HSB
Hochschule Bremen
City University of Applied Sciences

Faculty 4 - Electrical Engineering and Computer Science
engineering projekt

Shack Hartmann wavefront sensor with Raspberry Pi and Pi Camera

hardware setup and software implementation in python

Authors:	Hauke Janssen
Student IDs:	5010422
E-Mails:	haujanssen@stud.hs-bremen.de

Lecturer:	Prof. Dr. Ing. Friedrich Fleischmann
Submission Date:	23.12.2022

Contents

List of Figures

1 Abstract

This text will illustrate the development process and components of a wavefront sensor from consumer electronics. As the measurement principle, the Hartman test was used, where you place aperture arrays instead of microlenses as the optical component. For this project, an aperture array of 5x4 was placed in front of an image sensor. That can measure the wavefront by precisely calculating the position of the spots and their drift.

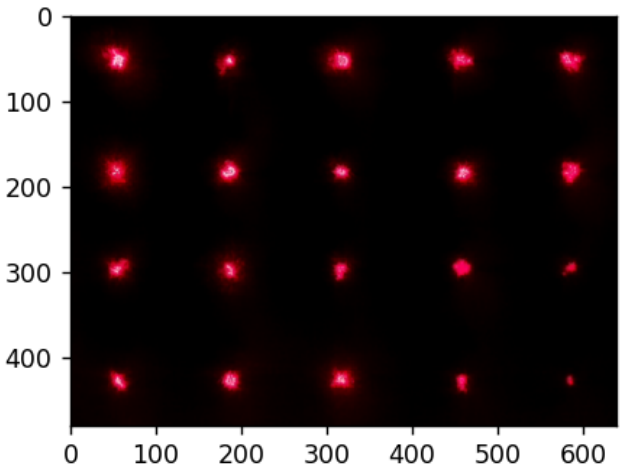


Figure 1: Raw sensor image with red laser beam hitting the aperture array

Furthermore, there will be a thorough discussion of the software decisions and specific implementations, because it has taken by far most of the time in the process of building the device. Some crucial elements for a truly usable device were not developed, e.g. a housing and a user interface. The performance of the whole system is not comparable with even the low-end wavefront- sensor on the market. They start at a price of around 3000 euros. But in that case, a device of around 100 euros is used. Because this work should just be a proof of concept and build a rough understanding of a wavefront, and the development process. Moreover, it has only been developed for offline analysis, so data is captured and the wavefront and other parameters are calculated afterwards. At the beginning that was not the goal, but throughout the project, it became clear that the focus should be more on algorithm prototypes than on real-time generated visualisations. This mistake was very severe and has led to major time mismanagement.

2 Theory

2.1 The wavefront of light and measuring techniques

In the field of optics, the propagation behaviour of light is one of the most important properties. This property is determined by the shape of the wavefront, of the light[2]. The

wavefront is also called the phase front. It describes a line of equal phase values which for example appear for the phase front of a lightbulb as well as for the phase front of a light beam. For further understanding, on the one hand, the phase front of a light bulb is spherical shaped [3], because it emits in all directions equally. And on the other hand, the phase front of a laser beam, which does not change its beam width while propagating, occurs as a flat line, perpendicular to the propagation direction [4]. So the shape of the phase front determines the propagation direction of the light. For that reason, the focus of the light is an important factor. However the measurement of the phase shape is not a trivial task, but its effects can be observed very easily. One possibility is to use a complex interferometer- setup, this requires mutable optical components and an extensive setup- process. That's why a simpler alternative was used, where a regular CCD camera and a lens or aperture array were placed in front of the chip. This setup contains fewer components because the measurement is always relative to a calibration measurement. The properties make the device simple to build, set up and use. For the

2.2 Prerequisite: The wavefront

The sensor is designed for measuring the phase-front of an electromagnetic wave, e.g. light. This will be called a "wavefront" in the following text. Because the understanding of this concept is fundamental for the explanation of the sensor, some further information will be delivered. Light can be described as a propagating wave in the medium of the electromagnetic field. The first visual equivalent that often comes to mind, is ripples on the surface of the water. This is very helpful but it is a simplification. The surface is just a 2D surface, where the amplitude is the displacement of the water surface in the 3rd dimension. So the amplitude is also a spatial dimension, which is not the case in the electromagnetic field, short: "EM-Field". This field has propagated in 3D-dimension, with the amplitude being a 4th-dimension and is not spatial[4]. It is not necessary to keep the higher dimension in mind. The water surface analogy can help tremendously, but it is incomplete and in some cases misleading. Coming back to the simplified analogy, the ripple on the Surface of the water can have different shapes. If, for example, a stone is thrown in the water, the ripples circulate around the incident point of the stone and move along a wooden panel back and forth, which can yield a more focused wavefront.

3 Principle of the Shack-Hartmann-Sensor

A simple Shack-Hartmann-Sensor wavefront has just one optical component, a lens or an aperture array. This component is focusing on segments of the incoming wavefront into a set of discrete points, which will be captured by the camera chip.

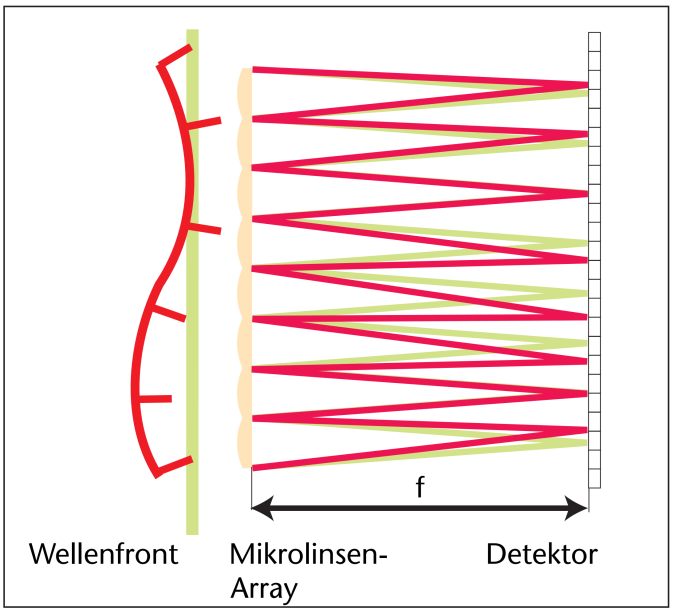


Figure 2: Wavefront hitting lens array, reference in yellow measurements in red [1]

The tilt of the inclining wavefront will be represented by a shift of the light points. So if the incoming wavefront is hitting the array with a specific angle, all light points will be shifted in the according direction. Even when the wavefront is randomly deformed, the sensor can be capable of resolving the original shape. But the sensor has a limited range of wavefront angles according to the distances of the apertures to one another and the distances of the apertures to the sensor. Also the number of apertures and the size of the sensor determine the resolution of the wavefront.

3.1 The reference wave

One important element is, in practice, the absolute position of the dots relative to the focus point of the aperture, which is a difficult way of estimating information about the wavefront. The shifts of the dots have to be measured against something form the sensor itself. It is the most important aspect of the sensor regarding correctness of the measurements. The established standard measuring means that the waveform deformation is relatively to one reference wavefront. This wavefront should be as close as possible to an even wavefront, because in all processing steps, the reference wavefront will be assumed to be completely flat. Any deviation from this shape will be presented in all measurements, which are basically not detectable without another device testing the accuracies.

