technische universität
dortmund

Bachelorarbeit

Saving Energy for Mobile Biosensors by Offloading

Dragana Popovic
Mai 2016

Gutachter:
Jian-Jia Chen
Kevin Wen-Hung Huang

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl XII
http://ls12-www.cs.tu-dortmund.de

Contents

Chapter 1

Introduction

1.1 Motivation and Background

The spread of infectious diseases has a massive increase in the last couple of years [12]. The main reason for this is that people travel daily, for example by plane, bus, car, ship etc. to other countries. Also the worldwide import and export supports the spread of diseases across countries. Wherever people go, it is possible to potentially receive unwelcome very little *presents* without immediately noticing. These *little presents* that are mentioned here, are viruses and only a few nanometers in size.

Figure 1.1: Virus under an electron microscope [1]

Figure 1.1 shows a single virus, which has a size of about 50 nanometers. Of course, some viruses are harmless, like regular flu viruses, but there are also some very dangerous kinds of viruses, like bird flu, swine flu or ebola. Although it is possible to analyze liquid samples, like blood or saliva in labs, this process usually takes a long time. And in some situations, for example at the airport it is important to detect infections fast. Every region has other specific viruses, like ebola which is widespread in Africa. If there is a plane arriving from a country struggling with an epidemic, it is crucial to quickly identify potentially infected passengers to prevent the epidemic to spread further.

There are already some methods to quickly detect viruses, for example the *Polymerase Chain Reaction* which is used to clone DNA-information to become more viruses in a sample

and by this to detect them easier. Another example is the *Enzyme-linked immunosorbent assay*. Thereby the virus stick to their antibodies on the layer. The best method is the detecting process with *PAMONO Biosensor* and the corresponding special software which is presented in detail in this work. It is an automatic method for detecting nano-objects in liquids. The measurement period to detect viruses is between 5 and 30 minutes as opposed to many hours in a lab. It is practically possible to detect single viruses [14]. The operation of this method is discussed in detail in chapter 2.

Summing up, with the PAMONO Biosensor it is possible to detect viruses quickly and to prevent an epidemic to spread. The main problem of this process is that the software needs a lot energy, because this sensor is a mobile sensor and uses battery power. The aim of this work is to analyze how much energy can be saved by offloading the processing data in order to evaluate them, for example on an external server.

1.2 Aim of the Project

The aim of the project is to find out if it is possible to save energy for Mobile Biosensors by offloading the obtained data to an external server and to process them there, instead of processing the data directly on the Odroid. To find out, if energy is truly saved by offloading the software will be firstly executed on the Odroid with different conditions, to be precisely this conditions are different parameter files. The software needs a parameter file, which contains specific parameter settings for the calculation algorithm. The parameter setting influences the quality of the measurement as well as the execution speed, because the parameters have to be adopted on the specific hardware environment. For this two different files were given. Chapter 5 is taking a closer look on the difference between them. One file is optimal for the Odroid and the execution process is very fast, while the second file is not optimal and the process takes considerably longer. The main difference between these files is es execution time on the Odroid.

After the measurements are performed on the Odroid, the offloading process is started. This will be done in two different ways. The first method is to offload the data from the Odroid to the server by Ethernet cable, whereby the Odroid is connected to the router. The second method is wireless offloading. For this a Wi-Fi module is used, which is explained in chapter 5. This module has an antenna, which can be used to get a wireless connection to a router and to offload the data that way.

When the offloading process is done, the values for energy consumption for offloading and the energy consumption for executing the software on the Odroid are examined. And by this is possible to analyze how much energy can be saved by offloading the data instead of processing them directly on the Odroid.

1.3 Structure of the work

The following section describes the structure of this work. It starts with the PAMONO application and its predecessor, the *Surface Plasmon Resonance (SPR)* method in chapter 2. Also the functionality, the advantages, disadvantages and the requirement of the sensor will be explained. After that the software which is needed to process the data obtained from the Biosensor is discussed in chapter 3. With respect to this software the used concepts are presented, especially the analysis process of the obtained data from the sensor. Furthermore, OpenCL is will be introduced, because the software is written in OpenCL to execute the software as fast as possible. OpenCL is an interface for parallel computing. Chapter 4 deals with the Odroid. It is a single board computer on which the software is executed. Thereby a closer look on the hardware properties of the Odroid will be taken. After the basic research has been presented, the experimental setup and the results of the measurement are discussed in chapters 5 and 6. Finally, chapter 7 gives a summary and an outlook on this topic.

Chapter 2

PAMONO Application for Virus Detection

This chapter focuses on the PAMONO application which is used for detecting specific viruses in different samples like blood or saliva. With respect to the PAMONO application the functionality of this Biosensor and the predecessor for this senor is introduced in section 2.1. Before the PAMONO application was developed, the *Surface Plasmon Resonance* (SPR) method was used for detecting viruses. At section 2.2 the advantages and disadvantages of the new application are presented and a comparison between the predecessor SPR and the PAMONO application is introduced. Finally, the last section 2.3 pays attention on the specific requirement of the sensor. The whole chapter is based on the documents [17, 14, 12].

2.1 Functionality of PAMONO Biosensor

PAMONO stands for *Plasmon-Assisted Microscopy of Nano-Objects* and is a new method for detecting viruses quickly. It is a further development of the *Surface Plasmon Resonance* (SPR) method, which was successfully contrived by the *Institut for Analytical Science* (ISAS). In fact the SPR method is a good way for detecting viruses, but the execution time was too long. More precisely for one process it takes between 30 and 60 minutes [14]. In order to shorten the execution time another method was needed.

The experimental setup of the SPR is similar to the PAMONO Biosensor and is shown in figure 2.1. There is a flow cell in which the sample, which has to be analyzed, is placed. The sample is composed of blood, saliva or any other liquid.

The sensor has a thin layer of gold with a size of about 50 μm. It is covered with one or more specific kinds of antibodies, more precisely with the corresponding antibodies of the viruses which have to be detected. After the sample is passed in the flow cell, the gold layer is illuminated with a laser. With a CCD sensor (*charge-coupled device*) the changing

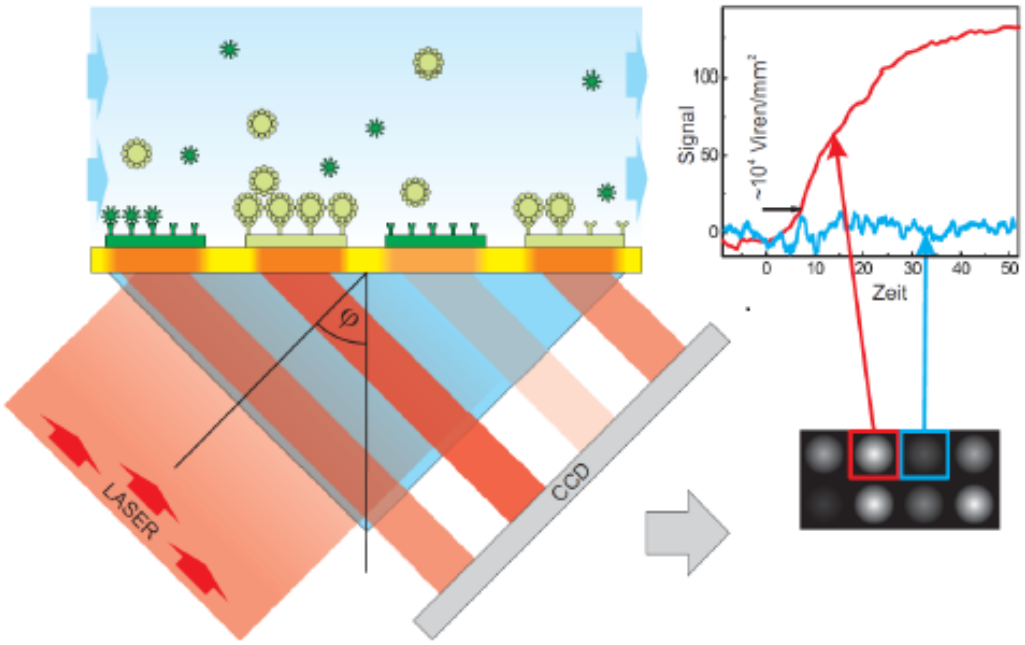

Figure 2.1: Experimental setup of SPR [14]

intensity of reflected light waves are recorded by the camera. For more information on the CCD sensor, refer to the document [18]. While the gold layer is illuminated, the viruses stick to their antibodies. The CCD sensor is able to detect viruses if the concentration is at least $10^4 \ \frac{viruses}{mm^2}$.

The diagram in figure 2.1 shows on the x axis the mapped time frame and on the y axis the mapped intensity of each pixel. In this context it has to be mentioned that during the processing each pixel is examined. If the intensity of one pixel rises up and stands clearly up in comparison to the other, it is an indication that a virus has been detected. At the diagram 2.1 the red curve shows the high intensity of the pixels. This suggests that a virus has been discovered. In contrast to this, on the blue curve the intensity of each pixel is not high enough, and because of that it is sufficiently certain that no virus has been discovered.

The main problem of this application is that viruses can only be detected after a certain concentration has been reached. By comparison the PAMONO sensor does not have this problem. With the PAMONO sensor it is practically possible to detect single viruses. The exact operation of this sensor is discussed in section 3.2. Other advantages of the PAMONO Biosensor will be presented in section 2.2. In order to remove some weaknesses of the SPR method, the PAMONO application has been developed. In figure 2.2 the experimental setup of the PAMONO Biosensor is introduced.

The experimental setup is described as follows:
The hoses are connected to the flow cell, in which the sample to be investigated is placed. A

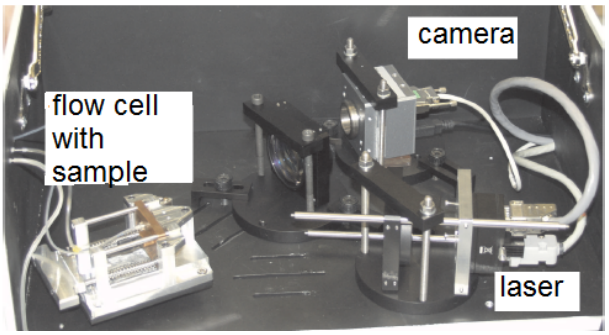

Figure 2.2: Experimental setup of the PAMONO Biosensor [14]

prism is attached at the bottom of the flow cell, its top surface is covered with a thin layer of gold. The laser on the left side under the prism illuminates the gold layer of the prism. The CCD camera on the right side records the reflected light. Additionally, a lens is added in front of the camera. For test purposes particle were added between a size of 40 and 280 nanometers in a sample. It turned out that even particles with a size of 40 nanometers were recognized. Furthermore, different cameras for taking the pictures can be used, but all cameras have the same color depth. An 8 Bit color depth is required. The sensor images, which were taken by the camera, are used for analysis. They are transferred to a special software which runs on a computer. It examines the sensor images. The examined data can be formatted in different files like pictures, videos and so on. The analyzing of the data is explained in section 3.2.

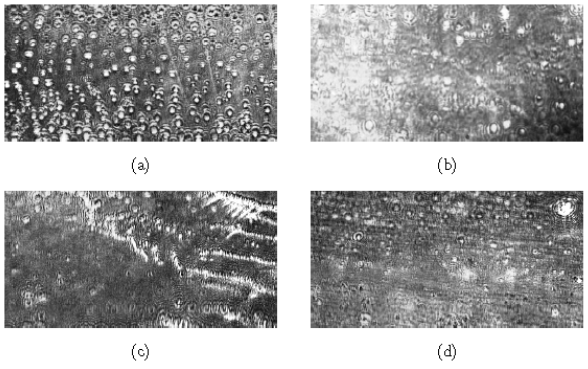

Figure 2.3: Examples for sensor images [14]

Examples for these sensor images are shown in figure 2.3. The gold layer is portrayed there, but it is obvious that the viruses are not visible with the naked eye on the images. In total, it can be seen that each picture has some stronger and some weaker brightness ranges. It is not possible to distinguish between a disturbance and a virus. With the aid of the software, which is introduced in 3.2, it is possible to distinguish viruses from disturbance even with the naked eye on the images.

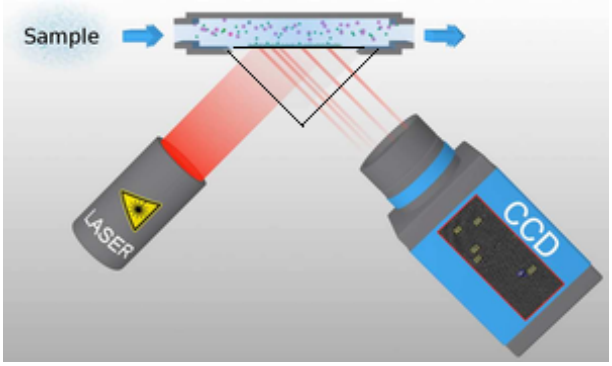

Figure 2.4: Working principle of PAMONO Biosensor [17]

In figure 2.4 the functionality of the PAMONO Biosensor is shown. The flow cell, in which the sample is placed in, is portrayed on the top. Before a sample can be analyzed, the gold layer of the prism has to be coated with specific antibodies, more precisely with the corresponding antibodies of the specific virus, which has to be detected. While the laser illuminates the gold layer and the sample passes through the flow cell the viruses stick to the corresponding antibodies. Through this process the light waves change. The CCD camera records the changing light waves and it is possible to detect whether viruses stuck to the antibodies.

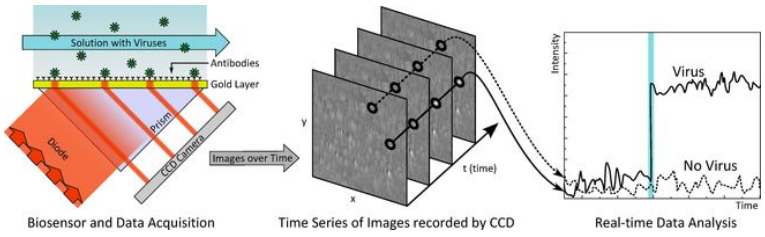

Figure 2.5: Detecting viruses by PAMONO Biosensor [11]

In figure 2.5 the functionality of the PAMONO Biosensor is shown once more and next to it the virus detection process is portrayed. The CCD camera records the reflected changing light waves by constantly taking pictures. The evaluation process happens as follows:

Each picture is considered independently by reviewing each pixel of the picture. The intensity of each pixel is measured and plotted in a diagram. On the x axis the time frame is mapped and on the y axis the intensity of each pixel. In case a virus is detected, the diagram shows a strong increase of intensity. Otherwise the intensity remains on the same level. Therefore, it is even possible to detect single viruses which is introduced in detail in the next section.

2.2 Comparison of SPR and PAMONO application

In this section the differences between the SPR and the PAMONO application are discussed. The main difference is the higher resolution at the PAMONO application. It is able to enlarge the particle between 1:5 and 1:7. This is achieved by a strong magnification of the particle. As a result, it is even possible to detect viruses with a size of less than 40 nanometers. In comparison, the magnification of the SPR method is just between 1:1 and 1:2. In contrast to the SPR method, the PAMONO application has the option to change the angle of the laser, which makes the detection more precisely. But, a detection is only possible if a concentration of 10^4 $\frac{viruses}{mm^2}$ is reached. The PAMONO application uses a CCD sensor as well as the SPR method to detect viruses. The higher precision enables to detect even single viruses by the PAMONO application. In the following figure 2.6 shows the differences of the SPR and PAMONO application.

	PAMONO	SPR
X-/Y resolution	0.5µm	30µm
insensitivity	very good	good
measurement duration	5-30 min	30-60 min
sample concentration	10^3- 10^4	10^8 - 10^{10}
types of viruses	to 400	to 400
continious control	yes	no
concentration	good	good

Figure 2.6: Comparison between PAMONO and SPR

Special emphasis should be placed on the high X- / Y-resolution. As already mentioned the magnification at the PAMONO application of the particle is much higher than at the SPR method. Furthermore, the insensitivity is excellent, which means that most disturbances were removed successfully. Also, the measurement time is between 5 and 30 minutes for the PAMONO application instead of between 30 and 60 minutes for the SPR method. The most interesting advantage of the PAMONO application is that viruses can be detect at a concentration of only 10^3 $\frac{viruses}{mm^2}$ and even single viruses in a sample can be recognized.

The last difference between the PAMONO and SPR application is the continuous control without interruption option of the PAMONO application. That means the sample is added evenly into the flow cell. Obviously the PAMONO and SPR application have some common properties. With both methods it is possible to detect up to 400 types of viruses and the needed concentration to detect them is considerable low and because of that both methods are in this relation good.

Even though the PAMONO sensor has a lot of benefits, there are also some disadvantages. The main problem of the sensor is the high energy demand. As already mentioned

it is very important to keep the running time short and the evaluation precisely, so the viruses can be detected quickly.

2.3 Requirement of the PAMONO Application

The PAMONO application is more precise than the SPR application. Therefore, a higher requirement for the evaluation process is needed. Especially the evaluation of the adhesion of the viruses represent a higher challenge, because viruses are recognized with brightness differences of the pictures. And as mentioned before each unevenness of the gold layer causes different brightnesses and it is difficult to distinguish between truly viruses and disorders. It is possible to detect single viruses with the PAMONO application through examining each pixel of the picture and because of this preciseness that challenge occurs. The SPR method just takes a look at the virus concentration in a sample. The software and virus detection process is will be presented in chapter 3. Another challenge, which arises, are some disturbances in the pictures, which have to be successfully mastered for a precise virus detection. Examples for these disorders are shown below.

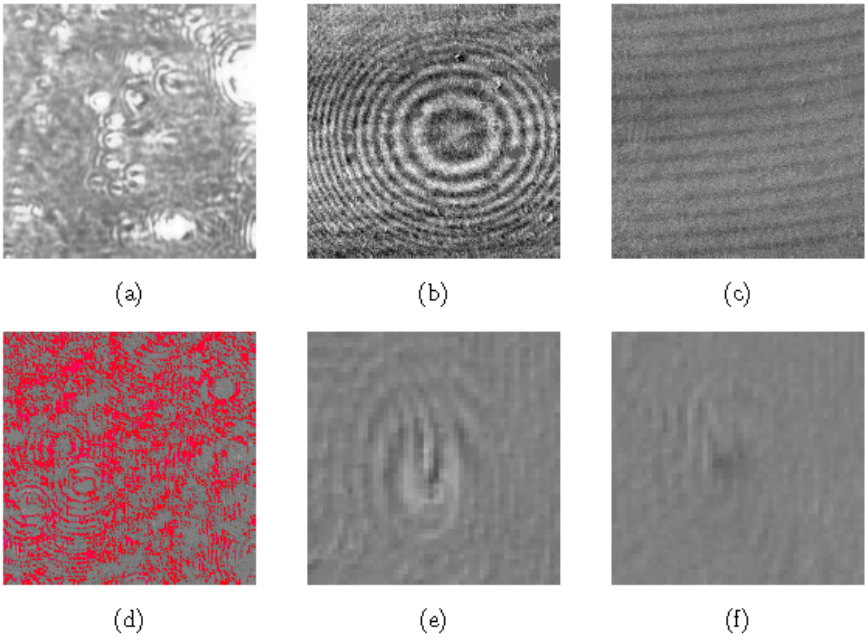

Figure 2.7: Disturbances of the sensor images [14]

Figure 2.7 shows different kinds of disturbances. In a) for example the sensor image shows a highly affected picture with disorders, which means that the viruses are overlapped and because of that the virus detection is impossible.

A disturbance of concentric circles is portrayed in b) and wavy pattern are illustrated in c). In d) signal noise, a typical problem of pictures, is demonstrated and illustrated through the red dots. In the last two sensor images dirt particles are portrayed. Such

disorders may include air bubbles or even dirt particles like dust. Even fine dust has a size of around 10 μm. All these disorders make it difficult to discover viruses.

As a summary it may be stated that the PAMONO application is more precise than the SPR method, but it has obviously more challenges on the evaluation process than the SPR method, because of the precision, effectiveness and the fast processing time. The solution for these problems will be presented in section 3.2.

Chapter 3

Processing of the Sensor Images

After the PAMONO application was introduced, the software, which processes the obtained data from the sensor, will be presented. The most important aspect is that the software is written in a language which supports massive parallel processing. Therefore, chapter 3.1 deals with OpenCL (*Open Computing Language*) which was used for implementing the software. In this section OpenCL and its functioning is summarized. All information are from the documents [14, 3, 26, 24, 10, 2]. The processing of the data is introduced in detail in section 3.2 and the last section 3.3 gives an overview of the exactly software process. These two sections are based on the document [14].

