Adaptive Alarm Clock Using Movement Detection to Differentiate Sleep Phases

Jasper Kuijsten
Eindhoven University of Technology
P.O. Box 513, 5600MB, Eindhoven, the Netherlands
Email: j.j.a.kuijsten@student.tue.nl

***Abstract*—Waking up can be very troubling. In many cases, being awoken by the alarm clock during a period of deep sleep causes a person to not feel fully rested after waking up. This paper describes how an adaptive alarm clock was developed addressing this problem. The alarm clock predetermines in what state the sleeper will be at the instant of supposed alarm-firing, and adjusts that instant to a more favourable one. Preferably for the sleeper to be awakened during a period of light sleep. For this the system gathers information about the sleeper during the night via a webcam. The videofeed is processed in real-time on a Nvidia videocard using CUDA, determining the amount of movement by the slumberer at that instant, which is a direct indication of which sleep phase he or she is currently experiencing. This information can be analyzed to predict future phases, to adjust the instant of alarm-firing. The possibility to use any webcam and a standard Nvidia GPU, makes this a solution available to a wide audience.**

***Index Terms*—Movement detection, CUDA, GPU, sleep phases, clock, video processing, approximate median, parallel computing, background subtraction.**

I. Introduction

WAKING up without the sound of an alarm clock is a privilige reserved for most of us only on weekends. Every weekday one runs the risk of being awoken by the alarm clock during a period of heavy sleep, resulting in feeling weak, being confused or experiencing a general fatigue after waking up. These situations can be avoided by determining the sleeping phase the slumberer is experiencing at the instant of supposed alarm-firing, and adapting the actual instant accordingly. Sleeping phases occur periodically, so information about them acquired during the night creates possibilities to predict future sleep phases and their moment of occurence. Preferably, the sleeper will be awakened during a period of light sleep. Commercial applications for this exist, but require expensive hardware and are often of discomfort to the sleeper.

In this paper we take a new approach on determining the sleeping phase of the sleeper. A relation exists between moving while asleep and sleeping phases[1]. For healthy sleepers, much movement indicates the sleeper is experiencing a period of light sleep, similarly, little or no movement indicates a period of deep sleep. This feature will be exploited by using a webcam to detect the amount of movement by the sleeper throughout the night, thus gathering information about the sleep phases of the sleeper.

For the system being able to properly detect sleep phases, three components are of special importance. First, there is the motion detection algorithm which determines the amount of movement by the sleeper in each frame of the videostream. This information is then used by a data analysis algorithm to determine the sleeping phase. Special considerations should also be made regarding the choice of hardware.

The rest of this paper is organized as follows: The next Section discusses the hardware platform selected. Sections three and four present respectively the motion detection and data analysis algorithms. Finally, results are discussed in Section five and the paper is concluded in Section six.

II. Hardware

The hardware for our application consists out of a computing platform and a webcam. Hardware choice is of influence on the choice and complexity of the software of our application, and is hence made first.

A. Computing Platform

Processing a video stream requires a lot of processing power. When algorithms get more complicated, CPU's are not suitable for real-time image processing. Instead, highly parallel SIMD (Single Instruction Multiple Data) devices are used, which are specialized in executing the same instructions in parallel on massive amounts of data at once. With this information, the computing platform choice should be made regarding a number of properties:

- Low-Cost
- Highly parallel (SIMD)
- Compability with a standard webcam
- Well documented

With these properties in mind the Nvidia CUDA platform[2] was chosen as computing platform for this application. Nvidia GPU's are high-speed SIMD devices which are capable of running many threads simultaneously. CUDA provides a runtime environment for using the GPU as a CPU extension, enabling the programmer to create processes to be computed on the GPU instead of CPU where it can be executed in parallel. This offers significant speedup for image processing algorithms and fully unloads the CPU, which can be used for other processes. Also, GPU's are widely available for low prices, and integrate perfectly with any laptop or desktop computer consumers already own.

