seconds, multiplied these results with different weights, and achieved a final score $x_i \in 0, 1$ given at each time $t = j$, where $x_j = 0$ meant inactive and $x_j = 1$ meant active.

Krüger modified the original method by using different weights for the input signal:

$$H(t) = \sum_{i=t_s}^{t_e} h(i) * N(t-i) \tag{3.4}$$

The input signal from the Kinect in the temporal domain was multiplied with the parameters in figure 3.4. $H \geq \alpha$ meant the subject was awake. The paper did not provide the value of α.

Kripke													
b_{-10}	b_{-9}	b_{-8}	b_{-7}	b_{-6}	b_{-5}	b_{-4}	b_{-3}	b_{-2}	b_{-1}	b_0	b_1	b_2	b_3
0.0064	0.0074	0.0112	0.0112	0.0118	0.0118	0.0128	0.0188	0.0280	0.0664	0.0300	0.0112	0.0100	0.0000
Gaussian													
h_{-10}	h_{-9}	h_{-8}	h_{-7}	h_{-6}	h_{-5}	h_{-4}	h_{-3}	h_{-2}	h_{-1}	h_0	h_1	h_2	h_3
0.0044	0.0104	0.0224	0.0440	0.0790	0.1295	0.1942	0.2661	0.3332	0.3814	0.3989	0.1295	0.0044	0.0000

Figure 3.4: Parameters for Kripke and Krüger's algorithms[14]

3.3.3 Results from Krüger

Recordings over two nights resulted in both a spatial motion summary and a temporal summary.

The spatial summary showed that the patient moved his head, hands and feet. And even though the patient used a duvet, the motion was clearly visible.

The temporal summary depicted by figure 3.5 shows the asleep and awake incidents during the second night of experiments in form of a curve illustrating at which times the subject displayed significant activity. The hypnogram marked with ground truth is measured with polysomnography. The rest of the hypnograms are based on equation (3.4), using the parameters from figure 3.4 for Kripke and Gauss, while the constant hypnogram used a constant parameter for all input.

I did not include the hypnograms for the first night of recordings because the article says the measurements were not reliable. The unreliable recordings was caused by a complex clinical sensor that the patient had to wear in order to get the ground truth (PSG) recordings. The sensors caused discomfort which disturbed the natural sleep pattern.

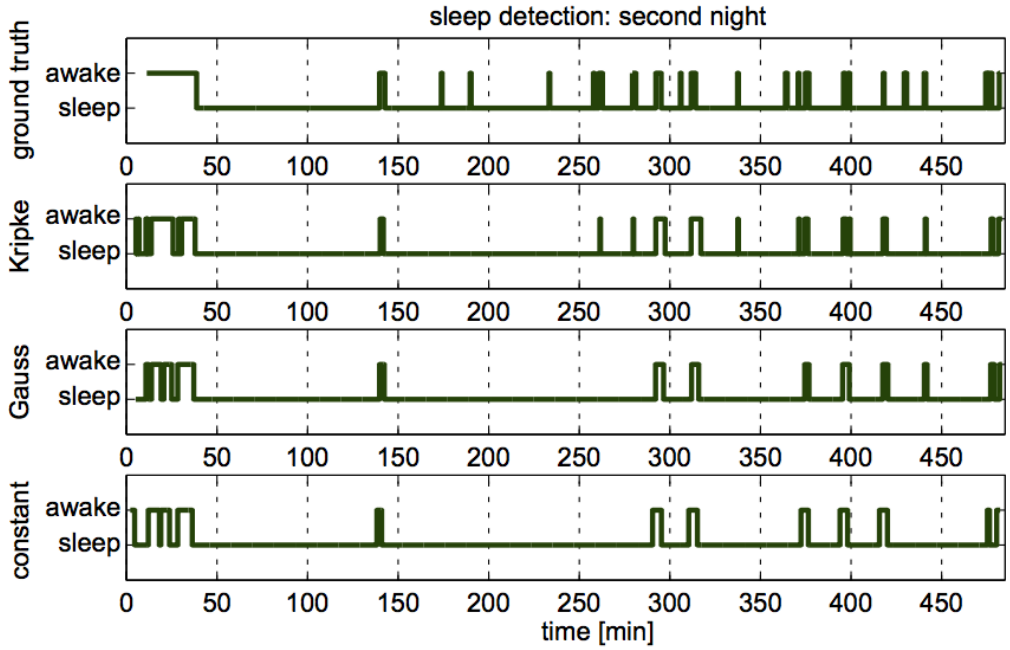


Figure 3.5: Temporal hypnogram for the second night from Krüger's experiments[14]

3.4 Sleep Monitoring with Kinect V2

The International Journal of Distributed Sensor Networks published a research article in 2015 called "Sleep Monitoring System Using Kinect Sensor"[15]. It is written by Jaehoon Lee, Min Hong and Sungyong Ryu from the Republic of Korea. They write about how traditional sleep monitoring requires many devices to be attached to the human body. In their paper they implemented a sleep monitoring system with Kinect v2 which could detect sleep movement and postures during sleep without using any body attached devices.

3.4.1 Sleep monitoring as part of a Smart Home

The sleep monitoring system proposed in the article can be one of the elements in a smart home[5], where it can take care of or improve human life through sleep state monitoring. Since the system does not require any inva-

[5]A smart home is a home equipped with lighting, heating, security, and electronic devices that can be controlled by a time schedule or remotely from any location by phone or computer.

sive sensing devices, the system can provide a natural and comfortable sleep environment while collecting sleep related information.

3.4.2 Sleep State Monitoring

There are two options to detect sleep state:

- **Body attached sensors:** Directly attach sensors to the body, which provides scientifically accurate sleep state information. A disadvantage with this option is that the sensors leads to an uncomfortable sleep environment, and this may negatively affect the measurements.

- **Video:** Recording body movements during sleep is more convenient, and more comfortable for the patient. A disadvantage with video is that it is more difficult to analyse the sleep state from images than with body-attached sensors. In order to monitor the sleep state, it is essential with image processing to extract the body from the background.

3.4.3 Microsoft Kinect v2

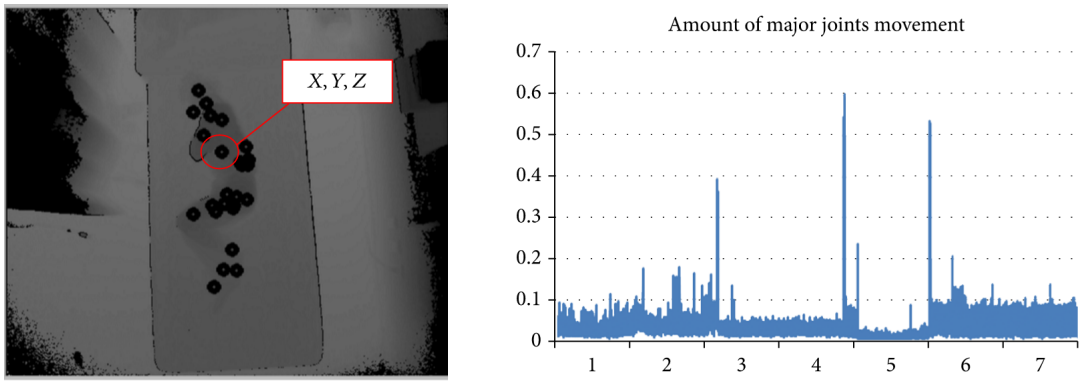
The project used Microsoft Kinect v2 to detect and track the body movements. The second version of Kinect provides the position of 25 joints, the detection range is better than the previous Kinect v1, and the v2 sensor uses the more accurate time-of-flight as opposed to v1's structured light[6]. Together with Kinect v2, they used Visual Studio with C++ and OpenCV to detect and track body movements during sleep.

3.4.4 Sleep Movement Measures

Kinect v2 provides x, y and z position of 25 joints, and the distance of objects to the depth sensor. They stored `(x,y,z)` for 19 critical joints every half second, and figure 3.6a) shows the detected joints from the article. The total sleep movement of 19 major joints is shown in figure 3.6b). They used Euclidian[7] distance between the previous and current position of the joints to obtain the total movement.

[6]The two depth sensing technologies time-of-flight and structured light are described in 2.8.1 in the Background chapter.

[7]The Euclidean distance between points p and q is the length of the line segment connecting them: $\sum_{i=1}^{n}(q_i - p_i)^2$.



(a) *(b)*

Figure 3.6: [15] a) Detected joints for tracking and b) Major joints movement per hour

3.4.5 Sleep Movement Analysis

To test the system, they monitored 20 students for 7 hours where their bodies were not covered with any blankets to accurately detect the motion. They found that the sleep movement value was sharply altered when the body position changed. When the body posture stabilized again, the detected motion was also stable. For each hour during the night they got 7200 measures of the motion from 19 joints – SM – where c denotes the current position, and p denotes the previous position:

$$SM = \sum_{i=1}^{19} \sqrt{(X_{ic} - X_{ip})^2 + (Y_{ic} - Y_{ip})^2 + (Z_{ic} - Z_{ip})^2} \tag{3.5}$$

By adding all the SM values for each hour, they got HMS (Hourly Sleep Movement) values ranging from 0.0 to 5.5, which they standardized to range from 0.0 to 1.0. They judged the sleep quality of the students based on the following HMS values:

- Good (deep sleep): $0 < HMS < 0.25$
- Normal (normal sleep): $0.25 \leq HMS < 0.59$
- Bad (light sleep): $0.59 \leq HMS \leq 1.0$

In the article, they included a table with the amount of movement per hour given in HMS values, and also an overall grade (Good, Normal or Bad) of the sleep quality for each of the students. The article does not say anything more about whether these values were accurate, or if the students in question agreed with the sleep quality grade they got.

3.5 Comparison of the articles

This section contains a short summary of each article, with the key findings that is used further in this thesis.

iWakeUp: The iWakeUp article used a near-infrared camera (not Kinect) to monitor sleep. They removed the noise in the images with morphological filters, and only used the bed area to avoid noise sources outside the ROI. They made wakeup rules where the global motion had to exceed a certain threshold for a certain amount of time for an incident to be considered "awake". At each second they gathered the motion from 7 seconds before and after, to perform asleep/awake-classification on this 15-dimensional feature vector. On average the system registered three "awake" cases in the 30 minutes before wakeup time.

Xia: Xia's method described in "Human detection using depth information by Kinect" proved not so relevant for sleep tracking because the method required people standing in order to track them. Further, the method was based on a accurate head detection, which can be difficult if the head is covered by a duvet. However, some knowledge was still relevant: They mentioned summing pixel values and comparing this with a certain threshold. The article also provided an assumption regarding object tracking: When an object moves, the neighboring frames should change smoothly.

Kinect V1: The most relevant article was about sleep monitoring using Kinect V1. They cropped the images to just contain the bed. The patients were allowed to use duvet, and the kinect recordings still showed a significant amount of movement. They found the pixel value difference between frames, and multiplied this input with certain parameters, in order to make hypnograms from the input with asleep or awake cases. The results from Kinect v1 was compared with results from polysomnograpy, and it turned out that the Kinect recordings obtained similar good results.

Kinect V2: The last article was mostly relevant for further works where a Kinect v2 sensor could be used. They used skeletal tracking, so the patients could not use blankets. They also used the more accurate time-of-flight depth sensing technology. An important point to take further was that they confirmed that when the body position changed, so did the sleep movement output in their system.