4 Hardware Components

4.1 The Raspberry Pi zero W2

The Raspberry Pi is a microcomputer platform that is offering high computing capability for a low price, in a small footprint. It lists vastly more features and speed than a typical microprocessor. Especially the chosen Zero W2 is a very inexpensive one, with 15 euros for the device and some additional costs for the power supplies (SD-card, Micro-USB to Ethernet and an Ethernet cable). A complete kit for a running system can be bought for about 40 euros. The most compelling feature, which is the origin of the project, is the different available cameras. One is a very capable camera sensor with a comparably large sensor size for this accessible hardware and price segment.

4.1.1 RAM considerations

One point of concern, in the beginning, was the low amount of available RAM. The Zero W2 variant with 512 MB is, in the Raspberry pi family, one, with the least amount, but it is also the cheapest one. One important piece of information regarding the RAM considerations was, how much there was available for the actual application. With the stripped-down version of the operating system, the Raspberry Pi OS light was, with the initial RAM use and without any other application running, about 45MB. Besides that, the use of a camera on the Pi, is decreasing the available RAM, because the integrated GPU is using the system RAM. The RAM allocated for the GPU can be chosen and should match with the requested number of pixel in the image or frame-rate. In this chase the 128 MB was chosen for the GPU to avoid issues in this regard. This option is limiting the image resolution, the whole resolution would be impractical and probably unnecessary for other reasons too.

4.1.2 Operating system choice

In this case the GPU was used because it is handling the capturing of the image, which is preprocessing in the camera-stack, that is available in the version of the Raspberry pi OS. The OS version used in this project, that is called ”buster” is not the most recent one available at the time. At the same time, as the parts for the project arrived, a new version became available, that is called ”bullseye”.
From other sources on the topic, one remark seems to be mandatory. The Raspberry Pi OS leans and its naming convention is connected to Debian, another operating system. So every major release is named after a Toy Story Character. [5].
However, the newly available versions of the operating system have a major changes in the use of cameras. It basically has changed the whole Camera-stack from a proprietary-close-source system to the new open-source variant ”libcamera”. This change had the inconvenient disadvantage of being incompatible with existing python packages ”picamera” for camera

interactions. In the beginning the option of using libcamera and c++ was investigated, but due to unfamiliarities with the language, this option was stashed and not used. In the end, an old operating system was used, where the convenient infrastructure of the python package ”picamera” is available.

4.2 The camera module - HQ camera

The ”Pi foundation”, the company behind all production related to Pi, has released three camera modules. The first one 2013, with 5MP another in 2016, with 8MP and in 2020, the here-used Raspberry Pi HQ camera with 12,3MP. This HQ module has a far larger sensor size, which makes it comparable to a smartphone, neglecting the point, that modern smartphones reach higher perceived image qualities due to very sophisticated image processing, which is not available to the HQ module. Besides this fact, the sensor is state-of-the-art and very cheap at 50 euros. Especially the sensor size of 6.287mm x 4.712 mm makes it very interesting to use in a wavefront sensor because it makes it easy to use an aperture array with multiple holes but reasonable spacing, to not have overlapping light points in the image.

4.2.1 Use of the camera

The capabilities of the sensor were only used in a rather limited fashion. It offers an absolute resolution of 4056 x 3040 pixels, but only 640x480 was actually used. This was chosen, because the number of light points was low, and it seem unnecessary to use a high resolution and resources, without any benefit. A higher resolution would yield lower samples per second. Despite the low resolution, the raw frame rate with the Python package is rather slow. It can offer a new image about every 50ms, so about 20 frames per second(fps). The sensor itself is promoted with up to 240fps for 1080p(1920x1080 pixel). The raw amount of data, by this bit-stream, would greatly exceed the capabilities of the raspberry pi and probably even the CSI-2 interface between the module and the pi. But this low frame-rate, not even 30fps, with this low resolution, was a disappointment on the python package. There are certainly ways to increase performance.
Some feature was not used at all, for example the metal mounting system for the lens of the sensor. Because no optics besides the aperture array, was needed, and the space between sensor and array should be small, so the mounting system got completely unmounted.

5 Software implementation

5.1 General algorithmic data flow

This section will explain what processing steps were used, from capturing an image to reconstructing a wavefront. This process can be separated into two major steps. The

first step, determine accurately the centre of mass, called ”centroid”, where just the laser spots can be seen. This process is rather straightforward, with one useful trick that has not been found in the literature, that is using the Voronoi diagram. The second major step is the calculation of the wavefront, or more accurately the wavefront shift from one, which is assumed to be a flat reference wavefront. The following first step will be discussed.

The captured image consists of multiple light-dots from which, at first, the accurate position has to be evaluated.

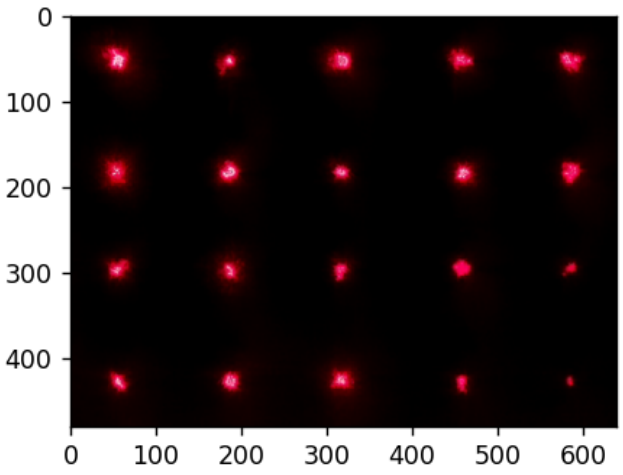


Figure 3: Raw sensor image

This will be performed by the very well-established momentum of the image. From which the centroid, the main emphasis, can be derived. For a 2D image the equation is shown below.

$$Mij = \sum_{x} \sum_{y} x^i y^j I(x, y)$$

These moments are a set of values described by the index i and j. $I(x, y)$ is the pixel intensity at position x and$y$. So for $i = 0$ and, $j = 0$ the result of the operation is just the sum over all pixel in the image. The momentum describes, in a statistical sense, the distribution of values over the image. To calculate the centre of mass in pixel space x, y , the following equation will be applied.

$$x = \frac{M_{01}}{M_{00}} y = \frac{M_{10}}{M_{00}}$$

The position of the centre of mass will be the value of x and y, with a floating-point accuracy, so the value will have sub-pixel accuracy.
This is the general approach for calculating the centre of mass of one image. The current situation requires some preprocessing, because not the whole centre of mass, of the image, is of interest. Just the areas for the illuminated spots on the sensor are important. So the image has to be split in separate areas, each corresponding to one illuminated

dot on the image. For this task, there are multiple approaches possible. With some prior knowledge, one common approach is to just have a predefined, even grid in the same dimensions as the overlying lens array. If the system should be as agnostic of the position of the lenses as possible, these areas can be evaluated dynamic from one initial sensor-image or even new for every sensor-image. This dynamic evaluation does offer the option to have the lens-array in a non-equally spaced grid, or if one has a self-made aperture-array with random locations of apertures.
This dynamic approach was chosen for this project. The evaluation of a before mentioned area detection, containing the illumination spots, will begin by finding just the maximum local peaks in the image. Just this, would most probably yield to a lot of maxima, because the image is obviously grained and has numerous peaks around the one big of interest in the middle of an illumination peak. To avoid small peaks around one large one, the function call from the used library has the parameter for minimum distance, between peaks. With a sensual value for this parameter, the actual peak, that a human would point at, will be found with a high enough precision in testing. Now, just the peak position as a discreet pixel is known. With these multiple positions and the use of an, already mentioned, ”Veronoi-diagram”, the maximum sized areas around each peak-point, can be assessed taking the other peak points into account. This means, the dividing boarder between two neighbouring peak-points, will always be exactly perpendicular to the midpoint of a connecting line. This approach will create a polygon areas around each peak point, and shows it, as a, so called ”segmentation mask”.

