Creating tailorable optical traps with a Digital Micromirror Device for the Dysprosium quantum gas experiment

Bachelor Thesis

Natascha FRITSCHLE

Presented to
Dysprosium Labor
Physikalisches Institut 5
Universität Stuttgart

Examiner: **Prof. Dr. Tilman PFAU**
Supervisor: **Dr. Lucas LAVOINE**

17 th August, 2024

ABSTRACT

In physics it is desired to be able to generate customized optical potentials, since they enable the study of e.g. exotic states of matter like supersolids and quantum droplets. [35]

This Bachelor Thesis details the design of a system capable of generating customizable potentials for capturing two-dimensional Bose-Einstein Condensates of Dysprosium. The potentials are formed by adjusting the amplitude of a 532 nm laser using a Digital Micromirror Device. The optimization loop, the traps precision and it's stability are also discussed. The setup is optimized for the iris aperture and the loop for the error of the gain. It is shown, that low gain values result in a stable converging approximation of the desired potential and an iris aperture is able to increase the flatness of it a bit.

ZUSAMMENFASSUNG

In der Physik ist man bestrebt maßgeschneiderte optische Potential erzeugen zu können, da diese das Erforschen von bspw. exotischen Zuständen wie supersolids oder quantum droplets ermöglichen. [35]

In dieser Bachelorarbeit wird der Entwurf eines Systems beschrieben, das in der Lage ist, maßgeschneiderte Potentiale zur Erfassung zweidimensionaler Bose-Einstein-Kondensate von Dysprosium zu generieren. Die Potentiale werden gebildet, indem die Amplitude eines 532 nm Lasers mit einem digitalen Mikrospiegelgerät eingestellt wird . Die Optimierungsschleife, ihre Genauigkeit und die Stabilität der Falle wird diskutiert.

Der Aufbau ist für die Verstärkung des Fehlers, sowie für die Iris-Blende optimiert. Dabei stellt sich heraus, dass kleine Verstärkungen besser geeignet sind, da ihre Approximation an des gewünschte Potential stabil verläuft und eine Iris-Blende die Flachheit der Potentiale etwas verbessern kann.

EHRENWÖRTLICHE ERKLÄRUNG

Hiermit bestätige ich, dass ich diese Arbeit selbständig verfasst habe, ich keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe, alle wörtlich oder sinngemäß aus fremden Werken übernommenen Aussagen als solche gekennzeichnet habe, die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahren war oder ist, der Inhalt des elektronische Exemplars mit dem des Druckexemplars übereinstimmt.

Ort, Datum

Unterschrift

CONTENTS

1 INTRODUCTION

The team in the 5 th physical institute in the university of Stuttgart achieved chromium condensation in 2004.[1] More recently, they have observed dipolar quantum droplets[2] and one-dimensional supersolids using Dysprosium atoms[3] .
Dysprosium is of striking interest, since the atoms have a high magnetic moment $\mu_m \approx 10\,\mu_B$[1], causing a dipolar interaction[4] between the atoms coming from the magnatic dipole-dipole interaction. It is anisotropic and compared to short range interactions typically employed in ultracold atoms, the interaction energy of dipoles decays slow with $1/r^3$, which is why the interactions have a long range.[23], Their interaction energy

$$E_{dip} = \frac{\mu_0}{4\pi}\left[\frac{m_1 m_2}{r^3} - \frac{3 m_1 r m_2 r}{r^5}\right] \tag{1.1}$$

is dependant on the two magnetic dipoles $m_{1,2}$ and their distance *r*, with μ_0 the permeability of free space.

Currently the team explores one-dimensional supersolids in a finite elongated trap. They are aiming towards studying supersolids in torus geometry.

After laser cooling, when the temperature is low enough, temperatures of about 1 mK, atoms can be trapped and manipulated using a far-off-resonance optical dipole trap.[6] Gaussian beams would result in a harmonic trap, to create non-harmonic traps there are various approaches. For example various trap shapes can be created using a Spatial Light Modulator. It can control and adjust the lights intensity or phase quickly.

There are different types of Spatial Light Modulators, one of them being the Digital Micromirror Device. It consists of an array of micromirrors controlled by a computer. Digital Micromirror Devices are widely used amongst different fields including digital projectors for home theaters, cinemas, but also physics for creating custom potentials.[34][24] These micromirrors only have two possible positions.

The goal of this thesis is to create tailorable optical traps using such a DMD. The first part of the final setup with outcoupler, high power fiber as well as the first stage of demagnification shall be set up. Next, a correction loop shall be implemented, characterized and optimized for the gain of the error and the iris aperture.