Chapter 4

Material and Methods

This chapter describes material and methods used to exploit the depth information from the Kinect. The chapter starts by explaining the choice of programming language and libraries necessary to program with the Kinect in section 4.1. Further, section 4.2 describes the setup of the system. Section 4.3 is dedicated to exploration of the depth information from the Kinect, and section 4.4 focuses on the frame difference on the depth images. Section 4.5 is a speed test, and section 4.6 explores the noise in the images. Preprocessing of the images is covered in section 4.7. Finally, the chapter ends with section 4.8 that covers an implementation of a scheduler used to start the Kinect recording at night, and the code for the actual recording in section 4.9.

4.1 Choosing programming language and libraries

A big decision in this project was choosing driver, image processing library and programming language. There were many options, but after some trial and test runs I ended up with OpenKinect, OpenCV and Python. This section explains the reason behind the choices.

4.1.1 OpenKinect

OpenKinect with the library libfreenect is one of the drivers for Kinect. It can be used to control the motor, accelerometer, led light and audio, and supports both color and depth images.

Advantages

An advantage with libfreenect is that it has support for motor control, which is important to get the Kinect in the correct angle before filming.

Disadvantages

One disadvantage with libfreenect is that, unlike OpenNI, it does not support skeletal tracking (see figure 4.1). Detection of the skeleton joints of the user can give useful results to compare with the results from the depth data. On the other hand, the skeleton feature requires that the subject is not covered by a duvet, which goes against the purpose of the project - uninvasive sleep surveillance.

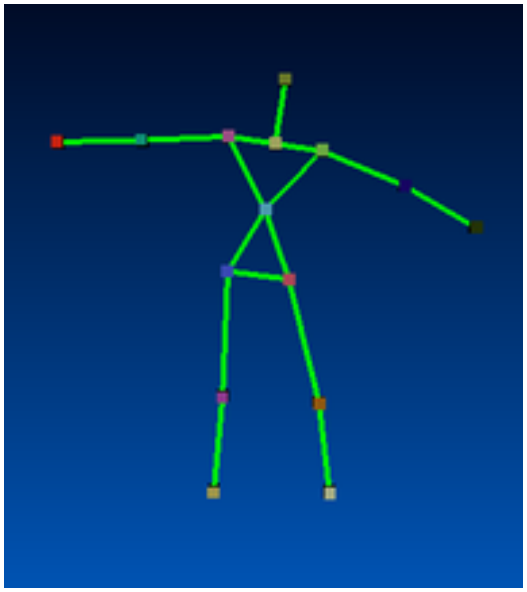


Figure 4.1: Skeleton joints from Kinect skeletal tracking

4.1.2 OpenCV

I first considered using MATLAB since I had previous experience with using MATLAB for image processing and image analysis earlier. MATLAB is an independent language, meaning there is no need for an extra language, but it is possible.

Advantages

Advantages with OpenCV is that it is open source. Another advantage is that it is faster than MATLAB, and it comes with an interface for Python. 1 hour of images with a frequency of 1 image per second produces 3600 images. For a whole night there is a lot of images to process, so speed is important.

4.1.3 Python

The original plan was to use C++ since this was used in some of the articles I had read about Kinect programming. However, C++ made the project a lot harder considering I had little experience with C++. Instead of learning a new language, I tested the Java-based Processing language, since I already know Java.

Processing uses SimpleOpenNI to connect with Kinect, and it provides its own image processing library. In regards to installation it was basically plug-and-play. One huge disadvantage was that the included image library did not have all the necessary functions, and using OpenCV together with Processing did not work. Instead, I tried using Python with OpenCV and libfreenect.

Advantages with Python

Advantages with Python is that I already was familiar with the language, and there is little code redundancy (as opposed to for instance Java). The installation process was more complicated, but there are a lot of tutorials available. The installation of Python with OpenCV and OpenKinect for OS X Mavericks is provided in Appendix A.

4.2 Setup of the Kinect

Regarding the setup of the system, I took the following requirements into account:

- The distance between the bed and the sensor can not be smaller than 0.8 meters, and not larger than 4 meters because of the range of the Kinect.

- There can be no natural sunlight, since sunlight ruins the structured light pattern from the Kinect's IR emitter.

- There can be no obstacles between the camera and the user.

4.2.1 First setup

Finding a good position for the Kinect was challenging. The first attempt was simply placing the Kinect at a table next to the bed. This did not work out since the bed and the person got registered at the same depth. The second attempt was duct taping the Kinect to a book shelf above the bed,

shown in figure 4.2. However, recording over night was not possible with the first setup because it was not safe for the person sleeping. One of the main goals with this project was to avoid intrusive methods to record sleep, and getting hit in the head with the sensor should probably be defined as intrusive. The first setup is used in section 4.4 and 4.7.

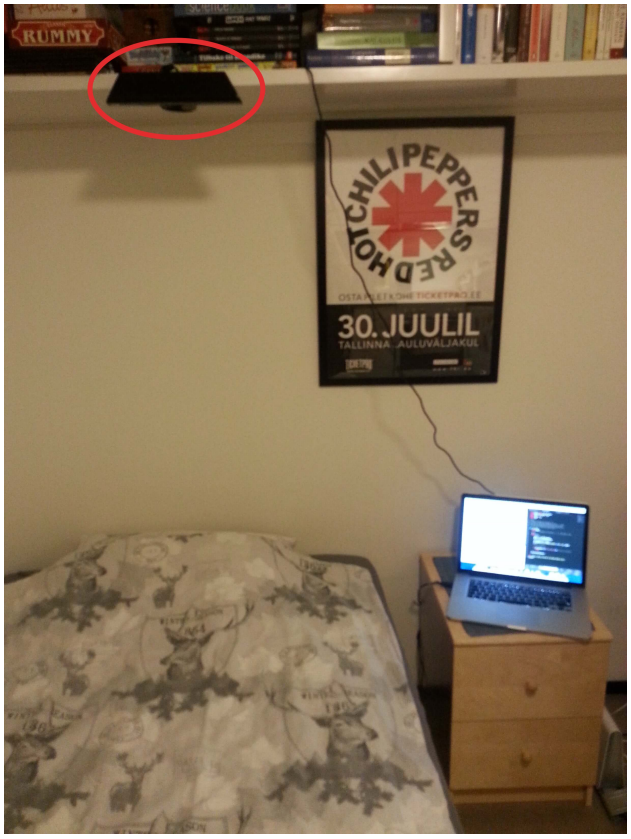


Figure 4.2: Setup 1

4.2.2 Second setup

The second setup worked out much better, see figure 4.3. The Kinect was placed on top of a book shelf with no risk of falling down. The distance from the sensor to the nearest point in bed was 1.5 meters (middle of body), and the furthest was 2.5 meters (position of head). For this setup, the Kinect default range of 0.8m - 4.0m suffices, since the upper body is within this range. The second setup is used in all the sections **except** 4.4 and 4.7.

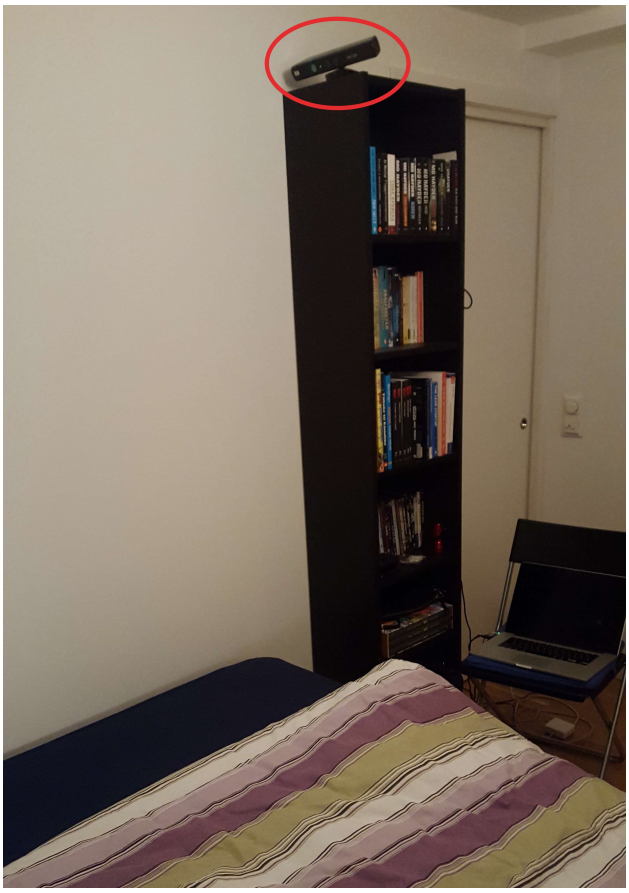


Figure 4.3: Setup 2

4.3 Depth Information from Kinect

The libfreenect library provides depth images from the Kinect with the function *sync_get_depth()*. This function returns raw depth values in millimeters (e.g. 1200 mm). On the other hand, OpenCV shows images where each pixel only has values ranging from 0-255. For the depth information to make sense for OpenCV, I converted the raw values to uint8, which is an eight bit unsigned integer. *Unsigned* means that there are only positive values. One bit can be either 0 or 1, and since there are 8 bits, that gives $2^8 = 256$ different values.

4.3.1 How the pixels are stored

The images from Kinect have a resolution of 640x480 pixels. An image is usually represented as a two-dimensional array, but the pixels from the Kinect are stored in a one-dimensional array. An advantage with a one-dimensional array is that the computations are faster than a two-dimensional array. However, the formula for getting a pixel from a specific position is a bit more advanced. Table 4.1 and 4.2 illustrates the mapping between how the pixels look and how they are stored.

Table 4.1: How the pixels look

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Table 4.2: How the pixels are stored

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

To find the pixel value in position `(x,y)` in the image, the following formula is used, where width is 640 for the Kinect images:

$$index = x + y * width \tag{4.1}$$

4.3.2 Testing the depth

The default mode of the Kinect has a range between 0.8m and 4.0m, with optimal light conditions. Two test cases was done to see how many pixels could be measured by the Kinect.

The first test was done with a white wall approximately 1.5 meters away from the sensor. The result showed that more than 98.5% of the pixels could be measured.

Another test run was done against a window at night. The result of this test showed that only 86.9% of the data was measured. By closing the curtains in front of the window this number was increased to 98.7%. These numbers will of course vary with different tests, but the conclusion is that the amount of unmeasured data is reduced in a closed environment.

4.4 Frame differencing on depth images

Frame differencing is comparing the values in the same pixel position in two consecutive frames/images to get the average change – the motion – per frame:

$$F(t) = \frac{1}{nm} \sum_{i=1}^{n} \sum_{j=1}^{m} |p_{i,j}(t) - p_{i,j}(t+1)| \tag{4.2}$$

By comparing the consecutive frames in an image sequence, I found the number of pixels that changed between frames. Figure 4.4 consist of 8 images with different sleeping positions. The pixel value in position `(x,y)` in image 4.4b was compared to the pixel value at the same position in the previous image 4.4a. If the difference exceeded a certain threshold (called *slack* in table 4.3), the amount of motion pixels is increased by 1.