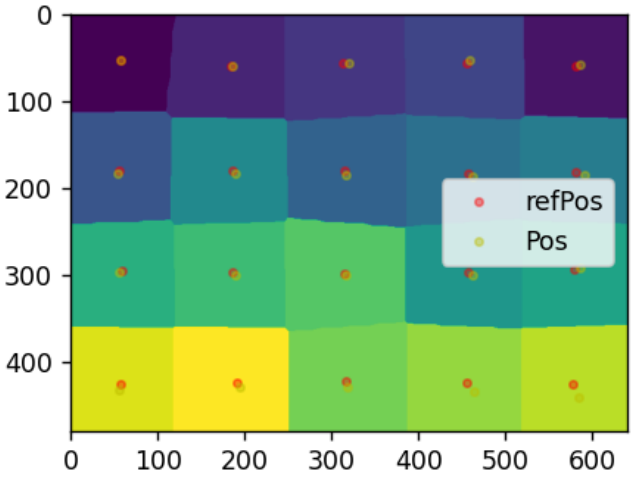


Figure 4: segmentation map

A mask, which is used often in image processing, has the number ”one”, for an area of interest, and ”zeros” for every uninteresting area in an image. This could be used in these situations, but they would need as many mask images as peak points. So these can be combined in a segmentation mask, where the area of interest, for the first peak, has the value ”zero”, while the second area has the value ”one”, the third has value ”two”

and so on. This has the need-benefit to yield to a very straight forward visualization for the separated areas.

5.2 Explanation of code details

5.2.1 Git hub and code structure

The whole project is tracked in a public git repository and is structured as a python library with the name SHSlib for **S**hak **H**artman **Sensor** library. This library consolidates all the necessary image processing and data manipulation into one easily accessible software entities. In all professional software development this is a mandatory setup.
Furthermore the library is placed in the `./src/SHSlib` and will be accessible for development from the folder `./src/dev`, where multiple Jupyter notebooks are placed in. For the development process, directly on the device, a Jupyter server was installed, so the code will directly be executed on the Raspberry Pi.
The SHSlib library has currently three subcategories, called ”module” in Python. These clusters simulate types of functions, like `analyse`, `simulation` and further `utils`. These modules will be automatically imported, when SHSlib is imported. That is not the default in python, but from personal experience the expected behaviour from other libraries. In every module folder, there is one `__init__.py`, which imports the module functions.
In one special case, a c-function will be compile as a shared library and import. This only happens, when the file is not present, and the system is linux. This applies for the `getMomentum` functions.

5.2.2 Function dispatch

The function dispatch, is in computer science, used to refer to a function, with different implementations. Before calling a function, there is a dispatching system, that is choosing the specific implementation to use. As an example, in python, the + operator is transformed to a function call `operator.add()`. But this function can perform different operations, depending on the input type. On integer, the operation is different from floats and more noticeable on string. This dispatching is aware of types. The notion was adopted for functions in the module `analyse`. All functions have multiple implementations, that have been tested during development, as well as an optional parameter `algorithm`, where different implementations can be used. In most cases, one has proven to be the best/fastest, which is here, set to be the ”default”. The functions `getMomentum` are available in python, as well as in a very fast C implementation. This C-version is the ”default” but will only be executed, if the system is linux, like on the Raspberry pi. Otherwise another implementation will be used. That is not a necessary step for this project, but it had the benefit of keeping old ideas available, with the option of quickly using it by changing one parameter.

5.2.3 Details on `getSeperation()`

This function bears some non obvious implementations, a more detailed view seemed reasonable.
Its task is to generate a segmentation map, displayed in figure 4, from the raw sensor image... The function is depending on the useful and widely used image libraries `scikit-images` and `openCV`.
The functions start with the search for local peaks in the image with the function `skimage.feature.peak_local_max(Sensor_image,min_distance=min_distance)`, to get the x and y positions of the spots. The parameter `min_distance` has to be chosen according to the number of pixel between the dots. In the current setup, there are 5 dots in the horizontal and a resolution of 640 $\frac{5\text{Dots}}{640\text{Pixel}} = 120\text{Pixel}$. So some min_distance value lower than 120 pixel, should be good, but choosing this too low, would yield unwanted detected peaks, due to roughness of the image. Another approach would be, to have a blurring operation happening, to filter smaller peaks.
The next step is to create the veronoi-diagram with these coordinates.

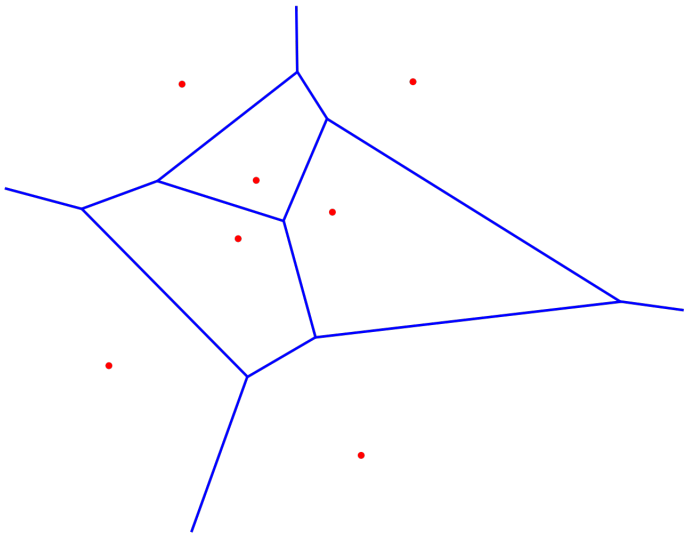


Figure 5: Voroni Diagram

This is an algorithm that creates polygon cells around any points in on a set of points. So that the borderline of two cells, close to one another, is exactly on half of the connecting line of its point and orthogonal to the connecting line. This process is rather quick and established. It is very useful because it breaks free from the rectangular grid that is often employed in the Shak-Hartman-Sensor and would make it possible, to have, non-equal distributed measuring points. One important consideration with this approach would be, that many wavefront reconstruction algorithms are requiring equal-grid spacing. More will be explained later. First, back to the Veronoi diagram.