3.1 Introduction to OpenCL

Since it is important that the examination process is fast, the software has to run very quickly too. The images obtained from the sensor are very large, especially when taking into account that there are hundreds or even thousands of them [14]. To ensure that this quantity can be analyzed fast, it is recommended to use a language which supports massive parallel processing. In this case the language OpenCL is chosen and will be presented here.

The first version of OpenCL was published in August 2009 [26]. According to Benedict et al. [3], it is defined as follows:
"It is a heterogeneous programming framework which supports a wide range of levels of parallelism and efficiently maps to homogeneous or heterogeneous, single- or multiple-device systems consisting of CPUs, GPUs, and other types of devices limited only by the imagination of vendors". Furthermore, it is important to mention that OpenCL offers a device-side language as well as a host management layer. The device-side language is used for an efficient mapping to a wide range of memory systems and the host language supports a rapid execution of complex concurrent programs [3].

In the following the documentation deals with the advantages of OpenCL first and after that a closer look at the construction of the model structure of OpenCL will be taken.

Summing up, OpenCL enables an efficient and parallel *General Purpose Computation* on graphic cards [14]. It is also important to mention that OpenCL is platform independent and portable. This feature permits that OpenCL can run on different hardware, e.g. *Central Processing Units*, *Graphic Processing Units*, *Digital Signal Processors*, mobile units etc. and also on diverse operating systems, like Linux, Windows, MacOs etc. [14]. Furthermore, OpenCL offers the advantage of consistent numeric precision and accuracy. Also, a high number of given functions are available. These characteristics are useful for a fast virus detection, for example at the airport. Through all these properties OpenCL excels as a very efficient programming platform, outstanding in comparison to other massive parallel processing programming platforms, for example *OpenGL*.

The model structure of OpenCL consists of four different types of models:

- Platform model: defines the cooperation of host and the other OpenCL devices
- Execution model: defines the utilization of resources
- Memory model: defines the different memory areas of the OpenCL devices
- Programming model: defines the parallel processing

In the next chapter the models will be discussed exactly to accurately present the working principle.

3.1.1 Platform model

The platform model consists of one host and at least one OpenCL device. Thereby the host manages the executions on all the devices. Figure 3.1 shows the abstract architecture of the platform model.

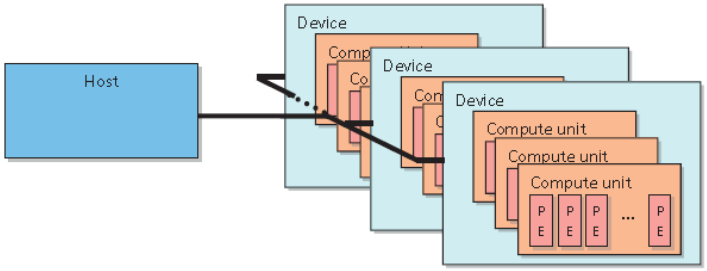

Figure 3.1: Platform model [3]

All these devices have several computing units. These computing units are further divided into more processing elements. The process takes places as follows:

While an OpenCL program, named *kernel*, runs on the host, the host chooses one or more devices on which the computation takes place independent from the rest. In principle the kernel includes the actual code, which is written in OpenCL.

The OpenCL program respectively kernel is spread to several compute units and on these one the program is divided into processing elements. On these processing elements the program runs in parallel independent from the rest. By this the computation is much faster, as if the program is executed sequentially. When the program is finished, the host can assign the next program to the devices and so on.

3.1.2 Execution model

The execution model gives an exact description of the OpenCL program execution. As already mentioned the program is divided up among several processors in order to execute in parallel and thus to keep the running time short. Each OpenCL program consists of two different parts: The host program and the kernels.

The host program manages the kernels and distributes them with the needed data on the OpenCL devices. It is also important to mention that not every OpenCL device is similar to the others. So the host divides the kernel corresponding to specific properties, for the number of compute units, clock frequency, memory storage and so on. Many kernels can be there and because of that the OpenCL devices need a command queue in which the data can be enqueued. These data can be *kernel executions, memory operations and synchronization operations* [10]. The OpenCL devices process the command queue step-by-step and an efficient and fast program flow is possible [24].

OpenCL C is based on the programming language C, but has a couple of differences. For example OpenCL C does not have the same standard functions like *printf()* or *malloc()*. Furthermore, it does not assist recursion or function pointers. On the other hand OpenCL has some additional functions that do not exist in C [24]. For more accurate information on this topic the reader is referred to the documentation [2].

Furthermore, the host defines a *context* which is an *abstract container* that coordinates the execution of the kernels, manages the *memory objects*, the *program object* and *kernels* that are created for each device [3, 26, 10]. For a detailed description of the execution model see [14, 26].

3.1.3 Memory model

The memory model defines the memory structure of OpenCL. The most interesting part on this occasion is the subdivision of the different memory models and the cooperation of them. In figure 3.2 the memory model of OpenCL is portrayed. Simply put if a device needs special data, these data have to be copied from the host to the global and constant memory, which are regions of the device memory. In the following the different memory models are discussed.

Figure 3.2 illustrates that the memory model is divided into two parts, Host and Device Memory. The differences between these two memories will be explained in the following:

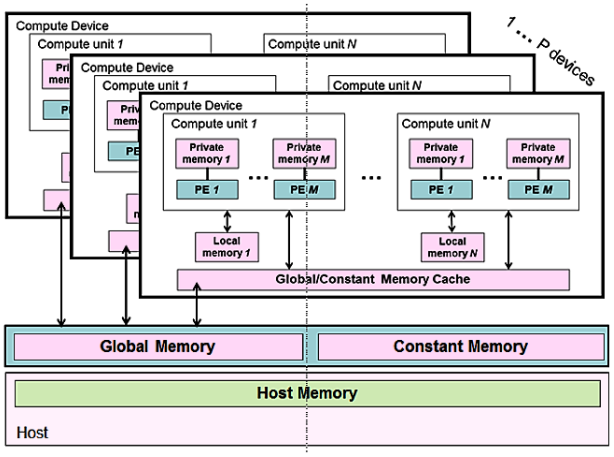

Figure 3.2: Memory model [2]

♦ The Host Memory:
The Host Memory works like a working memory, simply put, like a *RAM* (Random-Access Memory), and it is separated from the Device Memory. As a result every data the kernel needs has to be transferred to the Device Memory through a copy, an OpenCL API or through a virtual memory interface [2].

♦ The Device Memory:
The Device Memory represents the memory of each device. It should be noted that this memory is also divided into four different parts called *address spaces or memory regions* [2]. In the following those parts will be presented.

♦ The Global Memory:
This memory is comparable with the RAM and it is the biggest available memory, but also the slowest memory of all. Furthermore, it is visible for all compute units on the device. It is very important to mention that this memory permits read and write access to all work-items. If the data is transferred from the device to the host, every data is cached on the global memory.

♦ The Constant Memory:
The Constant Memory is a part of the global memory. The difference to the other memories is that the kernel-instances have a read access to all work-items, but not a write access. Memory objects which are allocated and initialized on the host, for example values which never have to be changed like π, are placed into the Constant Memory.

♦ The Local Memory:

This memory is local for every work-group. Every work-item of this work-group has the option to transfer data between the work-items and so it is possible to synchronize them during execution. That is the reason why it is important to have a shorter latency and a higher bandwidth than for the global memory. Summing up this memory can be used to allocate variables during the synchronization and by this to execute the program with the correct variables.

♦ The Private Memory:
This part of memory is used by work-items. Here variables can be defined individually for one work-item which cannot be seen from the other work-items. These variables can be mapped to registers.

To make the correlation between these kinds of memories the picture 3.3 is referred.

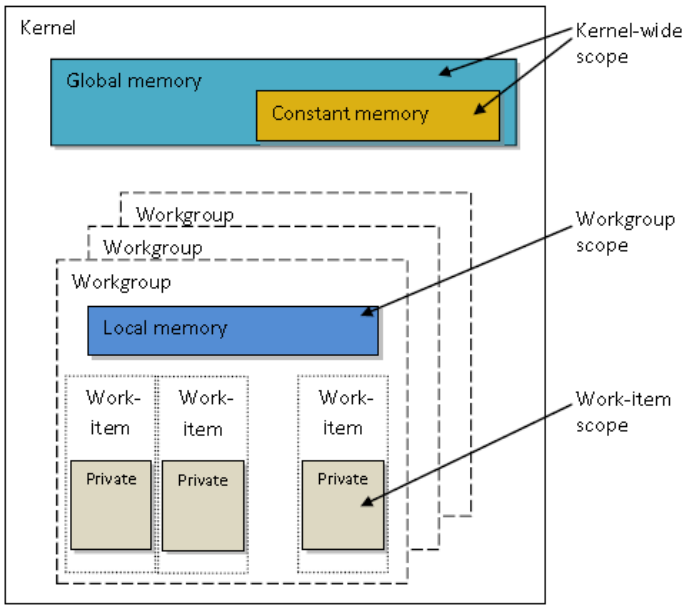

Figure 3.3: Memory model illustrated [3]

In summary, it is important to copy special data from the host to the global or local memory if a device needs them. Furthermore, it is an interesting aspect to know that the host has only read and write access to the global and constant memory, but not to the local and private memory. If the host needs data from these regions, they have to be copied to the global or constant memory, or in other words, if the kernel needs data, the host has to copy these data to the global or constant memory. After that the kernel can use these data on these memory regions or they have to be copied back to the private or local memory for processing. Every time data have to be copied, they have to pass the global memory. The constant memory, which is part of the global memory is used for constant variables like π

or for variables which only have to be calculated one time. The local memory deals like a *scratchpad memory* [18] and can be used for the memory devices of each work-group, while the private memory is used for every work-item. So in fact it is clear that each memory has a special task.

3.1.4 Programming model

The fourth and last model is the programming model. There are four general overview concepts for parallel processing. They are based on the *Flynnsche classification*, which are shown in table 3.1.

Table 3.1: Flynnsche classification [14]

	Single Instruction	**Multiple Instructions**
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

In the following the individual aspects of these concepts will be described. This information is taken from [14]:

♦ SISD: The program is executed serially and is working on a single data stream. For example a single-processor is working that way.

♦ MISD: The program is executed on multiple processors, but also only on a single data stream.

♦ SIMD: The program is executed on more processors and multiple data streams.

♦ MIMD: Each processor executes different instructions on different data.

3.1.5 Construction of an OpenCL Application

After the OpenCL models were explained in detail, a summary of the process execution follows. Listening 3.1 shows the process steps of an OpenCL application.

Listing 3.1: Approach of OpenCL application[24]

```
1. initialization of OpenCL
2. kernel source code compile
3. reserve global and constant memory for data
   transferring and copying data to the device
4. execute instances of kernels
```

5. copy the data back to the host

For further information to OpenCL the documents [3, 26, 24, 10, 2] were referred.

3.2 Workflow of the data

This section describes the function of the software. With this software the images which were obtained from the PAMONO application can be evaluated. The first part explains how the data, which were obtained from the sensor, are processed. The second part deals with the collection of the particle, in this case collection of the viruses, and the last part explains the follow-up phase of the data. After all these steps are performed it is possible to say whether or not a sample is contaminated with viruses. All that following information is taken from [14]. The pipeline in 3.4 illustrates the workflow process.

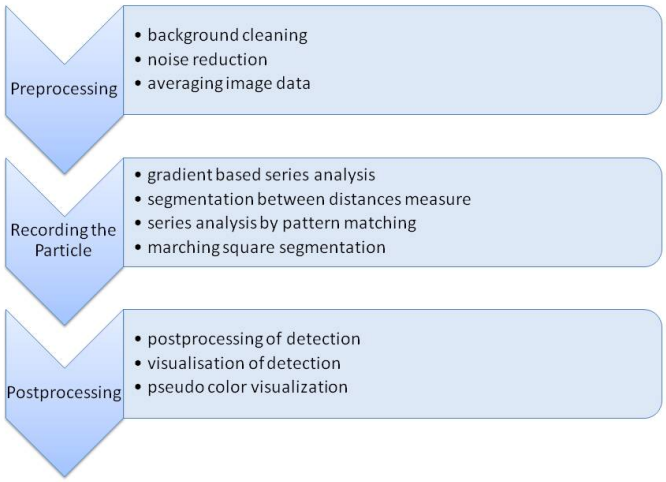

Figure 3.4: Pipeline of Workflow [14]

3.2.1 Preprocessing

Before it is possible to detect viruses, some intermediate stages have to be completed successfully first. There are two main problems that need to be solved. The first problem is that the surface structure dominates the brightness of the picture. And secondly the technology generates signal noises. Therefore, it is important to resolve these issues before it is possible to detect viruses. Otherwise, the detection process of the viruses would be extremely complicated. For example it would be difficult to distinguish between disturbance and true viruses. Simply put, these issues hide the virus adhesion. The following part describes the solution of these problems. The first section deals with the background cleaning, the second explains the reduction of the signal noise. The third one addresses

the segmentation of virus candidates and the last one explains how multicore processors accelerate processing.

Background Cleaning

The first problem (*surface structure dominates the brightness of the picture*) arises, because the surface of the sensor, more precisely the gold layer of the sensor, where the viruses should stick on, is not smooth. The objects, which have to be detected, are only a few nanometers in size and because of that even little irregularities change the intensity of the reflected light waves and wrong results occur. Precisely, unsmooth surfaces develop high brightness differences. After the sample has some viruses included the brightness differences are not that large anymore and it is actually possible to detect the virus adhesion with the naked eye. However, it is very difficult, because the differences are very small. To solve these problems the constant background cleaning is applied.

The constant background cleaning is executed once only, by summing up b pictures and average them. In this way it is accomplished to eliminate the signal noises in the background. After that the background remains constant.

After this first problem is solved, it is necessary to take care of the second one. Some structures of the gold layer background get visible in the picture, because it takes a few seconds to clean the background. That means additional to the particle of the sample also the background of the gold layer is illustrated. As mentioned earlier the gold layer is not smooth and while the laser illuminates it, it gets warmed up. Through the heat the structure of the gold layer gets visible on the pictures. As a result the truly virus attachment is overlapped by the background of the gold layer. To solve this problem a sliding background cleaning is needed. In contrast to the constantly background cleaning the background cleaning is executed in each time step. In this way it is possible to remove disturbances, until there are no more disorders and the pixel value remains constant. More information about that are given in chapter 4 from [14].

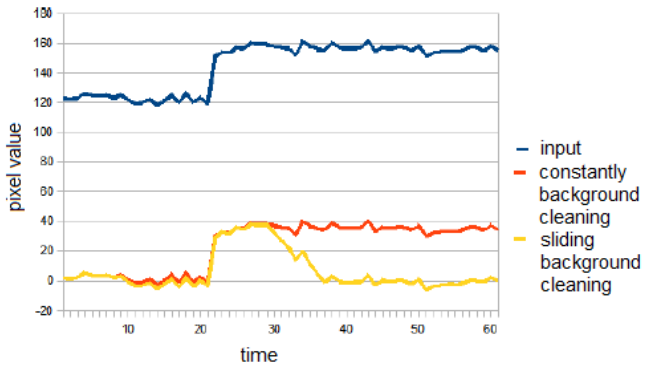

Figure 3.5: Background Cleaning [14]

Figure 3.5 shows the difference between the two methods of cleaning the background. The constant background cleaning just calculates the background once only and because of that the pixel values are postponed, but new disturbances are not removed. On the other hand the sliding background cleaning calculates the background every time step and because of that new disturbances can be removed instantly.

Signal Noise Reduction

The next important aspect of the data preprocessing is the elimination of signal noises. For each input signal noise arise. Figure 3.6 shows the visualization of signal noise of two successive frames.

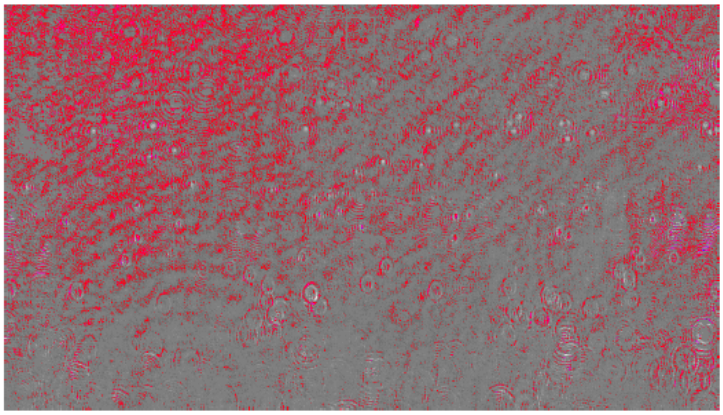

Figure 3.6: Signal Noise of two successive frames [14]

It can be seen clearly that the red areas, which visualize the signal noises, are not randomly. The largest part of figure 3.6 shows different structures, like circles, concentric semi-circles or lines. These structures result from the technology, for example from the chip of the camera. The remaining quantity constitutes from the disturbances of the time courses. Here no concrete structure is recognizable.

The signal noises can also develop from the rising sensor temperature. While the laser irradiates the sensor, the sensor set is warmed up and by this leads to a stronger signal noise [8]. In the following there are described two methods for signal noise reduction:

The first one is a simple method, which reduces the signal noises through averaging the pixel values of the pictures. This is made by a simple mathematical formula from [14].

$$I'_t(x,y) = \frac{1}{a} \cdot \sum_{i=0}^{a-1} I_{t-i}(x,y) \tag{3.1}$$

For this process only the position *(x,y)* of the pixel, the time t and the number of pictures a are needed. One example of this signal noise reduction is shown in diagram 3.7.

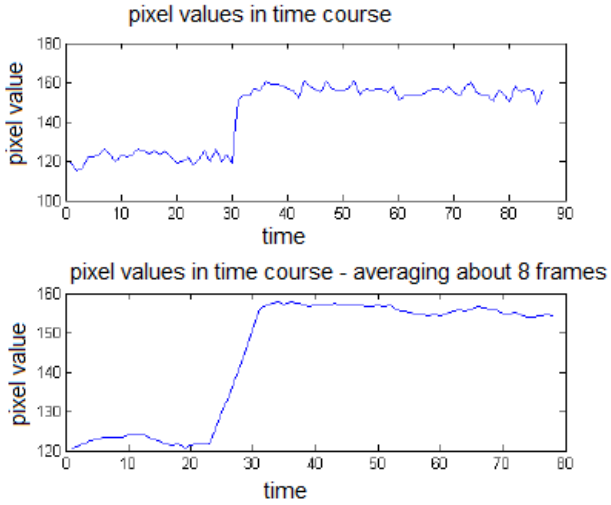

Figure 3.7: Signal Noise Reduction by averaging pixel values [14]

The upper diagram shows the pixel values in time course without signal noise reduction. Here the signal noises are clearly showing. At frame 30 a virus sticked to the gold layer and it is shown by rapid increase of intensity of the pixel values. In the lower diagram the averaging of the pictures is illustrated. It is recognizable that the signal noises are reduced, but also the strong jump is leveled. It is more difficult to recognize because of that at what point the viruses stuck.

The second method is more complex, but also offers a better reduction of the signal noises, so this method is used in the software. Here the signal noises are reduced by *Haar-Wavelets*. The main problem is that the signal noise reduction in the simple method also reduces the strong jump of the pixel values. But exactly this part should not be affected. In figure 3.8 the difference between the two methods is portrayed.

The upper one shows the pixel values in time course without signal noise reduction and the lower graph provides the graph after the Haar-Wavelets method was applied. It can be seen clearly that the signal noises are not reduced as strong as by the averaging method, but the strong rash is retained. Simply put, by this method the high frequency is being removed. This method works as follows:

The fact is that frequencies of signal noises are higher than frequencies of virus adhesiones. Therefore it is possible to remove only the high frquencies without influence in frequencies of the viruses. Instead of removing only the high frequencies, these one are become smooth and as a result the frequencies of the viruses produces strong hues. This is achieved by using the algorithms *One dimensional discrete Haar-Wavelet Transformation* and *Inverse one dimensional discrete Haar-Wavelet Transformation* which are illustrated and discussed in [14].