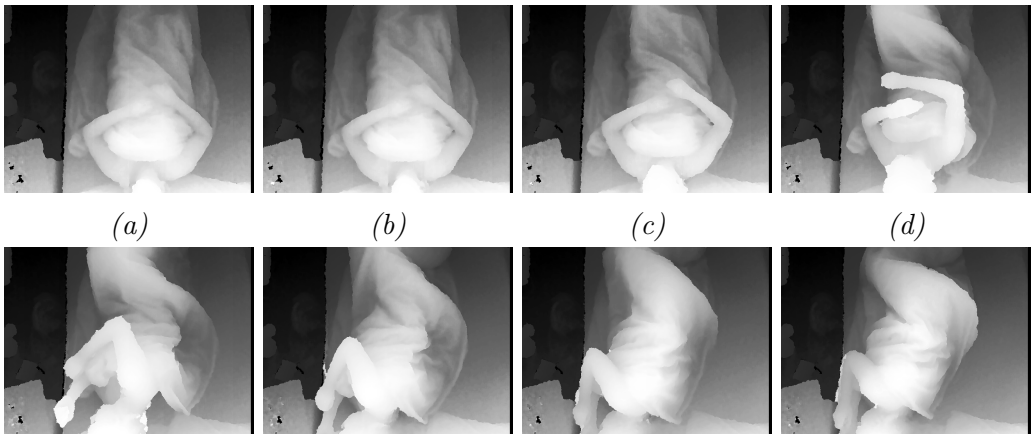
This first test is done with the first setup, described in figure 4.2.1. There is no preprocessing of the data, except that the raw data from the Kinect is reduced to 256 levels.

4.4.1 Testing various slack

Performing frame difference on the images requires a threshold to decide whether the difference is counted as a *motion pixel* or not. Deciding the value of this threshold – or *slack* – is done in this section. This section is also based on the image sequence in figure 4.4.

Table 4.3 shows the percentage of pixels that have moved compared to the previous image, i.e., difference `ab` means image b) (4.4b) compared to image a) (4.4a). This was done for all eight images.

The numbers marked with `bold` are supposed to be bigger than the other numbers, as there are major changes in these images, see images `d)`, `e)` and `f)` compared to their previous images. It is desirable that the images with



(a) *(b)* *(c)* *(d)*

(e) *(f)* *(g)* *(h)*

Figure 4.4: Image sequence with different sleeping positions with setup 1

little movement have the lowest possible percentage change, and that the images with large movement have a high percentage compared to the others.

Percentage of changed pixels with various slack

	0	**10**	**20**	**30**	**40**	**50**	**60**	**70**
ab	49.1%	1.1%	0.5%	0.3%	0.3%	0.2%	0.1%	0.1%
bc	77.0%	18.9%	9.7%	5.7%	3.5%	1.8%	1.0%	0.6%
cd	**83.6%**	**43.9%**	**25.0%**	**16.9%**	**12.5%**	**9.3%**	**6.7%**	**5.2%**
de	**80.1%**	**40.9%**	**21.8%**	**15.7%**	**12.1%**	**10.0%**	**8.4%**	**7.2%**
ef	**85.9%**	**41.0%**	**24.0%**	**16.9%**	**11.7%**	**8.1%**	**6.0%**	**4.3%**
fg	84.2%	32.4%	17.2%	9.3%	5.0%	2.9%	2.1%	1.8%
gh	79.6%	21.6%	11.5%	7.3%	4.4%	2.7%	2.0%	1.7%

Table 4.3: Motion pixels between frames with various slack. The images emphasized in bold text contains major movements, and the images in normal text contains minor or no movement.

Table 4.3 shows motion values with various slack:

- For zero slack it seems that there are more movement for `fg` than for `cd` and `de`, which is incorrect.
- A slack of 20 and above looks usable since the numbers in bold are higher than the others.

- From the table it seems that a slack of 40-60 gives a good result for these images since the percentage is significantly higher for the images with major movements.

4.4.2 Testing slack of 55 on a sequence of 30 images

Plot 4.5 shows a 30 seconds sequence of recording with a slack of 55. The exact values are included in table 4.4. In the table there are two peaks at image #12 and #13 where there is approximately 90°rotation of the upper body and some arm movement, and at #24 and #25 where there is another 90°movement, and motion of an arm and the head.

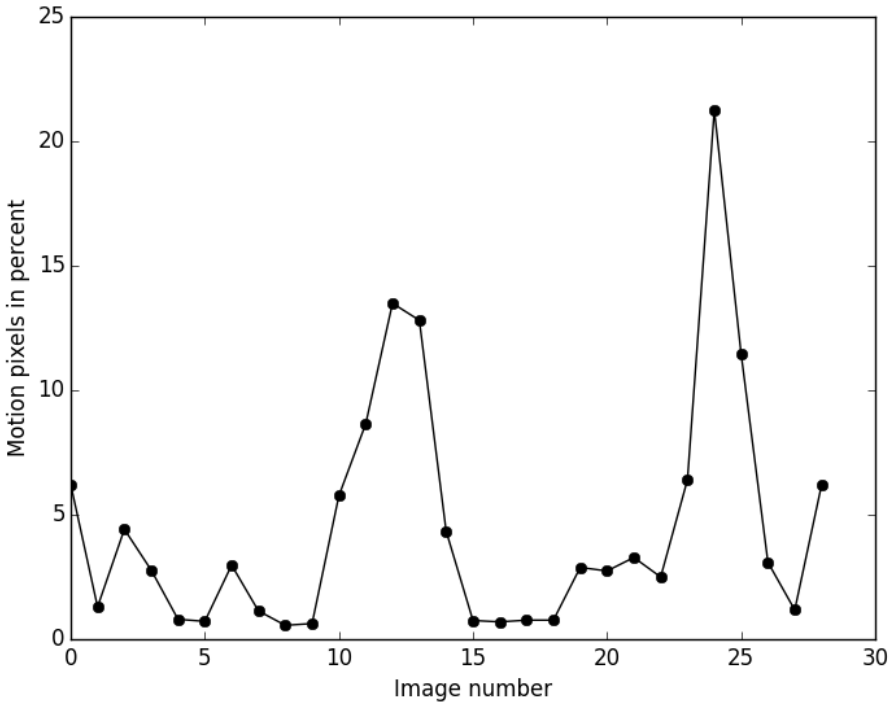


Figure 4.5: Motion pixels with slack=55 for a 30 seconds sequence

Img	Motion	Movement	Img	Motion	Movement
#0	6.21%	Arm	#15	0.75%	
#1	1.30%		#16	0.70%	
#2	4.42%		#17	0.77%	
#3	2.78%	Shoulder	#18	0.76%	
#4	0.80%		#19	2.88%	Hand
#5	0.72%		#20	2.74%	Hand
#6	2.97%	Forearm	#21	3.28%	Hand
#7	1.14%		#22	2.50%	Hand
#8	0.56%		#23	6.41%	Arm
#9	0.63%		**#24**	**21.25%**	**90°rotation**
#10	5.76%	Back	**#25**	**11.46%**	**Arm, head**
#11	8.65%	Arm	#26	3.09%	Forearm
#12	**13.49%**	**90°rotation**	#27	1.18%	
#13	**12.81%**	**Both arms**	#28	6.19%	Head
#14	4.34%	Hand			

Table 4.4: Frame difference of 30 seconds with 1 sec interval

4.5 Performance test of for-loops vs. NumPy

I mentioned in section 2.10.3 that since Python code is not compiled before runtime, regular for-loops are really slow. I wanted to see how much faster using NumPy arrays actually was, so I did a performance test on 1200 images where both programs found the frame difference of the images. The runtime for two nested for-loops was 1 minute and 47.9 seconds for processing all 1200 images. However, by using vectorization, the runtime was only 3.8 seconds. Quite a difference! So obviously NumPy is used for the rest of the project.

4.6 Noise test

This section is dedicated to localize and hopefully decrease the sources of noise in the depth images. Sources of noise comes from edges of the bed and wall. The noise test is based on a sequence of 1200 images with 1 image every second for 20 minutes. All the images depict an empty bed with no

actual motion.

One option to reduce noise is to only use the region of interest (ROI) in the images, which in this case is equivalent to the bed region. Figure 4.6 shows a depth image with the ROI as a black square. The gray and white part of the figure are the walls behind the bed.

Figure 4.6: A depth image with the black square representing the ROI

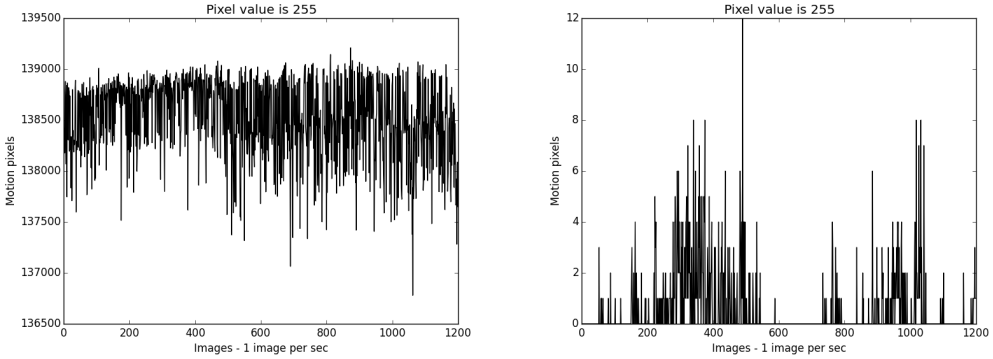
4.6.1 Minimum and maximum values

When the object is too close or too far away from the Kinect, the pixel values is assigned to 0 or 255. Figure 4.7 shows the number of pixel that have 255 as value. Plot `a`) have pixels from the entire image, and the maximum number of pixels is 139 000 which is equivalent to 45% of the total pixels. Plot `b`) contain pixels from a cropped image with only the bed area, and the maximum amount of pixels is 12, which is 0.017% of the total pixels in the cropped image.

Regarding the values set to 0, the cropped plot showed no zeros, and the full image showed maximum 12 zeros, therefore I did not include those two plots.

4.6.2 Motion pixels of noise

Figure 4.8 shows the motion pixels for the same image sequence. The black graph is the full original image, while the blue graph represents the cropped images. Plot 4.8a) shows that there is more actual noise in the full image (black graph), while plot 4.8b) shows that in percent it is actually the cropped images (blue graph) that has most noise.



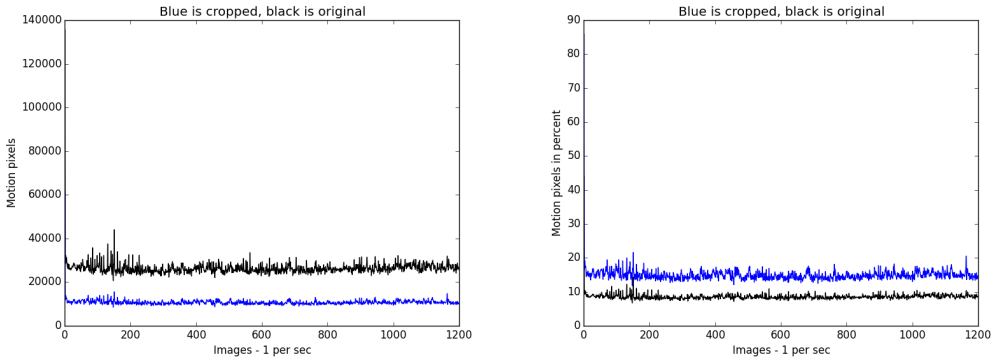
(a) *(b)*

Figure 4.7: Number of pixels with value 255 (max). Plot a) shows values from the entire image, plot b) shows values from a cropped image. Note that both y-axis is in actual value, and not percent.

