ZHAW School of Engineering

Projektarbeit

IT

Measurement of Rate Deviation of Watches with Raspberry Pi Camera

Authors:
Linda Helen Boedi
Valentin Bossi

Supervisor:
Hans-Joachim Gelke

December 18, 2017

Declaration of Autonomy

The undersigned certify with their signature that the work has been written independently and put into written form, that the involvement of other persons has been limited to consulting and proofreading, and that all documents and guarantors used are listed.

Place and Date Linda Helen Bödi

Place and Date Valentin Bossi

Abstract

Mechanical watches must be checked for accuracy both during manufacture and use. Most of the equipment used for this purpose is expensive. Almost all of the devices on the market apply acoustic measurements and a few complement this measurement with an optical measurement by means of a laser beam. In this work it is determined whether a Raspberry Pi and a Raspberry Pi camera can be turned into a cheaper and simple measuring device that can compete with the accuracy of a professional device. In this work a new approach is used: the measurement is carried out optically by means of image processing.

The work will give answers to the following questions: Is a complete optical measurement by means of image processing possible? How accurate will these measurements be and can they keep up with the results of professional equipment? What are possible sources of error? How can the measurement be further improved?

In a first step, possible methods were evaluated in order to carry out the optical measurement of the frequency of the balance wheel of a watch. A closer look was taken at the OpenCV library, the gstreamer, a one-to-one comparison of images, tracking a specific pixel and the Picamera library. Since the Picamera library offers a very detailed documentation and many of the functionalities needed for this work, this library was used. In order to better understand the motion vectors, various experiments were carried out, whereby the camera settings were also improved. After this basis had been established, a first rough calculation of the frequency was done by hand in Excel. Due to a promising result the implementation of a Python program was carried out. It was found that the watch reacts very sensitively to temperatures and runs very inaccurately. Therefore it was difficult to get reliable reference measurements. Therefore, a comparison was made with a blinking LED, which flashes in time to show how the test clock should ideally oscillate back and forth. The measurements of the flashing LED showed that the determination of the frequency by means of image processing worked very well and that sufficiently good results could be achieved. In order to get better direct reference measurements a parallel measurement of the watch with the camera and a Greiner measurement device was made. It showed that the results of both devices are very similar and therefore the measurement with the camera is accurate enough. To have a product, which can compete with the professional devices, some more implementations of measurements of various other things need to be made. Additionally one could do further research, if real-time measurements are possible.

Contents

Chapter 1

Introduction

Today's watch technology requires measuring instruments to perform a wide variety of measurements and analyses on almost all types of watches with great precision and reliability. Mechanical watches must be checked in production and service for their rate deviation. There are different techniques to measure this deviation; on one hand acoustically and optically or only acoustically. The aim of this work is to develop a handy device that can visually measure the rate deviation using inexpensive and simple means. To achieve this, an existing camera consisting of the best-known single-board computer Raspberry Pi and the corresponding camera from Raspberry Pi is used. The aim of this work is to use image processing to measure and analyse the rate deviation of mechanical clocks. The vision of this project is to be able to perform real-time measurements as established providers do with a much cheaper device.

The second chapter will give a background for this work. Acoustical and optical measurement will be explained and compared. One will have a look at the functionality and operation of mechanical watches and the camera module used in this project. In the third chapter different algorithms and libraries for image processing will be discussed and evaluated. The test setup, camera settings and all methods used to get a better understanding of the picamera library and motion vectors are listed in chapter four. The fifth chapter afterwards explains how the frequency was then calculated from the motion vectors and how it was implemented as a program in Python. All the measurements and results are then listed in chapter six. The last chapter summarizes those results roughly again and outlines all the problems that were faced during the work. It will also give an outlook of what could be improved and done further.

Chapter 2

Background

This chapter is intended to lay the foundation for this report. An overview of the existing measuring methods for the accuracy of a mechanical watch is given and the Raspberry Pi camera, which was used for the optical measurements in this project, is described. In addition, the functionality of a mechanical watch is explained and important information about the test watch is listed.

2.1 State of the Art

At the beginning a small historical excursion, or rather an overview of the existing measuring methods for the accuracy of a mechanical watch.

2.1.1 Optical versus Acoustical Measurement

There exist mainly two methods on the market to measure the frequency of the balance wheel of a mechanic clock; acoustical and optical measurement. The commonly used method is to analyse the beat noises of the lever escapement. More expensive devices use both acoustical and optical feedback. In order to classify the rate deviation well enough, it is usually measured in different positions of the watch.

2.1.1.1 Acoustical Measurement

The original method used for frequency measurements of the balance wheel was by using a vibrograph. For every "tick" of the clock a line was drawn onto a ongoing strip of paper. From the distance between of these lines the frequency was calculated [14]. But as this method wasn't very accurate, other techniques have evolved. Modern watch timing machines use an oscillating quartz crystal as comparison for the frequency of the balance wheel [14]. The beat noises of the lever escapement are recorded and amplified with a microphone whereat unwanted background noises need to be filtered out.

2.1.1.2 Optical Measurement

About hundred years later, optical watch timing machines joined acoustical ones. Even nowadays there are barely devices on the market, which employ only optical measurement, but several companies started to combine their acoustical with optical metering. One way to scale the frequency of the balance wheel optically is by using a laser. The beam is periodically interrupted by the balance wheel and thus the frequency can be calculated [12]. In this paper the frequency is quantified optically by using image processing. There is hardly to none company out there, which uses the last technique for measuring the frequency.

In table 2.1 a comparison of acoustical and optical measurements is made and the most important facts are listed.

Comparison acoustical and optical measurement		
	acoustical	**optical**
since	around 19th century [12]	end of 20th century [12]
accuracy	0.1 s/d	0.1 s/d
advantages	- most experience (exists for about 200 years) - tracks if balance wheel is damaged (different noises)	- no background noises, which need to be filtered out
disadvantages	- background noises need to be filtered out	- laser needs to be positioned precisely - balance wheel needs to be visible - good lighting is needed

Table 2.1: Comparison of Acoustical and Optical Measurement

2.1.2 Industrial Products

In the following section the three largest companies producing such measuring instruments are described and the most important facts listed.

2.1.2.1 Witschi - WisioScope S

WisioScope S tests mechanical watches acoustically and optically. The measurement is done in parallel and is more accurate as two signals are used for the calculation of the frequency. The optical metering is done using a laser and lighting. A camera helps to adjust the watch properly. The costs of their product are: CHF 10450.- .

2.1.2.2 Lepsi - WatchScope/WatchAnalyzer

The WatchScope and WatchAnalyzer from Lepsi are especially for watch lovers and not really for production. Both products are Swiss technologies. Both devices only work with an acoustic input with measurements of either a few seconds or up to 24 hours. All data can then be queried with the smartphone. WatchScope is the smaller, more convenient version. With the WatchAnalyzer you can measure the rate deviation of the watch in several positions. The prices for this product are CHF 369.- resp. CHF 929.- .

2.1.2.3 Greiner Vibrograf - Compact 900

The Compact 900 measures the beat noises of the lever escapement. The price is CHF 4070.- .

The following table summarizes all important information for the three professional devices.

Measurement rate deviation			
	Witschi - WisioScope S	**Lepsi - WatchScope/Watch-Analyzer**	**Greiner Vibrograf - Compact 900**
Scope	+/- 999.9 s/d	+/- 1000 s/d	+/- 1000 s/d
Resolution	0.1 s/d	0.1 s/d	0.1 s/d
Price	CHF 10450.-	CHF 369.-, resp. CHF 929.-	CHF 4070.-

Table 2.2: Measurement rate deviation with different professional devices

2.2 Camera Module

The documentation for the picamera library [2] provides a detailed introduction to the operation of the Raspberry Pi with camera, which is used for all measurements in this project. This chapter will briefly summarize this introduction to better understand the hardware and operation of the Raspberry Pi camera.

The camera module of the Raspberry Pi is basically a mobile phone camera module. Among other things, it uses a rolling shutter to take pictures. The pixel values of a frame are not captured completely at once, but are read line by line.

However, the sensor is configured via the registers with the number of lines to be read and the corresponding time. The sensor reads the lines and pushes the data with the configured speed to the Raspberry Pi.

This keeps the readout time of each line constant. However, the CPU does not "speak" directly to the camera, but the processing rather takes place on the GPU (VideoCore IV) of the Raspberry Pi, which operates its own real-time operating system (VCOS).

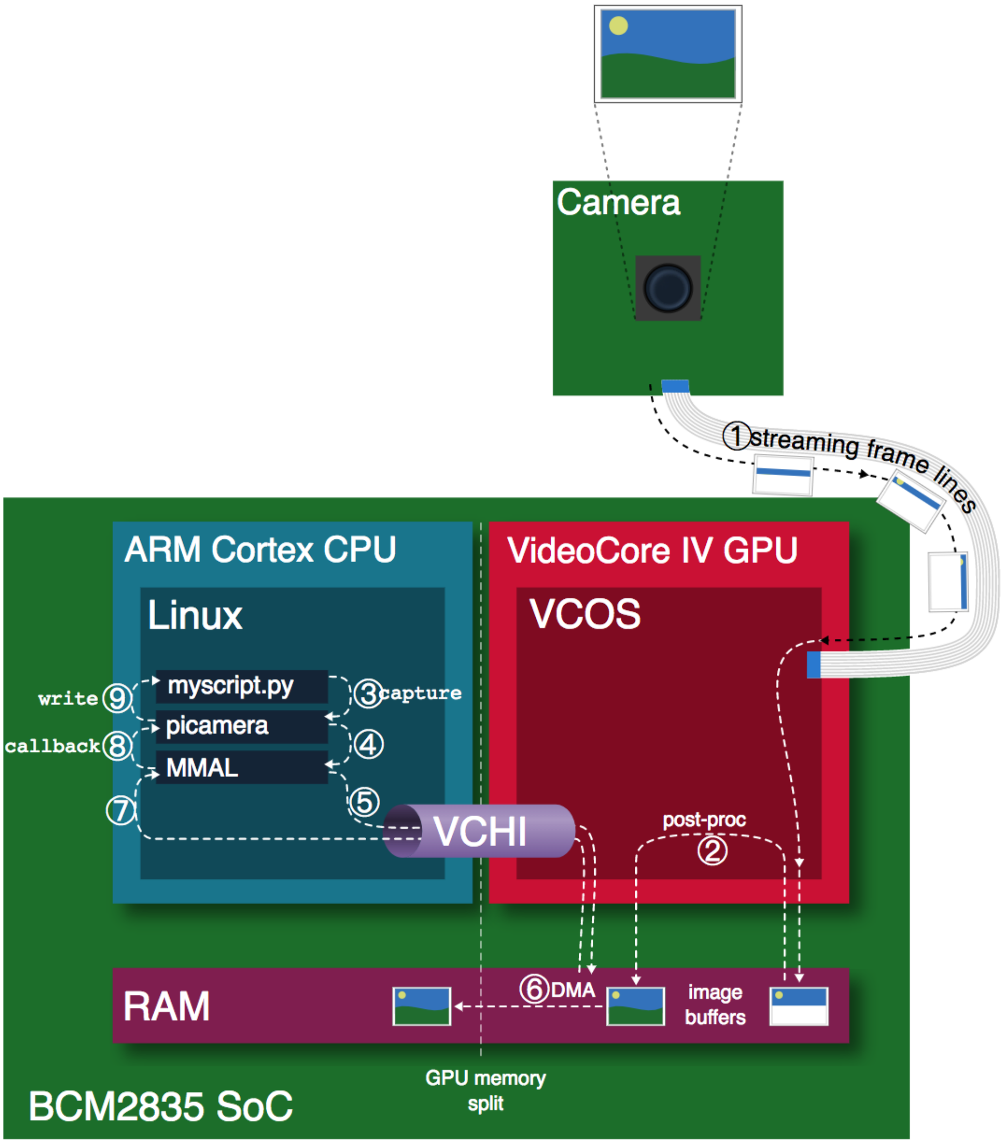


Figure 2.1: Camera architecture [2]

The figure 2.1 illustrates the processing flow of a frame, with the associated steps explained in the following section.