B. Camera Choice and Preparations

To use a webcam at night and still be able to sleep in the dark, infrared light is used to illuminate the environment. Infrared light is not visible by humans but digital camera's and webcams are highly sensitive to this wavelength. Normally, a webcam has an infrared light filter mounted before the CCD chip, preventing the infrared light from reaching the sensor and adding a purple glow to the videoframe. To use a webcam at night, this infrared filter will have to be removed. This is often a straightforward process, but research should be done before buying a webcam. Several other performance improving properties are a high framerate, high resolution and high infrared light sensitivity. Optimal camera settings regarding exposure, internal gain, and contrast should be experimented with.

The application described in this paper was developed using a Sony PS3 EYE camera, which offers true 50fps framerate at a resolution of 640x480 and is regarded as being highly infrared light sensitive. For illuminating the environment during testing, five standard 5mm 850nm LEDs were used.

III. Motion Detection

To determine the amount of movement by the sleeper several algorithms are executed sequentially on the videostream, provided by a webcam. For this, a videoframe is first captured on the CPU using the OpenCV[3] library, and is then sent to the Nvidia GPU for parallel processing. On the GPU each of the algorithms listed below is executed sequentially, while each algorithm by itself is executed in parallel.

- Colour to greyscale conversion
- Gaussian blurring
- Background subtraction
- Movement calculation

A. Colour to Greyscale Conversion

Converting colour frames to greyscale frames simplifies the remaining algorithms and increases performance. Due to the nature of the environment, where only infrared light is present, the colour frame will only contain multiple grades of purple. As a result there is no extra information in a 3-channel colour image compared to a 1-channel greyscale image. Thus converting to a greyscale image will not result in loss of information, while improving the processing performance of the remaining three functions threefold since only one channel is processed. For this the well known standard conversion equation[4] is used.

$$bw(x,y) = b(x,y) * 0.114 + g(x,y) * 0.587 + r(x,y) * 0.299$$

Where b, g, r represent the blue, green and red colour channels and bw the resulting greyscale frame.

B. Gaussian Blurring

Blur is applied to the videoframe to smooth the image. Videostreams contain noise, most commonly caused by the internal gain of the webcam and flickering pixels near sharp edges. Because the affected pixels quickly change in brightness they could be incorrectly interpreted as movement. To avoid this, the picture is smoothed using a blurring algorithm. Three of which were considered:

- Averaging blur
- Median blur
- Gaussian blur

These options were tested using the OpenCV library, with Gaussian blur being the method most effectively smoothing the noise while preserving image details. This method has thus been selected, and implemented for use on CUDA.

The Gaussian blur is applied by convolving the source frame with a 2-dimensional texture matrix. The texture matrix is filled with Gaussian components, forming a normal distribution. When implementing, it is best to split the blurring process into two passes. The first pass applies a horizontal blur using a 1-dimensional array of Gaussian values, after which the secondary pass will apply the same 1-dimensional blur vertically. This leads to the same result as applying a 2-dimensional blur in one pass due to the algorithms linear properties, but is less computationally intensive.

The Gaussian values are set according to the Gaussian function.

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}}$$

For this application the best results were achieved using an array five elements wide and a standard deviation σ of one, preserving image detail and completely removing noise.

C. Background Subtraction

Background subtraction provides an efficient method of detecting movement by comparing new frames against a background model. Pixels in the current frame deviating significantly from the background model are considered to be moving, and classified as foreground pixels. Many background subtraction methods exist, differing mostly in background modeling technique[5]. Three methods were explored.

- Frame differencing
- Approxmiate median
- Mixture of Gaussians

Frame differencing uses the previous frame $i - 1$ as background model for current frame i. This provides a very simple method for detecting moving objects. Unfortunately, objects standing still or uniformly coloured are identified as background after only one frame. This is known as the aperture problem. The aperture problem is especially noticable in low brightness environments, as is the case in our application.