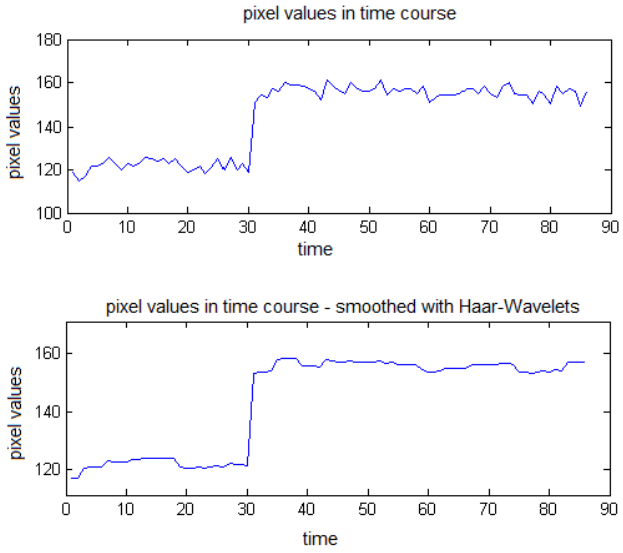

Figure 3.8: Signal Noise Reduction by Haar-Wavelets [14]

3.2.2 Recording of the particle

After the preprocessing is finished the recording of the particle starts. In this section three different methods of recording the particle are described, which mark pixel-based viruses which stick to the gold layer of the PAMONO sensor. The first method called *Jump detection with time series analyses and damped past* is the most simple alternative of detecting viruses. The second alternative called *Jump detection with time series analyses* uses recent pictures of the past to detect viruses. The last method is the most complicated. It is called *Time series analyses with pattern matching*. All three methods were developed for multicore processors to speed up the process.

The following pictures illustrate how the rapid arise of the intensity should look like. Figure 3.9 illustrates the value of a single pixel in time course. The strong increase of the intensity shows how a virus in the sample attaches to the gold layer of the PAMONO sensor. The used method will affect the amount of the increase in intensity. After the intensity got high and stand out of the remaining intensity, it remains at the same level. If the intensity falls off, it implies that a disturbance occurred. If no disturbance exists, the intensity does not fall off, because the intensity of the virus remains constant.

If a pixel is not just a disturbance, but truly a virus attachment, all the algorithms create a matrix for virus candidates. The matrix is a n $\times$ m matrix, where n and m stand for the width and the height of the matrix, respectively. If a candidate is found in position (x,y) it is marked with $k_{y,x} \neq 0$, otherwise it is just marked with 0.

It is also interesting to see how the virus detection is visualized in the picture. So in figure 3.10 it is illustrated what it looks like when the virus is seen. In part a) it shows

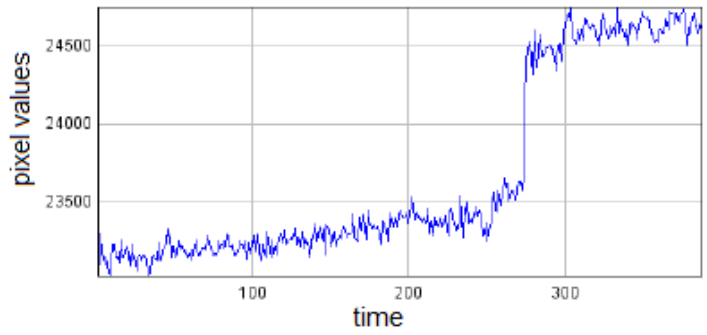

Figure 3.9: Virus attachment by time course [14]

one frame before the virus in the sample attaches to the gold layer. In part b) the virus attachment can be seen a little and in c) the virus attachment can be seen clearly. The size of the virus here is above 200 nanometers.

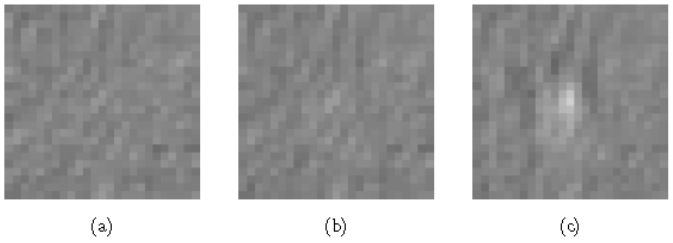

Figure 3.10: Virus attachment [14]

Jump detection with time series analyses and damped past

This method works on only one picture I_t for the current time *t*, the position *(x,y)* of the pixel and the damped past I_t^d. p describes a percentage value which expressed how strong the value of the past influences the new value.From the recursive equations 3.2 and 3.3 it is recognizable that the values of the past influence the new value less and less, because this one incorporates into the new value.

$$I_t^d(x,y) = p \cdot I_{t-1}(x,y) + (1-p) \cdot I_{t-2}^d(x,y), \text{ if t} > 0 \tag{3.2}$$

$$I_0^d(x,y) = I_0(x,y) \tag{3.3}$$

One example for p could be around 10 percent. This means that the actual value does not influence the new value much and that the past is more decisive. This ensures that the memory usage is kept as low as possible. To verify if a jump is truly a virus attachment it

is necessary to proof if the intensity of each pixel at this position stay constantly for the next pictures. For example, for the next 50 pictures this condition, if the intensity of this pixel stays constantly high, has to be verified for each time between u < t < (u+50) with the formula 3.4:

$$I_u(x,y) - (1,5 \cdot r) < I_t(x,y) < I_u(x,y) + (1,5 \cdot r). \tag{3.4}$$

On this occasion r stands for the signal noise and u is the time when a virus sticks to the antibodies. The signal noise is multiplied with the constant 1.5. The constant 1.5 is chosen, because of the quantization noise [13]. This represents the distance between the useful signal and the noise signal. If for example a signal is chosen that order curve is less step-like and signals with 1 and 2 are included, but the analog signal has the value 1.5, a rest remains. If the condition is true, the point (y,x) is added into the matrix for the candidates of a virus. All these three-dimensional images in 3.11 result from this method.

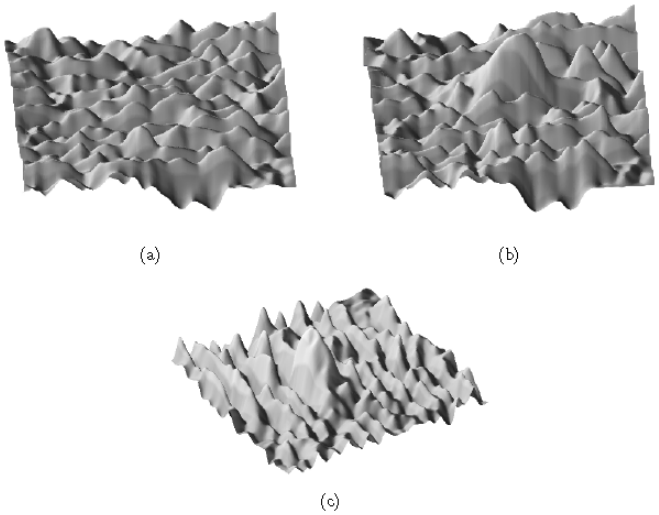

Figure 3.11: Virus attachment in three dimensions [14]

Part a) of 3.11 shows a picture without virus attachment. In b) the virus attachment is illustrated by the height of the mountains which show the intensity of each pixel and c) portrays the same as b) just by a different angel.

Jump detection with time series analyses

The next method for recording the particle is called *Jump detection with time series analysis* and only works with a section from the past. This section has a size of size of 4·a+b, whereby a and b stand for the dimensions of the picture. The main idea of this method is to look at 2·a frames before and after a possible virus attachment. After the process has finished and a virus stick to the gold layer, there are some conditions which have to

be checked. The surface of the gold layer has to be smoothed and the intensity of the pixel values has to be positive. To analyze the section of the past four averaging values are calculating for each time frame. Through these values it is possible to analyze this section. With the following formulas 3.5 - 3.8 these averaging values are calculated:

$$avg_1(x,y) = \frac{1}{a} \cdot \sum_{i=4a+b}^{3a-1+b} I_{t-i}(x,y) \tag{3.5}$$

$$avg_2(x,y) = \frac{1}{a} \cdot \sum_{i=3a+b}^{2a-1+b} I_{t-i}(x,y) \tag{3.6}$$

$$avg_3(x,y) = \frac{1}{a} \cdot \sum_{i=2a}^{a-1} I_{t-i}(x,y) \tag{3.7}$$

$$avg_4(x,y) = \frac{1}{a} \cdot \sum_{i=a}^{0} I_{t-i}(x,y) \tag{3.8}$$

After the main values were calculated, the following conditions 3.9 - 3.11 have to be reviewed. Thereby again is taking a closer look if the increase of the intensity stays for all further pictures constant high. The value r in the formulas for the condition describes the value of the signal noise again and can be held constant. As in the formula before for proofing the condition the signal noise is multiplied with the constant 1.5. The explanation for this constant is the same as at the section *Jump detection with time series analyses and damped past.*

$$|avg_1(x,y) - avg_2(x,y)| < 1,5 \cdot r \tag{3.9}$$

$$|avg_3(x,y) - avg_4(x,y)| < 1,5 \cdot r \tag{3.10}$$

$$\frac{avg_3(x,y) + avg_4(x,y)}{2} - \frac{avg_1(x,y) + avg_2(x,y)}{4} \geq 4 \cdot r \tag{3.11}$$

If the conditions are fulfilled, it can be presumed that a virus candidate exists.

Time series analysis with pattern matching

The last method uses a comparison of the pixel by time course with one or more master samples. Since the pixel value by time course shows a rapid ascent of the intensity in case of a virus attachment, this method uses this knowledge to compare the time row of a master sample with the received time row. In figure 3.12 an example for the comparison is visualized. It illustrates a candidate for a virus attachment, because it is clearly identifiable that the time row of the pixel matches the one from the master sample.

All three methods can be used for a virus detection. The first method (*Jump detection with time series analyses and damped past*) is the easiest way and optimal for a low memory

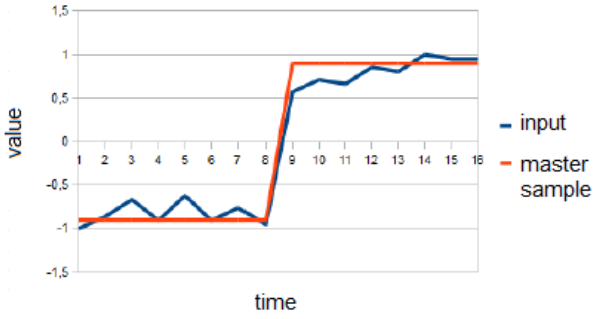

Figure 3.12: Comparison of pixel time course with master sample [14]

usage. The *Jump detection with time series analyses* and *Time series analysis with pattern matching* methods are more complex and because of that need more hardware resources. Each method has been optimized for a parallel processing. The processing of each pixel works independent from the others. Further information to the time row which has to be revised for a comparison to the master sample is described in [14] chapter 5.4.

3.2.3 Segmentation

Up to here each pixel was examined for a virus candidate. The next step is to bundle up the virus candidates that are located in the immediate vicinity to a structure. On the basis of the structure it is possible to classify the characteristics in particle or disturbance. To do this, two algorithms will be presented in this section. On the one side there is the *Scanline Algorithm*, which scans the picture pixel by pixel and maybe finds a structure of a particle and on the other side the *Marching Squares Algorithm*, which identifies the structure by polygons. The Scanline Algorithm and the Marching Squares Algorithm are based on [15] and [14].

Scanline Algorithm

As already mentioned the Scanline Algorithm examines each picture pixel by pixel and tries to condense the pixels, which were recognized as a virus candidate, to a segment. The algorithm can be explained by the following figure 3.13.

At the beginning the virus candidates are illustrated in green in part a). The Scanline Algorithm starts scanning the picture line by line. When a virus candidate is detected, which is shown in part b) and c) as a red point, this point is condensed either to an existing segment or a new segment is created. In part d) the result of the Scanline Algorithm is shown and all virus candidates have their own segment. It can be clearly seen that this method is very simple and has some disadvantages. It is working parallel and because of that each thread will be processed by an own working group (see section 3.1) it can

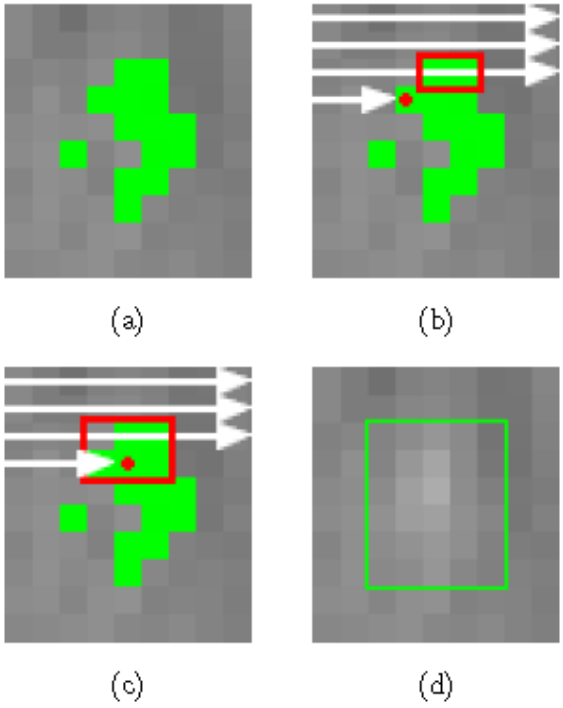

Figure 3.13: Scanline Algorithm [14]

happen that some segments are separated, although they should not be. Furthermore, a closer look should be taken at edge cases. Further information for managing this task will be described in chapter 6.2 from [14].

Marching Squares Algorithm

The Scanline Algorithm gives a rough structure of the viruses and because of that it is not possible to get a clear classification. To solve this problem, the Marching Squares Algorithm was developed. This section is based on chapter 6.3 from [14]. The algorithm process is illustrated in the following figure 3.14.

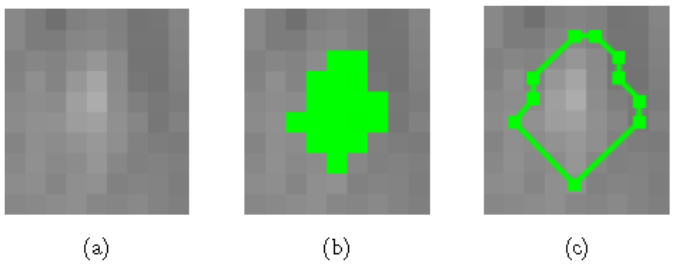

Figure 3.14: Marching Square Algorithm [14]

First the algorithm needs a n × m matrix, which gets processed by the algorithm from chapter 3.2.2, to know on which position the virus candidate stuck. These positions will be marked green on the picture, which is shown in part b) of 3.14. By this it is possible to construct closed polygons by which the structure can be classified in virus candidate or disturbance. Each polygon should be made of at least four points as a minimum and can be described by this formula:

$$P := (P_0, P_1, P_2, ..., P_{n-1}) \text{with } P_i \in \mathbb{R}^2 \text{ ,} 0 \leq i \leq n\text{-}1, n \geq 3 \text{ and } P_0 = P_{n-1}. \quad (3.12)$$

The algorithm works by processing two steps. The first one is to find out in which direction the polygon should be constructed and the second how the polygon is exactly built. Further information on this process are described in chapter 6.3 from [14].

3.2.4 Postprocessing

The last step of the software is the postprocessing. The segments are separated into virus candidate and non virus candidate. This is based on the structure and size of the segment. Also a few methods will be described how the virus candidates can be visualized so that the human eye can recognize the difference between virus and disturbance. Although it is truth that an algorithm is more exact than the human eye, it is also important to verify this algorithm. Before this algorithm was developed, these pictures were examined pixel for pixel by hand. Through the visualization of each virus it is possible to pursue the working process of the software. This section is based on chapter 7 from [14] .

Classification of Segments

Virus candidates should be segmented because disturbances can change the brightness as well as viruses and only through the brightness it is impossible to detect if this is a virus or a particle. If it is a virus candidate the segmentation has a certain structure by which a virus can be classified. The difference between virus and disturbance is shown in figure 3.15.

The images a) and b) illustrate viruses while the rest show disturbances. It is typical that viruses have a certain structure, for example circles or ellipses while disturbances have a random shape. They are crescent and stretched. As mentioned in chapter 3.2 particles and disturbances also cause a rise of the brightness. For a better representation the contrast of the picture is boosted. To do this the values range has to be stretched. It is necessary, because the virus attachment is taken place in a very small value range and so it would be impossible to see the virus attachment with the naked eye. This can be archived by the following formula 3.13:

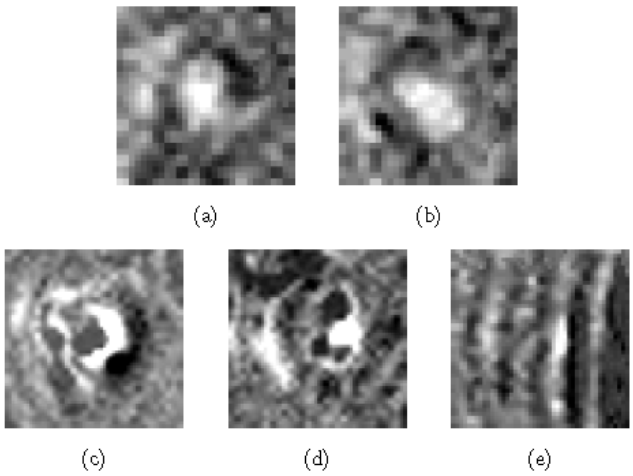

Figure 3.15: Comparision between virus and disturbance [14]

$$I_t'(x,y) = \frac{max_{out}}{max_I - min_I} \cdot I_t(x,y) + \big(max_{out} \cdot \frac{-min_I}{max_I - min_I}\big)[14]. \tag{3.13}$$

With this formula 3.13 it is possible to stretch an interval from [$\min_I$, $\max_I$] to [0,$\max_{out}$]. So it is possible to visualize the virus.

Pseudo Color Reproduction

All the images which were produced by the PAMONO sensor always have a gray tone recording which is a huge problem, because the human eye is not able to see subtle differences and because of that, it is important to transfer this gray tone into a color model. Here the HSV color model was chosen. Figure 3.16 presents the HSV color model.

In the model 3.16 it is not necessary to know about the proportions of red, green and blue. The colors are presented by the angle, where 0° and 360° represents red, 120° green and 240° blue. The stronger the saturation, the stronger is the intensity of the color. With the following formula 3.14 it is possible to calculate the angle to the color.

$$H = \begin{cases} (x \cdot (360° - H_S + H_E) + H_E) \mod 360° & \text{, for } H_S > H_E \\ (x \cdot (360° - H_S + H_E) + H_E) & \text{, else} \end{cases} \tag{3.14}$$

The variable H stands for color angle here. H_S describes the starting values while H_E stands for the ending value. Both variables are within the required interval [0° - 360°]. The last variable x entries within the specified interval [0-1] for one input value. What it looks like when the grey tone is transferred into a HSV color will be presented in figure 3.17.

After this transformation, it is much easier for the human eye to see the differences between viruses and disturbances. Figure 3.18 portrays the picture after processing. The

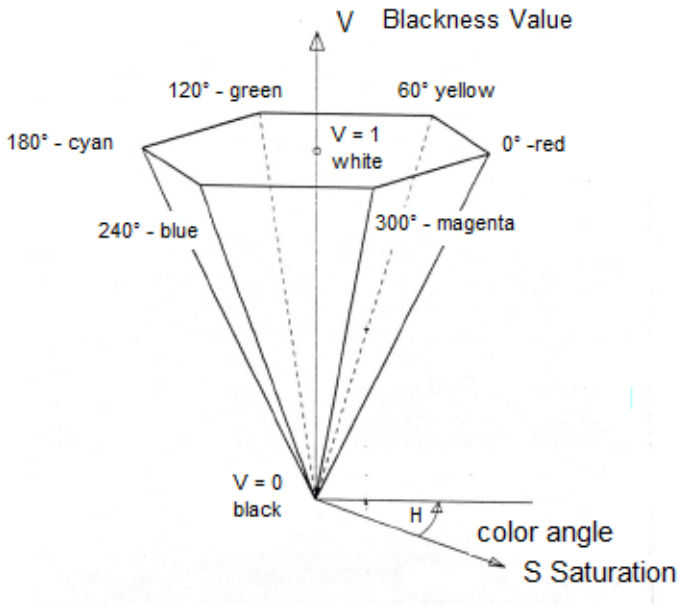

Figure 3.16: HSV colour model

Figure 3.17: Grey tone into HSV colour model [9]

yellow points with red border in the second quarter of the picture are virus attachments. The half circles with red or magenta border could be blurred virus attachments or disturbances.

3.3 Software

Taken together, all the strategies presented here were transcribed in a software. A general overview of these concepts are portrayed in 3.19.

The input for the software can either be images, a video or a camera. This input has to be decoded before the virus detection process can start. The exactly process was discussed in chapter 3.2. After this process an online visualization or a segmentation can be done or the images can be encoded. After this the obtained data from the sensor are evaluated. See [14] chapter 8 for a closer look at the software implementation.