One option for computing the desired cell shape from the peak positions would be

deriving the cell lines from the connecting lines' peak positions, cell line length from their interception and rastering them into one image. Luckily, this whole process was done as a by-product of an OpenCV function. One function, supplied by OpenCV is doing a morphological opening, which enlarged white areas on a black background. In a naive implementation, two close white areas can become one, during expansion. This can be desired in suppressing noise. In this special function, in OpenCV, it is actively avoided, by a prior segmentation of individual white areas and a map of maximal expansion. This maximal expansion map, is primarily helping the function to avoid merging separate areas. Luckily this map is built with a Voronoi algorithm and provides the exact segmentation, that is desired for the `getSeperation()` function.
The openCV function call is to `cv2.distanceTransformWithLabels`, DIST_L2

5.2.4 Details on `getMomentum`

There are many ways to implement the momentum calculation. Three different ones are present in this package. One uses a python image processing package, sci-kit image. It is rather slow but can be easily installed on every system. The second is using openCV, which has many very fast C-functions for image processing, also for centroid calculation. One drawback is, that these functions calculate higher-order momentum, which is not needed for the centroids. So there is a lot of computation and time wasted. A costume version in numpy is possible to be done but has no significant benefit. All proposed algorithms have one drawback, they rely on masking with the segmentation map e.g. figure 4. That means, when one centroid should be evaluated, all pixels are set to zeros, that is not in the defined area of the segmentation mask. For 20 centroids, that means at least 20 summations of the whole array. Again a lot of unnecessary operations. This lies in the necessity of array operations in python.
One neat thing, that could vastly speed up the implementation, is using the value of the segmentation mask, as an index and only performing the summations that are actually needed. This would interfere a loop as long as elements in the images, 307200, for 640x480, which are far too many for python. This was implemented in C and compiled as a shared library. This .so-files can be loaded into python with the native ctypes interface. This offers direct access to the numpy array without copying just like native numpy functions. The central hot loop is displayed in the listing 1

Listing 1: function

```
1
2 for (size_t i = 0; i < n_img; i++){
3     M00[segMask[i]] = img[i] + M00[segMask[i]];
4     M10[segMask[i]] = (img[i] * X[i]) + M10[segMask[i]];
5     M01[segMask[i]] = (img[i] * Y[i]) + M01[segMask[i]];
6 }
```

The runtime comparison of these algorithms:

sckit-image	ca. 500ms
openCV	ca. 200ms
C implementation	ca. 7ms

Table 1: Caption

5.3 Integration - the Modal way

For the reconstruction of the wavefront from the slope values, multiple methods are available. They integrate the derivative measurements from the sensor to the wavefront shape. This process can be done by the zonale method, where every measured slope point will be processed in an individual way and represent one zone. This process takes every individual slope point to be exact and fully represented in the wavefront reconstruction. This generally yields to a high spatial resolution, but can not by itself, be interpreted for optical properties like defocus. This reconstruction may happen by spline interpolation or matrix iterative approach, which generally requires a regular grid for the slope points.
Another approach is, to assume a linear combination of functions for the whole sensor surface, that is smoothly differentiable. These differentiated functions can be fitted to the measurement data and the coefficient will be applied to the basis function for the wavefront reconstruction. Besides that, it has the benefits of a compact notion and can yield to some physical interpretation in the case of the Zernike polynomial. The coefficients are associated with optical transformations like defocus. This approach of Zernike, the decomposing of the wavefront was chosen and will be described in the following section.

5.3.1 The Zernike Polynomial

The first orders of these Zernike Polynomials are depicted in the following table 2. These are expressed in terms of Cartesian coordinates to directly correspond to pixel coordinates. Because the acquired centroid movement correspond to wavefront gradient, the model was also differentiated with respect to x and y. The coefficient of the differentiated Polynomial are then used for the base Polynomial, that yields the actual wavefront.

i	n	m	$Z(x,y)$ Cartesian	$Z(x,y)\Delta x$	$Z(x,y)\Delta x$
1	0	0	1	0	0
2	1	1	x	1	0
3	1	0	y	0	1
4	2	2	$x^2 - y^2$	$2x$	$-2y$
5	2	1	$-1 + 2x^2 + 2y^2$	$4x$	$4y$
6	2	0	$2xy$	$2x$	$2y$
7	3	3	$x^3 - 3xy^2$	$3x^2 - 3y^2$	$-6xy$
8	3	2	$3x^3 + 3xy^2 - 2x$	$9x^2 + 3y^2 + 2$	$6xy$
9	3	1	$3y^3 + 3x^2y - 2y$	$6xy$	$3x^2 2y - y^3$
10	3	0	$-y^3 + 3x^2y$	$6xy$	$3x^2 - 3y^2$

Table 2: Zernike polynomials of order 1 to 10 and derivatives with respect to x and y [6]

The derivation in 2 where performed by the pyhton package Sympy.

Listing 2: function differentiation with Sympy

```
1 import sympy as sy
2 from sympy.parsing.sympy_parser import parse_expr
3 dx = sy.diff(parse_expr("x**2"),"x")
4 dy = sy.diff(parse_expr("x**2"),"y")
```

This could be performed faster by hand, but the goal was to have any height order of the Zernike Polynomial accessible in differentiable form. This did not work out, because the implementation of the Zernike Polynomial, for any n and m, especially in Cartesian coordinates, were more complex than expected. But extending the list of the function, it is only a new basis function needed.

5.3.2 construction of linear System for coefficients estimation

The contributions of the individual Polynomials will be estimated in a least square manner, by expressing them as a linear system of equation, in a form of a matrix. Furthermore the measured gradient data is constructed in one vectors for x-direction and y-direction. Cn_{dx} is the shift of the n'th centroid in x direction and Cn_{dy} shift in y-direction.

$$G = [C1_{dx}, Cn_{dx}, ..., C1_{dy}, ...Cn_{dy}]$$

The 2D Model Matrix consists of computed values for all centroid positions Cn_x, Cn_y, for all model functions. In this case, the functions in the table 2 introduce the derivatives of the Zernike Polynomial.

$$M = \begin{bmatrix} z1_{dx}(C1_x, C1_y) & ... & zn_{dx}(C1_x, C1_y) \\ ... & ... & ... \\ z1_{dx}(Cn_x, Cn_y) & ... & zn_{dx}(Cn_x, Cn_y) \\ z1_{dy}(C1_y, C1_y) & ... & zn_{dy}(C1_y, C1_y) \\ ... & ... & ... \\ z1_{dy}(Cn_x, Cn_y) & ... & zn_{dy}(Cn_x, Cn_y) \end{bmatrix}$$

Multiplication of the model matrix M with a coefficient vector V, that acts like weighting the individual contributions of the model order that yield to one possible differentiated wavefront.

$$G = M \cdot V$$

Reversing this process, by finding one V that yields to the exactly measured wavefront G, will tell us the coefficients vectors of the Zernike Polynomials V.

$$V = M \setminus G$$

The solving will be done, in this case, with the simple least-squares solution to a linear matrix equation from numpy.

Listing 3: numpy function for solving linear system of equation

```
1    V = numpy.linalg.lstsq(M,G,rcond=None)[0]
```

5.4 2D zonal integration with unordered data

The integrations method presented by Smith in 2021 [2] was implemented and analysed in the Notebook `src/dev/Implementation_zonale_integration.ipynb`. It yields somewhat good results for synthetic data, integration 2D-sin function to cos. But was terrible for the actual wavefront. Maybe some mistake was made, but this has led to the switch to Zernike decomposition, for the wavefront reconstruction.