1. The camera's sensor is configured and continuously streams frame lines to the GPU.
2. The GPU builds complete frame buffers from these lines and performs the post-processing of these buffers.

3. Myscript.py makes a capture call with the picamera on the CPU.

4. The picamera library uses the MMAL API to meet this requirement.

5. The MMAL API sends a message via VCHI requesting a frame capture.

6. The GPU initiates a DMA transmission of the next full frame from its RAM portion to the CPU portion.

7. The GPU sends back a message via VCHI that the capture is complete.

8. This causes an MMAL thread to trigger a callback in the picamera library, which in turn retrieves the frame.

9. Picamera calls "write" on the output object provided by myscript.py.

2.2.1 Exposure Time

The camera sensor detects how many photons hit the sensor elements, because the more impact, the more they increase their counter values. The sensor can perform exactly two operations; reset a set of elements or read a set of elements.

2.2.1.1 Minimal Exposure Time

Reading out a series of elements takes a certain amount of time, thus there is a limit to the minimum exposure time. Assuming one has 500 lines on a sensor and reading each line takes at least 20ns, then it will take at least $500 * 20\text{ns} = 10\text{ms}$ to read a full screen.

2.2.1.2 Maximum Framerate

The frame rate is the number of frames the camera can capture per second. The exposure time determines the maximum number of images that can be taken in a given time. Assuming it takes 10ms to read a complete image, then no more than $\frac{1\text{s}}{10\text{ms}} = \frac{1\text{s}}{0.01\text{s}} = 100$ images are read in one second.

So it is valid:

$$\frac{1\text{s}}{\text{min exposure time in s}} = \text{max framerate in fps.}$$

The lower the minimum exposure time, the higher the maximum frame rate and vice versa.

2.2.1.3 Maximum Exposure Time

To maximize shutter speed, the framerate needs to be reduced.

Therefore the following is valid :

$$\frac{1\text{s}}{\text{min framerate in fps}} = \text{max exposure time in s}$$

2.2.2 Hardware Limits

- The picamera originally has a very limited zoom to record the watch properly a lens was added to the camera in this project.
- The maximum horizontal resolution for the standard H264 recording is 1920 (this is a limitation of the H264 block in the GPU).
- The maximum frame rate of the camera depends on several factors. With overclocking 120 fps can be achieved, but 90 fps is the maximum supported frame rate.

2.3 Mechanical Watches

In the following, the function of a mechanical watch is explained in more detail and the role of a watch's balance wheel is explained. Subsequently, important information on mechanical watches, such as the duration of stroke and facts about the test watch used in this project, are described.

2.3.1 Operation of a Mechanical Watch

In figure 2.2 the components of a mechanical watch are illustrated. In the following the operation and interaction of those components is described. If a mechanical watch is wound up over the crown, the locking wheel (3) is moved by the winding shaft (1) and winding wheels (2). It turns the spring core (4) and pulls the tension spring in the mainspring barrel (5). The spring transmits the stored energy via the external toothed mainspring barrel (5) to the minute wheel (6), third wheel (7) and fourth wheel (8).

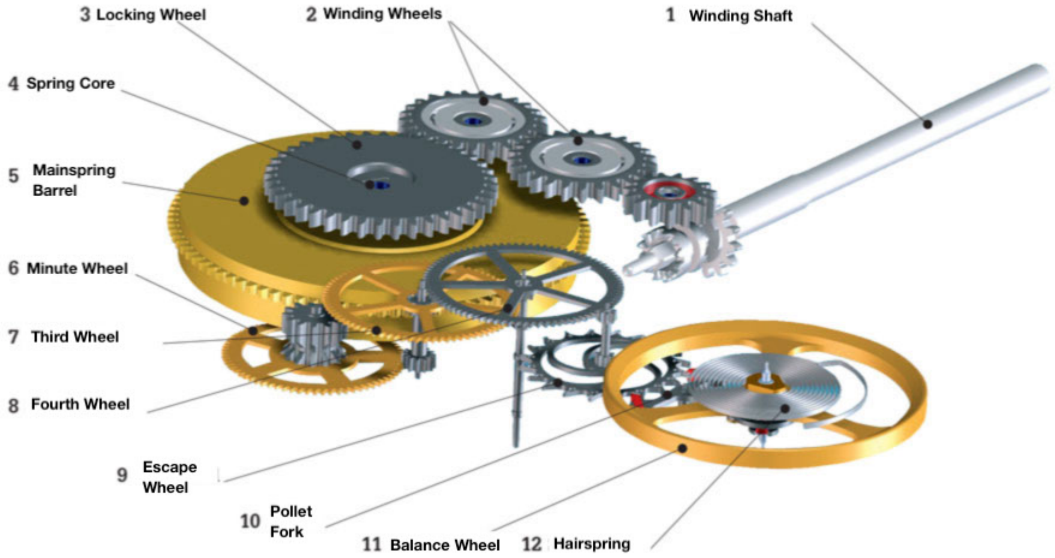


Figure 2.2: Operation of a mechanical watch [5]

The so-called escapement ensures that the gear train runs at the correct speed. The fourth wheel (8) drives the escape wheel (9); it gives an impulse to the pallet fork (10) which it passes on to the balance wheel (11), after which the armature, which moves back and forth like a seesaw, blocks the escape wheel. The balance wheel rotates, but is then retracted by the hairspring (12). It moves the armature back, which now releases the armature wheel again a little bit, which gives the armature with its rotation the next impulse. Due to the sudden braking and acceleration of the escapement wheel, the second hand of a mechanical watch also moves stepwise [5].

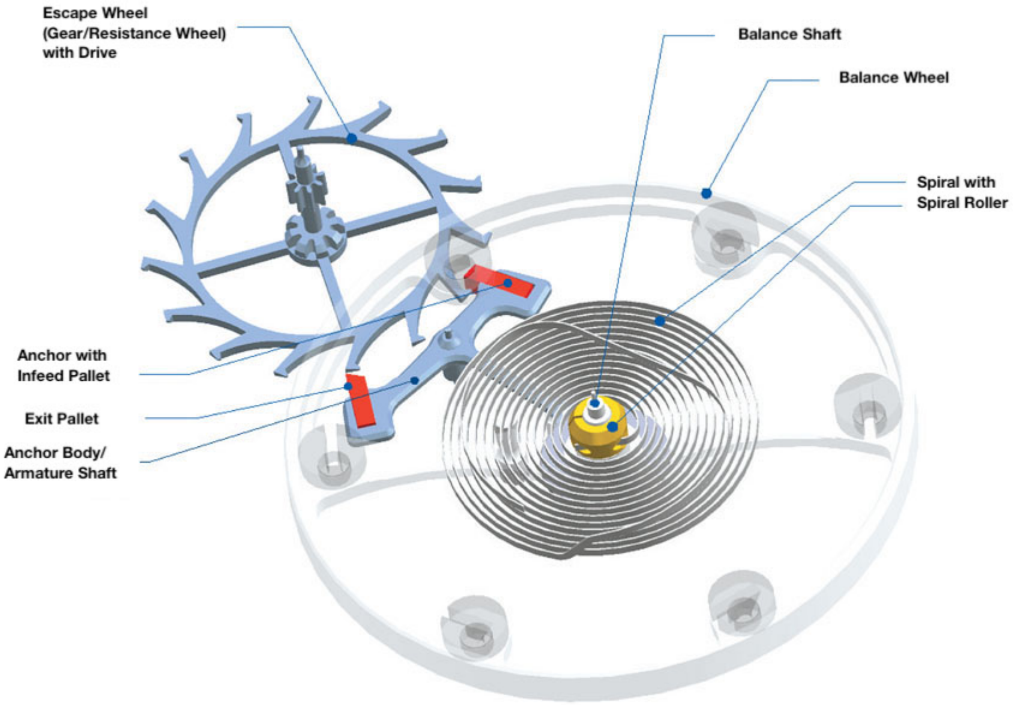


Figure 2.3: Lever escapement and balance wheel [5]

The balance wheel, which is shown in figure 2.3 is the heart of the oscillation system. It generates a time-defined movement, which in turn is transferred to the gear train and passed on to the watch hands. The occurring error, if the balance wheel does not run accurately, is called rate deviation. The deviation can be caused by different external and internal matters. Internal motives often are wear, resp. erosion or dirt. Whereas there are a lot more external causes as changes in temperature, air pressure or magnetism.

2.3.2 Oscillation of the Balance Wheel

In the following section the movement of the balance wheel, which will be recorded and used for the calculation of the frequency, is explained.

2.3.2.1 Period

The period of the balance wheel is the path of a point from one turning point to the other and back (A - B - A) as shown in figure 2.4 [3].

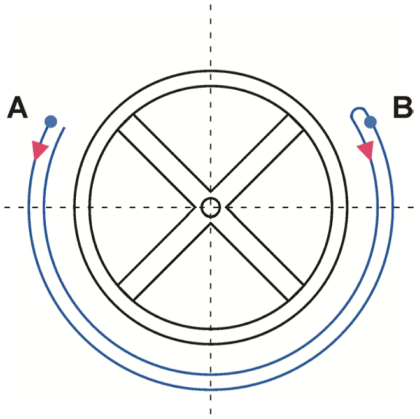


Figure 2.4: Period of the balance wheel [3]

2.3.2.2 Half a Period

In figure 2.5 half of a period, resp. one stroke, of the balance wheel is illustrated. It is the path of a point from one turning point to the other (A - B) [3].

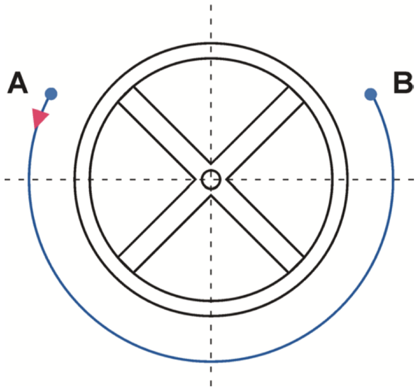


Figure 2.5: Half of a period of the balance wheel [3]

2.3.3 Number of Strokes

Since the rate deviation has been calculated in seconds per day and the number of strokes per hour, i. e. the number of audible impulses of a two-armed pallet fork of a mechanical watch per hour, is known, these two quantities must first be brought to a unit. The relationship between the number of strokes (n*) and frequency (f) is as follows [10]:

$$n* = 2f * 3600$$

and therefore:

$$f = \frac{n*}{7200}$$

The following table lists important facts about mechanical wristwatches. Among other things, it contains the number of strokes per hour, the duration for one stroke - i. e. half a period - and the frequency.