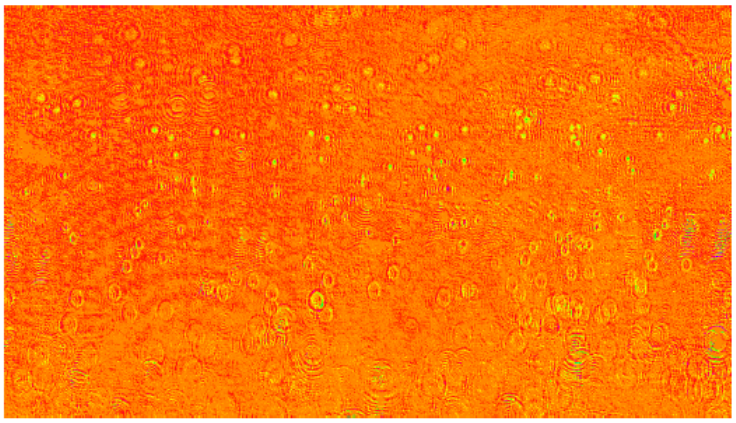

Figure 3.18: Result after Processing [14]

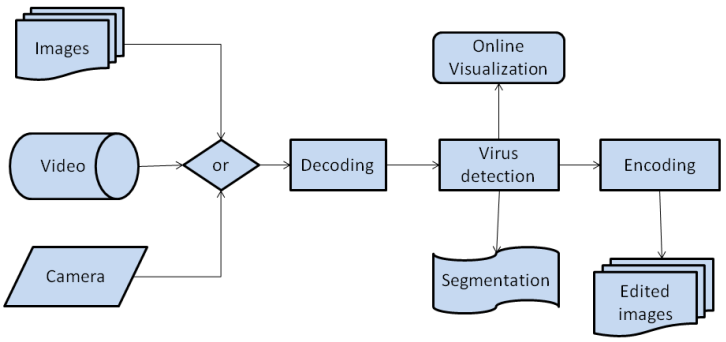

Figure 3.19: General overview of the Software

Chapter 4

Odroid

This chapter introduces the Odroid hardware on which the Software will run. Before the Odroid type XU 4 is described in section 4.2, the differences between a Single Board Computer and a typical PC are introduced in section 4.1. Odroid stands for *"Open + Android and it is a development platform with hardware as well as software"* [7]. It is a Single Board Computer and was developed by the company *Hardkernel*. In total 15 types of Odroid were manufactured, but only the types XU 4 and C 1+ are still being produced [6]. An Odroid offers diverse possibilities, for example it can be used *"as a home theater set-top box, a general purpose computer for web browsing, gaming and socializing, a compact toll for collage or office work, a prototyping device for hardware tinkering, a controller for home automation, a workstation for software development, and much more"* [7] page 6. The first Odroid came onto the market in fall of 2009 and the newest type will come onto the market in spring of 2016 which is the Odroid C2. In the last part of this chapter the SmartPower module will be introduced. With this device the power consumption can be calculated. The following sections are based on the references [7], [5], [6], [25] and [19].

4.1 Single Board Computer vs. typical PC

Although a Single Board Computer can be used as a typical PC, there are a few small differences between them. A personal computer has a motherboard which has a processor, a random access memory, a read only memory, a hard disc and so forth. A Single Board Computer also offers all these components, but with one difference. The motherboard on the Odroid is much smaller in comparison to the motherboard on a typical PC. The difference is that the motherboard on the Odroid does not offer further slots for extension boards. As already mentioned, a Single Board can also be used as a personal computer, because it also offers a normal operating system, which is highly optimized. Although there is a difference between the speed of a typical Intel processor and the ARM processor which is used on a Single Board Computer, with the right hardware it is possible to reach nearly

the same speed as a personal computer. These processors are also very powerful and have high clock rates. The Raspberry Pi 2 Model B, which is on the market since February 2015 has a clock rate up to 900 MHz [27]. But also there are more powerful models, like the Odroid XU 4 which offers a clock rate up to 2 GHz [6]. A hardware comparison between the different types of Single Board Computers is shown in 4.1. The boot partition of the Single Board Computer can be stored by a simple micro-SD card. The Odroids by Hardkernel, for example the Odroid XU 4, supports the much faster eMMC module. By this it is also possible to switch between operating systems much easier. Only a switch has to be flipped depending on whether the Odroid should boot from the eMMC module or the SD card. The next interesting aspect for comparison between Single Board Computer and typical PC is the energy consumption. A personal computer uses between 100W and 1000W or more, but an Odroid XU 4 for example only uses between 10W and 20W [7]. Summing up Single Board Computers are not only energy-saving, they are also cheaper and more flexible than personal computer. That makes them great choices for new development in all sectors, for example testing new applications.

	ODROID-C2	ODROID-C1+	RPi 2 Model B
CPU	Amlogic S905 SoC 4 x ARM Cortex-A53 2GHz 64bit ARMv8 Architecture @28nm	Amlogic S805 SoC 4 x ARM Cortex-A5 1.5GHz 32bit ARMv7 Architecture @28nm	Broadcom BCM2836 4 x ARM Cortex-A7 900MHz 32bit ARMv7 Architecture @40nm
GPU	3 x ARM Mali-450 MP 700MHz	2 x ARM Mali-450 MP 600MHz	1 x VideoCore IV 250MHz
RAM	2GB 32bit DDR3 912MHz	1GB 32bit DDR3 792MHz	1GB 32bit LP-DDR2 400MHz
Flash Storage	Micro-SD UHS-1 @83Mhz/SDR50 or eMMC5.0 storage option	Micro-SD UHS-1 @78Mhz/SDR50 or eMMC4.5 storage option	Micro-SD @ 50Mhz/SDR25 No eMMC storage option
USB2.0 Host	4 Ports	4 Ports	4Ports
USB2.0 Device / OTG	1 Port for Linux USB Gadget device or USB host	1 Port for Linux USB Gadget device or USB host	No
Ethernet / LAN	10 / 100 / 1000 Mbit/s	10 / 100 / 1000 Mbit/s	10 / 100 Mbit/s
Video Output	HDMI 2.0 4K / 60Hz	HDMI 1.4	HDMI 1.4 / RCA / DSI
Audio Output	HDMI / I2S	HDMI / I2S	MDMI / 3.5mm Jack / I2S
Camera Input	USB 720p	USB 720p	MIPI CSI 1080p
Real Time Clock	No (unless using an add-on module)	Yes (on-board RTC)	No (unless using an add-on module)
IR Receiver	Yes (on-board IR sensor)	Yes (on-board IR sensor)	No (unless using an add-on module)
IO Expansion	40 + 7 pin port GPIO / UART / I2C / I2S / ADC	40 + 7 pin port GPIO / UART / SPI / I2C / I2S / ADC	40 pin port GPIO / UART / SPI / I2S
ADC	10bit SAR 2 channels	10bit SAR 2 channels	No (unless using an add-on board)
Heat sink	Included	Included	Optional
Size	85 x 56 mm (3.35 x 2.2 inch)	85 x 56 mm (3.35 x 2.2 inch)	85 x 56 mm (3.35 x 2.2 inch)
Weight	40g (1.41oz)	40g (1.41oz)	42g (1.48oz)
Price	$40	$37	$35

Figure 4.1: Hardware Comparision between different Single Board Computers [6]

4.2 Odroid XU4

The following section is describing the Odroid XU 4 in detail. All the including components will be presented. This section is based on the references [7] and [5]. In figure 4.2 the Odroid XU 4 is portrayed. This type of Odroid is on the market since summer 2015 and is described

as: "The world's most affordable ARM Octa-Core big.LITTLE high-performance board computer" [5].

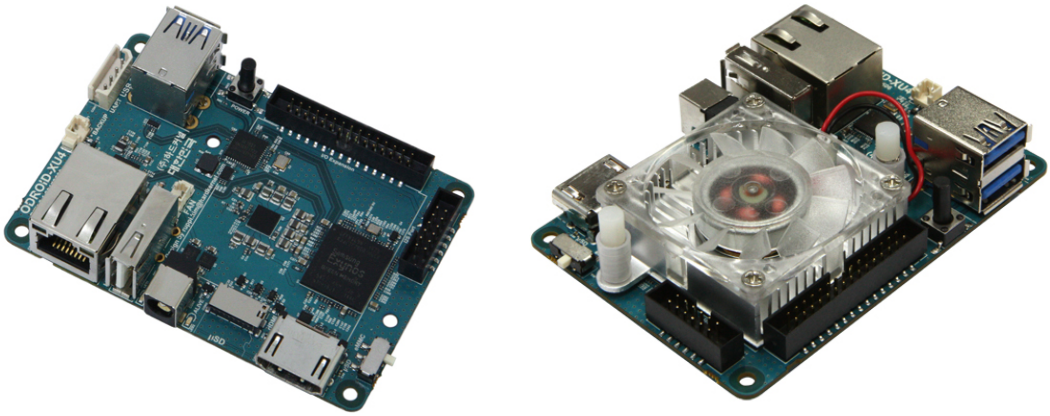

Figure 4.2: Odroid XU 4 [6]

The Odroid XU 4 is smaller than the types before, to be precise it has a size of half a credit card and because of that it is very easy to transport. Furthermore, it has a more powerful hardware compared to its predecessors. It offers new operating systems like some versions of Linux, e.g. Ubuntu 15.04, and also the two Android versions KitKat and Lollipop. The most important point is the excellent data transfer rate this version offers. By the eMMC module, the USB 3.0 and the Gigabit Ethernet interface this version has the fastest data transfer compared to all his predecessors. In diagram 4.3 a comparison between the standard SDcard and the eMMC module is shown. The SDcard or the eMMC module is needed for booting an operating system. But it can clearly be seen how much faster the Odroid works with an eMMC card. Furthermore, there are also some kinds of SD cards like the SD-class 10 or SD-UHS1. The difference between these two memory cards takes place in the writing and reading speed. In comparison with the SD-class 10 and SD-UHS1 the writing speed with the eMMC module is four to five times faster. The reading speed is much higher. It is nearly four to seven times faster as the reading speed of the SD-class 10 and the SD-UHS1. The measured values for the read and write speed at the USB type are also quite different. The write and read speed of the USB 3.0 is about five times faster as of the USB 2.0.

The next diagram 4.4 illustrates the streaming speed between the XU 4 and his processors XU 3 with 100 Mbps on board and XU 3 External 1Gbps. In this context, the term *streaming speed* means how long it takes to transfer a collection of digital data through a transmission channel. A comparison is made between the offload and download time, more precisely when the Odroid functions as a client and when the Odroid is functioning as a server.

As clearly can be seen the Odroid XU 4 functions faster than its predecessor XU 3 as server as well as a client. If the Odroid functions as a server the XU 4 is between two

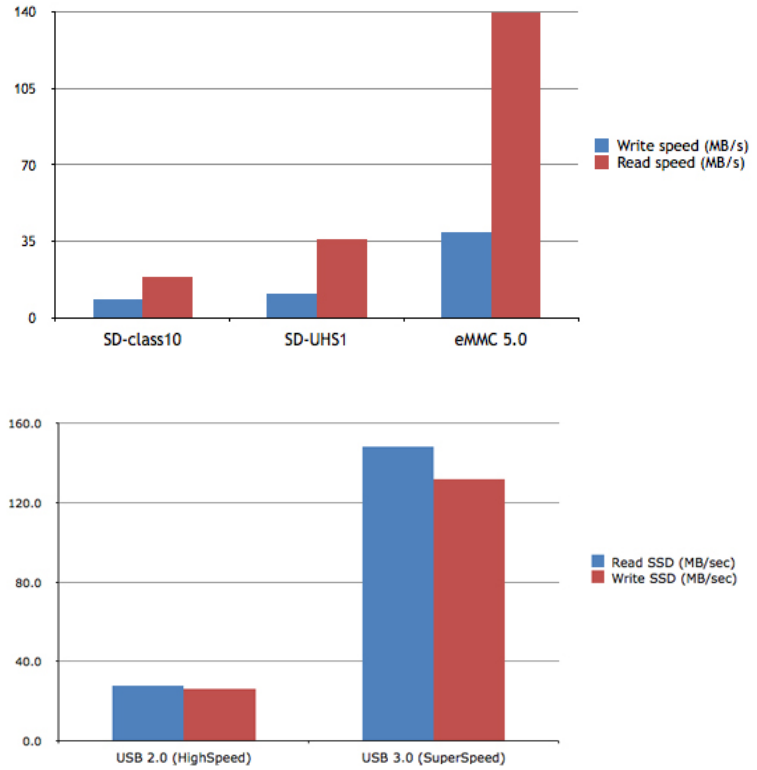

Figure 4.3: Comparison between the SDCard and eMMC 5.0 and USB 2.0 and USB 3.0 [6]

and more than seven times faster than his predecessor. If it functions as a client it is also between more than one and more than seven times faster. The Odroid XU 4 has a streaming speed of about 880 Mbps.

As seen in figure 4.5 and 4.6 the processor of the Odroid XU 4 is a Samsung Exynos5422 with four Coretex^{TM}-A15 2 GHZ and four Cortex^{TM}-A17 Octacore 1.4 GHz CPUs with Mali-T628 grafic card. The four Coretex^{TM}-A15 are used for computer intensive operations, while the four Cortex^{TM}-A17 are used for less energy demanding operations. By this it is possible to save power. If necessary the suitable kernels are used in order to use it as sparingly as possible. Furthermore, the XU 4 offers a 2 GByte DDR3-RAM as a memory and as indicated above it can also use eMMC 5.0 or a simple micro SDcard in different sizes. The size of the memory expansion goes from 8GB to 64GB. And as mentioned earlier the reading speed of the eMMC is up to seven times faster than the reading speed of the micro SDcard. The XU 4 can switch which module it wants to use with a hardware switch. It also has a 750 MHz clocking and a 12 $\frac{GB}{s}$ memory bandwidth with 2$\times$ 32 bit bus. This Odroid needs a $\frac{5Volt}{4Ampere}$ DC power source. In the standby operation it uses about 1 Ampere and while operating the current consumption rises above 3 Ampere. Further, the Odroid has two USB 3.0 Host ports and one USB 2.0 Host port. As mentioned before, the USB 3.0 port is about five times faster than the USB 2.0 port and offers all the same features that

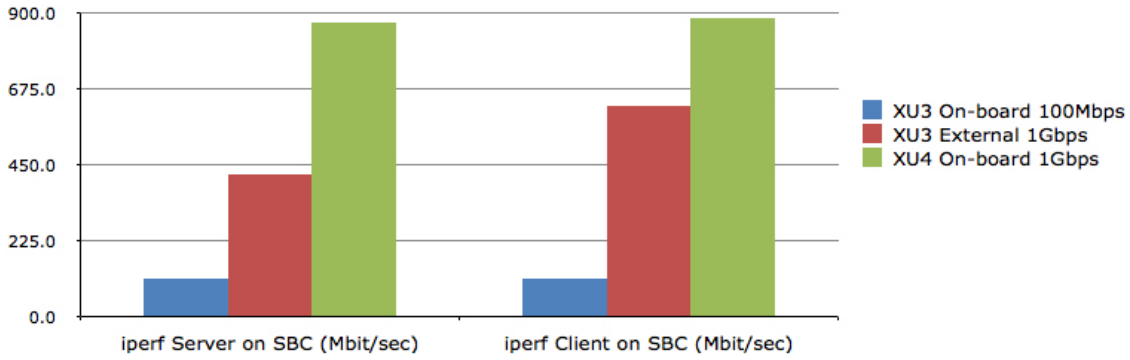

Figure 4.4: Comparison of the ethernet performance between the Odroid xU 4 and his predecessor XU 3. For the comparison two different versions of the XU 3 were used, a XU 3 with 100 Mbps on board and XU 3 External 1Gbp [6]

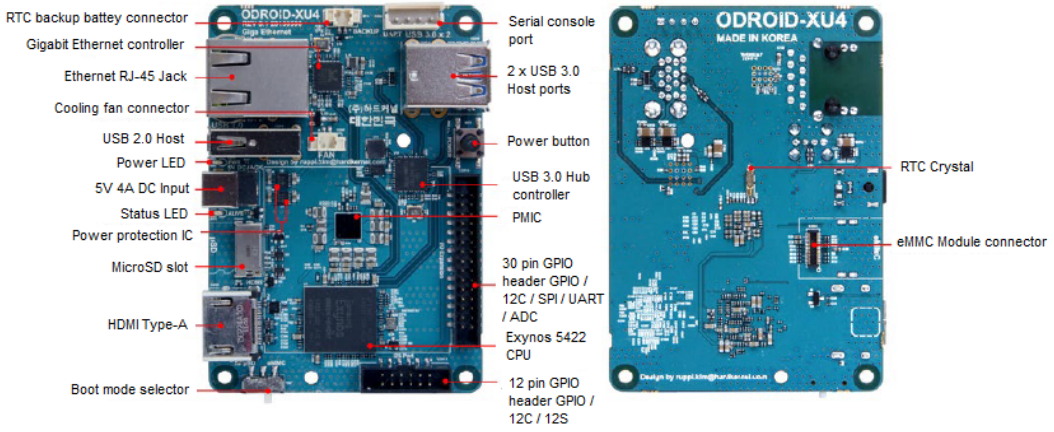

Figure 4.5: Annoted Board Image [7]

a typical PC has, too. For example it is possible to connect a Wi-Fi adapter and by this get a connection to the Internet or other devices as keyboard, mouse, or even an external USB hub. By this it is possible to have more than three USB ports. In addition, it also offers a standard type-A HDMI connector to connect it to a display. Furthermore, XU 4 has a installed Ethernet RJ-45 jack for LAN connection. It offers different speeds. There are 10 / 100/ 1000 Mbps available. To know with which speed the Odroid is connected to the Internet it has LED flashes. If the LED flash lights green the speed is 100Mbps and if it is yellow the speed is up to 1000Mbps. As shown in the diagram 4.4 the streaming speed of the Odroid is about 880 Mbps. With a Wi-Fi Module, which is illustrated in figure 4.8 a connection to the Internet can be established. The XU 4 also offers a USB-UART Module Kit which allows the user to connect the Odroid to a laptop and by this to view the bootloader and to change the network and video settings. Of course the XU

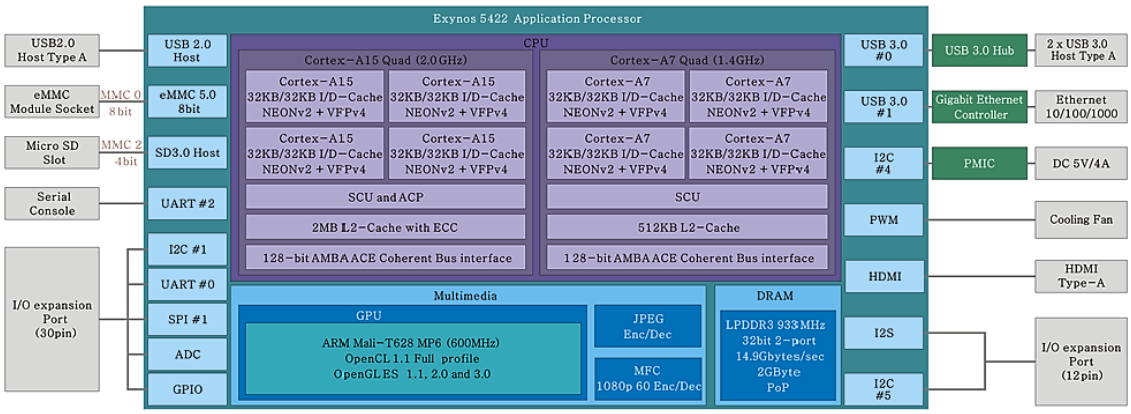

Figure 4.6: Block Diagramm of XU 4 [7]

also provides General Purpose Input and Output (GPIO) ports. Altogether there are two different GPIO ports. One has 30 pin and the second one has 12 pin. A GPIO port is a more general contact pin on an integrated circuit [4]. It can be used as input or as output, which is determinate by logic programming. This contact usually does not have a purpose and because of that it is by default idle. All these pins have a 2mm space and are operated with 1.8V DC. They can be used as an interface for other physical devices, but if the user does not need any further devices it is not necessary to control them. The important point here is that the ADC inputs are limited to 1.8 Volt and if a peripheral needs a higher voltage a XU 4 Shifter Shield 4.7 is needed. By this it is possible to rise the voltage up to 3.3 Volt or even to 5 Volt.