Approximate median method uses a recursive technique for estimating a background model. Each pixel in the background model is compared to the corresponding pixel in the current frame, to be incremented by one if the new pixel is larger than the background pixel or decremented by one if smaller. A pixel in the background model effectively converges to a value where half of the incoming pixels are larger than and half are smaller than its value. This value is known as the median.

Mixture of Gaussians (MoG) uses multiple Gaussian distributions to model a pixel in the background model, each with its own weight and standard variation. Current pixels are matched against all distributions of its corresponding background pixel. If a match with a certain distribution occurs, its weight and standard variation are respectivily increased and decreased. If no match can be found, a new distribution is created with initial low weight and large standard variance. This technique allows constantly moving objects, like waving trees, to be modeled into the background.

All three background subtraction techniques have been tested using MATLAB[6], each producing good results for our application. The Approximate median method has been selected, for it handles slow movements, which are often the case in our environment, better than the Frame differencing method and is less computationally intensive than the MoG method.

The Approximate median foreground detection compares the current video frame to the background model, and identifies the foreground pixels. For this it checks if the current pixel $bw(x, y)$ is significantly different from the modeled background pixel $bg(x, y)$.

$$|bw(x, y) - bg(x, y)| > T$$

A simplified pixel-wise implementation of the Approximate median background subtraction method in pseudo-code is given in Figure 1.

```
/* Adjust background model */
  if (bw > bg) then bg = bg + 1 ;
  else if (bw < bg) then bg = bg - 1 ;

/* Determenine foreground */
  if (abs(bw - bg) > T) then fg = 1 ;
  else fg = 0;
```

Fig. 1. Approximate median background subtraction

D. Calculating Movement

For our application only the amount of movement by the sleeper is of interest. Direction, speed, and which body parts are moving do not tell us anything about sleeping phases, and are thus not tracked. This simplifies calculating the total movement by the sleeper to simply summing up all the foreground fg pixels.

$$Movement = \sum_{y=0} \sum_{x=0} fg(x, y)$$

IV. Data Analysis

The raw data measured throughout the night needs to be formatted and then analyzed to extract any information about sleep phases from it. Figure 2 shows the movements of someone falling asleep at 11.00PM and waking up just before 6:00AM. Already periods of high and low activity can be roughly distinguished.

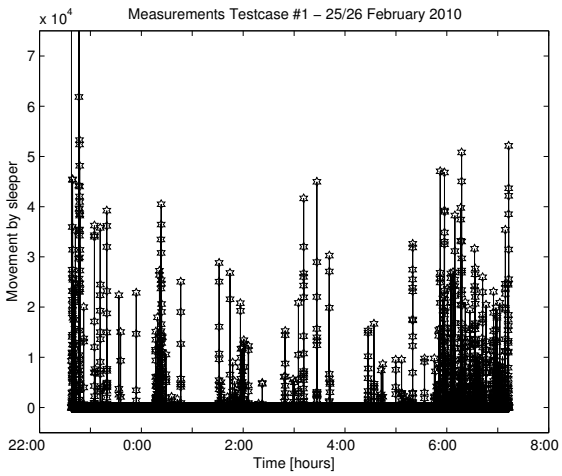

Fig. 2. Measurements

A. Formatting Measurements

Measurements are performed by the movement detection algorithm in real-time at the maximum samplerate allowed by hardware. These measurements are summed over a period of one second and saved to logfile as one sample. This method makes sure no measurements are discarded but keeps the amount of samples limited, which is beneficial when analyzing the data. The analyzing is done only once a night. The user defines this instant as the point from where on on the alarm is allowed to fire. For example, thirty minutes before supposed alarm-firing.