A short overview of the theory of optical trapping is given in Chapter 2. It contains the semiclassical approach for a two level system to calculate the dipole-dipole interaction potential. Chapter 3 introduces the Digital Micromirror Device and characterizes the diffraction patterns obtained by using one. The reasoning for the chosen setup and the Point-Spread-Function are also discussed. The algorithm that is used in the correction of the potentials in this thesis is explained in Chapter 4. Afterwards, the optical setup is explained and characterized in Chapter 5. Also here the Point-Spread-Function is discussed in the explicit framework of the Digital Micromirror Device, as well as the setup of the loop, the characterization of it's precision, it's initial test and performance for a rectangle and a torus. The loop is optimized for different parameters. The fluctuations of the setup are characterized and in an attempt to better the traps precision for the torus an intensity compensation is implemented.

At last an overview of remaining adjustments and a conclusion is given in Chapter 6.

1 Bohr magneton $\mu_B = 9.2740100657 \cdot 10^{-24} \mathrm{JT}^{-1}$

2 OPTICAL TRAPPING

Optical trapping, the art of using light to capture and control tiny particles, has opened new frontiers in science. By exploiting the subtle forces of light, researchers can manipulate everything from single molecules to living cells with extraordinary precision.[5] This chapter gives a short overview over optical trapping using a semi-classical approach. Considering a two level atom composed of $|g\rangle$ the ground state with energy
$E_g = 0$ and the excited states $|e\rangle$ with energy $E_e = \hbar\omega_0$,

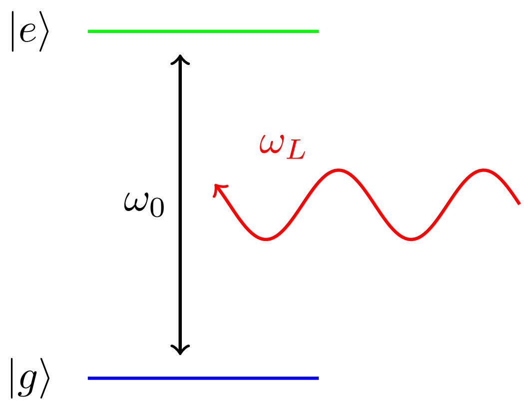


Figure 1: Schematic two level system. Figure inspired from [13].

as depicted in Figure 1.
The generic state of such a system

$$\psi = c_1 |g\rangle + c_2 |e\rangle , \quad c_{1,2} \in \mathbb{C}, \quad |c_1|^2 + |c_2|^2 = 1 \tag{2.1}$$

is given by the superposition of those two states. The density matrix operator is given by

$$\rho = \begin{bmatrix} \rho_{gg} & \rho_{eg} \\ \rho_{ge} & \rho_{ee} \end{bmatrix} = \begin{bmatrix} |c_1|^2 & c_2^* c_1 \\ c_1^* c_2 & |c_2|^2 \end{bmatrix} , \quad \rho_{xy} = |x\rangle \langle y| , \quad x, y \in \{e, g\} . \tag{2.2}$$

Placing an atom in an electric field

$$\mathbf{E}_L(\mathbf{r}, t) = \boldsymbol{\varepsilon}_L(\mathbf{r}) \cos(\omega_L t + \phi(\mathbf{r})) , \tag{2.3}$$

where $\boldsymbol{\varepsilon}_L(\mathbf{r})$ is the polarization vector, a dipole moment $\mathbf{p}$ is induced. This can be done by applying laser light on the atoms. The dipole moment oscillates with ω_L, also called *pulsation*. The dipole moment can be written as

$$\mathbf{p} = \alpha(\omega_L)\mathbf{E} \tag{2.4}$$

is dependant on the complex polarizability[2] α.[13]

Using the density matrix from equation (2.2) and the electric dipole operator
$\hat{\mathbf{d}} = -e\hat{\mathbf{x}}$ with $\hat{\mathbf{x}}$ being the position operator, the electric dipole moment can be described alternatively as the trace of the matrix of the position operator multiplied with the density matrix operator

$$\mathbf{p} = \mathrm{Tr}[\hat{\rho} \cdot \hat{\mathbf{d}}] . \tag{2.5}$$

To solve this equation, one can examine the evolution of the density operator. This is done by using a semi-classical approach, where the motion of atoms is described

2 Normally α is a tensor, for this discussion it is a scalar.

classically while considering the quantum nature of their internal dynamics, This allows to determine the average value of the dipolar interaction using the *Liouville-Von Neumann* equation

$$\frac{d\hat{\rho}}{dt} = -\frac{i}{\hbar}[\hat{\mathcal{H}}, \hat{\rho}] \,. \tag{2.6}$$