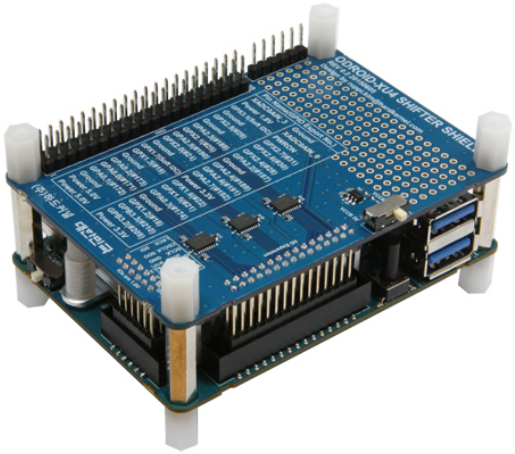

Figure 4.7: Shifter Shield for Odroid XU 4 to get a higher voltage [5]

Another aspect of the XU 4 is that it can be used on battery power. It offers a RTC (Real Time Clock) backup battery connector 4.9. By this it is possible to work with it while the Odroid is not connected to a power socket and in total the Odroid can run up to three years by this. This battery is a CR2032 $\frac{3Volt}{220mAH}$ and uses as connector a Molex

51021-0200. The Odroid XU 4 is not the only Odroid which has a battery power supply. The directly forerunners XU 3 and XU 3 Lite, the C1 and C1+, U3 and XU / XU Lite all offer this option.

Figure 4.8: Wi-Fi Module for XU 4 [5]

Figure 4.9: Backup Battery for the Real Time Clock for Odroid XU 4 [5]

Another problem which is known from typical PCs is that heat builds up while the system operates. Especially the XU 4 reaches up to 95° C. The XU 4 uses a cooling fan to solve this problem which is illustrated in 4.10. To prevent damages the processor will throttle itself if the temperate passes a critical threshold value. The processor does not work with full power then and the efficiency gets lost. With the cooling fan it is possible to regulate the temperature and the processor can still work with full power.

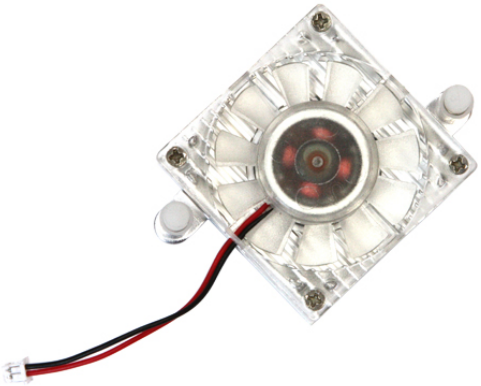

Figure 4.10: Cooling Fan for XU 4 [5]

Summing up, a Single Board Computer has all the technical possibilities that a typical PC also offers, it is just much smaller and cheaper. The Odroid XU 4 dimensions are 82 × 58 × 22 mm. As shown in figure 4.1 Single Board Computers like the Odroid XU 4 can be bought for as little as 74$. In all, they are a very good choice for testing and system

Processor	Samsung Exynos5422 ARM® Cortex™-A15 Quad 2.0GHz/Cortex™-A7 Quad 1.4GHz
Memory	2Gbyte LPDDR3 RAM PoP (750Mhz, 12GB/s memory bandwidth, 2x32bit bus)
3D Accelerator	Mali™-T628 MP6 OpenGL ES 3.0 / 2.0 / 1.1 and OpenCL 1.1 Full profile
Audio	HDMI Digital audio output. Optional SPDIF optical output (USB module)
USB3.0 Host	SuperSpeed USB standard A type connector x 2 port
USB2.0 Host	HighSpeed USB standard A type connector x 1 port
Display	HDMI 1.4a with a Type-A connector
Storage (Option)	eMMC module socket : eMMC 5.0 Flash Storage (up to 64GByte) MicroSD Card Slot (up to 64GByte)
Fast Ethernet LAN	10/100/1000Mbps Ethernet with RJ-45 Jack (Auto-MDIX support)
WiFi (Option)	USB IEEE 802.11b/g/n 1T1R WLAN with Antenna (USB module)
HDD/SSD SATA interface (Optional)	SuperSpeed USB (USB 3.0) to Serial ATA3 adapter for 2.5"/3.5" HDD and SSD storage
Power (included)	5V 4A Power
System Software	Ubuntu 15.04 + OpenGL ES + OpenCL on Kernel LTS 3.10 Android 4.4.2 on Kernel LTS 3.10 Android 5.1 is available as a community driven OS development. Full source code is accessible via our Github.
Size	82 x 58 x 22 mm approx. (weight: 60gram including cooling fan approx. 38gram without cooler)

Figure 4.11: Hardware components of Odroid XU 4 [5]

development. A general overview of all the components that the Odroid XU 4 offers is shown in figure 4.11.

4.3 SmartPower

To do measurements for the Odroid, especially power and electric current intensity, a special module was developed: ***SmartPower***. By this it is possible to measure the voltage *(Volt)*, the current in real time *(Amper)*, the power of the system *(Watt)* in real time and also the power consumption $(\frac{Watt}{hour})$. In figure 4.12 the Smart Power module is portrayed. This module can be connected to a computer or laptop with an USB cable and with the PC application a graphical representation can be displayed. It is possible to measure either the current or the power in real time. In figure 4.13 the diagram for the current graphical representation in the left picture is illustrated and the diagram for the power use is portrayed.

The Smart Power module is compatible with many types of Odroids, like the XU, XU 3 and XU 4 with the DC plug cable (5.5mm / 2.1mm) or with the DC plug cable (2.5mm / 0.8mm) for Odroid X, X2, U2, X2, U3 and C1. In the block diagram 4.14 the components of the Smart Power are illustrated.

The Smart Power module works on DC $\frac{12Volt}{3Ampere}$ input power and offers an output voltage between 3 Volt and 5.250 Volt. The maximum output current is 5 Ampere. As already mentioned, with the Smart Power it is possible to measure the voltage, the current and power in real time and also the used power per hour. The small tolerances of these results can differ from the real values around 2 percent. Besides the USB device port for a

Figure 4.12: Smart Power module for measurements with Odroids [5]

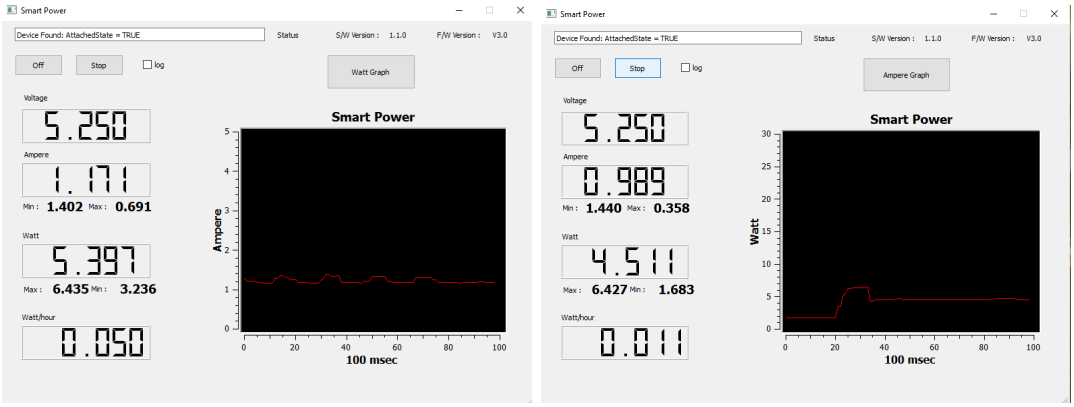

Figure 4.13: PC application for Smart Power. The left picture illustrates the current graphic and the right picture the watt graphic of the Smart Power module.

connection to a PC the Smart Power device also has 2 buttons. With the right button it is possible to start and stop the measurements and the left button just turns it on and off. With the button it is possible to set the desired voltage. The measurements can be done as follows: The Smart Power module is connected to the power outlet and the Odroid to the Smart Power. Further on, a laptop is wired on the Smart Power, so the PC application can be started on the laptop.

In total, this chapter introduced the Odroid with all its specifications and also the hardware accessory for the Odroid.

4.4 Conclusion

Further it is also an important aspect why especially an Odroid was chosen for this topic. It could also be possible to chose a typical PC or any other Single Board Computer, e.g. the *Raspberry Pi* and to measure the power and electric current intensity there. In comparison

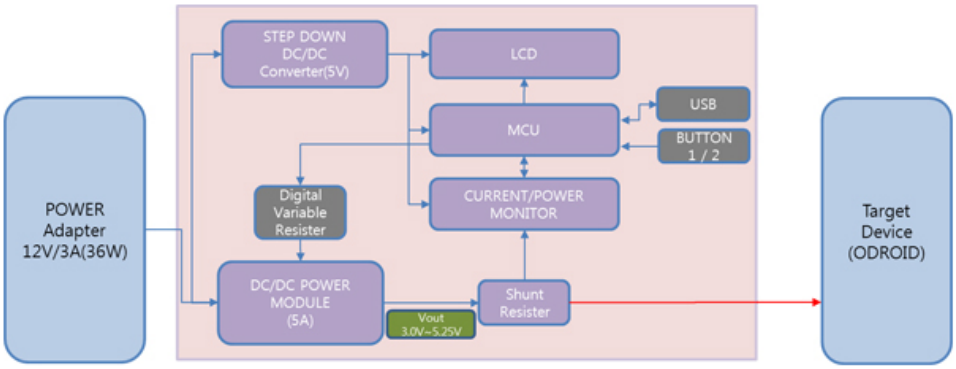

Figure 4.14: Block Diagram for the Smart Power device

to the Single Board Computer the Odroid sticks out because of its hardware opportunities. But not only this sticks out. Single Board Computer are in comparison to typical personal computers much cheaper and are therefore well-suited for experimental developments. In comparison to a Raspberry Pi [23] for example the here used Odroid XU 4 has a RAM with 2GB, instead of 1GB RAM for the *Raspberry Pi 3 Model B*. The processor which is used at the *Raspberry Pi 3 Model B* is the *Broadcom BCM2837* and the Odroid XU 4 uses the *Samsung Exynos5 Octa ARM Cortex*^{TM}*-A15 Quad*, which is more powerful. After these comparisons were made for the Hardware opportunities it turned out that the Odroid was the best opportunity for this topic.

The next question is why the Odroid XU 4 was chosen and not any other kind of them. The Odroid XU 4 is the one of the newest kinds of Single Board Computers from Hardkernel. The XU 4 is a further development of the XU 3 and the full software of the XU 3 ist compatible with the XU 4, but the XU 4 is more compact, more affordable and more expandable [6]. In total, this type has an energy-efficient technology and because in this work the power and electric current intensity should be under examination, this kind of Single Board was chosen.

Chapter 5

Experimental Setup

After the Software for virus detection and the Odroid hardware were introduced, this chapter deals with the experimental setup. As discussed earlier the measurements of the power consumption will be executed with the Smart Power module which was presented in detail in chapter 4.3. The first part of this chapter 5 explains which necessary preparations have to be taken to run the software. The second section presents the hardware preparation for this experiment. In the third section 5.3 of this chapter the experimental setup for the measurements on the Odroid is presented and a closer look at their realization is taken. The last part 5.4 of this chapter focuses on the offloading process. To be precise, how the data, which were obtained from the PAMONO application, were offloaded on a server and how much energy the Odroid needs for this process.

5.1 Preparation

As mentioned earlier, there are various ways to process the data from the PAMONO sensor, for example different ways of single noise reduction, time series analysis or segmentation. And also the results of a chosen algorithm are depending on parameter settings, e.g. the maximal merging distance, the detection threshold, usage of brightness correction and a lot more. In order to analyze the effcts of these differences, two different parameter documents were given. Firstly the optimized parameter and secondly the unoptimized parameter. The reason why some values are optimal and other not for the Odroid is dues to the hardware of the Odroid. As already shown in chapter 4 there are some differences between the hardware of a typical PC and the hardware of an Odroid. Furthermore, it is important to mention that the effect of the differences of these files is mainly its influence on the execution speed. As it will be seen in the measurements later, with the optimized values the software runs much faster. Another important point is the evaluation of these two different parameters. The optimized parameter found 108 viruses in the data which were obtained from the sensor, while the unoptimized parameter only detected 105 viruses. This

is no huge difference, so overall it can be said, that it may be stated that the difference between these files has a major on the execution time, but only a minor effect on the precision of the evaluation. In figure 5.1 the result of these processes is visualized. This has a lot to do with the chosen parameter file. The explanation for this is that the values for the not optimal parameter sum up some virus candidates to just one virus and because of that it seems that less viruses have been detected. In the Appendix A.1 the complete values of these two documents are portrayed and can be compared for their differences. The differences between them are for example the values for the *merging maximal frame distance for polygons* which is for the unoptimized parameter 17 and for the optimized 15. Furthermore, some parameters have different values, for example *the merging for the maximal distance* is for the unoptimized value *6.102633* and for the optimized value *3.499158*. But there are more differences. For example the *brightness correction* is activated at the unoptimized values, but not activated at the optimized values. The main difference between these two data is that the optimized values are adapted for the Odroid, and the unoptimized are not.

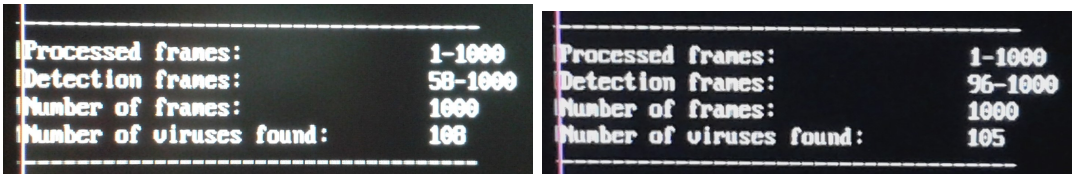

Figure 5.1: In the left picture the result of the software for the optimized values and in the right picture the results of the software for the unoptimized values is realized.

5.2 Hardware Preparation for the experiment

The experimental setup is presented in figure 5.2. Firstly the Smart Power module is connected to the power socket and then to the Odroid. Furthermore, the laptop is connected to the Smart Power, so the Smart Power application can run on it. Then, a display is plugged into the HDMI port of the Odroid and the keyboard is attached via USB port. A mouse is not necessary, because the used SD card has no graphical operating system. As mentioned earlier, just like a normal PC the Odroids slowly heats up while running. To ensure that the Odroid works most efficient, a script which regulates the fan of the Odroid, was prepared. This script is introduced at the beginning ot the next section.

Overall there are two different methods to offload the data from the Odroid to the server: The first method is to connect the Odroid to the router with a LAN cable to get an Internet connection that way. The second method is to set up a wireless connection with the router. For this the Odroid Wi-Fi Module 4 is used. It is portrayed in figure 5.3 [5].

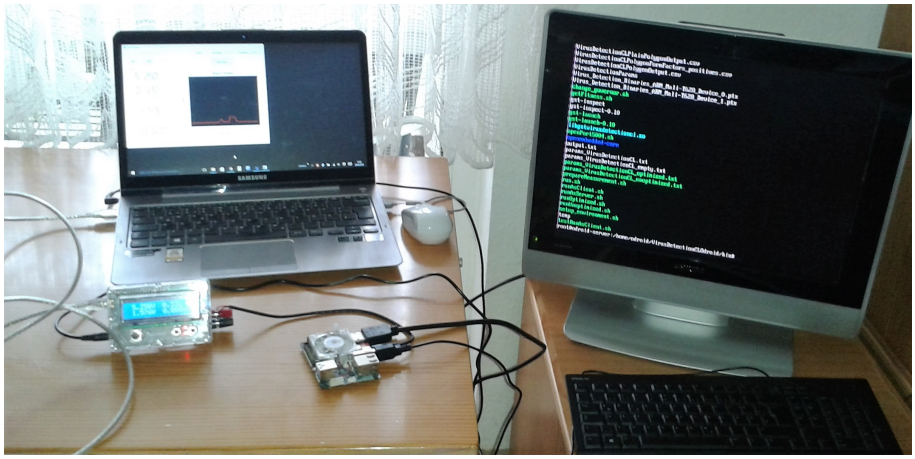

Figure 5.2: Experimental setup of the measurements

Figure 5.3: Odroid WIFI Module 4 [5]

The Wi-Fi Module 4 serves as an antenna to get a wireless connection to the router, to be precisely it is an IEEE 802.11a/b/g/n WLAN module with on-board 2.4Ghz and 5Ghz Dual band antenna [5]. The board image of this module is illustrated in figure 5.4.

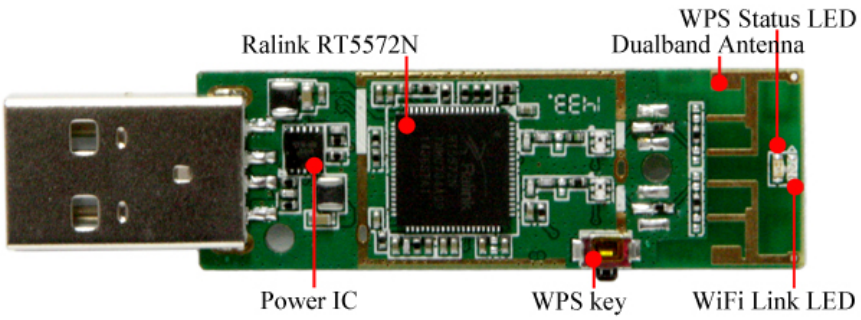

Figure 5.4: Board Image of the Wifi Module 4 [5]

It is shown that the Wi-Fi Module 4 has a Ralink RT5572N chipset and a dual band antenna. Furthermore, it has a WPS status LED, a WPS key and a Wifi link LED. The dimension of the module is 68.5 x 22.8 x 8.5 mm and it only weighs 12 g. This module can be used on different types of Odroids like the ODROID U3 / XU3 / C1 / XU4 / C0 / C2.

5.3 Measurements on the Odroid

While running the software on the Odroid heat arises and with the script in Listening 5.1 the fan can be regulated.

Listing 5.1: prepareMeasurements

```
sudo service lightdm stop
sudo /etc/init.d/cups stop sudo
echo 0 > /sys/devices/odroid_fan.14/fan_mode
sudo echo 255 > /sys/devices/odroid_fan.14/pwm_duty
sudo ./change_governor.sh f a7
sudo ./change_governor.sh f a15
if pgrep -x gst-launch-0.10; then sudo killall
gst-launch-0.10 -e -q -w; fi
if pgrep -x sh; then sudo killall sh -e -q -w; fi
```

It is possible to have automatically the fan start every time a given temperature has been reached or it can be triggered manually. It is also possible to set the fan speed. All measurements, which have been done, were done by setting the fan manually on full speed. In fact, the speed was set to 255. The results of these measurements will be introduced in chapter 6.1.

To get comparable values in relation to the power and electric current intensity, the software first has to be executed only on the Odroid and after this the obtained data from the sensor were offloaded to an external server for further processing. For th executing process on the Odroid two scripts were written. One script is using the optimized values and the other one the unoptimized. The script for the optimized values is shown in 5.2 and the script for the unoptimized values is illustrated in 5.3.

Listing 5.2: RunOptimized

```
#!/bin/sh
#Environment variables
export PATH=./:${PATH}
export LD_LIBRARY_PATH=../lib/:${LD_LIBRARY_PATH}
export GST_PLUGIN_PATH=../Plugins
taskset -c 0,1,2,3, ./RunVirusDetection -paramsFile
params_VirusDetectionCL_optimized.txt resultFilesOutputFolder=.
sourceStartIndex=1 num-buffers=-1 sourcePath=../images/training
sourceFileName=capture2013-04-11- sourceFileNameDigitCount=4
sourceFileExtension=png forceRecompile=0 saveOpenCLBinaries=1
silent=0 visualize=0 -useQueues queueMaxSizeBytes=52428800
```

Listing 5.3: RunUnoptimized

```
#!/bin/sh
#Environment variables
export PATH=./:${PATH}
export LD_LIBRARY_PATH=../lib/:${LD_LIBRARY_PATH}
export GST_PLUGIN_PATH=../Plugins
taskset -c 0,1,2,3, ./RunVirusDetection -paramsFile
params_VirusDetectionCL_unoptimized.txt resultFilesOutputFolder=.
sourceStartIndex=1 num-buffers=-1 sourcePath=../images/training
sourceFileName=capture2013-04-11- sourceFileNameDigitCount=4
sourceFileExtension=png forceRecompile=0 saveOpenCLBinaries=1
silent=0 visualize=0 -useQueues queueMaxSizeBytes=52428800
```

Obviously the only difference between these two scripts is the used parameter file. The other parameters are equal. Both scripts analyze the same pictures obtained from the PAMONO sensor. For fine tuning of the measurement process, more different parameters can be set. For example, it is possible that the analyzing time of each pixel can be displayed to setting the parameter *"ProfilingCPU"* or *"ProfilingGPU"* to true. With setting *"silent"* to true the output function is being deactivated. With the function *"-useQueues queueMaxSizeBytes"* the maximum size of the queue can be adjusted. The software process of the data was explained in detail in chapter 3.2.