6 Use of the Device

The intended way of use is based on the Jupyter Notebook. This allows the use of the interactive execution of python code and was immensely useful in the development process. The major benefit of developing right on the RasperyPi was first, that the live interactions with the camera were very simple and second, that some code changes and experimenting with it had no barrier. That could be evaluated without extra steps. Additionally, the Jupyter Notebook has simple and fast plotting capabilities, but they were not so easy with just an SSH connection.

6.1 development environment

The actual way of developing was, to start a Jupyter Notebook Server on the Raspberry Pi and configure it, in a way, that accepts connection not just from local host, which is the standard, but from any other computer. So any computer in the local Network could just type the right URL in the browser and could use the device and edit the code. This can also be done from the browser of a smartphone if needed.

6.2 connecting to the Raspberry Pi

To start the server, one must connect with the Raspberry Pi over SSH. It has the User `PI` and the computer host name RaspberryPi. Alternatively to the hostname, the IP address, that were given by the local DNS(most likely the router) can be used as well. That yields to command `"ssh pi@raspberypi"`, which should be executable in a PowerShell on windows or any Linux machine in the terminal. As a note from the author, on windows, the Windows Terminal app gives a very modern Linux-like experience for a terminal.

6.3 navigating to the right director

To understand how to use the python library of this project, the following will guide step by step. In the project folder, called `pishackhartmannsensor` is the folder `/src/SHSlib`, which contains all code. Besides that, there is the folder `/dev`, which contains the Jupyter Notebook, where the use of the library is explained.

6.4 launching the Jupyter Server

This is just done by running the commando `"jupyter notebook"`.

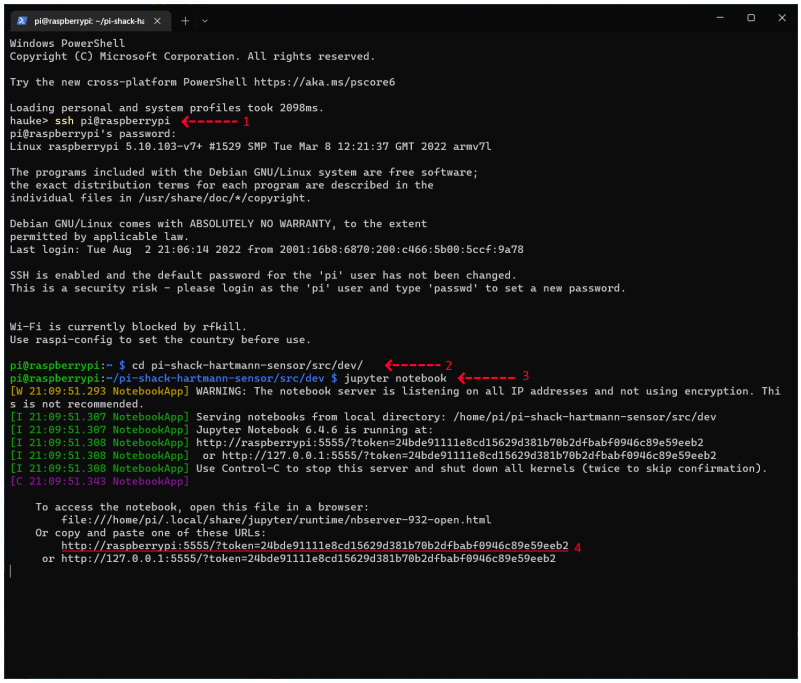


Figure 6: starting jupyter server

6.5 Run Jupyter Notebook

The Jupyter Notebook `./src/LivePlot.ipynb` contains two cells for live Viewing of the sensor. The second is for some simple live analyses and the first Cell is just for recording raw images. These can be saved as a .hdf5 file and processed later in `./src/GeneratePlots.ipynb` with the Zernike-decomposition. In this way, the following experiment was captured and analyses.

7 Experiment

The following experiment will be conducted with the developed Shak-Hartman-Sensor. The acquired data will be discussed in detail, in a step-by-step manner, to build confidence in the developed algorithms. The experiment has a quantitative character to test the implementation. All data generated and analysed have arbitrary units because many important scaling factors like the focal length of the apertures and the distance of the aperture to the sensor were unknown and not measurable with confidence accuracy. All of the following figures are created from the Jupyter-notebook `./src/GeneratePlots.ipynb` and the data is also contained in the Git-Repository. So this can be fully recreated by just cloning the repository.

7.1 Setup

The setup consists of a laser diode as a light source, shining directly on the aperture array of the Shak-Hartman-Sensor. Under this condition, the reference wave was captured. To introduce some wave disturbance a lens of unknown properties will be held in the propagation path of the laser. This lens will then be moved along the laser beam towards and away from the sensor. This should change the size of the laser-dot-projection, visible on the area around the aperture.
This size change of the laser is due to a change in the wavefront of the beam by the lens. In figure 7 the ideal reference wave is depicted in red and the one disturbed by the lens is green. The projection of the aperture shows an outwards movement due the lens, called centroid shift. This should be detectable by the Shak-Hartman- Sensor.

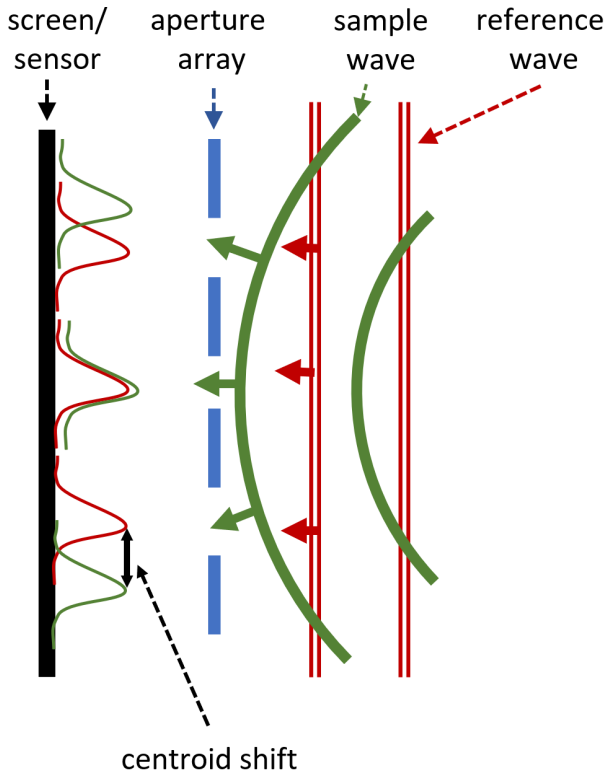


Figure 7: Expected influence of lens on wavefront

7.2 Data analysis

The captured data will be processed with the previously introduced Zernike decomposition and wavefront reconstruction.
If the lens is kept perfectly on the optical axis of the beam, only a wavefront similar to spherical shape should arise. This will be visible in Zernike Polynomial order > 2. When the lens position or rotation is off-axis, the beam direction will change. This would yield an X-Y shift on the sensor and should be expressed in Zernike orders 1 and 2.

7.3 Peak and Centroid investigation

The figure 8 of the experimental data depicts the reference wave used for the wavefront reconstruction. The intensity of the laser was controlled with a potentiometer to have as little clipping pixel as possible, while still having enough intensity in all peaks.