4.6.3 Test of noise compared to actual motion

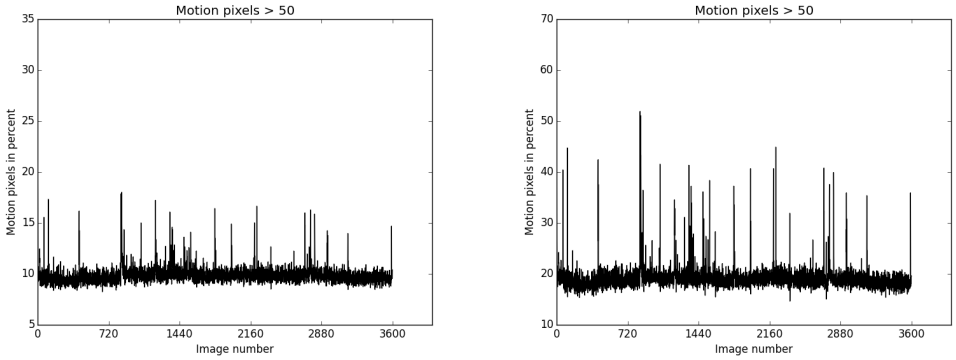
The reason why I wanted to use cropped images was because these images contained only the bed area, and processing this smaller area took approximately a quarter of the time. Section 4.6.1 shows clearly that using the full image is unnecessary, especially considering that almost half the pixel values is 255. However, based on the noise images described in the previous section (4.6.2) it would seem that using the full image is less noisy percentage-wise, so I was in doubt of whether to use cropped images or full images.

For this reason I also had to investigate how the motion plots looked with actual motion compared to the plots with no motion. Figure 4.9 shows two plots where 4.9a is motion pixels in the full image, and 4.9b is motion pixels within the cropped image. The plots are based on a 3600 long image sequence recorded over 5 hours. The noise becomes the minimum values in the plot, and represents no motion. The plot shows that the noise in the full image is about 10 percent, and in the cropped image it is about 20 percent. However, the actual motion pixels from the cropped image has also increased, which makes separating the motion from the noise much easier. This means that I can continue with the cropped images.



(a) *(b)*

Figure 4.8: Motion pixels from the 1200 noise images. Plot a) is the actual values, and b) is in percent relative to the total amount of pixels in the original or cropped image.



(a) *(b)*

Figure 4.9: Motion recorded every 5 seconds for 5 hours. Plot a) is based on the full image, and plot b) is based on cropped images.

4.7 Preprocessing

This section is dedicated to preprocessing methods to improve the images in order to get a better result. One approach to reduce noise in the images is by convolving the image with a low-pass filter. A low-pass filter passes low values, and reduces or removes high values. It is especially good for reducing noise within small neighborhoods, by finding single pixels with a lot different value compared to its neighboring pixels, and reducing the outliers. One disadvantage with low-pass filtering is that it blurs out the edges.

4.7.1 Low-pass filters

The algorithm iterates over all the pixel positions in the original image, for each `(x,y)` position it calculates a new value for `(x,y)` in a new image based on the values inside a size*size window. The actual calculations of this new pixel value depends on the type of kernel used, and how blurred the image becomes is decided by the size of the window.

Mean kernel

The mean kernel emphazises the average of the neighborhood. If the size of the window is 3, the mean kernel has $1/9$ at every space in a 3x3 matrix.

$$Mean = 1/9 * \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gauss kernel

Sometimes it can be preferable to use a kernel which emphasizes other features than the average. One option is a gaussian kernel which emphasizes the center pixel the most, and gives less weight to the neighbors. A Gauss-filter blurs less than a mean filter of the same size. A 3x3 Gauss-filter looks like this:

$$Gauss = 1/16 * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

4.8 Kinect Scheduler

For recordings in middle of the night I used a Python scheduler library called *schedule*. With this library it is possible to set day and time when the program should run. For instance run the program every Monday to Friday at 6AM.

```
import schedule, time, datetime
from kinect_record import Kinect

kinect = Kinect()

def job():
   now = datetime.datetime.now()
   strf = now.strftime("%Y-%m-%d %H:%M")
   print 'Scheduler is running.. ', strf
   kinect.runLoop()

schedule.every().day.at("02:30").do(job)

while True:
    schedule.run_pending()
    time.sleep(1)
```

4.9 Kinect Recorder

This section contains the code for recording with Kinect.

```
#!/usr/bin/env python
import freenect
import os, time, datetime
import cv2

class Kinect:
    def get_depth(self):
        array,_ = freenect.sync_get_depth()
        array = array.astype(np.uint8)
        return array

    def runLoop(self):
        frequency = 1 # frames per second
        terminate = 60*60*5/frequency
```

```
15
16          now = datetime.datetime.now()
17          strf = now.strftime("%m-%d-%H-%M")
18
19          # New directory every time - don't overwrite other images
20          pathWithFilename = os.path.abspath("kinect_record.py")
21          splitted = os.path.split(pathWithFilename)
22          directory = splitted[0] + '/29april/kinect' + strf + '_' +
                str(frequency) + 'sec_'
23
24          j=0
25          while os.path.exists(directory+str(j)):
26              j+=1
27          directory += str(j)
28          os.makedirs(directory)
29
30          print 'Kinect running every', frequency, 'seconds. Folder',
                os.path.split(directory)[1]
31
32          i = 0
33          while True:
34              depth = self.get_depth()
35              cv2.imwrite(directory + '/depth' + str(i) + '.jpg', depth)
36              i += 1
37
38              # Close window on ESC or after X time
39              if cv2.waitKey(5) == 27 or i > terminate:
40                  break
41
42              time.sleep(frequency)
43
44          cv2.destroyAllWindows()
45          print 'Kinect is finished!'
```

Chapter 5

Results and Discussion

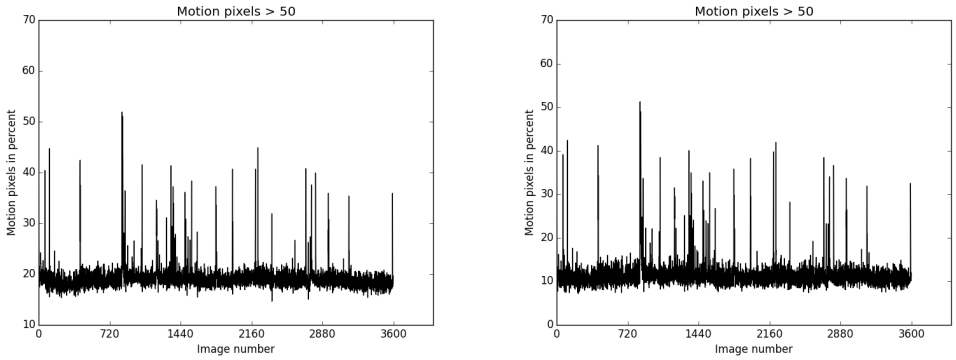
This chapter contains the results from my sleep movement program, and discussion about the results. Section 5.1 discusses the results of the low-pass filtering methods described in the chapter of Material and Methods. Section 5.2 introduces a threshold to separate noise from the actual motion in the plots. Section 5.3 decides a vector size in order to find the largest motion incidents during a recoring. Section 5.4 compared a Kinect recording to a recording done with an Android application called "Sleep as Android". Finally, the chapter ends with chapter 5.5 discussing motion plots based on recordings from entire nights, using the parameters decided in the previous sections of this chapter.

5.1 Result of low-pass filtering

The low-pass filtering described in section 4.7 was performed on the 3600 cropped image sequence from section 4.6.3. The amount of motion pixels for each image was based on a slack of 50, which means that a pixel value change between two frames were only counted if the absolute difference exceeded 50.

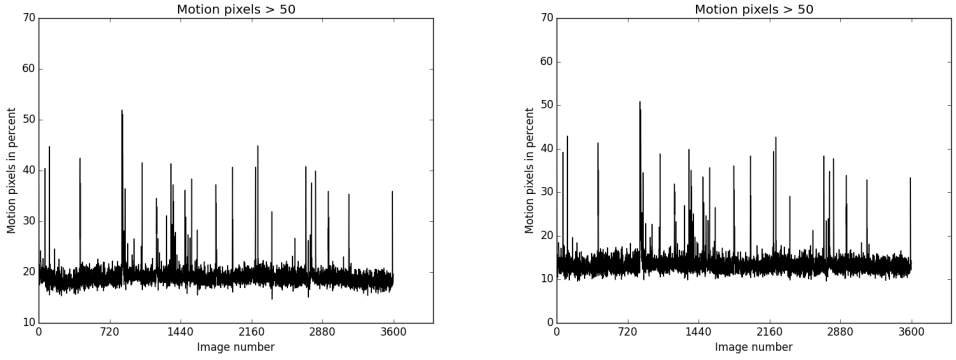
Figure 5.1 shows the original cropped image compared with the result when using a 11x11 **mean** kernel. Figure 5.2 shows the same, only with an 11x11 **gauss** kernel.

Both b-plots has reduced the noise from 20% to approximately 10-15%, and still kept the motion values on the same high value. Conclusion of this section is that low-pass filtering improved the result. However, it did also increase the runtime on the 3600 image sequence from 8.54 to 10.65 seconds, or 25% time increase.



(a) *(b)*

Figure 5.1: ***Mean:*** *Plots based on 3600 cropped images. Motion > 50 means that a slack of 50 was used. Plot a) is the original images, while b) is low-pass filtered with a 11x11 mean kernel.*



(a) *(b)*

Figure 5.2: ***Gauss:*** *Plots based on 3600 cropped images. Motion > 50 means that a slack of 50 was used. Plot a) is the original images, while b) is low-pass filtered with a 11x11 gauss kernel.*

5.2 Plot threshold

As discussed in section 4.6 in the chapter of Material and Methods, the plots contains 10-20% noise. This noise forms the base of all the plots. In order to separate the actual motion from the noise, I decided on a threshold above 20% to represent the motion. Figure 5.3 depicts the threshold as blue dashes in the motion plot.

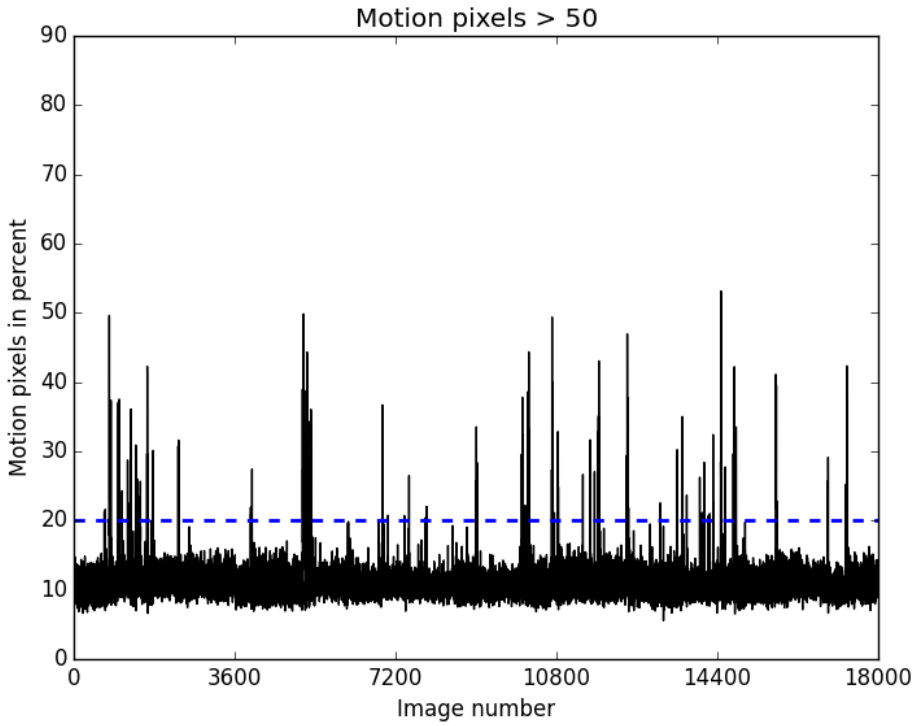


Figure 5.3: Motion plot with blue threshold

The threshold could not be lower than 20% for two reasons:

- The threshold should be general, and work for all motion plots.