5.4 Measurements by Offloading from Odroid to a Server for Computation

After the electric current intensity *(Ampere)* and power consumption $\frac{Watt}{hour}$ of the software on the Odroid were determined, it is interesting to find out how much power and electric current intensity the Odroid needs to offload the data to an external server for executing. For this a server was provided on which the data were offloaded. To realize this, two further scripts were given. One script has to run on the server and the other one on the Odroid. The script in Listening 5.4 describes the work steps of the server and the script in Listening 5.5 the offloading process of the Odroid.

Listing 5.4: runAsServer

```
#!/bin/sh
#Environment variables
export PATH=./:${PATH}
export LD_LIBRARY_PATH=../lib/:${LD_LIBRARY_PATH}
```

```
export GST_PLUGIN_PATH=../Plugins
./RunVirusDetection -paramsFile
params_VirusDetectionCL_unoptimized.txt
resultFilesOutputFolder=. sourceStartIndex=1 num-buffers=-1
sourcePath=../images/training sourceFileName=capture2013-04-11-
sourceFileNameDigitCount=4 sourceFileExtension=png
forceRecompile=0 saveOpenCLBinaries=1 silent=0 visualize=0
-useQueues queueMaxSizeBytes=52428800 -udpReceiver
udpReceiverCaps="application/x-rtp, media=(string)video,
clock-rate=(int)90000, encoding-name=(string)RAW,
sampling=(string)RGB, depth=(string)8, width=(string)706,
height=(string)167, colorimetry=(string)SMPTE240M,
payload=(int)96, ssrc=(uint)2490757164,
clock-base=(uint)3577112882, seqnum-base=(uint)14889"
udpReceiverPort=5004 cameraInputWidth=706
cameraInputHeight=167 cameraInputBPP=16 cameraInputDepth=16
```

The data from the Odroid were offloaded to the server with the script *runAsServer*. The server has already the parameter file with the settings for the calculation. Further, the server needs the information what kind of data is offloaded. This may include more for example the depth, width and height of the offloaded media. To realize the offloading process the *udpReceiverPort* must be set up. In this case the port 5004 was used.

Listing 5.5: runAsClient

```
#Environment variables
export PATH=./:${PATH}
export LD_LIBRARY_PATH=../lib/:${LD_LIBRARY_PATH}
export GST_PLUGIN_PATH=../Plugins
./gst-launch-0.10 -v multifilesrc
location="../images/training/capture2013-04-11-%04d.png"
index=1 num-buffers=-1
%caps="image/png,framerate=(fraction)60/1,
pixel-aspect-ratio=(fraction)1/1" ! %pngdec !
%"video/x-raw-gray,bpp=16,endianness=4321" ! ffmpegcolorspace !
%%%rtpvrawpay ! udpsink host=129.217.43.72 port=5004
```

The Odroid functions as the client and offloads the data with the script *runAsClient* to the server. For the process the folder has to be indicated in the script. Furthermore, the IP address of the server has to be initialized and the port has to be equal the one in the script for the server.

Chapter 6

Evaluation

After the experimental setup in chapter 5 was described, the measurement results will be presented. First, the measurements were done on the Odroid, which means that the electric current intensity and power consumption was measured while running the software on the Odroid. And after this a closer look is taken on how much electric current intensity and power the Odroid needs to offload the data to the external server. The measurements on the Odroid are presented in 6.1 and the measurements of the offloading process in section 6.2. A distinction is made here according to the needed power and electric current intensity for offloading the data via LAN-cable and via wireless. The last part of this chapter is summing up the most important aspects of these measurements.

6.1 Measurement results on the Odroid

The software, which was presented in chapter 3.2, was running on the Odroid with the two different kinds of parameters, the optimized parameter and then the unoptimized parameter. The Table A.2 which is in the Appendix represents the measurement results for the optimized parameter. Thereby different aspects were examined. One aspect is how much additional power and electric current intensity the Odroid needs when unnecessary equipment is connected to the Odroid, during the software is executed or the data were offloaded to the server. Here the software was running in two different cases. The first one is with a connected keyboard and the second is without the keyboard. This comparison is made to see the difference in power consumption on the Odroid with respect to plugged and unplugged keyboard. In this experiment the static power consumption of the keyboard is observed, because the keyboard was not used while the measurements were running. In this way it is ensured that with unplugged keyboard only the electric current intensity and power consumption of the software is considered. In order to calculate the power consumption and electric current intensity of the software, it is also necessary to know how

much electric current intensity and power the Odroid used in standby mode. This is shown in figure 6.1.

It can be seen that the Odroid runs with a constant voltage of 5.250 Volt. Furthermore, it used approximately between 1.56 Watt and 1.57 Watt and about 0.33 Ampere in the standby mode. The last line with the unit $\frac{Watt}{hour}$ indicates how much power in total the Odroid uses per hour. This display only shows the value if the Smart Power is started. Although only the usage of power and electric current intensity for one moment is displayed.

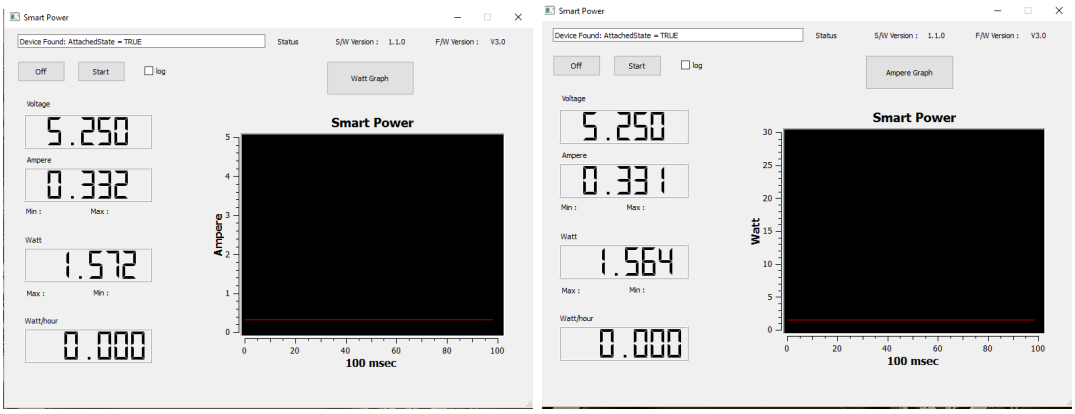

Figure 6.1: Power and Energy consumption of the Odroid in standby mode. In the left picture the electric current intensity is shown and in the right one the power consumption.

As mentioned earlier, the obtained data from the sensor were processed with two different parameter files. The main difference between these files is that the optimized values are optimal for the Odroid, which means that the running time of the software is faster than with the unoptimized parameter. In total it can be seen that the consumed power is much higher by processing with the unoptimized parameter. The precise difference between them is presented in Table A.2 and A.3 which are in the Appendix.

In total the software was running twenty times for each parameter file and another twenty times to see the different results whether the keyboard is connected to the Odroid or not. For every passage the fan of the Odroid was working in full speed. This ensured that the process performance was most effective.

For the optimized parameter the results are as follows. The electric current intensity was equal for each passage, to be precisely the average value accounted for an amount of 0.064 $\frac{Watt}{hour}$. The time, which the software needs for completion, in average is around $32.52 \cdot 10^9$nanoseconds, roughly around 32.5 seconds. This result correspond to the measurements with unconnected keyboard.

In comparison to the execution time in chapter 2 [14] which was between 5 and 30 minutes, the execution time with optimal parameters is more than 10 times faster. This may be due to the fact that the hardware of the Odroid is more powerful than on a typical PC. Another point could be that the Odroid was used without graphical interface. The

editing pictures are thus not directly visualized on the desktop and the processing of them works in the background. This could also be one point why the execution process on the Odroid is much faster.

The results for the measurements without connected keyboard are as follows. The power consumption average is around 0.063 $\frac{Watt}{hour}$. In comparison, with the power consumption with connected keyboard, the difference is only around0.001 $\frac{Watt}{hour}$ lower. The difference does nearly not exist and can be classified to be within the error range. The lead time of the software without keyboard is in average $32.51 \cdot 10^9$nanoseconds, roughly around 32.5 seconds. If the comparison is made by means of seconds, there is no truly difference between them. It could therefore be concluded that there is no matter if the keyboard is connected to the Odroid or not, because the power consumption stays considerably equal. Although the lead time and the electric current intensity of the software has barely noticeable differences, there are two more aspects which can be compared, the power consumption and the electric current intensity while the software is running. To illustrate this there are two graphics for the measurements with connected keyboard in 6.2 and two further graphics for the measurements without connected keyboard in 6.3.

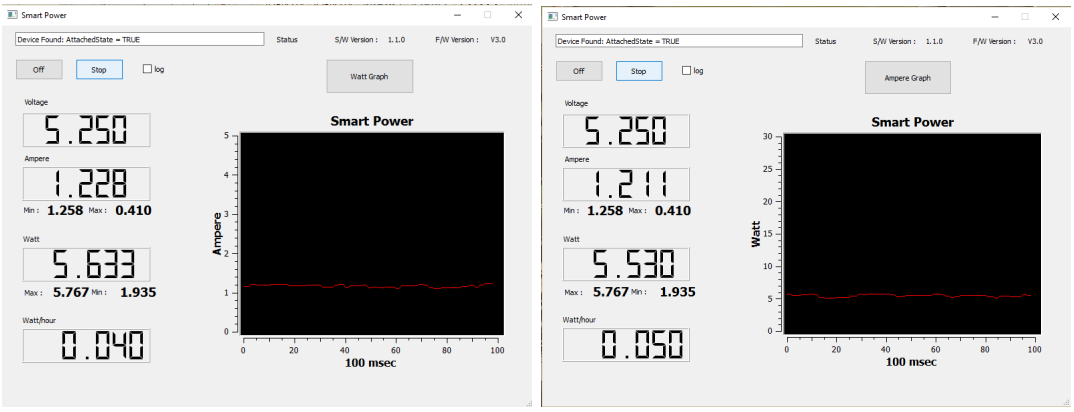

Figure 6.2: Graphical illustration for connected keyboard with optimized parameter on the Odroid. In the left figure the electric current intensity and in the right picture the power consumption while running the software is illustrated.

By means of the time course in the figure 6.2 for the process with the optimized parameter and connected keyboard the upshot of this is relatively uniform. For the electric current intensity the value is more than 1.2 Ampere. The value fluctuates between plus and minus 0.1 Ampere. The graphical progress of the electric current intensity is similar. The development of this is relative uniformly and the average of this amounts to 5.3 Watt. The value fluctuates stronger and is between plus and minus 0.2 Watt.

The results for the unconnected keyboard with optimized parameter are as follows. Both curves are similar and are relatively uniform, but they are a little shifted along the x axis. The electric current intensity value amounts to 1.15 Ampere and fluctuates

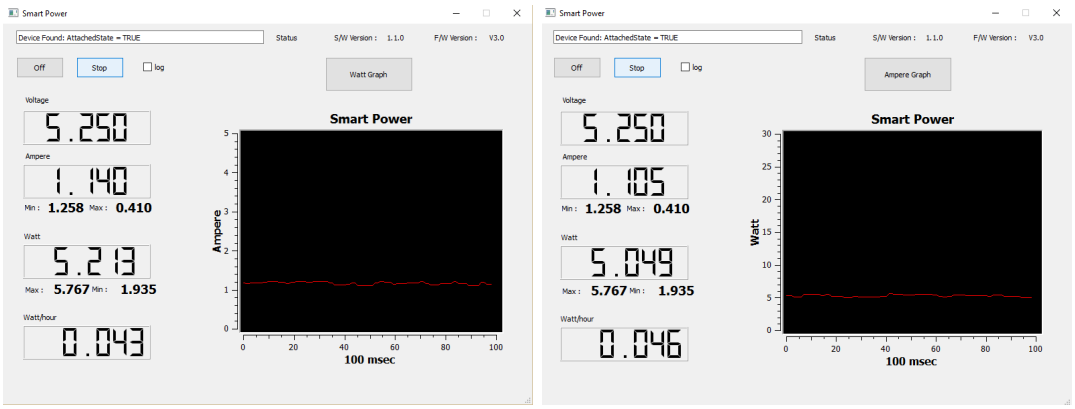

Figure 6.3: Graphical illustration for unconnected keyboard with optimized parameter on the Odroid. In the left figure the electric current intensity and in the right picture the power consumption while running the software is illustrated.

between 1.1 and 1.2 Ampere. Therefore, it can be seen that the electric current intensity with unconnected keyboard is around 0.1 Ampere lower than with connected keyboard. Similar results arise with the power consumption. The average value amounts 5 Watt with variations around 0.2 Watt either way. In total there is a difference of approximately 0.3 Watt.

The overall conclusion is that without a connected keyboard a small, but recognizable difference in power consumption is noticeable. But this difference could be classified to be within the error range and because of this it can be specified that there is no matter if the keyboard is connected to the Odroid or not.

As mentioned earlier the unoptimized parameter is not optimal for processing on the Odroid and so the lead time of the process is much higher than for the optimized parameter. That is why the power consumption is also much higher. In fact the average for the power consumption with connected keyboard is around 0.21985 $\frac{Watt}{hour}$ and the lead time of the software amounts around $117.1 \cdot 10^9$ nanoseconds, roughly around 117.16 seconds. If the keyboard is unconnected, the following results occur. The average value for the power consumption is around 0.2185 $\frac{Watt}{hour}$ and the running time is around $117.0 \cdot 10^9$ nanoseconds, roughly around 116.98 seconds. In the following graphics 6.4 and 6.5 the exactly course of power and electric current intensity is portrayed.

As in the results for the optimized parameter also shown a slight variation, but the fluctuation is a little smaller. The average value with the connected keyboard amounts to around 1.3 Ampere and 5.9 $\frac{Watt}{hour}$. For the electric current intensity there are variations between plus and minus 0.05 Ampere and the power consumption shows fluctuations between plus and minus 0.1 $\frac{Watt}{hour}$. The interesting point here are the graphics without connected keyboard. Here the fluctuations of the curves are hardly recognizable. The electric current intensity is around 1.3 Ampere and varies around 0.01 Ampere. The characteristics of the

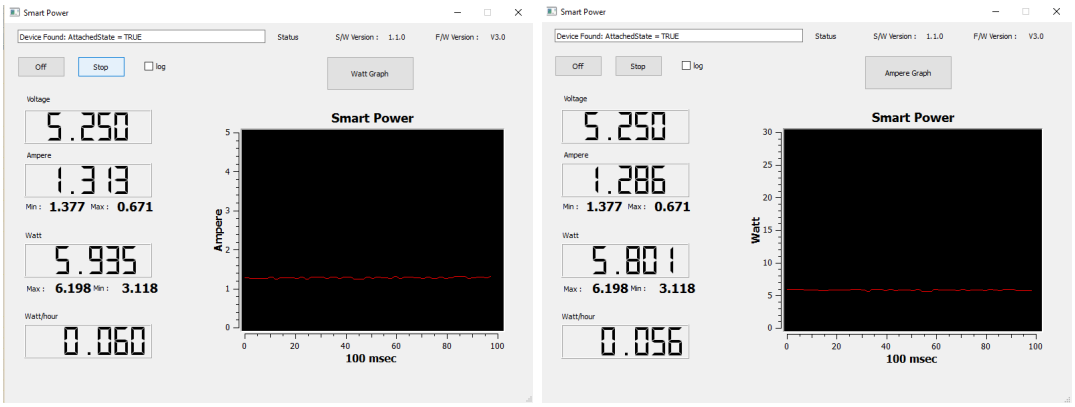

Figure 6.4: Graphical illustration for connected keyboard with unoptimized parameter on the Odroid. In the left figure the electric current intensity and in the right picture the power consumption while running the software is illustrated.

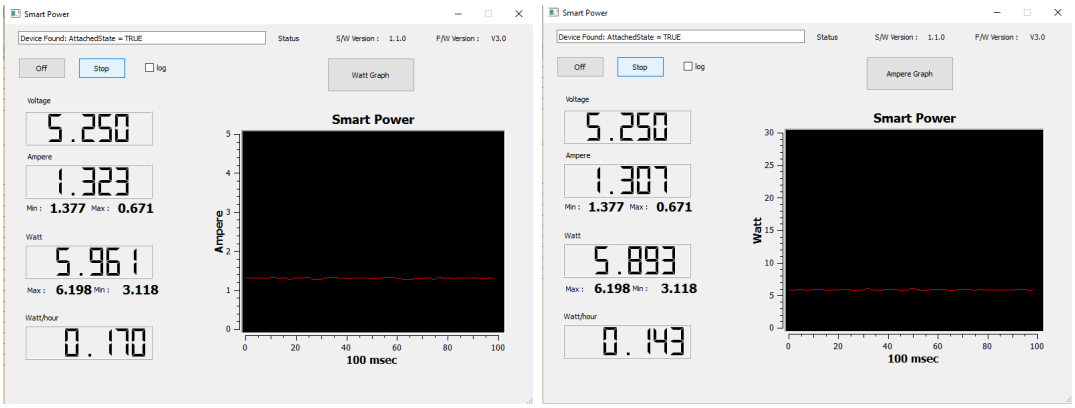

Figure 6.5: Graphical illustration for unconnected keyboard with unoptimized parameter on the Odroid. In the left figure the electric current intensity and in the right picture the power consumption while running the software is illustrated.

course for the power consumption is similar. The value amounts around 5.8 Watt and also varies only around 0.01 Watt.

In total it is recognizable that with the unoptimized parameter the lead time and also the electric current intensity is much higher, but the course is constant. But again it is clearly seen that the difference between connected and unconnected keyboard has no matter on the power consumption.

6.2 Measurement Results by Offloading to an server

After the measurements were done on the Odroid and comparable values are available, it is possible to see the difference of the power and electric current intensity for offloading. Here, the only interesting point is how high the power and electric current intensity is to offload the data to the server. The power consumption of the server to process this information,

is irrelevant for this work. The first paragraph 6.2.1 deals with the measurement results for offloading the data via LAN-cable and the second paragraph 6.2.2 with the results for offloading the data wireless.

6.2.1 Offloading via LAN-cable

The results for offloading the data from the Odroid to the server with an Ethernet cable are as follows. With connected keyboard the power consumption was in average around 0.0275 $\frac{Watt}{hour}$ and the time for offloading was in average $19.06 \cdot 10^9$ nanoseconds, roughly 19.06 seconds. In contrast to this, the results with unconnected keyboard were quite similar, only for the power consumption a little higher. For the power consumption arises a value around 0.0278 $\frac{Watt}{hour}$ and the time for offloading in average is around $19.05 \cdot 10^9$ nanoseconds, roughly 19.05 seconds. The graphics in figure 6.6 and 6.7 illustrate the electric current intensity and power consumption for the offloading process. The values for each offloading process via Ethernet cable is illustrated in A.4 which is located in the Appendix.

By means of the graphical time course of the power and electric current intensity it can be clearly seen that the process is relative constant. With connected keyboard the electric current intensity is around 1.1 Ampere and almost constant. The power consumption is around 5.2 Watt and variates between plus and minus 0.01 Watt. The measurement results with an unconnected keyboard are quite similar. The electric current intensity again amounts to around 1.1 Ampere and fluctuates between plus and minus 0.1 Ampere. The power consumption is also the same and amounts to around 5.2 Watt.

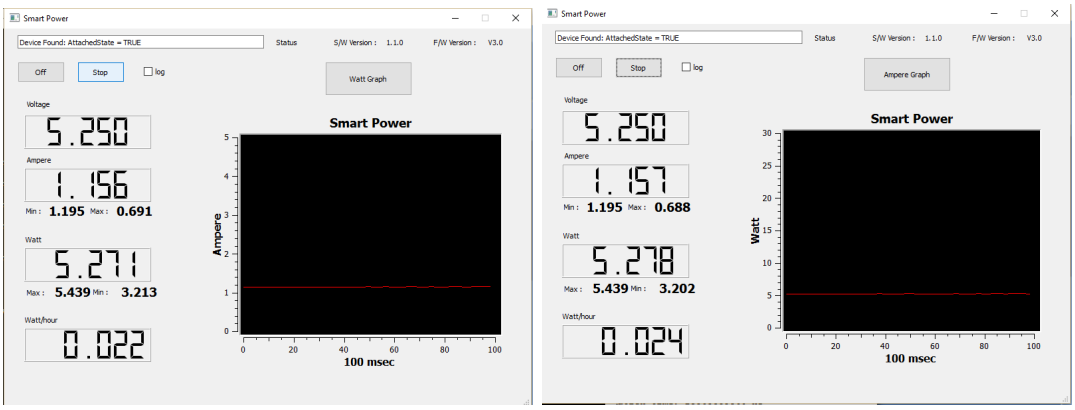

Figure 6.6: Graphical illustration with connected keyboard for offloading process via LAN-cable. In the left figure the electric current intensity and in the right picture the power consumption for offloading process is illustrated.