Number of strokes [1/h]	**Stroke duration** [s]	**Oscillation number** [1/h]	**Period duration** [s]	**Frequency** [Hz]
18'000	0.200	9'000	0.400	2.50
19'800	0.182	9'900	0.364	2.75
21'600	0.166	10'800	0.333	3.00
28'800	0.125	14'400	0.250	4.00
36'000	0.100	18'000	0.200	5.00

Table 2.3: Number of strokes, period duration and frequency of the balance of automatic wristwatches [10]

2.3.4 Test Watch

The test watch, which was used, is of the caliber ETA C01.211. It is a chronograph, with a diameter of 31 centimeters. It strokes 21600 times per hour with a stroke duration of 0.1666 seconds. It therefore has a frequency of 3 Hz. Additionally the watch has a power reserve of 43 hours [1].

Chapter 3

Technologies and Algorithms

The first library is **OpenCV**, an open source library, which is suitable and widely used for the programming languages C++, C, Python and Java. OpenCV combines different algorithms for image processing and machine vision. It offers modules that implement algorithms that can solve common problems such as face recognition, gesture recognition, object recognition, tracking and optical flow, to name but a few [13].

The next library is **Gstreamer**, which is also open source. However, it is mainly programmed in C. As the name suggests, this library is mainly used for multimedia data streams, such as media players and video editing software [7].

As the last library, the eye was casted on the **Picamera** library. This is also an open source project and can only be programmed in Python. The library is an interface to the MMAL -written in C- which in turn is an interface to the firmware of the GPU. This means that the camera can be accessed using a simple programming language such as Python. The picamera library is very strongly tailored to the raspberry pi camera, but offers a wide range of functionalities [2].

3.1 Optical Flow

The optical flow represents the vector field of the velocities of the points of an image sequence. It is a useful representation of motion information in image processing. The local optical flow is an estimate of patterns in an image in the vicinity of a viewed pixel.

One possibility would be to follow the course of a velocity vector and thereby calculate a period of the frequency of the balance wheel. The calculation of the flow at selected points is also called feature point tracking. Another approach would be to calculate the frequency based on the velocity obtained from the optical flow, since it is more or less a circular movement.

One difficulty with this procedure is that one has to commit to certain pixels that one examines. This can cause problems if the clock is placed differently, because wrong points can cause problems for the calculation. In the worst-case scenario points, in which movement never takes place, would be examined.

3.1.1 Sparse Optical Flow

The OpenCV library uses the Lucas-Kanade method to calculate the optical flow. The Lucas-Kanade method also assumes that the flow in the local environment of the pixel for which the flow is intended is the same. So more or less a whole set of pixels is considered. The flow can thus be determined by calculating the derivatives (= gradients) [13].

One characteristic of this method is that it does not provide a dense flow this means the flow information disappears quickly with the distance from the edges of the image. The advantage of the method is its relative robustness against noise and smaller defects in the image.

3.1.2 Dense Flow

The dense optical flow calculates the flow over all pixels of an image, so the whole image is processed. This method is more accurate than the sparse optical flow method, but also much slower.

The implementation in Open CV is baed on the Gunner Farneback's algorithm which is explained in "Two-Frame Motion Estimation Based on Polynomial Expansion" by Gunner Farneback in 2003 [4].

3.2 Kalman Filtering

Using the Kalman filter, errors in real measurement values can be reduced and estimates for unmeasurable system variables can be provided. For this purpose, the values which are to be examined in more detail are described by a mathematical model, for example in the form of equations of motion. The special mathematical structure of the Kalman filter makes it possible to use it in real-time systems in different areas, such as for example for the evaluation of GPS data to determine the position of moving objects - tracking- or for augmented reality [6].

3.3 Object tracking

Also called video tracking and is a process in which a moving object is tracked in a video. To achieve this, not only an algorithm for object tracking is used, but also an algorithm for object detection. Depending on the application, this process is computationally intensive. This procedure could be used to track the movement of the balance wheel and find out how fast it moves and when it stops [9].

3.4 Motion vectors

Motion vectors are used in video compression to reduce resources required to store and transmit data. A motion vector describes the displacement of a pixel or pixel block within an image sequence.(referenz zu wiki oder buch.) The motion vectors can be used to describe how much change is taking place in an image sequence. Thus can be used to identify a moving or still balance wheel.

3.5 Conclusions and Evaluations

Because in this project a mini computer is used and a realtime measurement is to be made, many presented technologies and algorithms are already excluded because they are too computing-intensive. This means that at 90 frames per second, a calculation on the image data must be done within $1/90$ seconds, no matter how computing-intensive it is. In addition, there is the problem with object tracking, for example, that there are many different shapes of balance wheels and that the software would have to be adapted for each new shape of balance wheel in order to make the object detection work again The same applies to procedures that work with colors. As soon as a balance wheel has a different color, the procedure would have to be adapted.

In table 3.1 it is listed which methods are already implemented in which library, how high the computational effort is and if it is generalizable, so that the measurements are always more or less the same and the camera does not need to be positioned precisely in the exact same position every time.

			OpenCV	**Gstreamer**	**Picamera**
	Generalizability	Computational Intensity	Implemented and Easy Accessible		
Optical Flow	✓	high	✓	✗	✗
Kalman Filtering	✓	high	✓	✗	✗
Object Tracking	✓	high	✓	✗	✓
Motion Vectors and SAD	✓	very low	✗	✗	✓

Table 3.1: Evaluation of Different Algorithms

As the computational effort is relatively low with the picamera library and the motion vectors are already reprocessed and easy accessible this library was chosen for the calculation of the frequency of mechanical watches via image processing.

Chapter 4

Experimental approaches

After deciding which library to use, some experimental sessions took place to get a better understanding of the so called motion data values or motion vectors. With the picamera library it is easy to access these motion data values so the hard part wasn't to get the values but to understand how they are constructed. The motion data object from picamera consists of multiple values: a signed 1-byte x vector, a signed 1-byte y vector and and unsigned 2-byte SAD value for each macro-block of 16x16 pixel of a frame.

4.1 Test setup

Figure 4.1 shows the test setup. On average the camera was positioned approximately four centimeters from the test watch. In order to ensure that the images are still well illuminated and a good measurement can be carried out due to the high shutter speed, a small light source directed at the test watch was used. The test watch was positioned on a turn table so it was easy to adjust the position of the watch.

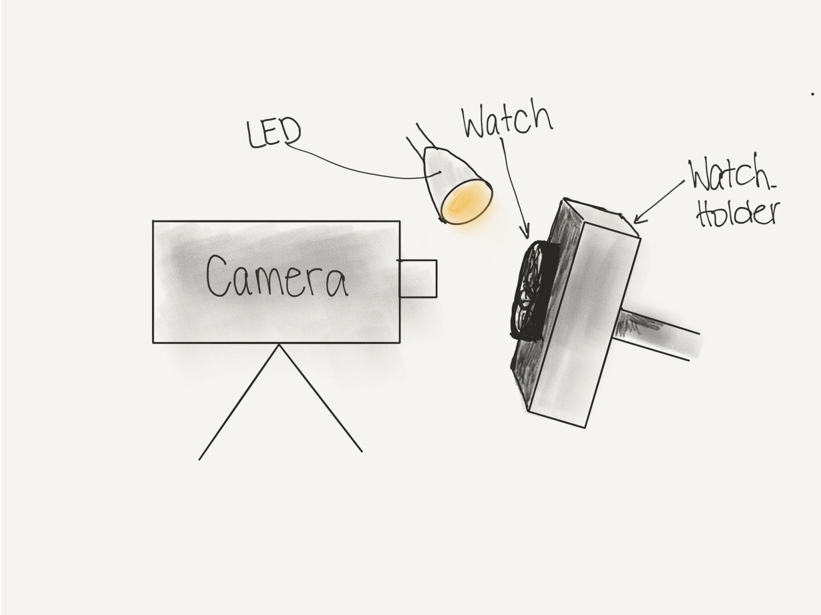


Figure 4.1: Test setup for records

4.2 Gesture Detection

A first attempt was to move the hand in front of the camera down, up, right and left and to determine the direction in real time based on the motion vectors.
For this, the average of all x-values and all y-values of the motion vectors of a frame are calculated and compared with a minimum value of movement that was defined. Depending on whether the x-average value is lower or higher than zero, a movement to the left or right has been detected. Likewise, it works for the y-average with the movement up or down.
As this worked quite well, the balance wheel of the test watch was recorded and it was tried out to detect the movement with the same program. Unfortunately, this program can only detect very clear movements. With very minimal movements this program fails, therefore the frequency of the balance wheel could not be calculated with this implementation.

4.3 Animation of Motion Vectors

Another approach was animating all SAD values with colors to get a better understanding of how good the information they keep is and whether they can be used to calculate the frequency of the movement of the balance wheel. In figure 4.2 a snapshot of the animation of the SAD values is shown. The lighter the colour of a pixel, the higher is the value of the SAD.

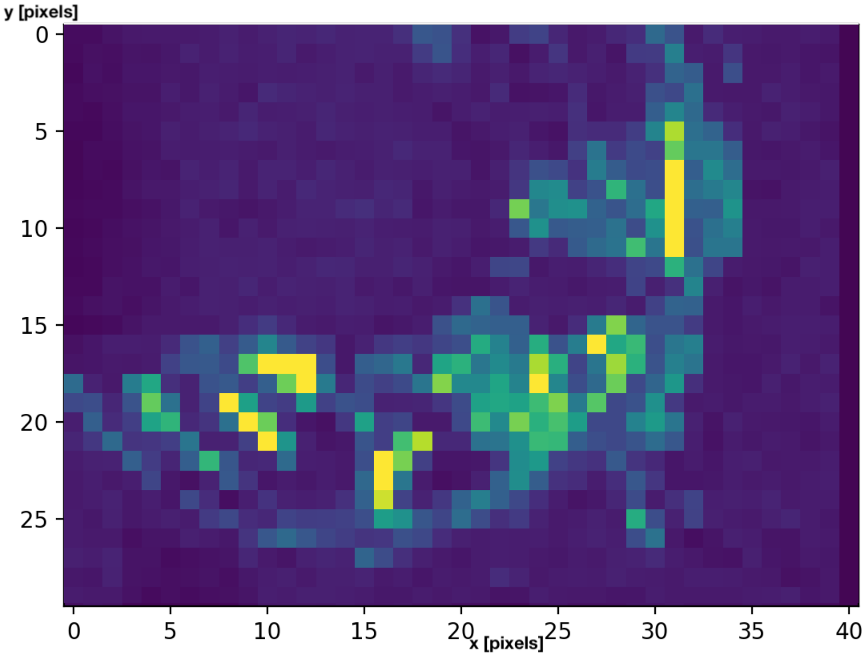


Figure 4.2: Snapshot of the animation of the SAD values

Further the x and y vector were analyzed. But as those vectors itself are hard to use and only give partial information (y-vector: up or down, x-vector: right or left), the hypotenuses were calculated with Pythagoras' theorem and displayed similar to the SAD values as displayed in figure 4.3. The hypotenuses give information about the amount as well as the direction of the motion. The lighter the colour of a pixel, the higher is the value of the hypotenuse.