A typical movement, like rolling over from side to side, only takes a few seconds and occurs a few times per hour. This means several hundred zero-value samples, where no movement is detected at all, can sometimes be found in between two movement detections, which themselves only last a few samples. Because of this, the raw data can not simply be smoothed with a low-pass filter. Instead a weighting system is used. A sample becomes more weighted when more large samples are close by, where nearby samples contribute more than ones located further away and large samples contribute more than ones where only a small movement is detected. Further explaining this method is done most effectively by studying the pseudo-code shown in Figure 3.

```
define W 900 // 15 minute window size

for each m -W to W
  if (raw[n + m] > 0) then num_move = num_move + 1 ;
  formatted[n] =
    formatted[n] + (raw[n + m] * weight[m]) ;

formatted[n] = formatted[n] * num_move ;
```

Fig. 3. Formatting data algorithm

The value of sample n is a summation of itself and nearby samples multiplied with a m, distance from n, dependant weighing factor. The weighing factor is a Gaussian array with width equal to and standard deviation one third of the window

size, assigning more weight to samples as they get closer to n. Finally, sample n is multiplied with the number of movement detections that occured shortly before or after n.

Figure 4 shows the result of formatting the measurements of Figure 2 using discussed method. Periods of much and little movement are now very apparant.

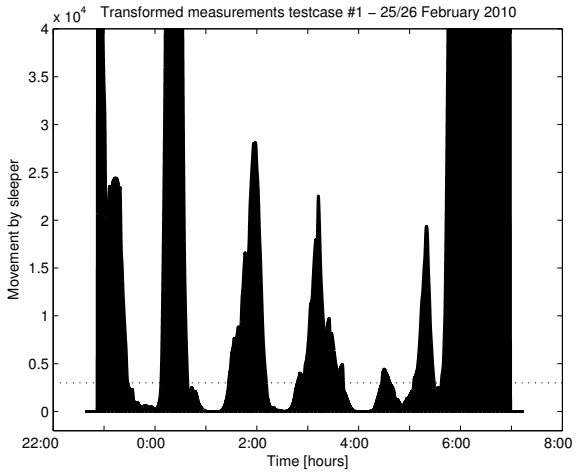

Fig. 4. Transformed measurements

B. Sleep Phases

Light and deep sleep phases are distuingished by the amount of movement detected. After formatting the measurements, distuingishing them is as simple as setting a treshold value. Periods of sleep where less movement is measured than the treshold value are classified as periods of deep sleep, whereas a measured value greater than the treshold is interpreted as light sleep. The exact treshold value differs from person to person, and should be determined experimentally or adapted recursively during usage. This can be done by asking the user a simple yes/no question in the morning. This is the method implemented into our application. For testcase one, the optimal treshold was set to 3000. The treshold is also shown in Figure 4.

Now that different sleep phases can be identified, information about them is collected to predict the occurence of future sleep phases. Transitions between sleep phases, the moment of transition, and the duration of each sleep phase are learned from the formatted data. This information is then extrapolated to determine to most favourable instant of alarm-firing, preferably a moment of light sleep.

V. Results

Discussed application was implemented on a Nvidia ION device and tested with the help of volunteers. Results are divided into two sections. First the technical results are presented, after which the effect on volunteers having been wakened by our application will be discussed.

A. Technical Results

On CUDA performance: Developed application was optimized for use with CUDA. Optimizing decreases processing time, enabling increasing samplerate and thus improving accuracy of the system. Table 1 shows the performance gain on the ION system running this papers image processing algorithms. With an image processing time of 6.6ms, samplerate is limited by the maximum framerate of the webcam instead of the CPU's computing capability. This secures the highest amount of accuracy possible while still using low-cost hardware.

TABLE I
Execution times in miliseconds on CPU and GPU

Process	CPU	GPU	Speedup
Greyscale Conversion	3.8	1.8	2.1x
Gaussian Blur	14.2	2.8	5.1x
Background Subtraction	15.1	0.5	30.2x
Movement Calculation	1.1	2.7	0.4x
Total	32.5	6.6	4.9x

The movement calculation function was implemented for CUDA, but ultimately not used. Summing the elements of a matrix is, in our application, executed faster on a CPU. Nvidia provides documentation how to efficiently sum a matrix in parallel. The proposed method was tested but did not give better results than using the CPU. Total processing time differs from the individual processing times summed up due to poor memory performance. To process data on the GPU, data needs to be copied to and from sytem RAM to GPU RAM. This is a slow process and not taken into account for the measured individual process execution times.