- It is better to classify an awake instance as asleep, than vice versa.

5.3 Motion vector size

The iWakeUp system[16] described in section 3.1, used a frequency of 1 second between images. At each time interval, they collected the amount

of motion of the previous and following 7 seconds to form a 15-dimensional feature vector.

Inspired by iWakeUp system, I also made a vector of motion incidents. Unlike iWakeUp, I decided to only look at motion above the 20% threshold to avoid including noise as motion. For this reason, the vectors can only include events where there were actual motion, so my vector size is smaller than the one from iWakeUp.

The goal with the vector was to find the largest motion incidents where there was major movement for a long time. This section tests different vector sizes in order to decide which size is the best. The vector sizes is denoted as `axb`. This means that there were `a` consecutive incidents within `b` seconds.

The time span of the vector is based on the vector length. For a vector with length 7, the minimum possible difference between all indices in the vector is 6 seconds, since there is 1 second between images. This time span was decided to be 2 times the minimum time span. For a vector with length 7, the sum of the absolute difference between the indices in the vector could be maximum 6x2, so this vector size becomes 7x12. Note that most of the indices is not included because their motion value did not exceed the threshold of 20%.

5.3.1 Motion plots with different vector size

The following three motion plots (5.4, 5.5 and 5.6) are all based on the same image sequence. The sequence is 18 000 images long with 1 second between images, which equals 5 hours of recording.

The red dots in all plots shows where there was major movement according to the vector. Some of the dots are occluded, so in addition to the plots I printed all the image indices for testing purposes.

The vector size, the number of major incidents, and the number of incidents that happened during the last half hour are included in the caption of the plots.

5.3.2 Choice of vector size

The final choice of vector size was based on the amount of big incidents in the last 30 minutes of recording. This time period is the most important if this data should be used as input to an alarm clock, as they did in the article about the iWakeUp system (described in 3.1). The vector with most incidents in the last half hour was 9x16.

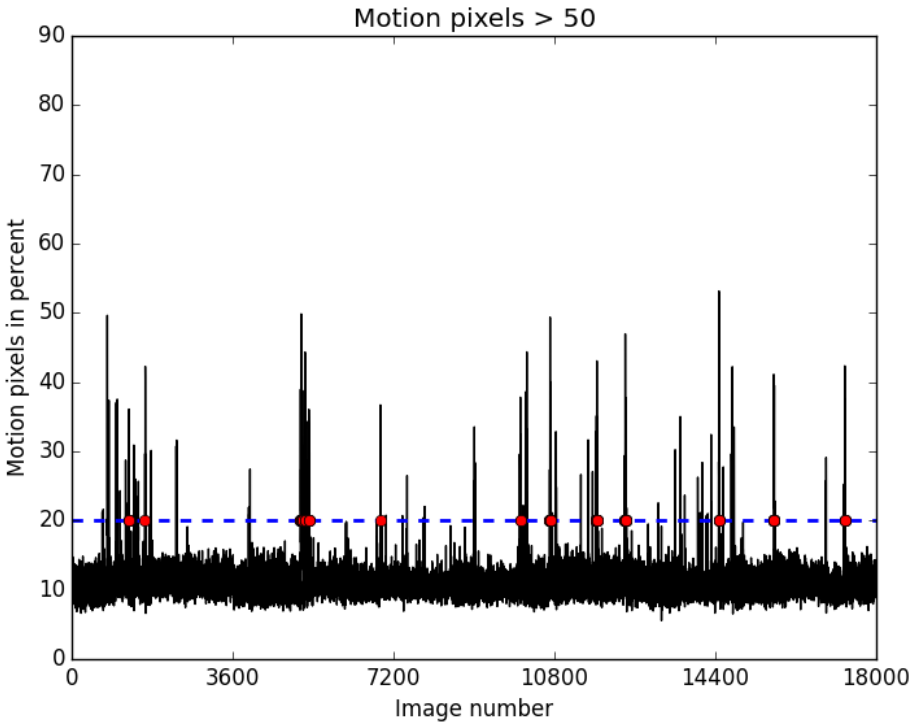


Figure 5.4: Vector size 7x12 - 108 major incidents - 5 in the last half hour

False positives and false negatives

This section uses the terms false positives and false negatives. Since I am looking for awake incidents, awake is considered positive. A false positive means an incident was classified as awake, when the subject was actually asleep. A false negative is the opposite; an incident classified as asleep when the subject was awake.

As input to an alarm clock, it is preferable with false negatives from the recordings. If the alarm clock starts ringing based on a false positive this will result in an unpleasant awakening since the subject is actually in deep sleep. Fewer incidents where I am sure that the subject is actually awake, is a more reliable input.

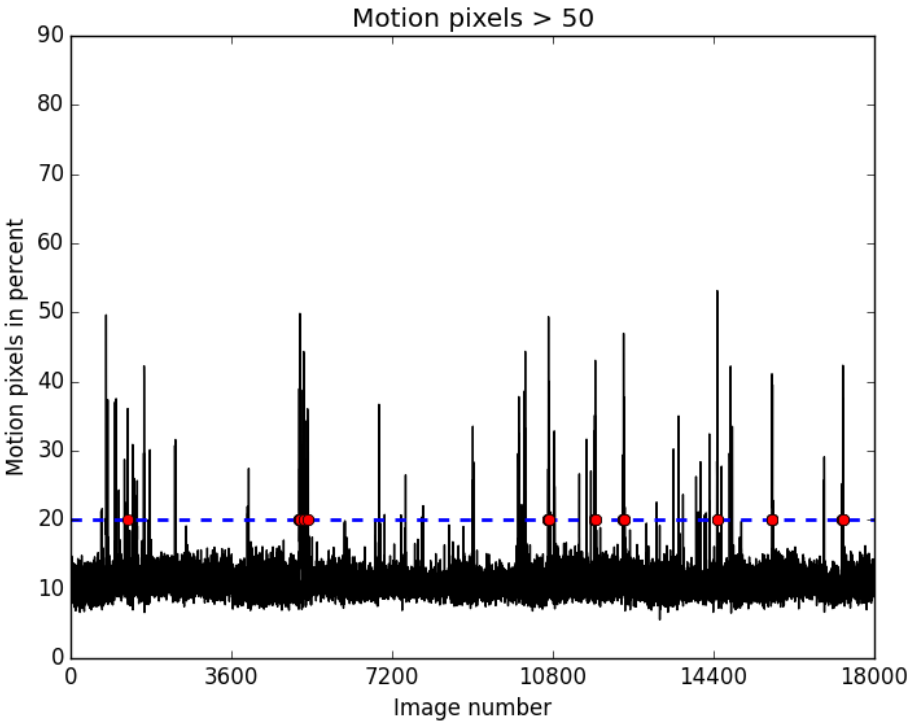


Figure 5.5: Vector size 9x16 - 89 major incidents - 6 in the last half hour

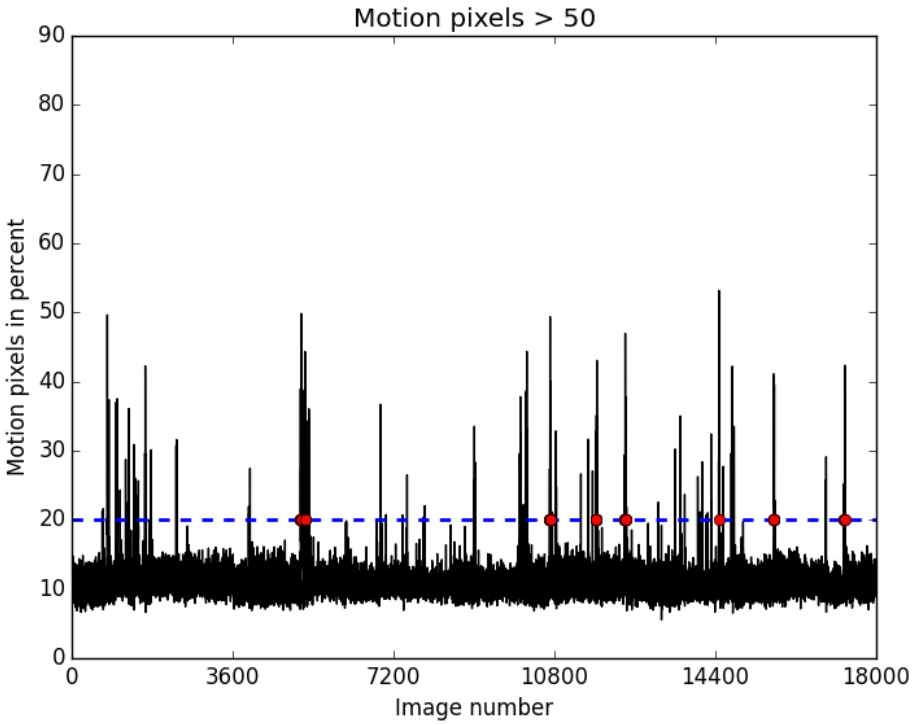


Figure 5.6: Vector size 11x20 - 71 major incidents - 4 in the last half hour

5.4 Sleep as Android application

The application "Sleep as Android" described in section 2.4.3 measures light and deep sleep with actigraphy. The build-in accelerometer sensor in my phone measured sleep movements during the same Kinect recording as the last section (5.3).

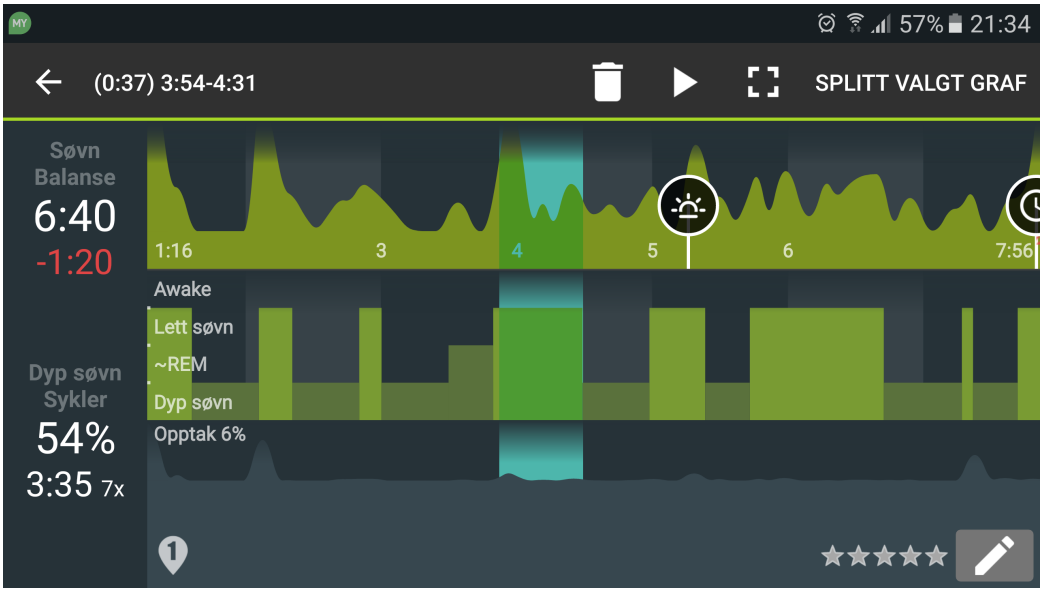


Figure 5.7: Sleep recording from Sleep as Android

Sleep as Android classified these times as light sleep for that particular night:

- 02:05-02:20
- 02:50-03:00
- 03:50-04:30
- 05:00-05:25
- 05:45-06:45
- 07:20-07:26
- 07:45-07:55

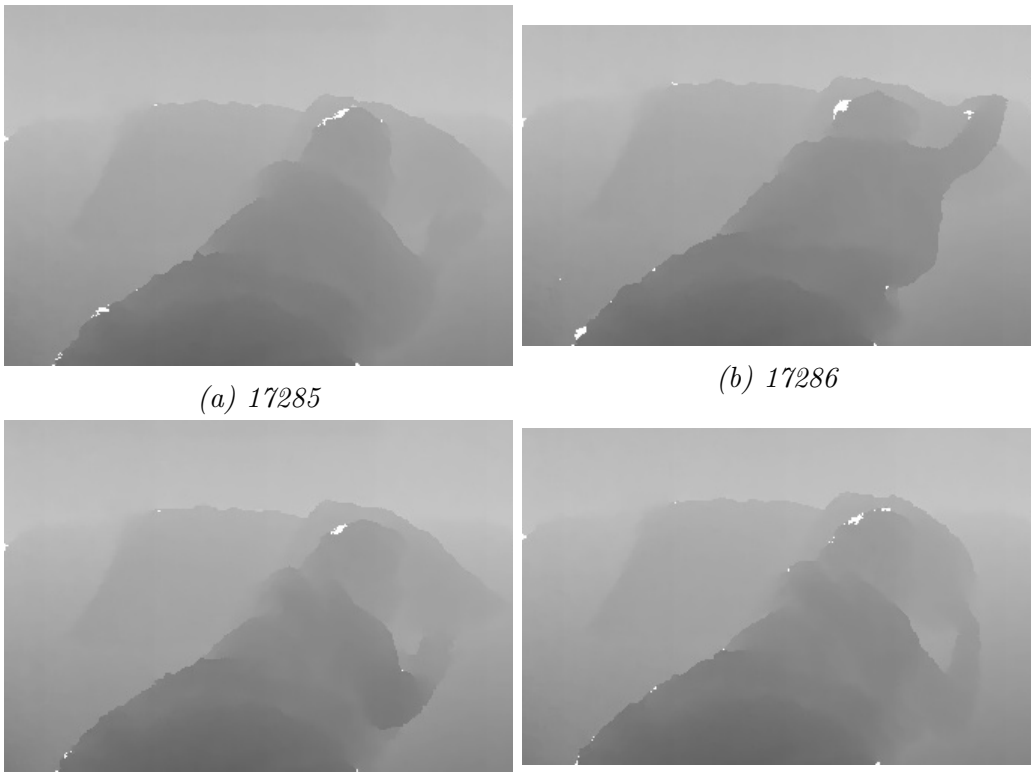
The Kinect recordings started at 02:30 AM, so in order to compare the Android times with the Kinect plot, the indices have to be translated. Index 0 in the plot equals 02:30:00, and every consecutive index is 1 more second. For index 3600, 1 hour has passed, so the time is 03:30:00. End time (image 18 000) is 07:30:00.

5.4.1 Comparison with vector 9x16

Since the red dots in the motion plots does not give the accurate time, I printed the times. Times in black matches the times recorded with "Sleep as Android", and times in blue are almost a match – decided to be maximum 5 minutes. There was no times that was completely wrong. The times below does not match Sleep as Android perfectly. However, looking at the depth images in figure 5.8, the numbers in blue does contain actual motion.

['2:51:12', '3:55:19', '3:55:20', '3:55:21', '3:55:22', '3:55:25', '3:55:27', '3:55:28', '3:55:29', '3:55:30', '3:55:31', '3:55:32', '3:55:34', '3:55:35', '3:55:36', '3:55:37', '3:55:38', '3:55:39', '3:55:40', '3:55:41', '3:55:45', '3:55:46', '3:55:47', '3:55:48', '3:55:49', '3:55:50', '3:55:51', '3:55:53', '3:56:59', '3:57:00', '3:57:01', '3:57:02', '3:58:26', '3:58:27', '5:28:16', '5:28:17', '5:28:18', '5:28:20', '5:28:21', '5:28:22', '5:28:23', '5:28:24', '5:28:25', '5:28:26', '5:28:27', '5:28:30', '5:28:31', '5:28:32', '5:28:33', '5:45:32', '5:45:34', '5:45:36', '5:45:37', '5:45:38', '5:45:39', '5:45:40', '5:45:43', '5:45:44', '5:45:46', '5:56:13', '5:56:14', '5:56:16', '5:56:17', '5:56:20', '5:56:23', '5:56:24', '5:56:25', '5:56:26', '5:56:27', '5:56:28', '6:31:14', '6:31:15', '6:31:16', '6:51:47', '6:51:48', '6:51:49', '6:51:50', '6:51:51', '6:51:52', '6:51:53', '6:51:54', '6:51:55', '6:51:56', '7:18:03', '7:18:04', '7:18:05', '7:18:09', '7:18:11', '7:18:12']

The last 6 times shown ('7:18:03', '7:18:04', '7:18:05', '7:18:09', '7:18:11', '7:18:12') equals image 17285, 17286, 17287, 17291, 17293, 17294. Figure 5.8 shows the depth images with index 17285-17288. Looking at the original images there is clearly motion of arms and upper body. (It is harder to see the motion from the images included in this PDF, since you cannot place them on top of each other to see the differences.)



(a) 17285 *(b) 17286*

(c) 17287 *(d) 17288*

Figure 5.8: Depth images for index 17285-17288.

5.5 Final motion results with vector 9x16

The next four pages contain motion plots of 5 hours of recording for consecutive 4 nights, where the vector size was 9x16. The code for making these plots are provided in Appendix B. The spikes in the diagrams marked with a red dot means that there were major movement.

The first night (figure 5.9) the red dots were evenly spread out over the plot. That much major movement during the entire night, could indicate that I did not sleep well the first night. It takes some time getting used to the idea of recording yourself while you sleep. Also the light and noise from the Kinect were disturbing.

The second night (figure 5.10) had the most major incidents, and the least amount in the last half hour. This was the night when the "Sleep as Android"-recording was done. The Android recording shows many incidents of light sleep for this night.

The third night (figure 5.11) stands out compared to the other nights.

Night 1, 2 and 4 had 84-89 major incidents, while the third night had only 50 major incidents. I slept well the third night, or at least I did not move as much as the other nights. There is a wide spike after 10800 which should probably have a red dot.

The fourth night (figure 5.12) should probably have a red dot for the thick spike before reaching 3600 images. This was the night with the most incidents in the last half hour.

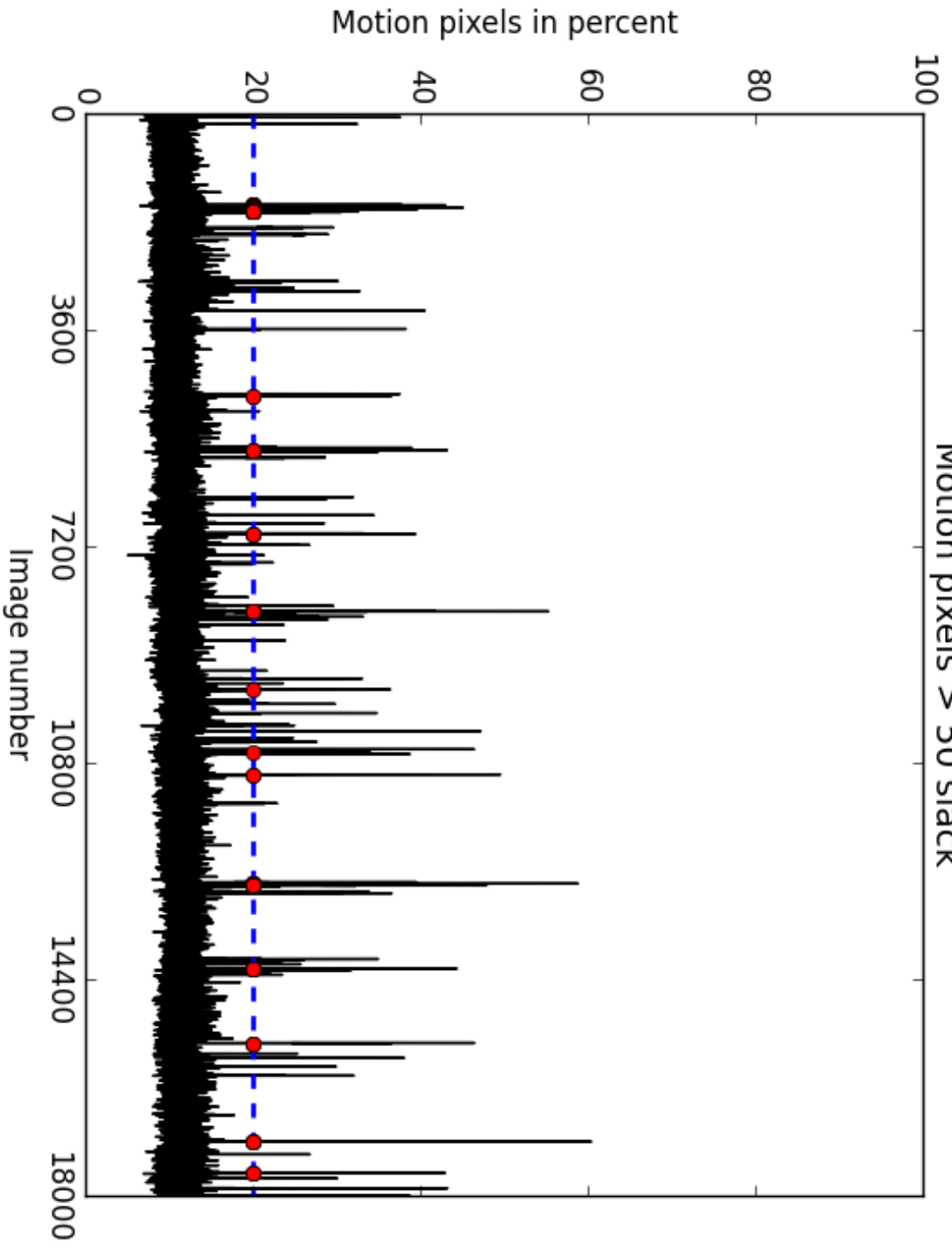


Figure 5.9: First night. 9x16 vector resulted in 88 major incidents, where 8 occured during the last half hour.