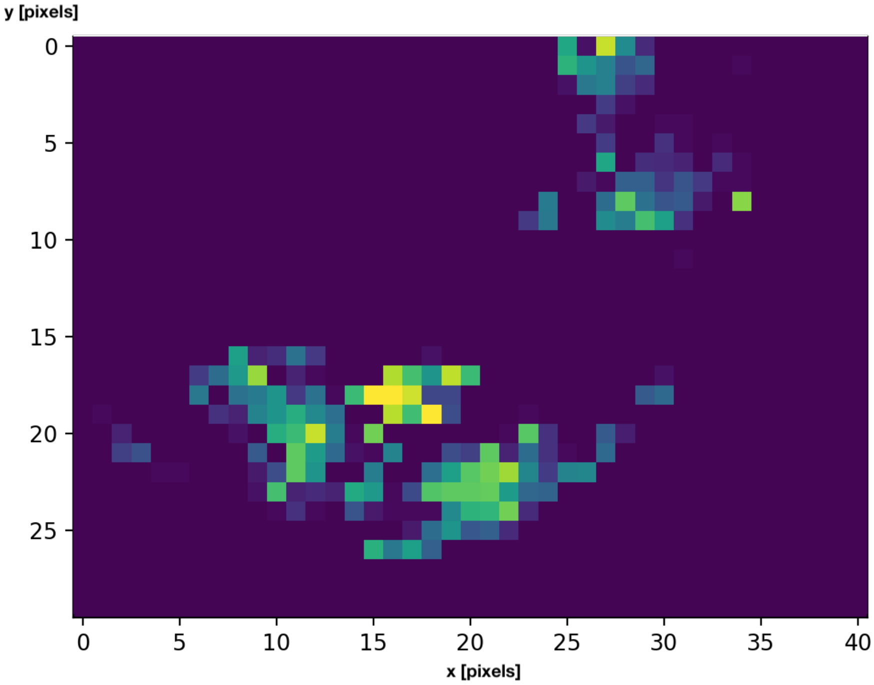


Figure 4.3: Snapshot of the animation of the hypotenuses

4.4 Motion Vectors Visualized in a Frame

Another experimental approach was displaying the motion vectors as arrows directly in the video. This approach was helpful to get a better understanding of the conditions that yield the best result of the motion vectors. Codecvisa was used to display the motion vectors of a frame as arrows. The larger the arrow, the higher is the motion of this pixel. As one can see in figure 4.4 the movement of the balance wheel is detected very well, as the arrows are bigger there.

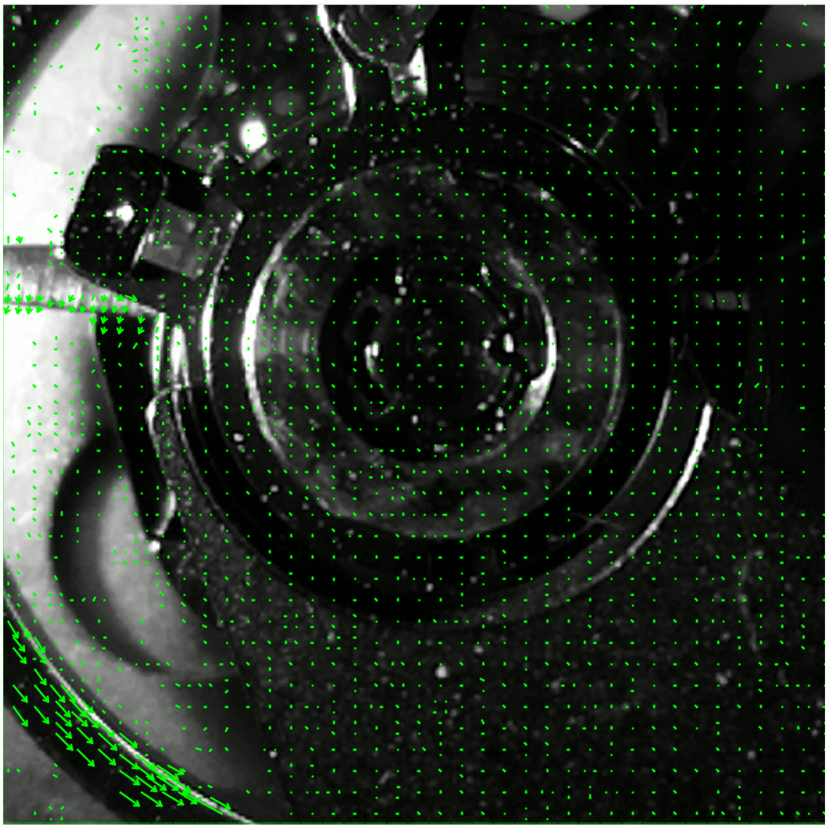
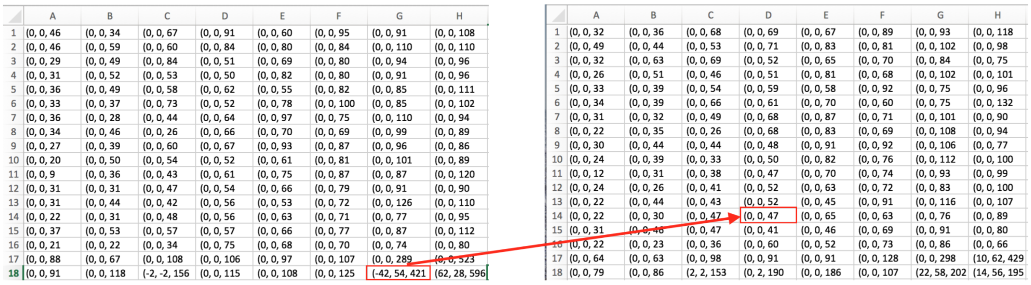


Figure 4.4: Motion vectors drawn in one frame

4.5 Analysis of Motion Vectors in Excel

Another approach used Excel was to determine whether a motion vector from one frame is the predecessor of the motion vector in the next frame. The hope was that a moving pixel could be followed and therefore the location of the pixel would always have been known.



	A	B	C	D	E	F	G	H
1	(0, 0, 46	(0, 0, 34	(0, 0, 67	(0, 0, 91	(0, 0, 60	(0, 0, 95	(0, 0, 91	(0, 0, 108
2	(0, 0, 46	(0, 0, 59	(0, 0, 60	(0, 0, 84	(0, 0, 80	(0, 0, 84	(0, 0, 110	(0, 0, 110
3	(0, 0, 29	(0, 0, 49	(0, 0, 84	(0, 0, 51	(0, 0, 69	(0, 0, 80	(0, 0, 94	(0, 0, 96
4	(0, 0, 31	(0, 0, 52	(0, 0, 53	(0, 0, 50	(0, 0, 82	(0, 0, 80	(0, 0, 91	(0, 0, 96
5	(0, 0, 36	(0, 0, 49	(0, 0, 58	(0, 0, 62	(0, 0, 55	(0, 0, 82	(0, 0, 85	(0, 0, 111
6	(0, 0, 33	(0, 0, 37	(0, 0, 73	(0, 0, 52	(0, 0, 78	(0, 0, 100	(0, 0, 85	(0, 0, 102
7	(0, 0, 36	(0, 0, 28	(0, 0, 44	(0, 0, 64	(0, 0, 97	(0, 0, 75	(0, 0, 110	(0, 0, 94
8	(0, 0, 34	(0, 0, 46	(0, 0, 26	(0, 0, 66	(0, 0, 70	(0, 0, 69	(0, 0, 99	(0, 0, 89
9	(0, 0, 27	(0, 0, 39	(0, 0, 60	(0, 0, 67	(0, 0, 93	(0, 0, 87	(0, 0, 96	(0, 0, 86
10	(0, 0, 20	(0, 0, 50	(0, 0, 54	(0, 0, 52	(0, 0, 61	(0, 0, 81	(0, 0, 101	(0, 0, 89
11	(0, 0, 9	(0, 0, 36	(0, 0, 43	(0, 0, 61	(0, 0, 75	(0, 0, 87	(0, 0, 87	(0, 0, 120
12	(0, 0, 31	(0, 0, 31	(0, 0, 47	(0, 0, 54	(0, 0, 66	(0, 0, 79	(0, 0, 91	(0, 0, 90
13	(0, 0, 31	(0, 0, 44	(0, 0, 42	(0, 0, 56	(0, 0, 53	(0, 0, 72	(0, 0, 126	(0, 0, 110
14	(0, 0, 22	(0, 0, 31	(0, 0, 48	(0, 0, 56	(0, 0, 63	(0, 0, 71	(0, 0, 77	(0, 0, 95
15	(0, 0, 37	(0, 0, 53	(0, 0, 57	(0, 0, 57	(0, 0, 66	(0, 0, 77	(0, 0, 87	(0, 0, 112
16	(0, 0, 21	(0, 0, 22	(0, 0, 34	(0, 0, 75	(0, 0, 68	(0, 0, 70	(0, 0, 74	(0, 0, 80
17	(0, 0, 88	(0, 0, 67	(0, 0, 108	(0, 0, 106	(0, 0, 97	(0, 0, 107	(0, 0, 289	(0, 0, 523
18	(0, 0, 91	(0, 0, 118	(-2, -2, 156	(0, 0, 115	(0, 0, 108	(0, 0, 125	(-42, 54, 421	(62, 28, 596