On detecting sleep phases: Periods of much and little movement throughout the night are visible very clearly in Figure 4. From this, the occurence of different sleep phases is very apparant. However, they are still estimations. It is difficult to state how accurate these estimations exactly are. To do this our measurements would have to be compared to ones taken by an EEG machine, capable of directly measuring brain activity via magnetic fields. Due to practical reasons this was not possible. Having said this, a number of properties of sleeping patterns do enable us to verify our measurements to atleast some extent. One feature of sleep cycles is that they are periodic of nature, with a known period of 90 to 110 minutes. Figure 4 shows a similar pattern. A second feature of sleep cycles is that periods of light sleep occur more often and the sleep becomes more irrigular in general as the sleeper nears the moment of natural awakening, this also can be seen in Figure 4.

With this information, it can be said our estimations on sleeping phases are a reasonably accurate representation of the actual sleeping phases. Our application is probably not accurate enough to wake the sleeper at the most favourable time possible, but is be able to select a moment very close to it. Most certainly, it can prevent users being woken up at the worst time possible, a period of deep sleep.

B. Effects After Waking up

The effect on how people feel after being woken up by our application is difficult to measure objectively. How people

feel after waking up is subject to many more variables than just the sleeping phase they were experiencing at moment of awakening and are subjective by nature. None of the volunteers felt particularly better than usual after being awoken by the developed alarm clock, but also none felt like they were awoken at a wrong instant. It should be noted all volunteers are healthy sleepers and do not suffer from problems waking up and getting out of bed. Perhaps the effect of our application is more noticable on persons who suffer from sleeping conditions.

Future research regarding this subject is necessary, but is best performed by a biomedical engineer, sleep researcher or professional sleep center. A physical and psychological view on sleep is out of scope of this paper and the authors expertise.

VI. Conclusions

In this paper we have shown a complete system capable of detecting sleep phases and determining a convenient time for the sleeper to be woken up. For this we measure the amount of movement by the sleeper using a webcam. The videoframe is first converted to greyscale, then blurred. Next a background subtraction is performed, classifying all moving objects as foregroud. Last, the foreground objects are counted, which translates to the amount of movement in that frame. All processing algorithms are run on a Nvidia CUDA device, which is highly parallel. The measurements are then formatted using a weighting system, to eventually extract information about sleeping phases from them. Sleep phases are distuingished by the amount of movement being larger or smaller than a personalized treshold value. Transitions between sleep phases and their duration are analyzed to predict future sleep phases. The results prove accurate enough to reliably distuingish sleeping phases. The sleeper will not always be woken up at the most favourable time, but likely close to it. So far, no positive nor negative effect has been observed regarding how people feel after being awakened by our alarm clock. Further research on this subject is needed, but best performed by someone with more expertise regarding the medical science of sleep.

Promising results and the use of low-cost hardware mean our application could be of interest to a consumer market.

References

[1] (2010, March) Neurofeedback. [Online]. Available: http://www.neurofeedback.nl/slaap_dynamische_activiteit.htm
[2] (2010, March) Nvidia CUDA website. [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html
[3] (2010, March) OpenCV website. [Online]. Available: http://opencv.willowgarage.com/wiki/
[4] (2010, March) Introduction to OpenCV programming. [Online]. Available: http://www.cs.iit.edu/ agam/cs512/lect-notes/opencv-intro/index.html
[5] S. Cheung and C. Kamath, ”Robust techniques for background subtraction in urban traffic video” in 2003.
[6] (2010, March) Background Subtraction MATLAB models by DSP DesignLine. [Online]. Available: http://www.videsgnline.com/howto/210002167