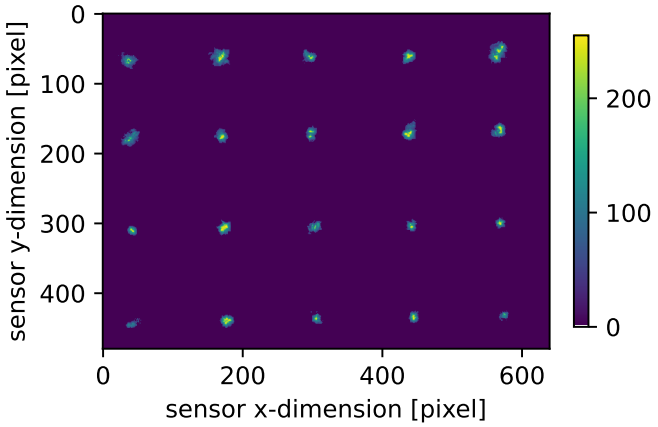


Figure 8: image of reference wave

The segmentation map was calculated from the peak position explained in chapter ”Details on getSeperation()” and is depicted in figure 9

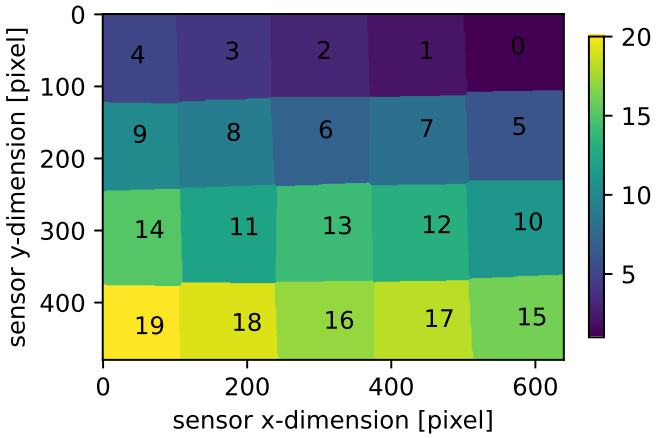
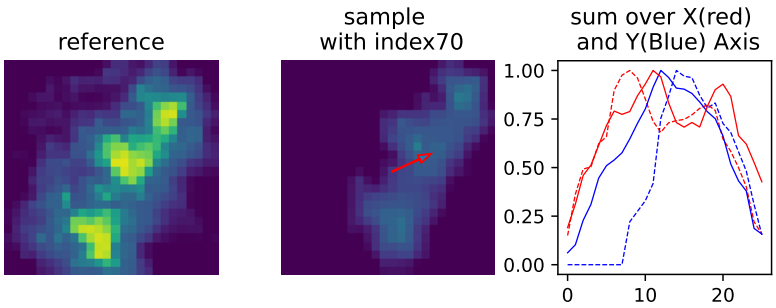
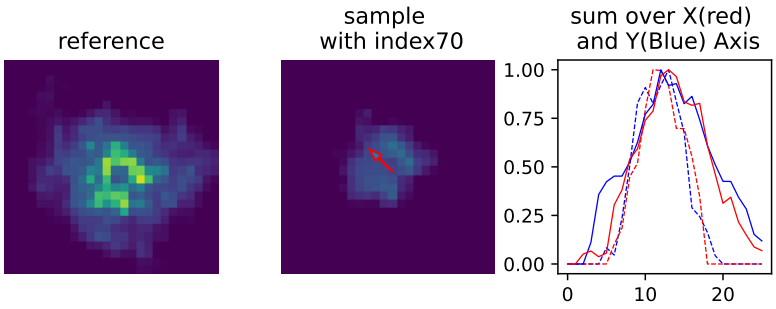


Figure 9: segmentation map for centroid calculation

To evaluate the correctness of the centroid estimation and the movement, a close-up view is presented. A 30x30 pixel crop of the reference wavefront and sample wavefront peaks will be displayed, the centre is in both images from the reference wave. Figure 10 depicts the crop centre at one centroid of the reference wave. In all images, the colours express the pixel intensity and are capped at 255 to the cut-off value of 40. That shows that the sample wave has a far lower intensity because the lens diverges the beam. In the depiction of the sample peak, the shifted centroid is displayed with a red arrow. Moreover, the normalised sum, over the X and Y dimension of both images, is shown.



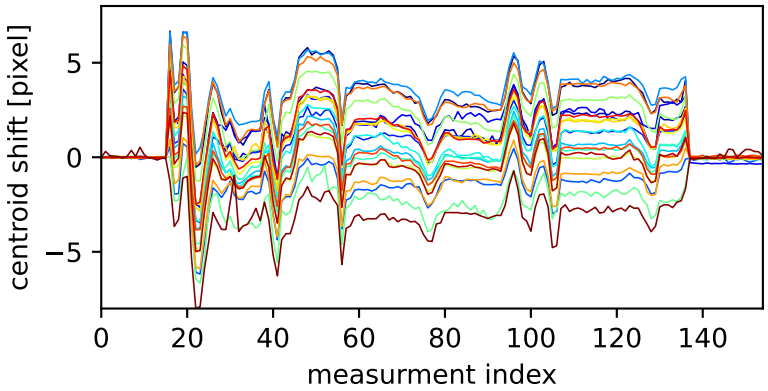
(a) centroid index 0 (x:565 y:55)



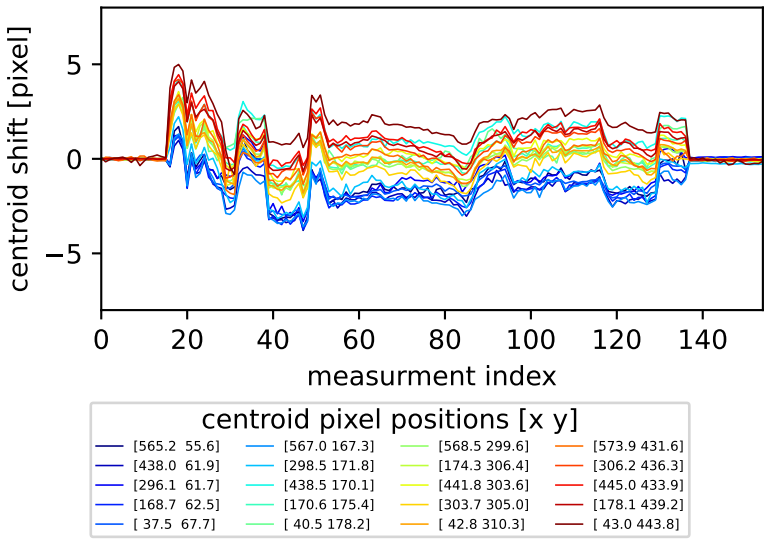
(b) centroid index 4 (x:37 y:67)

Figure 10: crop view of the peak with according centroid move
dotted line: sample, solid line reference

In the following graph 11 the movement of the centroid over many measurements is depicted. At around index 18, the lens was introduced to the beam, while at index 139 it was removed. This was captured in a span of about one Minute, with a periodic interval. In this graph, the context of the position of the centroids is lost. This makes it difficult to interpret. Besides that, it should function as a sanity check on the data, so the movement is consistent and has no drastic outliers. That could have happened, if it would have been caused by a wrong centroid mask or a bad aperture placement in front of an image-sensor.