	A	B	C	D	E	F	G	H
1	(0, 0, 32	(0, 0, 36	(0, 0, 68	(0, 0, 69	(0, 0, 67	(0, 0, 89	(0, 0, 93	(0, 0, 118
2	(0, 0, 49	(0, 0, 44	(0, 0, 53	(0, 0, 71	(0, 0, 83	(0, 0, 81	(0, 0, 102	(0, 0, 98
3	(0, 0, 32	(0, 0, 63	(0, 0, 69	(0, 0, 52	(0, 0, 65	(0, 0, 70	(0, 0, 84	(0, 0, 75
4	(0, 0, 26	(0, 0, 51	(0, 0, 46	(0, 0, 51	(0, 0, 81	(0, 0, 68	(0, 0, 102	(0, 0, 101
5	(0, 0, 33	(0, 0, 39	(0, 0, 54	(0, 0, 59	(0, 0, 58	(0, 0, 92	(0, 0, 75	(0, 0, 96
6	(0, 0, 34	(0, 0, 39	(0, 0, 66	(0, 0, 61	(0, 0, 70	(0, 0, 60	(0, 0, 75	(0, 0, 132
7	(0, 0, 31	(0, 0, 32	(0, 0, 49	(0, 0, 68	(0, 0, 87	(0, 0, 71	(0, 0, 101	(0, 0, 90
8	(0, 0, 22	(0, 0, 35	(0, 0, 26	(0, 0, 68	(0, 0, 83	(0, 0, 69	(0, 0, 108	(0, 0, 94
9	(0, 0, 30	(0, 0, 44	(0, 0, 44	(0, 0, 48	(0, 0, 91	(0, 0, 92	(0, 0, 106	(0, 0, 77
10	(0, 0, 24	(0, 0, 39	(0, 0, 33	(0, 0, 50	(0, 0, 82	(0, 0, 76	(0, 0, 112	(0, 0, 100
11	(0, 0, 12	(0, 0, 31	(0, 0, 38	(0, 0, 47	(0, 0, 70	(0, 0, 74	(0, 0, 93	(0, 0, 99
12	(0, 0, 24	(0, 0, 26	(0, 0, 41	(0, 0, 52	(0, 0, 63	(0, 0, 72	(0, 0, 83	(0, 0, 100
13	(0, 0, 22	(0, 0, 44	(0, 0, 43	(0, 0, 52	(0, 0, 45	(0, 0, 91	(0, 0, 116	(0, 0, 107
14	(0, 0, 22	(0, 0, 30	(0, 0, 47	(0, 0, 47	(0, 0, 65	(0, 0, 63	(0, 0, 76	(0, 0, 89
15	(0, 0, 31	(0, 0, 46	(0, 0, 47	(0, 0, 41	(0, 0, 46	(0, 0, 69	(0, 0, 91	(0, 0, 80
16	(0, 0, 22	(0, 0, 23	(0, 0, 36	(0, 0, 60	(0, 0, 52	(0, 0, 73	(0, 0, 86	(0, 0, 66
17	(0, 0, 64	(0, 0, 63	(0, 0, 98	(0, 0, 91	(0, 0, 91	(0, 0, 128	(0, 0, 298	(10, 62, 429
18	(0, 0, 79	(0, 0, 86	(2, 2, 153	(0, 2, 190	(0, 0, 186	(0, 0, 107	(22, 58, 202	(14, 56, 195

Figure 4.5: Motion vectors drawn in one frame

Two successive frames are illustrated in figure 4.5. Every field with the three values x, y and sad is a 16x16 macro block. The idea was to calculate from those values where the pixel will go. For example the field G18 contains for the x-value -42 and for the y-value 54. If those values are divided by 16, as the macro block has a size of 16x16, the idea was to add those resulting values rounded to the considered field and get the next field, where this macro block might have moved. This is displayed as the red field, D14, in the right excel snippet. From the fields with high values in it, which is an indicator for a lot of movement, it was expected to get the movement of the balance wheel. The result was sobering because one frame and motion vector cannot be associated with another frame and motion vector. This experiment rests on the idea of the optical flux with the movement vectors of the MMAL (Multi-Media Abstraction Layer) as displayed in figure 4.6 .

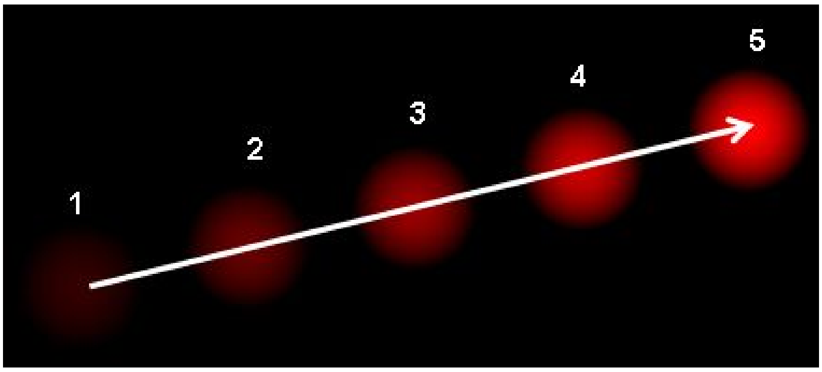


Figure 4.6: It shows a ball moving in 5 consecutive frames. The arrow shows its displacement vector.

4.6 Influences of Camera Setup and Parameters

Testing with different settings showed that the results can be improved. For example, the black and white records are the best choice. Reducing the shutter speeds can reduce motion blur, which also contributes to a better result. The consequence of the shorter exposure time is that the light source must be very strong and close to the subject, otherwise the image is too dark and has a negative effect on the result. Records with changed angles were also taken. The angle was changed by about 30 degrees. With this adjustment, no significant changes in the measured values could be detected. Furthermore, it was recognized that the selected image detail influences the measured values. During the video recordings it was taken care that the balance wheel takes up the entire image section.

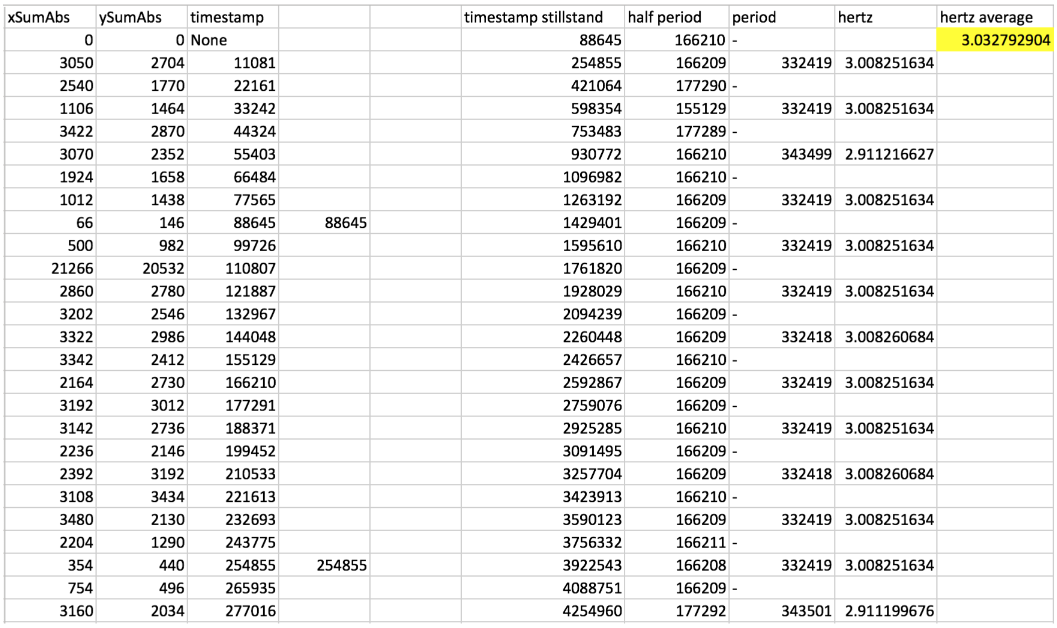
Chapter 5

Calculation of the frequency

After the experimental sessions with the motion vectors it became clear, how the information contained in the motion vectors could be used to calculate the oscillation of the balance wheel. At first a crude calculation of it was made by hand in Excel and the gained insights were then used to start implementing a program in Python. The following chapter will explain how those calculations and implementations were made.

5.1 Calculation of Frequency in Excel

With the help of the motion vectors and the corresponding time stamps per frame, a rough calculation of the frequency can be made. All x, y or SAD values contained in the image are summed up. If the sum of all values is small, it is very likely that the balance wheel is at a standstill before turning back again. For small values, the time stamp is registered and the difference to the next standstill is calculated. Since the balance wheel triggers half a oscillation, it has moved from one stop to the next half point. In order to roughly calculate the frequency, an Excel sheet was created - figure 5.1 is a snippet - in which all the accumulated x- and y-values were entered and the standstills of the balance wheels were determined by eye. The half-oscillations and the frequency were calculated from the time intervals of the standstills. Afterwards, the average of all calculated frequencies was determined and it was recognized that the results with this method should be good enough to determine the frequency and thus the rate deviation.



xSumAbs	ySumAbs	timestamp			timestamp stillstand	half period	period	hertz	hertz average
0	0	None			88645	166210	-		3.032792904
3050	2704	11081			254855	166209	332419	3.008251634	
2540	1770	22161			421064	177290	-		
1106	1464	33242			598354	155129	332419	3.008251634	
3422	2870	44324			753483	177289	-		
3070	2352	55403			930772	166210	343499	2.911216627	
1924	1658	66484			1096982	166210	-		
1012	1438	77565			1263192	166209	332419	3.008251634	
66	146	88645	88645		1429401	166209	-		
500	982	99726			1595610	166210	332419	3.008251634	
21266	20532	110807			1761820	166209	-		
2860	2780	121887			1928029	166210	332419	3.008251634	
3202	2546	132967			2094239	166209	-		
3322	2986	144048			2260448	166209	332418	3.008260684	
3342	2412	155129			2426657	166210	-		
2164	2730	166210			2592867	166209	332419	3.008251634	
3192	3012	177291			2759076	166209	-		
3142	2736	188371			2925285	166210	332419	3.008251634	
2236	2146	199452			3091495	166209	-		
2392	3192	210533			3257704	166209	332418	3.008260684	
3108	3434	221613			3423913	166210	-		
3480	2130	232693			3590123	166209	332419	3.008251634	
2204	1290	243775			3756332	166211	-		
354	440	254855	254855		3922543	166208	332419	3.008251634	
754	496	265935			4088751	166209	-		
3160	2034	277016			4254960	177292	343501	2.911199676	

Figure 5.1: Excel sheet with summed x-, y-values, timestamps and calculated frequency

Figure 5.2 is a graph of all summed x-values respectively y-values over time and it was created out of the same excel sheet in figure 5.1. The value over 20000 at the beginning almost always appeared, but as it does not have any impacts on the measurements of the standstills of the balance wheel and only showed up at the beginning, it can be ignored.