In total, it is clearly seen that in case of offloading there is no truly difference between the measurement results with or without connected keyboard. It is also an important point, that the courses of this offloading process are in total constant and also that the offloading process is very fast. Summing up, only about 19 seconds are needed to offload

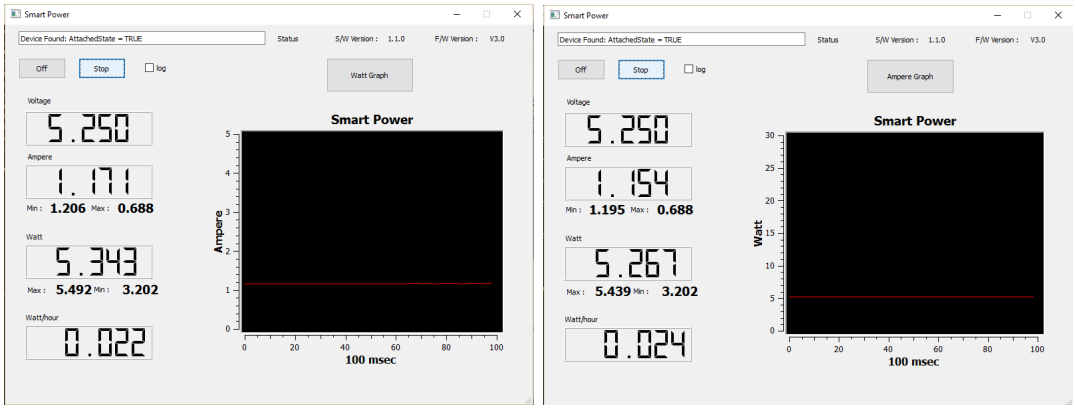

Figure 6.7: Graphical illustration with unconnected keyboard for offloading process via LAN-cable. In the left figure the electric current intensity and in the right picture the power consumption for offloading process is illustrated.

the data to an external server for processing and in fact can be said that the keyboard in this case has no influence on the results. The conclusion can therefore be drawn that the different results for the first measurement on executing the software with and without keyboard on the Odroid are only within the error range.

6.2.2 Wireless Offloading

The second method for offloading the data to an external server is the wireless offloading process. The measurement results show that the power consumption as well as the lead time for offloading is not constant.

Values ranged from 0.122 $\frac{Watt}{hour}$ to nearly 0.179 $\frac{Watt}{hour}$ for a connected keyboard. Without connected keyboard the power consumption fluctuates between 0.099 $\frac{Watt}{hour}$ and 0.184 $\frac{Watt}{hour}$. On average the power consumption is 0.14415 $\frac{Watt}{hour}$ and it takes about $88.04 \cdot 10^9$nanoseconds with a connected keyboard. Loosely speaking the offloading process takes about 88.04 seconds in average. Without a connected keyboard the power consumption in average is around 0.13195 $\frac{Watt}{hour}$ and $81.81 \cdot 10^9$ nanoseconds for the offloading process, so about 81.81 seconds. All the values are illustrated in A.5 which is again located in the Appendix.

In total, can be clearly seen that the power consumption and the offloading time points out a difference nearly seven seconds. Furthermore, the wireless offloading process also fluctuates highly in relation to power consumption and lead time. This also can be justified in the way that the wireless signal strength is never the same. The differences could arise because of the wireless signal strength and it is difficult so say how much the values are influenced by connected or the unconnected keyboard.

By means of the graphics in 6.8 and 6.9 can be seen that the course is again fairly constant. For the connected keyboard the results are as follows. The electric current intensity is around 1.3 Ampere and oscillates between plus and minus 0.1 Ampere. For the

power consumption a similar course is presented. The value results around 5.8 Watt and ranges only between plus and minus 0.1 Watt. As expected from the values in the Table A.5 the results for the measurements without a connected keyboard are the same.

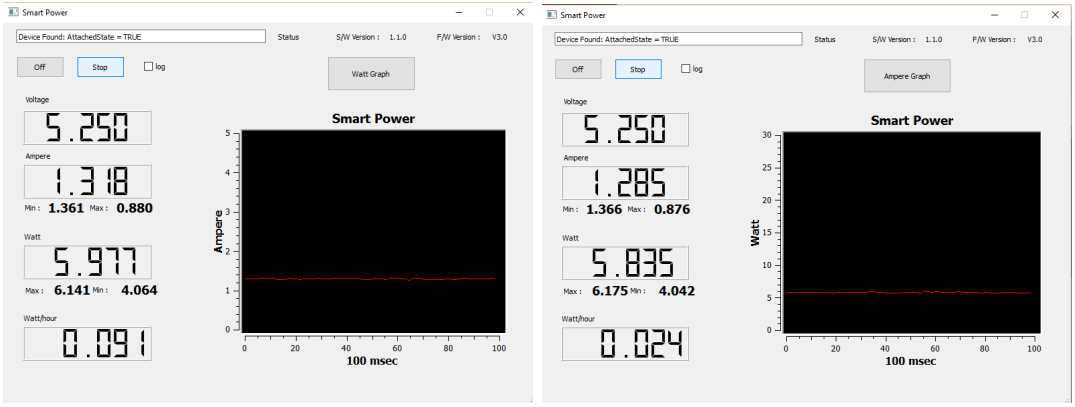

Figure 6.8: Graphical illustration with connected keyboard for offloading process via Wireless. In the left figure the electric current intensity and in the right picture the power consumption for offloading process is illustrated.

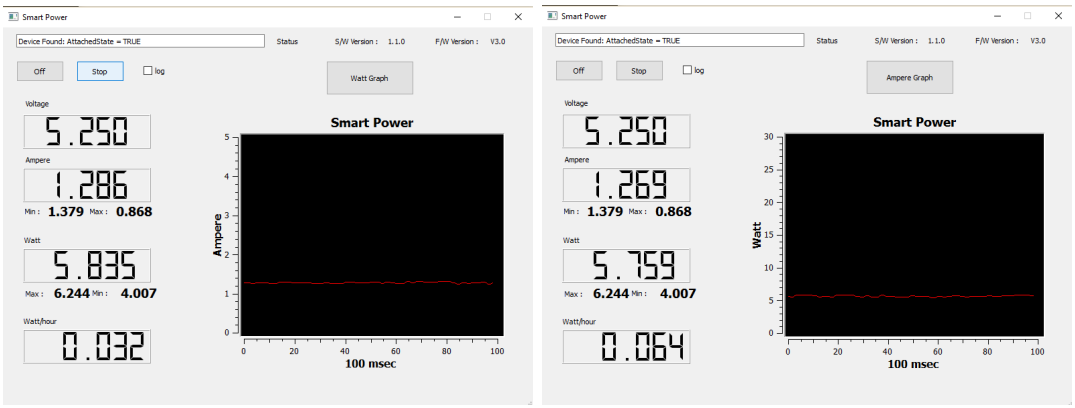

Figure 6.9: Graphical illustration with unconnected keyboard for offloading process via Wireless. In the left figure the electric current intensity and in the right picture the power consumption for offloading process is illustrated.

For the electric current intensity the value is 1.3 Ampere and has fluctuations between plus and minus 0.1 Ampere. The power consumption is similar. The result is 5.8 Watt and ranges around 0.1 Watt.

Therefore it can be be seen that there is no truly difference if the keyboard is connected to the Odroid or not. After a sort can be made the conclusion that connected or unconnected keyboard does not have any effect on the results and the low difference is only within the error range. But, it is interesting how much varying strengths the wireless has and because of that in comparison with the offloading process via LAN-cable, the wireless offloading process takes a lot of time.

The reasons for these fluctuations could be also important to know and by this to solve the issue. For private networks, there are some solutions for this [22]. The first one is to place the router in the same room as the laptop, computer, tablet, mobile phone and so on. As farther away from the router, as lower is the signal strength. A further barrier for the wireless signal strength can be also a wall or any other devices like a television, a radio or even a microwave. To enhance the signal strength and to ensure that in a apartment for example in every room the signal strength is the same there are some methods [20]. The first opportunity is to change the router. Older versions use the 802.11g-Standard and have a transmission rate of 54 $\frac{MBit}{second}$. Newer router use an 802.11n or 802.11ac Standard [21]. The 802.11n Standard offers transmission rates up to 150 $\frac{MBit}{second}$. If the router has three antennas it is even possible to become transmission rates up to 450 $\frac{MBit}{second}$. The newest wireless standard, the 802.11ac, offers up to 540 $\frac{MBit}{second}$.

Although it is possible to become so strong signal strengths, barriers like walls obstruct also these high signals. For this two further ways were developed:

Firstly a Wireless Repeater which works like a signal repeater. For example it is set at the half distance between router and laptop for example and by this it strengths the signal strength. The second is a Powerline Adapter. It is connected to the router with an Ethernet cable and plugged into a power outlet. A second adapter works like a Wlan-Assess-Point and by this it is possible for devices to connect to the Internet. Unfortunately this opportunities were missing and because of that they could not be tested.

In total, it is known from everyday examples that Internet access with wireless connection needs a lot of power. One best known example is the mobile phone. If the wireless connection is activated, it is apparent that the battery for the mobile phone discharge faster. In many magazines, e.g. [16] is given the advice to turn off the wireless connection to save energy.

6.3 Conclusion

In this respect a conclusion will be made at which point it is recommended to offload the data for calculation on an external server. It turned out that the measurement results for connected and unconnected keyboard on the Odroid are nearly similar, only the values which arises with connected keyboard here are discussed.

The first diagram in 6.10 illustrates the comparison between the executed software on the Odroid with optimized and unoptimized parameter and the offloading process via LAN-cable and wireless in relation to the execution time. It is clearly seen that the execution time for the executed software with unoptimized parameter needs the most time. In average the execution time takes about 117 seconds, but by means of the course the execution time stays considerably constant. In comparison to the execution time for the optimized parameter this process needs only about 32 seconds. Furthermore, on the course is seen

that the execution time stays constant. The same course shows the offloading process via LAN cable. The time for offloading is around 19 seconds. The only strong fluctuations arises with the offloading process via wireless. As mentioned earlier the transfer rate of this process fluctuates considerably over all passes. In average the offloading time is approximately 88 seconds.

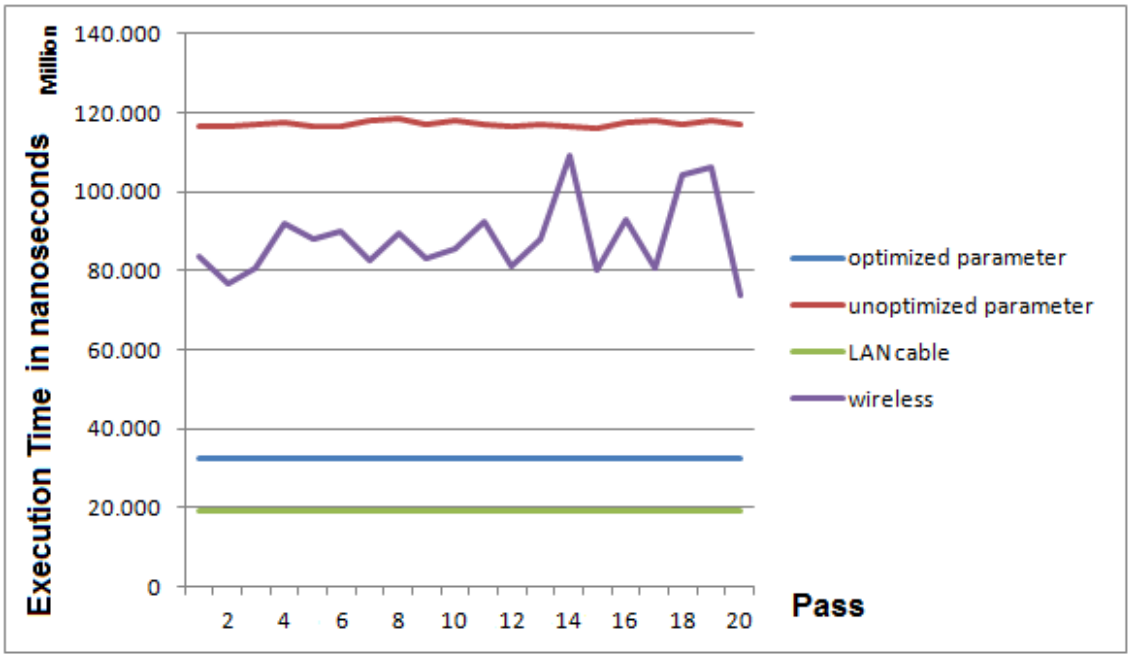

Figure 6.10: Graphical illustration of the execution time in nanoseconds. A comparison between the optimized and unoptimized parameter file on the Odroid and the offloading process via LAN cable and wireless is illustrated.

The second diagram 6.11 portrays the comparison between the executed software on the Odroid with optimized and unoptimized parameter and the offloading process via LAN-cable and wireless in relation to the power consumption. The courses of the power consumption are quite similar to the courses of the execution time. The power consumption of the executed software with unoptimized parameter setting is in average 0.220 $\frac{Watt}{hour}$ and for the optimized parameter setting is about 0.064 $\frac{Watt}{hour}$. For the offloading process via LAN cable around 0.028 $\frac{Watt}{hour}$ are consumed and via wireless it amounts around 0.144 $\frac{Watt}{hour}$.

In total, the main conclusions which can be drawn from the surveys are the following:

The main question of this work is how much power can be saved if the data was offloaded instead of executing this software directly on the Odroid. The overall conclusion is that the offloading process via LAN-cable is most effective in relation to power consumption and duration. The wireless offloading is unfortunately inadvisable. Even in the fastest way, the wireless offloading process takes nearly more than three times than the offloading by LAN-cable.

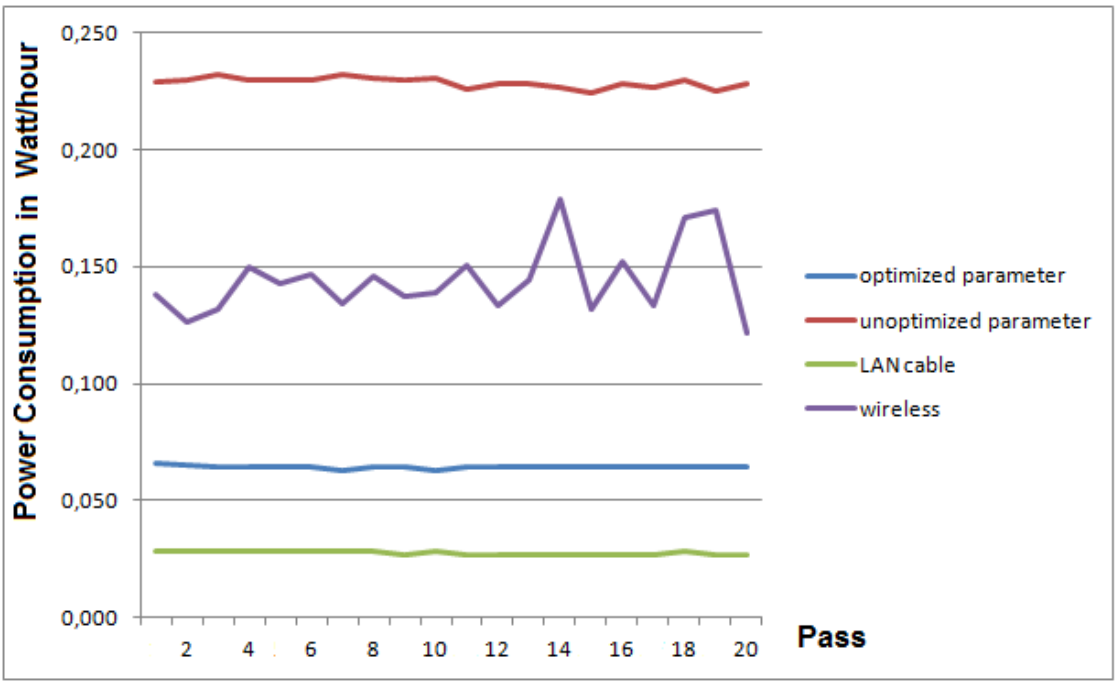

Figure 6.11: Graphical illustration of the power consumption in $\frac{Watt}{hour}$. A comparison between the optimized and unoptimized parameter file on the Odroid and the offloading process via LAN cable and wireless is illustrated.

And also in relation to the software executed on the Odroid, in some cases it needs more power and the lead time is longer. To be precise, if the software is executed with the optimized parameter setting for the Odroid, the offloading process needs more than three times as much electric current intensity and time. Simply put, it is possible to do three measurements on the Odroid, to offload the data to the server once.

For this comparison with the executed software on the Odroid by using the unoptimized parameters, the figures are as follows: Here in fact it would be worth to offload wireless the data instead of executing the software with this parameter. In this case executing the software on the Odroid would need almost 1.5 times more electric current intensity and time.

In summary it can be said that a clear distinction needs to be made between the selected parameter files. If parameter files were selected, which are optimal for the Odroid, it is recommended to offload the data via LAN-cable, but not wireless. In this case it is better to execute the data on the Odroid. But in the other case the offloading process has more advantages. There it is even irrelevant if the offloading process is done via LAN-cable or wireless. In total, it is obvious that the offloading process with LAN-cable is the best way for offloading.

The aim of this project was to find out if it is possible to save energy for Mobile Biosensors through offloading the data to an external server. After all the measurements

were done, it can be specified that especially the offloading process via LAN cable is optimal for saving power. It is very fast and the transmission rate stays nearly constant over the entire period and in total this method for offloading needs lower power than directly executing the software on the Odroid. Saving energy for wireless offloading is more difficult. The offloading process takes a lot of time and the transmission rate fluctuates very strong. And exactly here is the main problem. Although the power consumption is nearly constant and has no relation to the transmission rate, the offloading process takes much longer and because of that also more power is consumed. If it is possible to strengthen the transmission rate of the wireless offloading with the methods, which were presented in 6.2.2, it would be worth to offload the data via wireless. But in which relation this offloading process would stay in comparison to the offloading process via LAN cable has to be investigated further.

Chapter 7

Summary and Outlook

In the last chapter of this work a summary and an outlook of the main aspects of this theme follows.

7.1 Summary

The aim of this work was to analyze the energy consumption of the Odroid by means of an application example. The application that was used is called PAMONO application. With this Biosensor it is possible to detect viruses in samples like blood or saliva. During the investigation, pictures are made of this process and these pictures can be examined with a special software, which was explained in chapter 3.2.

To get comparable values and to recognize if the offloading process truly saves energy, the software was first executed on the Odroid. The processing time and the energy consumption depend on the parameter settings for the PAMONO software. Therefore a distinction was made between an optimized and unoptimized parameter file. With the optimized parameters are optimal for the Odroid this process is much faster than with the unoptimized parameter file. The result is that the software executes the calculation more than three times faster optimized than with the unoptimized parameter. In total, the energy consumption of the Odroid for processing with the optimized parameter was around 0.064 $\frac{Watt}{hour}$ and takes around 32 seconds, while it was around 0.220 $\frac{Watt}{hour}$ for the unoptimized parameter and takes around 88 seconds.

After the measurements on the Odroid are completed, the offloading process can be started. Two ways of offloading were examined. The first was offloading the data via LAN-cable and the second was offloading via wireless. This results in an extreme difference in relation to energy, power and time execution. The results for offloading via LAN-cable were pretty fast and the energy consumption was also very low. The measured values are roughly about 0.028 $\frac{Watt}{hour}$ for energy consumption and around 19 seconds for offloading. By offloading via wireless various different values occurred. The energy consumption lies

within the range of 0.100 $\frac{Watt}{hour}$ and 0.184 $\frac{Watt}{hour}$ and the lead time between 73 seconds and 109 seconds. Therefore the offloading process is more effective by offloading via LAN-cable.

Hence, it is possible to note that the offloading process via LAN-cable is most effective and it is possible to save energy by this. But this can not be said for the offloading process via wireless. If the process has to be done with a non optimal parameter setting, it is better to offload them, instead of executing the software on the Odroid. But if there are optimal parameter settings executing the process on the Odroid is more energy-saving.