(a) subtitle for figure



(b) other subtitle

Figure 11: Centroid movement over many measurments with lens

The movement appears to be in a similar range for all centroids and seems to have relatively similar behaviour in one measurement. This builds confidence in further processing steps in which some error, may become difficult to detect.

The following figure 12, despite a sample wavefront, of index 70. The red arrows are showing the centroid movement relative to the reference wave. The movement was multiplied by 10 to be visible in relation to the size of the image. Furthermore, the border pixels of the segmentation mask are displayed as maximal values in this image.
The expectation on the arrow movement, that they all point outwards is for most arrows visible in this image.

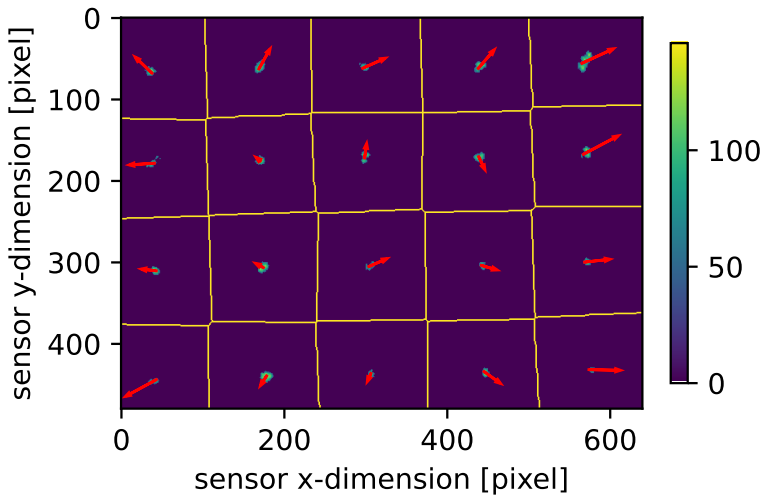
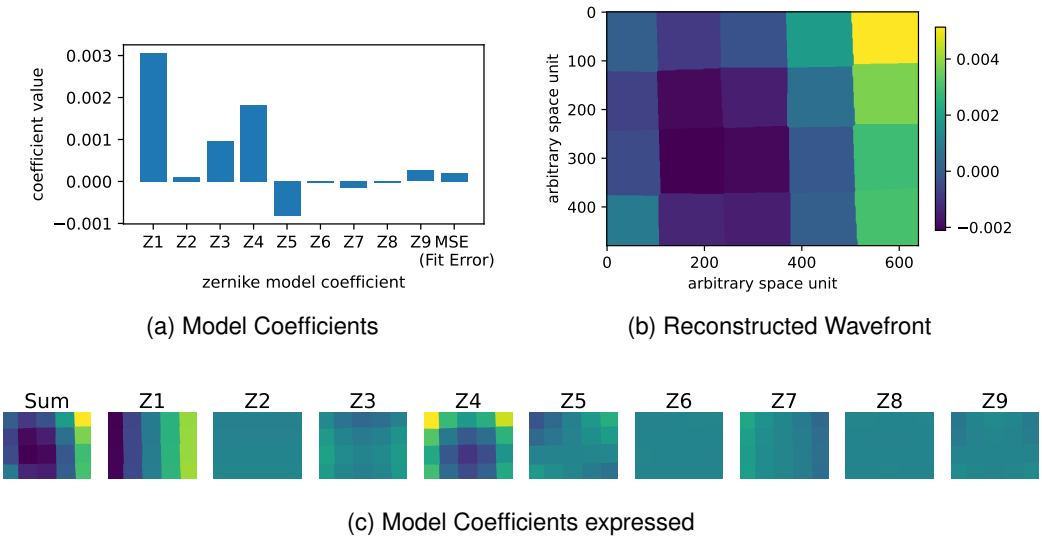


Figure 12: Sample wavefront of index 70

7.4 wavefront reconstruction

The next figure 13 depicts the actually reconstructed wavefront from the fitted Zernike model. In this figure, a valley at around x-200 and y-300 is clearly visible. This is consistence with the expectation of a spherical wavefront shape. The dark valley is, from the perspective of the sensor, the peak point of the wavefront. This point is very sensitive to the position of the lens in the beam, as the next plot will show. The xy-shift of the valley is expressed in the Z1 and Z2 coefficients of the model. Because the X-shifting of the valley from the centre of the image is far larger than its y-shifting and the Z1 value is much larger than Z2 as well.

Whenever a model is fitted to some data, there are some residuals, that can not be expressed by the models. These have to be relatively low to have a meaningful model representation of the data. The mean square error is one common way of representing this difference in real data and model representation. In the case of the orthogonal Zernike polynomials, these residuals will converge to zero if the model order reaches infinity. The last entry in the coefficient bar graph is the value for this fit. The value is not directly comparable to one another, because it was squared, but it is far smaller than many other model contributions. This shows, that the data can be represented rather well by the model.



(a) Model Coefficients

(b) Reconstructed Wavefront

(c) Model Coefficients expressed

Figure 13: Sample Wavefront reconstruction of index 70

To evaluate the whole data from the performed experiment and investigate the measurability of the movement of the lens, figure 14 depicts the Zernice coefficients of all captured wavefronts.

The first two coefficients Z1 and Z2, depict the x and y are shifting very rapidly compared to the other. This is probably due to the unstable holding of the lens in the laser beam. This was one concern during data capturing. The most interesting coefficient, in this experiment, is the red line Z4. It represents the equation $x^2 - y^2$, so its expression shapes similar to a sphere. This coefficient is extremely stable over the whole measurement and only varies with the movement of the lens, so it supported the expected influence of the lens.

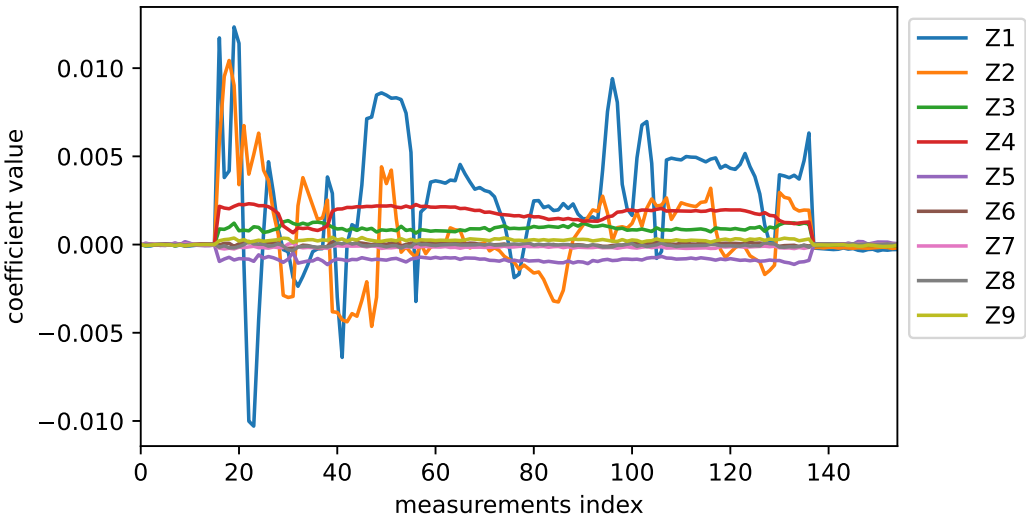


Figure 14: Zernike coefficients for every measurement

The model coefficient Z4 is better visible in figure 15, where the coefficients of Z1

and Z2 were omitted. Especially the section from index 60 to 100 is interesting. It was captured during very linear movement of the lens towards the sensor. This movement is very well represented with the same width and linear descending movement of the model coefficient.
Interesting to see here is, that the distance of the lens width Z4 is very well distinguished from lateral or rotational movement, expressed in Z1 and Z2.