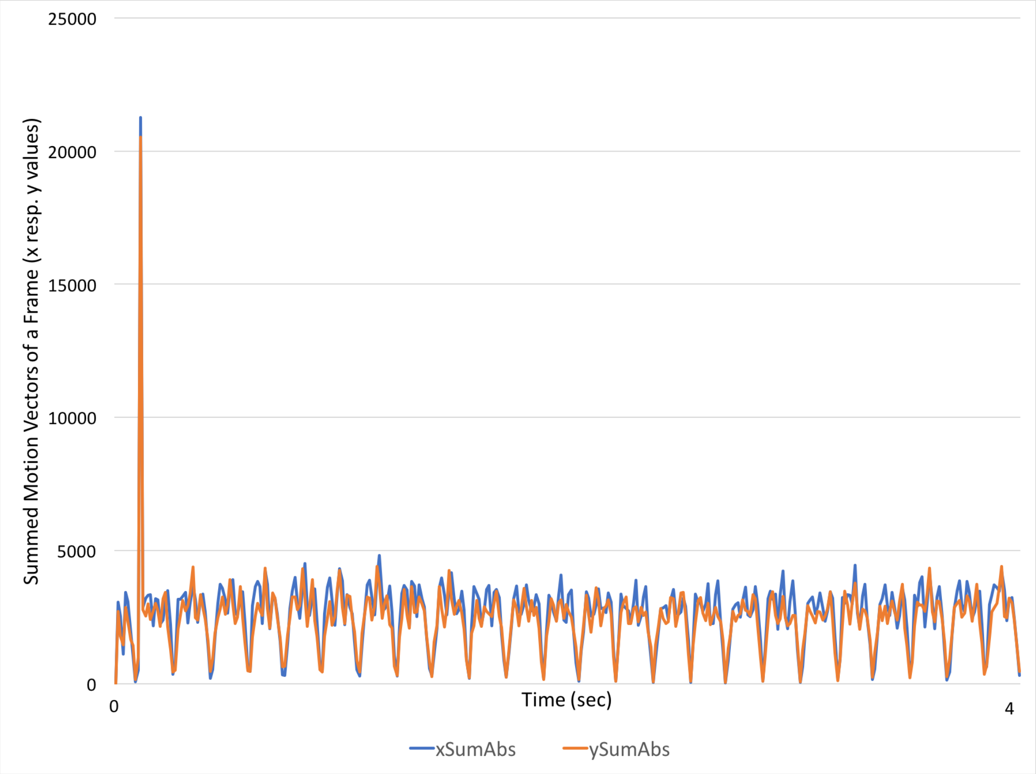


Figure 5.2: Illustration of summed x-(blue), y-values (red) over time. The x-axis is the time and the y-axis are the x- and y-values.

5.2 Implementation of the Algorithms

The implementation of a Python program was based on the procedure in the rough calculation from the Excel spreadsheet. During the first implementation, it turned out that there are different methods for calculating the frequency and it resulted in three different implementations, which are explained in more detail in the following sections.

5.2.1 Counting Minima with Upper Limit

The greatest difficulty proved to be the detection of the minimia, which characterize the standstill of the balance wheel. These were relatively easy visible to the unaided eye, but in an automatic calculation, appropriate conditions had to be set in order to take into account the correct values and correct the inaccuracies caused by the image noise. The setting of an upper limit for the minima proved to be a suitable condition. All minima below this limit are therefore taken into account.

Furthermore, the calculation of the frequency in the Python program has been further simplified by not calculating each period individually, but by selecting the first and the last minimum over the whole period of time, in which it was recorded. This time span is then calculated by half of the number of minima minus 1. Half, because every half of the period the balance wheel stops and minus 1, because otherwise a half-period would be taken into account too much. This approach reduces the number of rounding errors, resulting in higher accuracy. After this calculation, the period duration T is now available, from which the frequency f can be calculated quite simply:

$$f = \frac{1}{T}$$

5.2.2 Counting Minima with Stepsize

To ensure that minimas are not missed during counting, step sizes were used, that means the specifications where the minimas are to be searched for. In the measurements with the test watch, which has a stroke number of 6 per second, a frame rate of 90 was used and therefore every 15th value should show a standstill or a minimum. To detect deviations, a search window of size 5 has been defined. For this way, two further values are compared to the left and right of the nominal step size and 15th steps are added from the smallest value of these 5. This method does not require defining a threshold for the minima searching, which is often difficult to set. On the other hand one has to know the ideal frequency of the watch to calculate the stepsize.

5.2.3 Counting Minima with Tic Toc and Stepsize

The implementation of the program for calculating the frequency by counting the minimums with a defined step size has been adapted so that the calculation is carried out in a similar way as with the professional devices. In the previous implementation, the length of one half-period after another was measured and then the frequency calculated. However, the semi-oscillation may differ in one direction from the semi-oscillation in the other direction, so the accuracy of the program can be improved if the length is measured from one tic to the next and exactly the same for the beat on the other side, the tac, as shown in the figure 5.3.

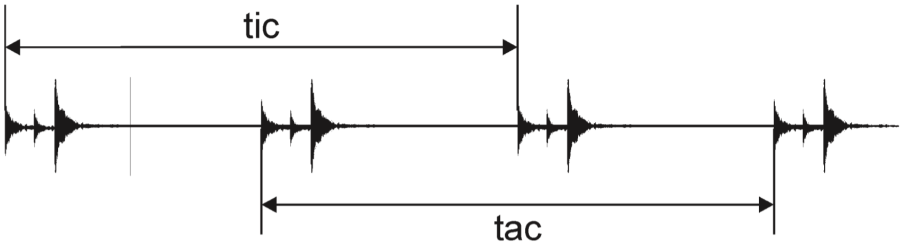


Figure 5.3: Period duration of tic, resp. tac

This results in overlapping periods, which is why the following calculation must be carried out at the end:

$$\text{period duration} = \frac{\text{period duration of tic} + \text{period duration of tac}}{2}$$

and therefore:

$$\text{frequency} = \frac{1}{\text{period duration}}$$

Chapter 6

Results

The first reference measurements were performed on the test watch on three days. Two measurements were carried out with the Greiner Compact 900 and one with the Witschi Watch Expert II. Calculations using the Greiner Compact diverge greatly from those taken with the Watch Expert II. All those first measurements weren't done in parallel with the camera measurements. In the first subchapter the results, this divergence and its causes will be discussed in more detail. In a next step as the results with the first reference measurements weren't useful, a verification of the accuracy of the measurements with the camera were made with a blinking LED. The last subchapter will focus on the meterings of the Greiner Compact 900 and the Raspberry Pi camera, which were made in parallel and where very similiar results were obtained.

6.1 Measurements

As the test watch was in a distinctly colder environment (about 5 degrees Celsius) shortly before the Greiner Compact meter was used to measure the reference values, it is highly likely that the small metal spring has contracted and the watch ran faster as a result. Before the calculation with the Watch Expert II time scale, the test watch was in an ambient temperature of about 25 degrees Celsius for a long time. A difference of only 5 degrees Celsius can affect the accuracy of the watch. [3] As the reference measurements are based on a difference of about 20 degrees Celsius, the above-mentioned difference occurs with the measured values obtained. The optical measurements of the picamera were carried out on an average at about 25 degrees Celsius, therefore the reference values determined by the Wisio Scope time scale are best suited for the comparison.

All measurements were taken in different positions of the watch with different length of recordings whereas each position was recorded in 10, 20, 40 and 80 minutes.

From the timestamp (t_1) of the first stillstand and from the timestamp (t_2) of the last timestamp, the duration ($\triangle$t)of the recording can be calculated.

$$t_2 - t_1 = \triangle t$$

Out of this duration and the number of stillstands (n_1) the stroke duration can be distinguished as follows:

$$\triangle t / n_1 = \text{stroke duration}$$

Additionally the deviation per hour is defined as

$$3600\text{sec} - (\text{stroke duration} * n_2) = \text{deviation per hour}$$

where (n_2) ist the number of strokes per hour, which is 21600.

The figure and table below show the deviation per day based on the following calculation:

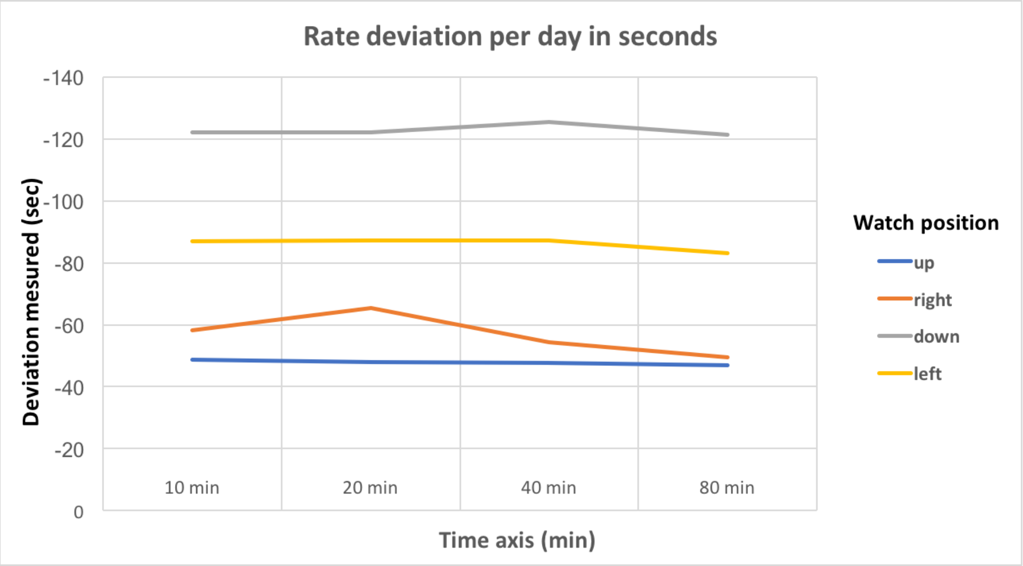


Figure 6.1: Rate deviation of different time and position recordings

Positions	**Greiner Compact 900**	**Watch Exper II**	**10 min**	**20 min**	**40 min**	**80 min**
up	-13	-31	-48.7	-48.0	-47.6	-47.0
right	-30	-64	-58.3	-65.5	-54.4	-49.4
down	-20	-46	-122.2	-122.2	-125.4	-121.4
left	-14	-34	-87.0	-87.1	-87.1	-83.1

Table 6.1: Variations per day in seconds

6.2 Verification of Accuracy

The calculated frequency is far too imprecise. In theory, however, the calculation should become more precise after a certain period of time and should even be accurate to the microsecond. But this has never been achieved. For this reason, a reliable artificial clock was used in the form of an LED that is clocked by a 50 Mhz quartz and is therefore sufficiently accurate.

Because trust in the motion vectors has diminished, a color-based method was used. The new method analyzes certain pixels in each image of the video sequence and detects the LED's light when a threshold value is exceeded in the RGB color space. This represents an artificial standstill of the balance wheel. The LED has been programmed to mimic the test watch's balance wheel and provide 21600 pulses per hour - respectively a frequency of 6 Hz as half periods.

Table 6.2 shows the result of the measurements. Different durations have been selected to show the measurement becomes more accurate over time and multiple runs have been made to show that it is reproducible.

Duration [s]	**run 1** [µs]	**run 2** [µs]	**run 3** [µs]	**run 4** [µs]	**run 5** [µs]
10	166591.5	166585.0	166397.2	165655.4	166585.0
20	166679.0	166117.1	166675.0	166675.0	166581.9
40	166486.4	166673.1	166675.0	166440.3	166673.1
60	166673.7	166672.4	166673.7	166672.4	166673.7
300	166665.2	166671.7	166640.4	166665.3	166640.4
600	166668.2	166649.6	166668.2	166655.8	166668.2
900	166667.2	166667.3	166638.4	166638.4	166667.2
1200	166660.4	166668.3	166669.7	166668.2	166668.2
1500	166668.8	166650.3	166650.3	166668.8	166668.8
1800	166652.8	166668.2	166669.2	166668.2	166669.2
2100	166668.6	166668.6	166668.6	166655.4	166654.5
2400	166668.1	166652.7	166656.5	166668.1	166668.1

Table 6.2: Measured impulses of led board through RGB analysis. Expected value 166666

In table 6.2 the assumption has been confirmed that longer records leads to more accurate measurement results. Surprisingly 60 seconds of measure time already shows a close approach to the expected 166666 microseconds. The error in measurement of run 1 during 60 seconds measure time would be 3.64 seconds per day.