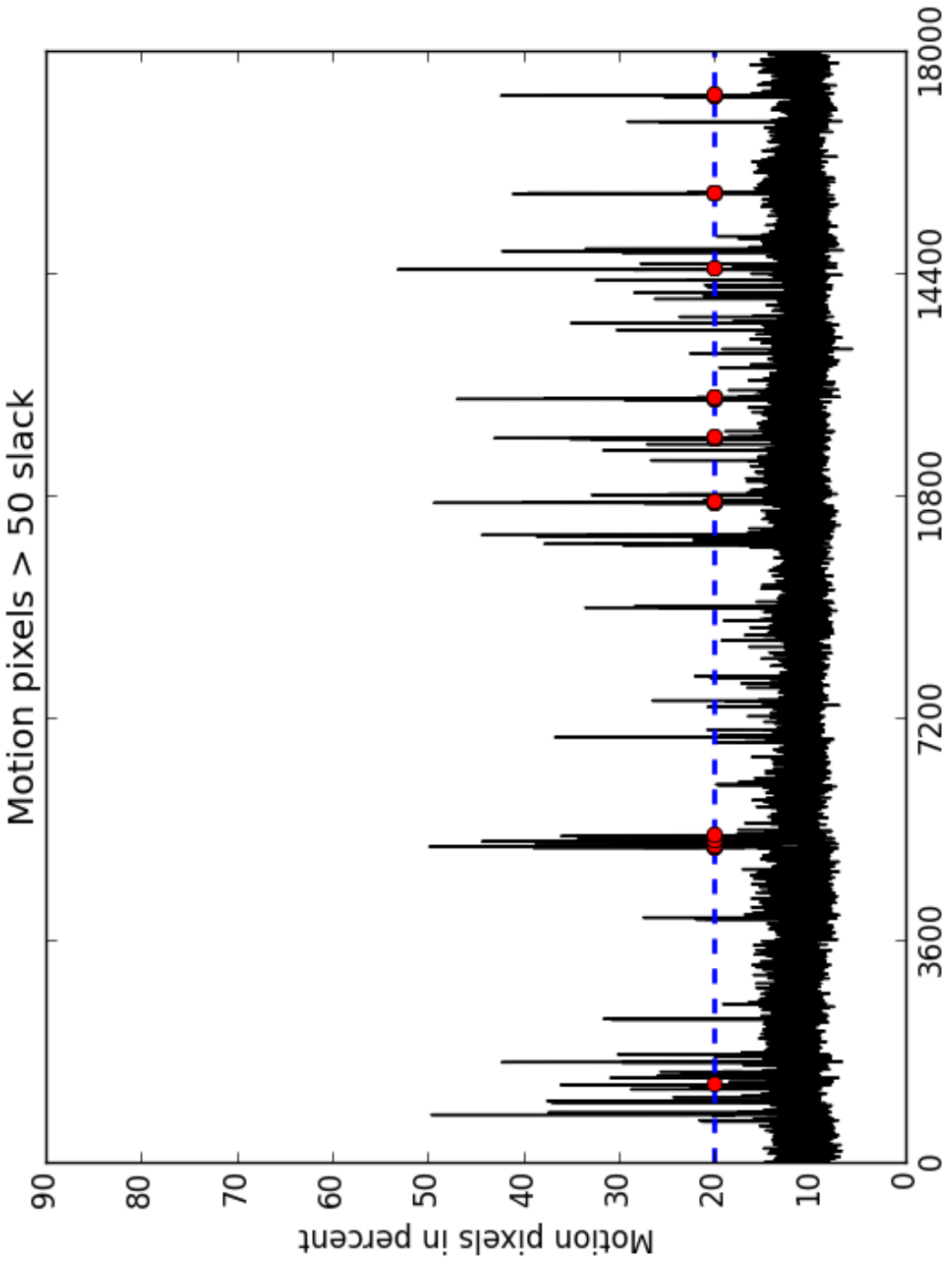


Figure 5.10: Second night. 9x16 vector resulted in 89 major incidents, where 6 occured during the last half hour.

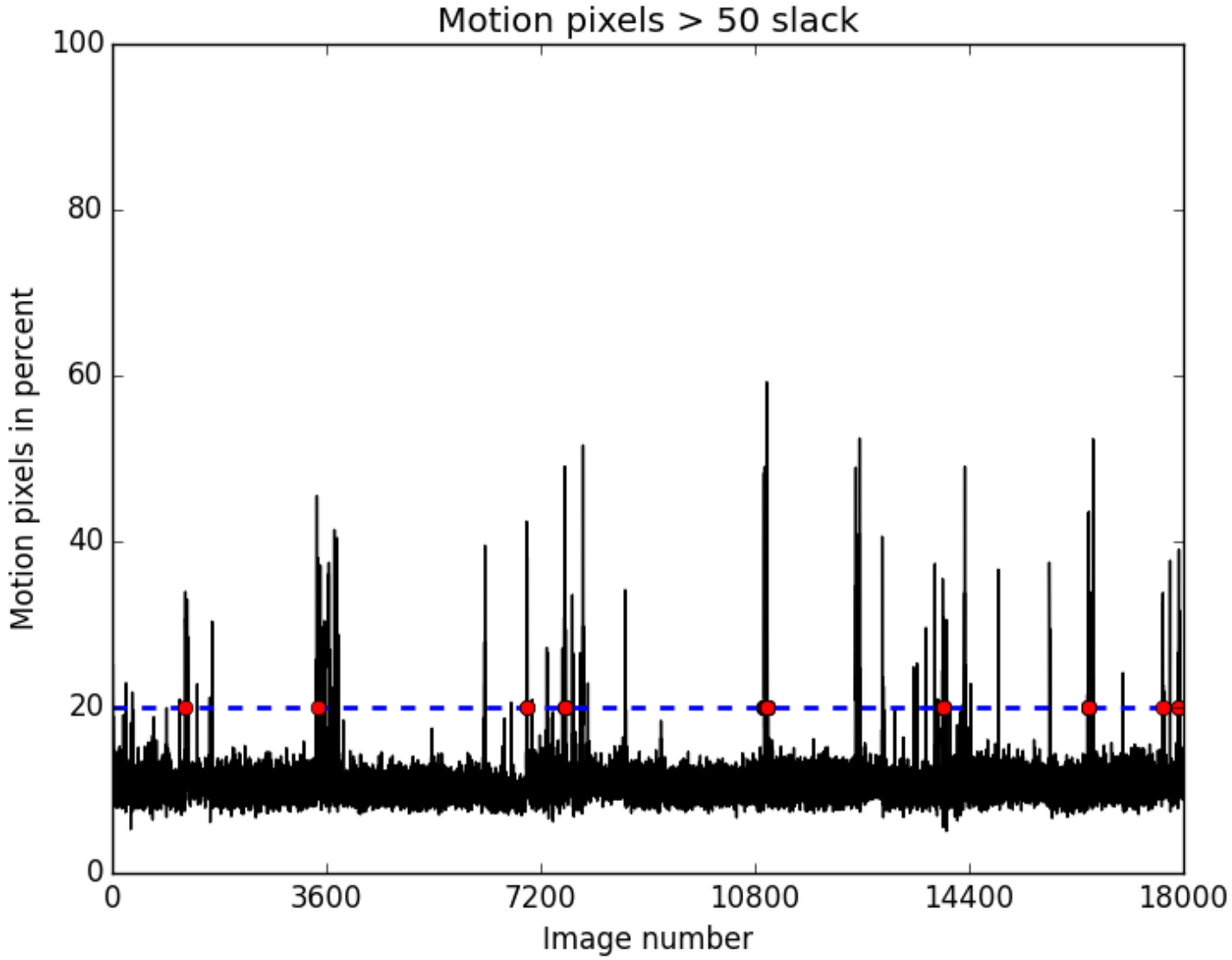


Figure 5.11: Third night. 9x16 vector resulted in 50 major incidents, where 10 occured during the last half hour.

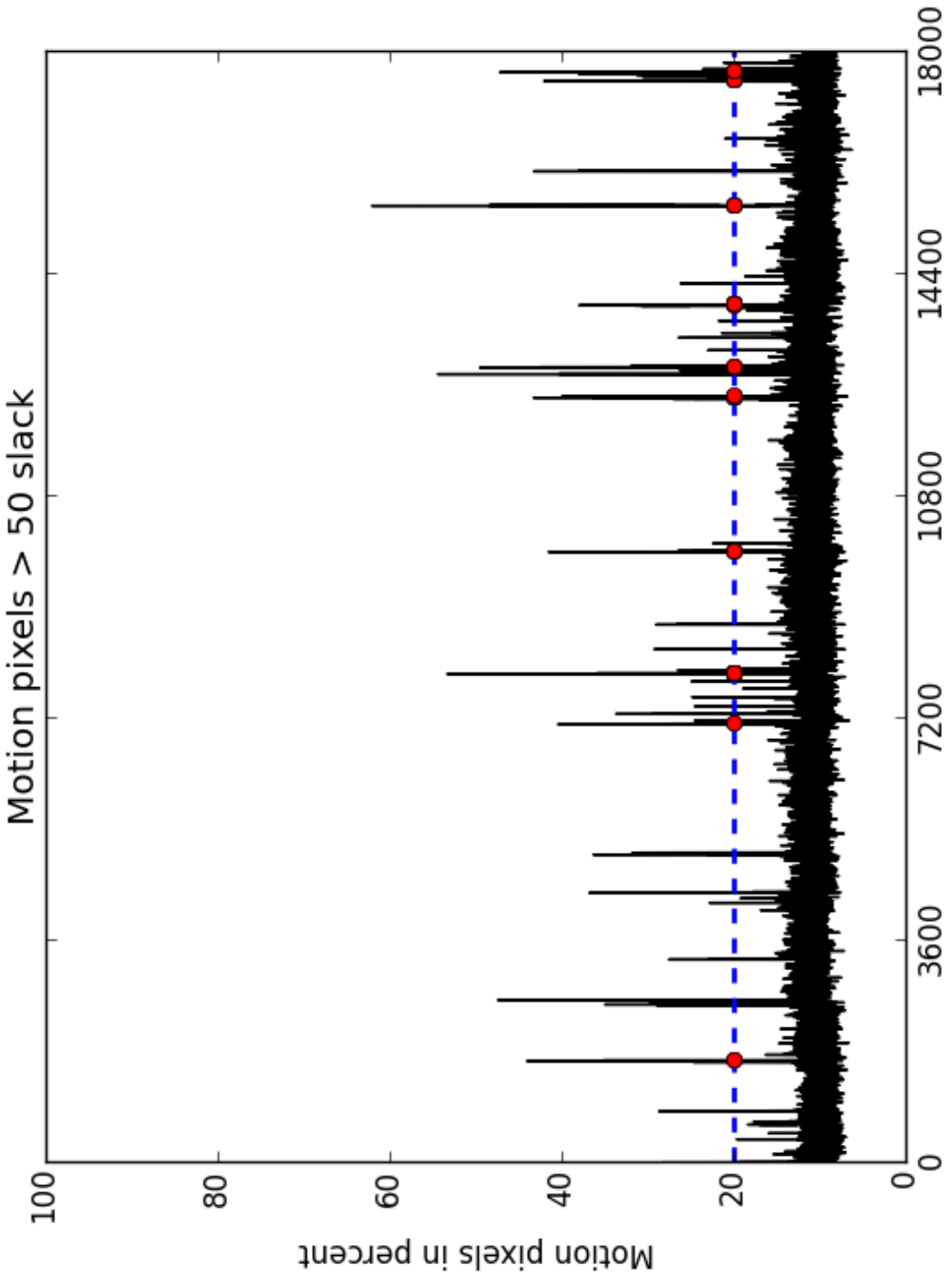


Figure 5.12: Fourth night. 9x16 vector resulted in 84 major incidents, where 11 occured during the last half hour.

Chapter 6

Conclusion and Further Work

Sleep movement analysis is a useful tool for improving sleep quality and for determining when a person is sleeping heavily, and when a person is almost awake. Performing sleep movement analysis at home is more comfortable and cheaper than with PSG in a laboratory. I explored the usage of Kinect to determine how well a person is sleeping. For this project I chose to use OpenCV, Python and OpenKinect.

This project focused on sleep movements based on the theory that large body movements during sleep means that the person is about to wake up. This is based on research provided in chapter 3.

The setup of the Kinect turned out to be more challenging than first assumed; the Kinect had to be placed above the bed without the risk of falling down on the person sleeping. Multiple programs were made to analyse the Kinect images. This project tried different thresholds and vector sizes in order to improve the results.

The evaluation of the results were done by studying the plots and images. Another comparison was done by comparing the times of the body movements with the application Sleep as Android.