7.2 Outlook

The offloading process via wireless is not optimal, because of that it would be important to try offloading via LTE *(Long Term Evolution)* and observe the differences. And also to find a solution for the high power consumption of using wireless. Another point would be to optimize the software for the Odroid and by this to ensure that every parameter file, with which the software should be executed, is close to optimal. Of course a further aspect is to improve the software and by this to reduce the lead time and also the power consumption, but with the guarantee of a reliable data evaluation. It is also an important point to realize the aspect why wireless offloading causes so many fluctuations in power consumption and offloading time and to find a solution for this problem. In total, it is important to detect the reasons for the high power consumption for offloading, e.g. via wireless and to find a solution to reduce this. It would be also interesting to try different offloading methods, maybe not directly over the Internet. For example the differences via Bluetooth offloading to a personal computer in comparison to offloading via Ethernet cable or wireless could be examined. But here the problem is that Bluetooth only works for small distances and also has the problem that is needs many powers to work effective.

A further aspect to take a closer look at is to do these measurements on other Single Board Computers, e.g. the *Raspberry Pi* and to take a comparison between them. In this relation can be examined in the case that one Single Board Computer needs lower power what hardware aspect causes this.

Appendix A

Further Additional

A.1 Parameter values for detecting viruses

Table A.1: Comparison between optimized and unoptimized documents

	optimized parameter	unoptimized parameter
sourceStartIndex	1	1
sourcePath	" "	E:/ISAS-Images2-repo/isas_images2/public_data/200nm_11Apr13_1/synth2/pcount100/nstt/training
sourceFileName	"capture2013-04-11-"	"capture2013-04-11-"
sourceFileExtension	"png"	"png"
sourceFileNameDigitCount	4	4
pauseAtFrame	-1	-1
pauseAtFrame2	-1	-1
sinkStartIndex	0	0
sinkPath	" "	" "
sinkFileName	"Image"	"Image"
sinkFileExtension	" "	" "
sinkFileNameDigitCount	3	3
forceRecompile	0	0
writeAllFeaturesToFile	0	0
writeFinalFeaturesToFile	0	0

fastRelaxedMath	0	0
delay	0	0
startMergingAfterXFrames	0	0
background	1	1
backgroundRefs	40	40
backgroundGap	-1	-1
backgroundOption	1	1
foreground	1	1
foregroundRefs	40	40
foregroundBackground Option	1	1
foregroundGroupNFrames	1	1
averageImage	0	0
averageImageKernelWidth	3	3
averageImageKernelHeight	4	4
medianImage	0	0
medianImageKernelWidth	3	3
medianImageKernelHeight	3	3
gaussImage	1	1
gaussImageSigma	1.500000	1.500000
brightnessCorrection	0	0
brightnessCorrection ReferenceBrightness	0.000000	0.000000
haarNoiseReduction	0	0
haarNoiseReductionKeep	8	8
haarNoiseReduction Quality	0.100000	0.100000
combineDetections	0	0
combineDetections CombineXFrames	16	16
combineDetections CombineEveryNthFrame	1	1
enlargeDetections	0	0
closeDetections	0	0
closeDetectionsCircle Radius	4.200000	4.200000
openDetections	0	0
openDetectionsCircle Radius	4.200000	1.000000
detectOutshinedSpots	0	0
convertScale	0	0
convertScaleMin	0.000000	0.000000
convertScaleMax	0.000000	0.000000

convertScaleMinMaxEvery NthFrame	16	16
convertScaleKeepConstant FromFrame	-1	-1
fuzzyNoiseRemoval	0	0
fuzzyNoiseRemovalX1	0.046264	0.046264
fuzzyNoiseRemovalX2	0.148658	0.148658
detection	1	1
timeSeries	1	1
timeSeriesOption	14	14
detectionThreshold	0.400000	0.400000
detectionThresholdDown	0.861650	0.861650
timeSeriesDistancePatterns	"-1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0,1.0"	"-1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,-1.0,-1.0,-1.0,-1.0, -1.0,-1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0,1.0,1.0, 1.0,1.0,1.0,1.0,1.0"
timeSeriesDistancePattern Size	16	16
timeSeriesDistancePattern Count	1	1
timeSeriesDistancePattern SlopeRelaxation	0	0
timeSeriesDistance PatternA	0	0
timeSeriesDistance PatternB	0	0
timeSeriesDistanceMin Step	0.009152	0.009152
timeSeriesDistanceMax Step	0.149582	0.149582
timeSeriesDistanceMin NegativeStep	-1.000000	-1.000000
timeSeriesDistanceMax NegativeStep	-1.000000	-1.000000
timeSeriesAutomatically UpdateVariablePatterns	1	1
fuzzyDetection Enhancement	0	0

fuzzyDetection EnhancementX1	0.054554	0.054554
gaussHesseFeature	0	0
gaussHesseFeatureSigma	0.700000	0.700000
gaussHesseFeatureNum Scales	8	8
gaussHesseFeature Detection Threshold	0.000000	0.000000
gaussHesseFeaturePolygon Threshold	0.000000	0.000000
gaussHesseBlobFeature DetectionThreshold	0.000000	0.000000
gaussHesseBlobFeature PolygonThreshold	0.000000	0.000000
hesseFeature	0	0
templateMatchingFeature	0	0
templateMatchingFeature Threshold	0.000000	0.000000
templateMatchingFeature Normalize	1	1
clDeviceType	4	4
clDeviceId	0	0
clPlatformName	"NVIDIA Corporation"	"NVIDIA Corporation"
roiStartX	0	0
roiStartY	0	0
roiEndX	0	0
roiEndY	0	0
stepFeature	0	0
merging	1	1
mergingOption	0	0
mergingMaxDistance	4.000000	4.000000
mergingMaxFrameDistance ForPolygon	8	8
classify	1	1
classifyVirusCount	0	0
classifyVirusCount MinDetections	0	0
classifyVirusCount MaxDetections	20	20
polygonMinAxisFirst	3.000000	3.000000
polygonMaxAxisFirst	40.000000	40.000000
polygonMinAxisSecond	2.000000	2.000000
polygonMaxAxisSecond	40.000000	40.000000

polygonMinArea	0.000000	0.000000
polygonMaxArea	20000.000000	20000.000000
polygonMinCircularity	0.400000	0.400000
polygonMinCircumference	0.000000	0.000000
polygonMaxCircumference	20000.000000	20000.000000
polygonBoundingBoxMax AspectRatio	4.000000	4.000000
polygonMinRoundness	0.300000	0.300000
polygonMinCompactness	0.000000	0.000000
polygonMinShape	0.000000	0.000000
polygonMaxRectangularity	1.000000	1.000000
visualize	1	1
visualizeOption	0	0
drawTheVirusBoundingBox	0	0
drawTheVirusPolygons	1	1
visualizePseudoColorHSV MappingHStartDegree	0.000000	0.000000
visualizePseudoColorHSV MappingHEndDegree	60.000000	60.000000
visualizePseudoColorHSV MappingMinDegree	0.000000	0.000000
visualizePseudoColorHSV MappingMaxDegree	360.000000	360.000000
visualizePseudoColorHSV Saturation	1.000000	1.000000
visualizePseudoColorHSV Value	1.000000	1.000000
resultFilesOutputFolder	" "	" "
quickStart	0	0
copomoCutoffParameter	-1	-1
foregroundBackground Option	0	1
foregroundRefs	12	27
backgroundRefs	26	26
gaussImage	1	1
gaussImageSigma	1.594505	2.106963
brightnessCorrection	0	1
closeDetections	0	0
closeDetectionsCircle Radius	2.788893	1.70609
openDetections	0	0
openDetectionsCircle Radius	2.90899	4.355153

detectOutshinedSpots	0	0
timeSeriesOption	17	14
detectionThreshold	0.073741	0.113192
timeSeriesDistance PatternSize	12	28
timeSeriesDistance PatternSlopeRelaxation	3	3
timeSeriesDistance MinStep	0.013227	0.00926
timeSeriesDistance MaxStep	0.191839	0.127521
timeSeriesDistanceMin NegativeStep	-0.576513	-0.099111
timeSeriesDistanceMax NegativeStep	-0.701246	-0.560982
mergingMaxDistance	3.499158	6.102633
mergingMaxFrameDistance ForPolygon	15	17

A.2 Measurement Results on Odroid and for Offloading

Table A.2: Measurement Results with the optimized parameter on the Odroid

optimized	with keyboard		without keyboard	
Pass	Power Consumption $\frac{Watt}{hour}$	Execution time ns	Power Consumption $\frac{Watt}{hour}$	Execution time ns
1	0.066	32 473 729 920	0.063	32 499 621 004
2	0.065	32 512 826 254	0.063	32 510 739 963
3	0.064	32 510 289 004	0.063	32 528 677 585
4	0.064	32 533 680 837	0.063	32 508 057 295
5	0.064	32 566 582 671	0.063	32 472 741 046
6	0.064	32 496 639 962	0.063	32 488 461 754
7	0.063	32 496 614 421	0.063	32 480 446 671
8	0.064	32 499 554 796	0.063	32 539 321 504
9	0.064	32 508 595 670	0.064	32 521 873 129
10	0.063	32 503 371 337	0.063	32 502 890 420
11	0.064	32 556 904 588	0.063	32 496 771 713
12	0.064	32 540 652 254	0.063	32 569 874 420
13	0.064	32 566 125 670	0.063	32 517 011 046
14	0.064	32 553 434 754	0.063	32 507 714 628
15	0.064	32 513 066 540	0.063	32 524 262 671
16	0.064	32 476 403 504	0.063	32 508 760 712
17	0.064	32 563 077 045	0.063	32 524 402 879
18	0.064	32 512 696 754	0.063	32 523 336 587
19	0.064	32 481 053 962	0.062	32 486 187 837
20	0.064	32 520 066 337	0.063	32 510 241 879
average	0.064	32 519 268 314	0.063	32 511 069 737

Table A.3: Measurement Results with the unoptimized parameter on the Odroid

unoptimized	with keyboard		without keyboard	
Pass	**Power Consumption** $\frac{Watt}{hour}$	**Execution time ns**	**Power Consumption** $\frac{Watt}{hour}$	**Execution time ns**
1	0.218	116 717 339 347	0.220	117 311 783 681
2	0.219	116 728 867 223	0.220	116 941 711 431
3	0.220	117 002 950 097	0.220	116 770 889 347
4	0.223	117 556 784 223	0.218	116 660 150 097
5	0.219	116 318 189 014	0.218	116 318 976 680
6	0.218	116 420 285 972	0.220	117 442 926 389
7	0.223	117 896 093 056	0.219	116 649 110 889
8	0.225	118 353 542 347	0.219	117 179 200 722
9	0.222	117 270 315 972	0.219	116 827 268 181
10	0.223	117 935 271 223	0.218	116 782 269 180
11	0.216	116 850 462 264	0.215	116 040 525 180
12	0.217	116 733 371 597	0.218	117 164 706 264
13	0.218	116 972 914 722	0.218	116 919 870 514
14	0.218	116 654 775 723	0.217	116 574 687 680
15	0.216	116 054 377 264	0.218	117 051 727 639
16	0.221	117 335 286 430	0.220	117 845 978 722
17	0.221	117 816 836 139	0.219	117 861 871 305
18	0.219	116 951 409 805	0.217	116 757 696 681
19	0.222	117 988 078 056	0.218	117 250 986 097
20	0.219	117 154 517 889	0.219	117 158 568 639
average	0.21985	117 135 583 418	0.2185	116 975 545 265

Table A.4: Offloading Results by LAN-cable from Odroid to an external server.

unoptimized	**with keyboard**		**without keyboard**	
Pass	**Power Consumption** $\frac{Watt}{hour}$	**Execution time ns**	**Power Consumption** $\frac{Watt}{hour}$	**Execution time ns**
1	0.028	19 028 946 335	0.028	19 052 737 585
2	0.028	19 024 229 336	0.028	19 018 654 127
3	0.028	19 159 829 002	0.028	19 027 453 835
4	0.028	19 138 726 836	0.028	19 071 580 335
5	0.028	19 081 834 711	0.028	19 092 511 877
6	0.028	19 104 421 628	0.027	19 056 275 544
7	0.028	19 024 389 377	0.027	19 035 794 544
8	0.028	19 118 736 169	0.027	19 115 639 003
9	0.027	19 023 022 960	0.028	19 063 089 002
10	0.028	19 043 116 919	0.028	19 046 127 919
11	0.027	19 038 844 586	0.028	19 029 071 044
12	0.027	19 065 534 211	0.028	19 065 526 336
13	0.027	19 036 474 961	0.028	19 020 022 252
14	0.027	19 064 781 044	0.028	19 098 487 002
15	0.027	19 070 859 294	0.027	19 024 237 836
16	0.027	19 045 600 544	0.028	19 031 341 377
17	0.027	19 059 206 211	0.028	19 070 742 336
18	0.028	19 044 162 710	0.028	19 042 515 877
19	0.027	19 046 237 169	0.028	18 972 377 210
20	0.027	19 050 390 336	0.028	19 092 364 794
average	0.0275	19 063 467 217	0.0278	19 051 327 492

Table A.5: Offloading Results by Wi-Fi from Odroid to an external server.

unoptimized	**with keyboard**		**without keyboard**	
Pass	**Power Consumption** $\frac{Watt}{hour}$	**Execution time ns**	**Power Consumption** $\frac{Watt}{hour}$	**Execution time ns**
1	0.138	83 652 256 927	0.132	81 688 954 927
2	0.126	76 629 326 425	0.119	74 050 513 676
3	0.132	80 542 662 052	0.099	61 343 144 383
4	0.150	92 092 527 302	0.128	79 817 852 010
5	0.143	87 913 014 635	0.110	68 026 345 925
6	0.147	90 175 954 428	0.141	87 602 136 010
7	0.134	82 660 097 176	0.126	77 472 847 801
8	0.146	89 378 978 760	0.131	80 603 741 259
9	0.137	83 348 090 135	0.129	85 850 255 552
10	0.139	85 778 639 427	0.110	68 545 804 341
11	0.151	92 398 565 886	0.133	82 541 984 593
12	0.133	81 282 463 843	0.133	82 145 592 468
13	0.144	87 875 553 178	0.167	103 084 415 971
14	0.179	109 092 968 888	0.183	112 909 011 305
15	0.132	80 080 317 427	0.144	87 332 031 510
16	0.152	92 814 575 845	0.184	113 496 359 639
17	0.133	80 720 995 760	0.130	80 463 066 093
18	0.171	104 305 984 512	0.134	82 043 431 052
19	0.174	106 200 901 638	0.105	65 367 340 300
20	0.122	73 827 399 218	0.101	61 822 872 799
average	0.1445	88 038 563 673	0.13195	81 810 385 080

List of Figures

Listings

List of Tables

Bibliography

[1] 123RF: *Virus-Infektion medizinische Symbol von einer Gruppe von bakteriellen Eindringlings Zellen verursacht Krankheiten und Gebrechen zu gesunden Patienten vertreten.* http://de.123rf.com/lizenzfreie-bilder/flu_____%C3%82%C4%BCkrebs.html?mediapopup=10945949. [Online; accessed 02-January-2016].

[2] Aaftab Munshi, Lee Howes and Bartosz Sochacki: *The OpenCL C Specification.* https://www.khronos.org/registry/cl/specs/opencl-2.0-openclc.pdf. [Online; accessed 12-January-2016].

[3] Benedict R. Gaster, Lee Howes, David R. Kaeli Perhaad Mistry-Dana Schaa: *Heterogeneous Computing with OpenCL.* Revised OpenCL 1.2 Edition edition, 2013. Chapter 1,2.

[4] Bernet, Marcel: *GPIO.* https://developer.mbed.org/teams/ch-open-wstage2015/wiki/GPIO. [Online; accessed 02-May-2016].

[5] Hardkernel: *ODROID Support Page.* http://odroid.com/dokuwiki/doku.php. [Online; accessed 06-March-2016].

[6] Hardkernel: *Odroid XU4.* http://www.hardkernel.com/main/products/prdt_info.php?g_code=G143452239825. [Online; accessed 09-March-2016].

[7] Hardkernel: *Odroid XU-4 Beginners Guide.* 2015. [Online; accessed 06-March-2016].

[8] Henner.Info: *Bild-Rauschen.* http://www.henner.info/rauschen.htm. [Online; accessed 29-February-2016].

[9] Hässler, Ulrike: *Vom Farbnamen zum Farbmodell.* http://www.wisotop.de/hsv-und-hsl-farbmodell.shtml. [Online; accessed 05-March-2016].

[10] Homberg, Willi: *OpenCL Basics Parallel Computing on GPU and CPU.* http://www.fz-juelich.de/SharedDocs/Downloads/IAS/JSC/EN/slides/gpu/gpgpu-6-opencl.pdf?__blob=publicationFile, 2011. [Online; accessed 12-January-2016].

[11] Informatik 12, Leibniz-Institut für Analytische Wissenschaften ISAS e.V. Technische Universität Dortmund Lehrstühle Informatik 7 und: *Fortschritte.* https://ls7-www.cs.tu-dortmund.de/pamono-sensor/fortschritte.html. [Online; accessed 02-January-2016].

[12] Informatik 12, Leibniz-Institut für Analytische Wissenschaften ISAS e.V. Technische Universität Dortmund Lehrstühle Informatik 7 und: *PAMONO Virensensor.* https://ls7-www.cs.tu-dortmund.de/pamono-sensor/projekt/beschreibung.html. [Online; accessed 02-January-2016].

[13] Lexikon, HiFi: *Signal-Rausch-Abstand / signal-to-noise ratio (SNR).* http://www.fairaudio.de/hifi-lexikon-begriffe/signal-rausch-abstand-snr.html. [Online; accessed 10-May-2016].

[14] Libuschewski, Pascal: *Massive parallele Verarbeitung von Bildseuqnzen zur Erkennung von Nanopartikeln.* Diplomarbeit, Juni 2011.

[15] Müller, Prof. Dr. Heinrich: *Vorlesung Mensch-Maschine-Interaktion*, Wintersemester 2014/2015. Chapter 3 Graphische Datenverarbeitung.

[16] Netzwelt: *Smartphones: Die zehn Gebote für eine längere Akkulaufzeit.* http://www.netzwelt.de/news/89651_3-smartphones-zehn-gebote-laengere-akkulaufzeit.html. [Online; accessed 04-May-2016].

[17] Pascal Libuschewski, Dennis Kaulbars, Björn Dusza Dominic Siedhoff Frank Weichert-Heinrich Müller Christian Wietfeld Peter Marwedel: *Multi-Objective Computation Offloading for Mobile Biosensors via LTE.* MobiHealth 2014, November 2014.

[18] Patrick Wagner, Fa. ScanDig: *CCD-Sensor (Chip oder Zeile).* http://www.filmscanner.info/CCDSensoren.html. [Online; accessed 19-March-2016].

[19] Prasad, Mayank: *Introduction to Single Board Computing.* http://maxembedded.com/2013/07/introduction-to-single-board-computing/. [Online; accessed 09-March-2016].

[20] Seemann, Michael: *So erhöhen Sie die WLAN-Reichweite: 3 Tipps.* http://www.pc-magazin.de/ratgeber/wlan-empfang-verbessern-erweitern-erhoehen-router-repeater-1506595.html. [Online; accessed 04-May-2016].

[21] Seemann, Michael: *So verdoppeln sie Ihre WLAN-Geschwindigkeit.* http://www.pc-magazin.de/ratgeber/

wlan-802-11ac-standard-fakten-details-alle-infos-1942309.html. [Online; accessed 04-May-2016].

[22] Toschka, Patrick: *WLAN-Verbindung schwankt stark – das können Sie tun.* http://praxistipps.chip.de/wlan-verbindung-schwankt-stark-das-koennen-sie-tun_34790. [Online; accessed 04-May-2016].

[23] Welt, PC: *Die besten Ein-Platinen-PCs im Vergleich.* http://www.pcwelt.de/ratgeber/Die_besten_Ein-Platinen-PCs_im_Vergleich-Kleine_Helfer-8998742.html. [Online; accessed 04-May-2016].

[24] Wiersdoerfer, Tim: *OpenCL.* http://ps.informatik.uni-siegen.de/downloads/Seminare/multicore-ws2011/wiersdoerfer.pdf. [Online; accessed 12-January-2016].

[25] Wikipedia: *Odroid.* https://de.wikipedia.org/wiki/ODROID. [Online; accessed 06-March-2016].

[26] Wikipedia: *OpenCL.* https://de.wikipedia.org/wiki/OpenCL. [Online; accessed 05-January-2016].

[27] Wikipedia: *Raspberry Pi.* https://de.wikipedia.org/wiki/Raspberry_Pi. [Online; accessed 09-March-2016].

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kentlich gemacht habe.

Dortmund, den May 11, 2016

Dragana Popovic