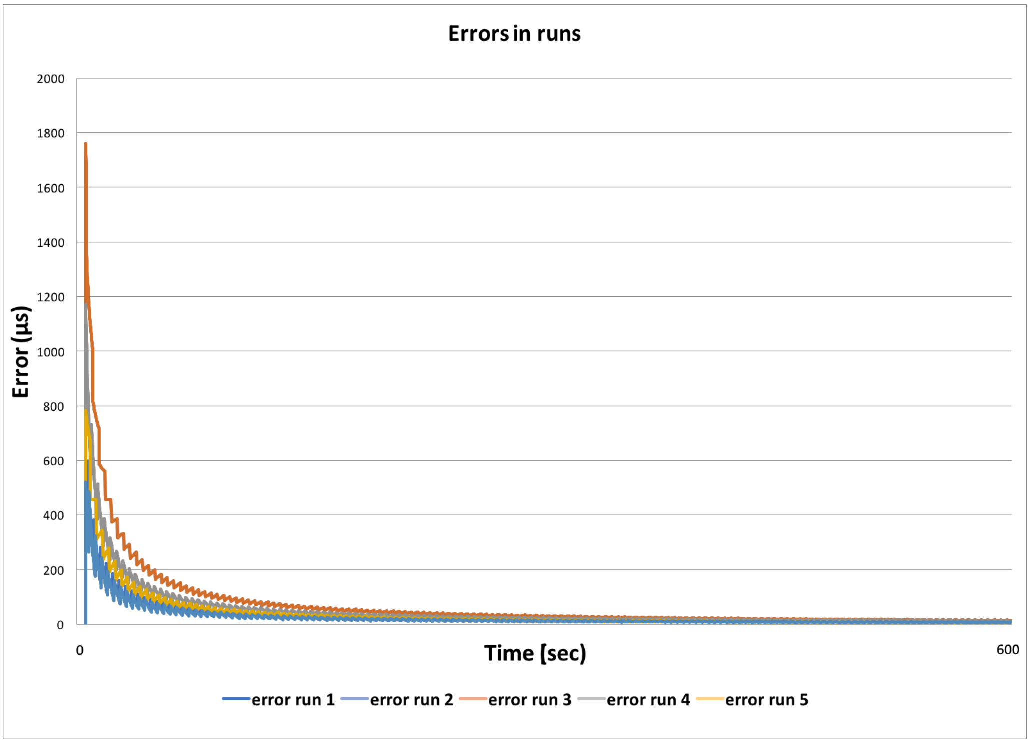


Figure 6.2: Illustration of convergation of the measurement error in 600 seconds.

Figure 6.2 shows the measurement over 600 seconds and the convergation of the measurement error. Since exact results of the LED pulses were possible with the RGB measurement, this should also be possible with the movement vectors respectively with the additional SAD values. Instead of reading the RGB values, in the next measurement the motion vectors respectively the SAD values were used again to detect the LED pulse.

Duration [s]	**run 1** [µs]	**run 2** [µs]	**run 3** [µs]	**run 4** [µs]	**run 5** [µs]
10	166636.1	166628.1	166628.1	166628.1	166628.1
20	166700.3	166602.3	166700.3	166602.3	166602.3
40	166685.6	166638.1	166638.1	166638.1	166638.1
60	166649.5	166649.5	166649.5	166649.5	166649.5
300	166667.4	166667.4	166673.6	166667.4	166667.4
600	166666.5	166666.5	166666.5	166669.6	166666.5
900	166668.2	166666.2	166668.3	166668.3	166651.7
1200	166669.1	166656.7	166658.3	166658.4	166669.3
1500	166669.7	166659.8	166658.5	166669.6	166659.7
1800	166661.7	166660.7	166668.9	166669.9	166668.9
2100	166668.4	166661.4	166668.4	166669.3	166662.3
2400	166669.7	166669.6	166663.5	166663.5	166662.7

Table 6.3: Measured half periods of led board using SAD. Desired value 166666

Table 6.3 proves that even with the SAD values accurate measurements can be made. This implies also the motion vectors can be used for precise measurement.

Knowing that the methods do not only work in theory but also in practice, a metering of a Greiner Compact 900 and the Raspberry Pi camera was executed in parallel. In figure 6.3 the setup is shown. The watch was placed on a nimble mounting of the Greiner device, which includes the microphone, which is used for measuring the oscillation of the balance wheel

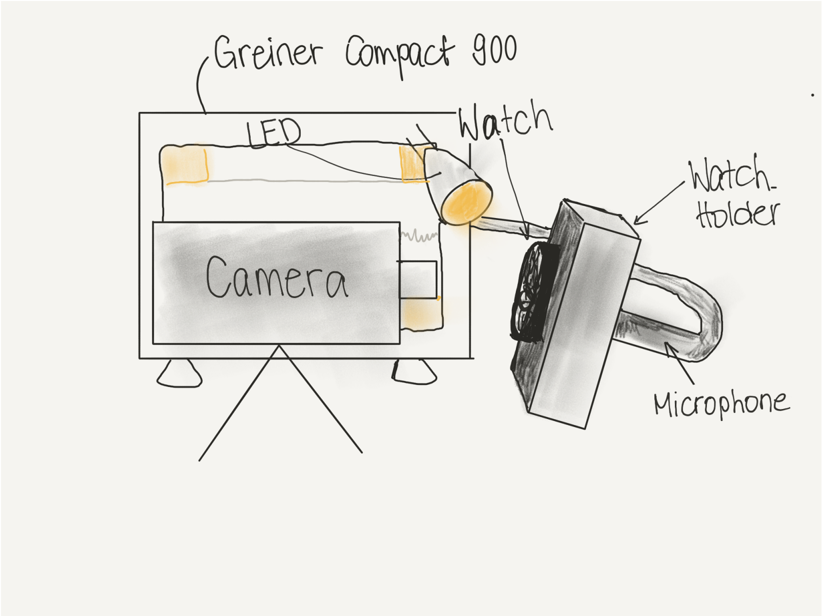


Figure 6.3: Test setup for records

The oscillation of the test watch was adjusted manually for seven times. Each change of this oscillation and the subsequent measurement of this new rate deviation are displayed as a row in table 6.4.

Greiner Compact's observed rate deviations during measurement [s]	**Camera's measured rate deviations** [s]
-50/-52/-55	-52.3
-13/-14/-15	-13.8
-4/-5/-6	-5.9
0/1	1.1
5/7	6.8
17/19/21/	18.1
43/44/45/	43.7

Table 6.4: Results of pararell measurement with Greiner Compact 900 and Camera

During the measurement at Greiner, the agitation of a watch with a frequency of 2.5 Hz was measured for 600 seconds. The mode of the Greiner meter would be set to measure in an interval of 4 seconds. Correspondingly, the Greiner meter has in some cases outputted up to 3 different values during the 600 seconds, which are shown in Table XXX. The camera's measurement is based on the summed y-values of the motion vectors and the minimas were counted with the algorithm of 5.2.2. As shown in Table xxx, the camera's measured values are very similar to those of the professional device Greiner Compact 900.

Chapter 7

Regression of the measurement error

- Grafik von Excel wo man sehen kann, dass die Messung nicht besser wird, weil es gewisse Messfehler gibt.
- als schlusswort: Messungen mit Kamera sind genau genug
- wenn annäherung funktioniert: "agrees with our theory. this verifies error proportional 1/n"

Chapter 8

Discussion

Different approaches of companies were first compared and then different algorithms and libraries for image processing were evaluated. With the help of the picamera library and motion vectors the frequency of the balance of a mechanical watch was calculated with different implementations. Overall, the frequency was determined relatively accurate and without large deviations. Actually, all methods worked, but the implementation with the step size and the consideration of the direction of the semi-oscillation did best. The results of image processing correlate very well with the values of professional devices. The deviation is very slight and therefore hardly weighs in. Even the results with the professional devices fluctuate slightly and so the measured values obtained in this work keep up very well with the devices.

- We came up with 1/t for the error[see section ...]

8.1 Difficulties

In this work there were also some difficulties, which are specified in the following section.

The first difficulty proved to be that the standstill of the balance was easily detected by eye in the summed x- and y-values of the movement vectors, but it took a while until the Python program could reliably detect these standstills.

The right camera settings and lighting conditions turned out to be another crucial point. Especially due to too weak light, the first recordings were not usable.

Furthermore, at the beginning too much information, i. e. complete frames, was stored on the Raspberry Pi, which caused memory overflow during measurements over 2 hours. Once this problem had been identified, only the sums of the motion vectors of a frame were stored, which saved a lot of memory.

Probably the greatest difficulty was to get good reference measurements, which could also be used as reference values. Thanks to Greiner, however, this problem could be resolved at the end by enabling parallel measurement of the professional Greiner Compact 900 and the camera.

8.2 Outlook

In this work it was shown that with relatively inexpensive hardware, sufficient good measurements can be carried out to determine the accuracy of a watch. Compared to the professional equipment, which costs thousands of dollars to buy, a good user interface could create a product that is much cheaper and can still keep up with the measurements of more expensive devices.

- mit Implementierung folgender Funktionalitäten könnte die Kamera ein konkurrenzfähiges Produkt zu professionellen Geräten werden

Glossary

API application Programming Interface. Clear defined interface to communicate between components. . 29

Balance wheel is a wheel that regulates or stabilizes the motion of a mechanical watch and is responsible for the clock to run precisely. 29

Codecvisa is a powerful real-time analyzer for different video codecs. http://www.codecian.com/ 29

CPU (Central Processing Unit) is the electronic circuitry in a computer, which executes the instructions of a computer program [11].. 29

DMA (Direct Memory Access) allows specific hardware systems of a computer system to access main system memory independently of the CPU [11].. 29

MMAL (Multimedia Abstraction Layer) is a framework designed by Broadcom to provide a host-side, simple and relatively low-level interface to multimedia components running on the Videocore IV GPU on the Raspberry Pi. 29

. 29

Photon is an elementary particle, which includes electromagnetic radiation (e.g. light) and moves at the speed of light within a vacuum [8].. 29

Rolling shutter When the camera needs to capture an image, it reads out pixels from the sensor a row at a time rather than capturing all pixel values at once. 29

SAD (sum of absolute differences) is a positive number, which results from the formation of the difference between two digital images. It serves as a measure of the difference between two images and is used in image processing and pattern recognition. It is obtained by subtracting the color values of the images pixel by pixel from each other and adding them up by amount. 29

Time scale is used to measure the rate deviation of a mechanical watch in different positions. 29

VCHI videoCore Host Interface is a message passing system provided to permit communication between CPU and GPU. [2]. 29

Bibliography

[1] *Chronoskop Zeitwaagen von Timegraphers.* URL: http://www.chronoskop.com/index.php/chronoskop-zeitwaagen-von-prelislisit-timegraphers/ (visited on 12/01/2017).