The results showed that Kinect depth sensor can provide images which are accurate enough to decide whether the patient is asleep or awake at different times during the night. By choosing a threshold of 20% and a vector size of 7x12, I was able to find major body movements with the program. Comparison of these results and study of the images showed that all body movements found by the Kinect was indeed movements. Comparison with Sleep as Android showed that this project was almost as precise at finding body movements as the phone application.

Finally, my conclusion is that Kinect is a useful tool for determining the amount of motion during sleep, and that this recorded motion can be used as input to an alarm clock. The results can be improved by modifying the

program further or extend the functionality.

6.1 Further Works

There are multiple desirable extensions or changes of the program that would improve the results. This section covers some of these extensions or changes with a short discussion of their effect.

6.1.1 Newest Kinect version

This project used Kinect v1 since this was the only available sensor from the robotics lab at the university at the time this project was started. A change could be to use the newest Kinect version. Kinect v2 uses Time-Of-Flight which is more accurate than Structured Light for v1.

6.1.2 Other programming languages

Another change would be to test other languages for the image analysis. This project chose to use OpenCV, Python and OpenKinect. However, it would be interesting to try other languages and libraries and compare the performance.

6.1.3 Different sample sizes

This project focused on a small sample size and I was the only sleeping subject. Further testing with other subjects and other sample sizes would probably be useful data for improving the program even more.

Bibliography

[1] Biber MP Aaronson ST Rashed S and Hobson J. "Brain state and body position: A time-lapse video study of sleep". In: *Archives of General Psychiatry* 39.3 (1982), pp. 330–335. DOI: 10.1001/archpsyc.1982.04290030062011. eprint: /data/Journals/PSYCH/12353/archpsyc_39_3_011.pdf. URL: http://dx.doi.org/10.1001/archpsyc.1982.04290030062011.

[2] Ethem Alpaydin. *Introduction to machine learning*. MIT press, 2014.

[3] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. 1st. O'Reilly, 2008. ISBN: 9780596516130.

[4] Penelope A. Bryant, John Trinder, and Nigel Curtis. *Nature Reviews Immunology*. http://www.nature.com/nri/journal/v4/n6/fig_tab/nri1369_F1.html. (Retrieved 16-October-2014). 2004.

[5] *C++*. https://www.techopedia.com/definition/26184/c-programming-language. (Retrieved 6-May-2016).

[6] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[7] William Dement and Nathaniel Kleitman. "Cyclic variations in EEG during sleep and their relation to eye movements, body motility, and dreaming". In: *Electroencephalography and Clinical Neurophysiology* 9.4 (1957), pp. 673–690. ISSN: 0013-4694. DOI: http://dx.doi.org/10.1016/0013-4694(57)90088-3. URL: http://www.sciencedirect.com/science/article/pii/0013469457900883.

[8] Marko Heikkila and Matti Pietikainen. "A texture-based method for modeling the background and detecting moving objects. IEEE transactions on pattern analysis and machine intelligence". In: 28.4 (2006), pp. 657–662.

[9] *How the Kinect Depth Sensor Works in 2 Minutes.* https://www.youtube.com/watch?v=uq9SEJxZiUg. (Retrieved 9-August-2015).

[10] Timo Kahlmann. "Range imaging metrology: Investigation, calibration and development". PhD thesis. University of Hannover, 2007.

[11] *Kinect Range.* https://msdn.microsoft.com/en-us/library/hh973078.aspx#Depth_Ranges. (Retrieved 22-April-2016).

[12] Jeff Kramer et al. *Hacking the Kinect.* 1st edition. New York: Apress, 2012.

[13] Daniel F Kripke et al. "Wrist actigraphic scoring for sleep laboratory patients: algorithm development". In: *Journal of sleep research* 19.4 (2010), pp. 612–619.

[14] Björn Krüger et al. "Sleep detection using a depth camera". In: *Computational Science and Its Applications–ICCSA 2014.* Springer, 2014, pp. 824–835.

[15] Jaehoon Lee, Min Hong, and Sungyong Ryu. "Sleep monitoring system using kinect sensor". In: *International Journal of Distributed Sensor Networks* 2015 (2015).

[16] Wen-Hung Liao et al. "iWakeUp: A video-based alarm clock for smart bedrooms". In: *Journal of the Chinese Institute of Engineers* 33.5 (2010), pp. 661–668.

[17] David G Lowe. "Object recognition from local scale-invariant features". In: *Computer vision, 1999. The proceedings of the seventh IEEE international conference on.* Vol. 2. Ieee. 1999, pp. 1150–1157.

[18] *MATLAB.* https://en.wikipedia.org/wiki/MATLAB. (Retrieved 6-May-2016).

[19] Ernst Niedermeyer and F. H. Lopes da Silva. *Electroencephalography: Basic Principles, Clinical Applications, and Related Fields.* 5th edition. Lippincott Williams & Wilkins, 2005.

[20] *NREM.* https://en.wikipedia.org/wiki/Non-rapid_eye_movement_sleep. (Retrieved 24-April-2016).

[21] *OpenKinect.* http://openkinect.org/wiki/Main_Page. (Retrieved 11-June-2014).

[22] *Polysomnography.* http://en.wikipedia.org/wiki/Polysomnography. (Retrieved 12-November-2014).

[23] *PSG image.* http://www.pittsburghdentalsleepmedicine.com/. (Retrieved 29-April-2016).

[24] Hartmut Schulz. "Rethinking sleep analysis". In: *J Clin Sleep Med* 4.2 (2008), pp. 99–103.

[25] *Sleep*. http://en.wikipedia.org/wiki/Sleep. (Retrieved 11-June-2015).

[26] *Sleep as Android*. http://sleep.urbandroid.org/documentation/core/background/. (Retrieved 15-April-2015).

[27] *Sleep Research Timeline*. http://www.howsleepworks.com/research.html. (Retrieved 28-April-2016).

[28] *Support vector machine*. http://en.wikipedia.org/wiki/Support_vector_machine. (Retrieved 12-November-2014).

[29] Lu Xia, Chia-Chih Chen, and JK Aggarwal. "Human detection using depth information by kinect". In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*. IEEE. 2011, pp. 15–22.

[30] Zhengyou Zhang. "Microsoft Kinect Sensor and its Effect". In: *MultiMedia, IEEE* 19.2 (2012), pp. 4–10.

Appendices

Appendix A

Installation for OS X Mavericks

This section contains the instructions I used for installation of Python, OpenCV and libfreenect (OpenKinect) on OS X 10.9 (Mavericks). The installation also worked for 10.10 (Yosemite).

A.1 Homebrew

Homebrew (http://brew.sh/) is often referred to as the missing package manager for OS X. Homebrew installs packages to their own directory and then symlinks their files into /usr/local. Homebrew will not install files outside its prefix, and you can place an installation wherever you like. Homebrew formulas are just simple Ruby scripts.

Installing Homebrew

- Install Homebrew from http://brew.sh/
- Paste in Terminal (no blank spaces): ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
- Run brew doctor in Terminal. This is probably going to tell you that /usr/bin comes before /usr/local/bin, which Homebrew depends on, so change the PATH so /usr/local/bin comes first in your PATH. This only affects the current shell, so add the following line to your .bash_profile: export PATH=/usr/local/bin:/usr/local/sbin:$PATH
- Might have to change permissions to be able to install packages: sudo chown -R $USER:admin /Library/Caches/Homebrew

A.2 Installing OpenCV and Python

- Add homebrew/science, which is where opencv is located: "brew tap homebrew/science"
- See options when installing: "brew info opencv"
- Install OpenCV: "brew install opencv"
- You can find OpenCV at "cd /usr/local/Cellar/opencv/"
- For the newest version of Python: "brew install python"

Linking Python to the OpenCV files (my OpenCV version is 2.4.11_1)

- cd /usr/local/lib/python2.7/site-packages/
- ln -s /usr/local/Cellar/opencv/2.4.11_1/lib/python2.7/site-packages/cv.py cv.py
- ln -s /usr/local/Cellar/opencv/2.4.11_1/lib/python2.7/site-packages/cv2.so cv2.so
- Make sure everything works by running "python" in the terminal and then typing "import cv2" in the python shell
- Also install matplotlib for plotting: "pip install matplotlib"

A.3 Installing libfreenect (from OpenKinect)

- Open a Terminal and type: brew install libfreenect
- Run "freenect-glview" (from anywhere in the terminal) to test that libfreenect works properly.
- For libfreenect to work with Python, I had to make python bindings. Go to https://github.com/OpenKinect/libfreenect
- Go inside the wrappers folder, and copy the python folder so you have it locally on your computer
- Open the Terminal and cd to your new wrapper/python folder in the terminal, and run: "python setup.py install".

- Plug in the Kinect. While still in the same wrapper/python folder, run: "python demo_tilt.py".

Appendix B

Python code for plotting motion based on the depth images

```
#!/usr/bin/env python
import numpy as np
import cv2, os, datetime, sys
import matplotlib.pyplot as mp

height = 480
width = 640
slack = 50
frames = 18000
motion_list = []

start = datetime.datetime.now()

fileprefix = '24april/kinect04-24-03-30_1sec/depth'
print fileprefix
prev_orig = cv2.imread(fileprefix + '1.jpg', cv2.IMREAD_GRAYSCALE)
prev_blur = cv2.blur(prev_orig, (11,11))
#prev_blur = cv2.GaussianBlur(prev_orig, (11,11), 0)

for i in range(2, frames):
   filename = fileprefix + str(i) + '.jpg'
   img_orig = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
   img_blur = cv2.blur(img_orig, (11,11))

   prev = prev_blur[240:480, 20:320]
   img = img_blur[240:480, 20:320]

```

```
28      diff = np.subtract(img, prev)
29      abs_diff = np.absolute(diff)
30      motion_int = (abs_diff > slack).sum()
31      motion_list.append(motion_int * 100.0/img.size)
32
33      prev_blur = img_blur
34
35  end = datetime.datetime.now()
36  print "Total processing time =", end-start
37
38  array = np.asarray(motion_list)
39  motion_len = (array > 20).sum()
40  motion_indices = np.where(array > 20)
41
42  seconds = np.asarray(motion_indices[0])
43  print "Number of incidents", motion_len
44
45  # Find incidents
46  incident_len = 9
47  incident_time_list = []
48  incident_index_list = []
49
50  for i in range(0, (len(seconds)-incident_len+1)):
51      part = seconds[i:(i+incident_len)]
52      diff_in_seconds = np.ediff1d(part)
53      diff_seconds_sum = diff_in_seconds.sum()
54
55      if(diff_seconds_sum < 16):
56          # convert to hours, minutes and seconds
57          start_time = 3*3600+30*60 # start 03:30
58          incident_time =
                str(datetime.timedelta(seconds=seconds[i]+start_time))
59          incident_time_list.append(incident_time)
60          incident_index_list.append(seconds[i])
61
62  print "Number of big incidents:", len(incident_time_list)
63  print "Big incidents in time:", incident_time_list
64  print "Big incidents with index:", incident_index_list
65
66  def plotMotion(motion_list, titleString):
67      mp.ylabel('Motion pixels in percent')
68      mp.xlabel('Image number')
69      mp.title(titleString)
```

```
70      mp.plot(motion_list, 'k') #motion
71      mp.plot([0, frames], [20, 20], 'b--', linewidth=2) #threshold
72      y = 20 * np.ones(len(incident_index_list))
73      mp.plot(incident_index_list, y, 'ro')
74      mp.xticks(range(0, frames+2, 3600))
75      mp.show()
76
77  plotMotion(motion_list, ('Motion pixels > ' + str(slack) + '
        slack'))
```