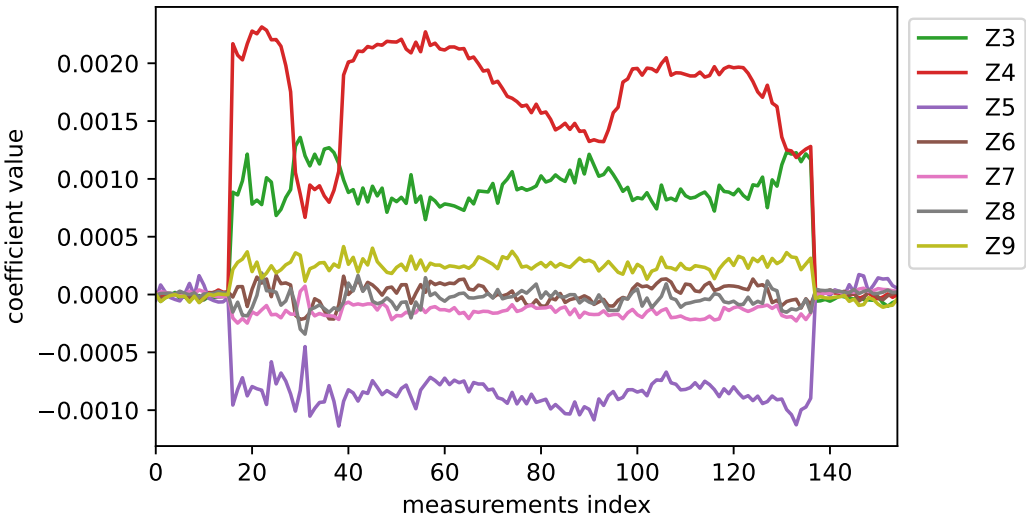


Figure 15: Zernike coefficient Z3-Z9 for every measurement

To present a complete representation of the data figure 16 depicts all reconstructed wavefronts from the measuring series. The colour range for all images was capped to the total maximal and minimal value happening in this series.

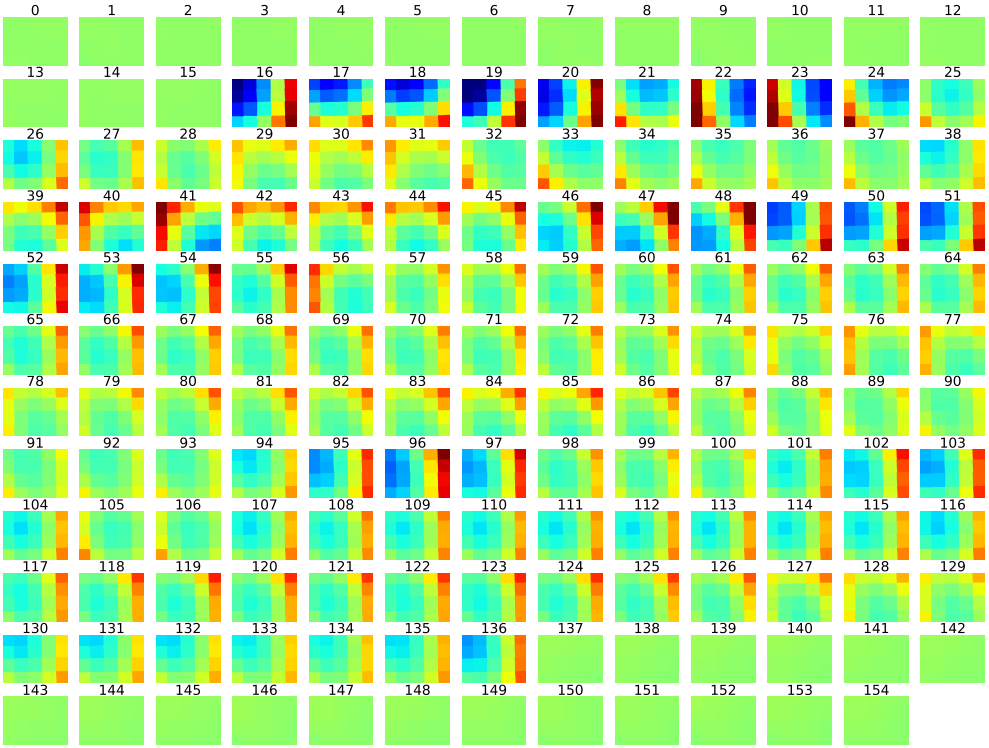


Figure 16: All captured wavefront reconstruction

8 Conlusion and Reflection

The base functionality of the sensor and reconstruction algorithm was proven to work and a plausible wavefront has been constructed. This marks the project to some degree as complete and successful. But there are many improvements and changes that were planned to be included. So the initial expectation was not fully met, with this final product. The Pi Camera can deliver far higher resolution images and with 10-bit, it was not very thoroughly investigated how this could be used in an efficient way and yield more precise results. Also, The processing was switched to be performed offline on a regular x86 Computer, so the performance of the Pi for analyses was not further investigated at a certain point in the development. The number of apertures could also be increased without problems, and the resolution of the images and the minimal movement of the peaks could accommodate this change. Also, the physical accuracy was completely ignored. Creating real results and measuring wavefront with a priori known properties for calibration, would be also very interesting.

In the end, this project has delivered huge learning using Linux, Raspberry Pi in general, the PiCamera, creating a structured python library with some inclusion of C-Code. Besides that also version control with GitHub was completely new to the author. The implementation of a novel algorithm for a paper was new and all the image processing and program structuring was newer done at this scale and over this timespan. This is

also the most frustrating factor, seeing how much time was spent on very unnecessary elements e.g. all the detailed investigation of fast centroid calculation, which is completely irrelevant to the state of the project. Also, the communication with the advising professor regarding the Project status was poorly done.
Besides these negative elements, it bought much joy working on the individual components and seeing plausible wavefront and very measurable effects of moving the Lens work satisfying well in the end.

References

[1] M. Hettwer, “Der shack-hartmann-wellenfrontsensor.”

[2] G. A. Smith, “2d zonal integration with unordered data,” *Applied optics*, vol. 60, no. 16, pp. 4662–4667, 2021.

[3] A. Schicht, “Fokussierendes radarverfahren im millimeterwellenbereich zur radialspaltmessung in kraftwerksturbinen,” doctoralthesis, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 2012.

[4] B. E. A. Saleh, M. C. Teich, and J. W. Goodman, *Fundamentals of Photonics*, ser. Wiley series in pure and applied optics. New York, USA: John Wiley & Sons, Inc, 1991. [Online]. Available: http://onlinelibrary.wiley.com/book/10.1002/0471213748

[5] S. Nazeem Basha, Dr. S.A.K. Jilani, Mr.S. Arun, “An intelligent door system using raspberry pi and amazon web services iot.”

[6] Hedser van Brug, “Efficient cartesian representation of zernike polynomials in computermemory.”