[2] Dave Jones. *6. Camera Hardware: Picamera 1.13 Documentation.* 2016. URL: https://picamera.readthedocs.io/en/release-1.13/index.html (visited on 12/01/2017).

[3] "Einführung-Grundkurs". In: (). URL: http://www.witschi.com/assets/files/sheets/Witschi%20Grundkurs.pdf.

[4] Gunnar Farnebäck. *Two-Frame Motion Estimation Based on Polynomial Expansion.* Springer, Berlin, Heidelberg, 2003. DOI: 10.1007/3-540-45103-X_50.

[5] Gisbert L. Brunner. *Das mechanische Uhrwerk.* 2016. URL: https://www.watchtime.net/uhren-wissen/das-mechanische-uhrwerk/ (visited on 12/01/2017).

[6] Mohinder S. Grewal and Angus P. Andrews. *Kalman filtering : theory and practice.* Prentice-Hall, 1993, p. 381. ISBN: 9780132113359.

[7] *GStreamer: open source multimedia framework.* URL: https://gstreamer.freedesktop.org/ (visited on 12/12/2017).

[8] David Halliday, Robert Resnick, and Jearl Walker. *Fundamentals of physics.* 7th. USA: Wiley, 2005. ISBN: 0471232319.

[9] Bahadir Karasulu and Serdar Korukoglu. *Moving Object Detection and Tracking in Videos.* 2013. ISBN: 9781461465331. DOI: 10.1007/978-1-4614-6534-8_2.

[10] Günter. Krug. *Mechanische Uhren : Einzelteile, Baugruppen, Werk- u. Hilfsstoffe.* Verl. Technik, 1987. ISBN: 3341003568.

[11] David J. Kuck. *The structure of computers and computations.* Wiley, 1978, p. 12. ISBN: 0471027162.

[12] Michael Lombardi. “A Century of Time Measurement: From Pendulum to Optical Clocks”. In: (2011). URL: https://www.ncsli.org/c/f/p11/286.314.pdf.

[13] *OpenCV library*. URL: https://opencv.org/ (visited on 12/12/2017).

[14] *Zeitwaage Uhren-Wiki: Die besten Uhren und Uhrenmarken*. URL: https://www.uhren-wiki.net/index.php?title=Zeitwaage.

List of Figures

Appendix - Python programs

searchMinima_tic_toc.py

```python
import numpy as np
import math

watch_hertz = 3
frames_per_second = 90
hours_per_day = 24
seconds_per_hour = 3600
one_second = 1
microseconds_in_second = 1000000
strokes_per_hour = 2 * watch_hertz * seconds_per_hour
watch_hertz_half_period = watch_hertz * 2
half_period_duration_in_sec = one_second /
   watch_hertz_half_period
resolution_grid_in_sec = one_second / frames_per_second
step_size = math.ceil(half_period_duration_in_sec /
   resolution_grid_in_sec)

directory = "path"
xValues = np.loadtxt(directory+"path")
timestamps = np.loadtxt(directory+"path")

def is_minima_search_window_5(array, index):
    left_right_increase = 2
    if index+left_right_increase < len(array):
        search_window = np.copy(array[index-
           left_right_increase-1:index+left_right_increase])
        array_to_find_index = np.copy(array[index-
           left_right_increase-1:index+left_right_increase])
        search_window.sort()
        minima = search_window[0]
        index_in_search_window = np.where(array_to_find_index
            == minima)[0][0]
        return index + (index_in_search_window -
           left_right_increase)
```

```
def find_start_minima(array):
    first_20_x_values = np.copy(array[:20])
    first_20_x_values.sort()
    if first_20_x_values[0] == 0:
        return np.where(array[:20] == first_20_x_values[1])
           [0][0]
    else :
        return np.where(array[:20] == first_20_x_values[0])
           [0][0]

durations_of_period_in_microsec = []
durations_of_period_in_microsec_tic = []
durations_of_period_in_microsec_toc = []
search_window_increase = 2
minima_tic = []
minima_toc = []
start_index = find_start_minima(xValues)
minima = start_index
switch_to_tic = False
counter_tic = 0
counter_toc = 0
start_timestamp = timestamps[start_index]
while (minima + step_size + search_window_increase) <= len(
   xValues):
    next_minima = is_minima_search_window_5(xValues, minima +
        step_size)
    if switch_to_tic:
        minima_tic.append(next_minima)
        switch_to_tic = False
        counter_tic = counter_tic + 1
        if counter_tic % 2 == 0:
            durations_of_period_in_microsec_tic.append((
               timestamps[minima_tic[-1]] - timestamps[
               minima_tic[-2]]))
    else:
        minima_toc.append(next_minima)
        switch_to_tic = True
        counter_toc = counter_toc + 1
        if counter_toc % 2 == 0:
            durations_of_period_in_microsec_toc.append((
               timestamps[minima_toc[-1]] - timestamps[
               minima_toc[-2]]))
    minima = next_minima
    if len(durations_of_period_in_microsec_tic) > 0 and len(
       durations_of_period_in_microsec_toc) > 0:
        durations_of_period_in_microsec.append((
           durations_of_period_in_microsec_tic[-1]+
           durations_of_period_in_microsec_toc[-1])/2)
```

```
end_timestamp = timestamps[minima]
duration_of_period_in_microsec = np.sum(
   durations_of_period_in_microsec) / len(
   durations_of_period_in_microsec)

print("start␣timestamp", start_timestamp)
print("end␣timestamp", end_timestamp)
print("number␣of␣found␣minimas", len(minima_tic)+len(
   minima_toc))
print("length␣of␣half␣period␣over␣all:␣",
   duration_of_period_in_microsec/2)
print("variation␣per␣day:␣", (seconds_per_hour-(((
   duration_of_period_in_microsec/2)/microseconds_in_second)*
   strokes_per_hour))*hours_per_day)
```

searchMinima.py

```
import numpy as np
import math

watch_hertz = 3
frames_per_second = 90
hours_per_day = 24
seconds_per_hour = 3600
one_second = 1
microseconds_in_second = 1000000
strokes_per_hour = 2 * watch_hertz * seconds_per_hour
watch_hertz_half_period = watch_hertz * 2
half_period_duration_in_sec = one_second /
   watch_hertz_half_period
resolution_grid_in_sec = one_second / frames_per_second
step_size = math.ceil(half_period_duration_in_sec /
   resolution_grid_in_sec)

directory = "path"
xValues = np.loadtxt(directory+"path")
timestamps = np.loadtxt(directory+"path")

def is_minima_search_window_5(array, index):
    left_right_increase = 2
    if index+left_right_increase < len(array):
        search_window = np.copy(array[index-
           left_right_increase-1:index+left_right_increase])
        array_to_find_index = np.copy(array[index-
           left_right_increase-1:index+left_right_increase])
        search_window.sort()
        minima = search_window[0]
        index_in_search_window = np.where(array_to_find_index
            == minima)[0][0]
        return index + (index_in_search_window -
           left_right_increase)

def find_start_minima(array):
    first_20_x_values = np.copy(array[:20])
    first_20_x_values.sort()
    if first_20_x_values[0] == 0:
        return np.where(array[:20] == first_20_x_values[1])
           [0][0]
    else :
        return np.where(array[:20] == first_20_x_values[0])
           [0][0]

durations_of_half_periods_in_microsec = []
search_window_increase = 2
minimas = []
start_index = find_start_minima(xValues)
minima = start_index
start_timestamp = timestamps[start_index]
while (minima + step_size + search_window_increase) <= len(
   xValues):
    next_minima = is_minima_search_window_5(xValues, minima +
        step_size)
    minimas.append(next_minima)
    minima = next_minima
    durations_of_half_periods_in_microsec.append((timestamps[
       minima] - start_timestamp) / len(minimas))

end_timestamp = timestamps[minima]
duration_of_half_period_in_microsec = (end_timestamp -
   start_timestamp) / len(minimas)

print("start␣timestamp", start_timestamp)
print("end␣timestamp", end_timestamp)
print("number␣of␣found␣minimas", len(minimas))
print("length␣of␣half␣period␣over␣all:␣",
   duration_of_half_period_in_microsec)
print("variation␣per␣day:␣", (3600-((
   duration_of_half_period_in_microsec/microseconds_in_second
   )*strokes_per_hour))*hours_per_day)
```

searchMaxima.py

```
import numpy as np
import math
```

```
watch_hertz = 3
frames_per_second = 90
hours_per_day = 24
seconds_per_hour = 3600
one_second = 1
microseconds_in_second = 1000000
strokes_per_hour = 2 * watch_hertz * seconds_per_hour
watch_hertz_half_period = watch_hertz * 2
half_period_duration_in_sec = one_second /
   watch_hertz_half_period
resolution_grid_in_sec = one_second / frames_per_second
step_size = math.ceil(half_period_duration_in_sec /
   resolution_grid_in_sec)
beginning_led_on = 100
lower_limit_start_maximum = 300

directory = "path"
sadValues = np.loadtxt(directory+"path")
timestamps = np.loadtxt(directory+"path")

def is_maxima_search_window_5(array, index):
    left_right_increase = 2
    middle = 2
    if index+left_right_increase < len(array):
        search_window = np.copy(array[index-
           left_right_increase-1:index+left_right_increase])
        array_to_find_index = np.copy(array[index-
           left_right_increase-1:index+left_right_increase])
        maxima = "unset"
        for i in search_window:
            if i > beginning_led_on:
                maxima = i
                break
        if maxima == "unset":
            print("never␣found␣a␣maxima␣at:␣", index)
        try:
            index_in_search_window = np.where(
               array_to_find_index == maxima)[0][0]
        except:
            index_in_search_window = middle
        return index + (index_in_search_window -
           left_right_increase)

def find_start_maxima(array):
    first_30_x_values = np.copy(array[:30])
    for i in first_30_x_values:
            if i > lower_limit_start_maximum:
                return np.where(array[:30] == i)[0][0]
```

```
durations_of_half_periods_in_microsec = []
search_window_increase = 2
maxima = []
start_index = find_start_maxima(sadValues)
maximum = start_index
start_timestamp = timestamps[start_index]
while (maximum + step_size + search_window_increase) <= len(
   sadValues):
    next_maximum = is_maxima_search_window_5(sadValues,
       maximum + step_size)
    maxima.append(next_maximum)
    maximum = next_maximum
    durations_of_half_periods_in_microsec.append((timestamps[
       maximum] - start_timestamp) / len(maxima))

end_timestamp = timestamps[maximum]
duration_of_half_period_in_microsec = (end_timestamp -
   start_timestamp) / len(maxima)

print("start␣timestamp", start_timestamp)
print("end␣timestamp", end_timestamp)
print("number␣of␣found␣minimas", len(maxima))
print("length␣of␣half␣period␣over␣all:␣",
   duration_of_half_period_in_microsec)
print("variation␣per␣day:␣", (seconds_per_hour-((
   duration_of_half_period_in_microsec/microseconds_in_second
   )*strokes_per_hour))*hours_per_day)
```