Using the potential for the coupled levels depicted in Figure 1

$$V_{dip} = \mathbf{p} \cdot \mathbf{E} = -p_0 E(|e\rangle \langle g| + |g\rangle \langle e|) \cos(\omega_L t + \phi) \,, \tag{2.7}$$

inserting the Hamiltonian

$$\hat{\mathcal{H}} = \hbar\omega_0 |e\rangle \langle a| + V_{dip} \tag{2.8}$$

and computing

$$\frac{d\rho_{eg}}{dt} = -i\omega_0 \rho_{eg} - i\Omega \cos(\omega_L t - \phi)(\rho_{gg} - \rho_{ee}) \tag{2.9}$$

with $\Omega = \frac{p_0 E}{\hbar}$ defining the strength of the laser coupling and p_0 the reduced atomic dipole

$$\rho_{eg}(t) = \frac{\Omega}{2}\left[\frac{e^{-i(\omega_L t - \phi)}}{\omega_L - \omega_0} - \frac{e^{i(\omega_L t - \phi)}}{\omega_L + \omega_0}\right] = \rho^*_{ge} \tag{2.10}$$

is obtained.
To do so, the fact that the laser pulsation is far from resonance $|\omega_L - \omega_0| \gg 0$ is used, which allows to approximate $\rho_{gg} = 1$ and $\rho_{ee} = 0$.

Given that, the value of the dipole operator can be calculated

$$\mathbf{p} = \mathrm{Tr}[\hat{\rho} \cdot \hat{\mathbf{d}}] = \frac{2p_0\omega_0}{\hbar(\omega_L^2 - \omega_0^2)} E = \alpha(\omega_L) E \,, \tag{2.11}$$

defining $\alpha(\omega_L) = \frac{2p_0\omega_0}{\hbar(\omega_L^2 - \omega_0^2)}$. [7]
Finally, the mean dipole interaction potential evaluated over time

$$V_{dip} = -\frac{1}{2} \langle \mathbf{p} \cdot \mathbf{E} \rangle = \frac{\mathrm{Re}(\alpha) E^2}{2} = -\frac{\mathrm{Re}(\alpha) I}{2\varepsilon_0 c} \tag{2.12}$$

is obtained. It holds the factor $\frac{1}{2}$ to consider that the dipole moment is induced. It depends on the lasers intensity *I* and to the real part of the polarizability α. This real part corresponds to the in-phase component of the dipole oscillation, which is crucial for determining the dispersive characteristics of the interaction. [6]

Figure 2 shows an energy diagram and illustrates how a gaussian beam forms a trap. Detuning the laser frequency slightly lower than the resonance frequency, $\omega_L < \omega_0$, provides a red-detuned trap. The potential (2.12) becomes negative, attracting atoms.
For the blue-detuned trap the laser frequency is a bit higher than the resonance frequency, $\omega_L > \omega_0$, creating a positive potential and thus rejecting atoms.[19]

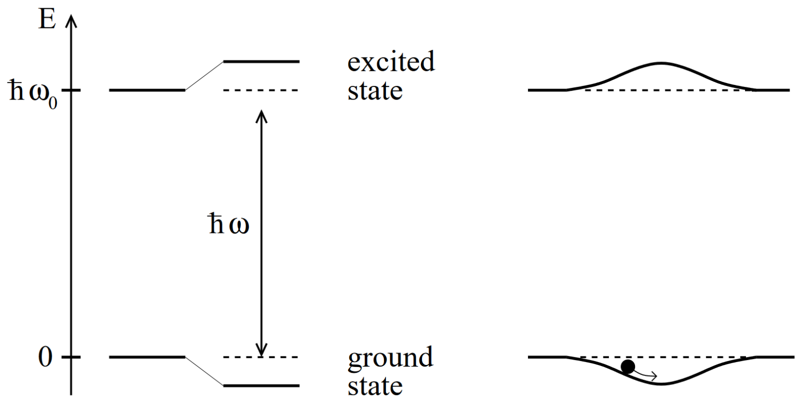


Figure 2: Energy diagram for a two level atom. On the left, red-detuned light ($\omega_L < \omega_0$) lowers the ground state energy and raises the excited state energy equally. On the right, a Gaussian laser beam, forms a potential well in the ground state, trapping the atom. Figure from [6].

The Dysprosium ground state has an angular momentum $J = 8$ and $m_J = -8$, meaning it is an anisotropic medium. This means the polarizability α must be treated as a tensor. Regarding this [20] and also that in this experiment a 532 nm laser is chosen to trap the Dysprosium atoms, already a small deviation in the wavelength can already change the polarizability significantly. To provide sufficient trapping, one has to choose the polarization of light carefully.[14]

3 DIGITAL MIRCROMIRROR DEVICES

As mentioned earlier, the dysprosium team wants to explore the physics of dipolar supersolids in a torus. One of the first steps towards that is to create an optical trap with torus geometry.
Digital Micro Mirror Devices (DMDs) have emerged as powerful tools for creating highly customizable optical traps.[35] Here, the V-9501 from Vialux is used.
The DMD surface of the chip consists of an array of 1920 x 1080 squared aluminium mircomirrors of width $d = 10.8\,\mu m$. The effective width of the mirrors is $d_{eff} \approx 10.2\,\mu m$, since the fill factor is 94 %. Each micromirror can be tilted for $\theta_B = \pm 12\,°$ around it's diagonal axis.[10]

The mirrors have three possible states, named On, Off and the Parked state, depicted in Figure 3.

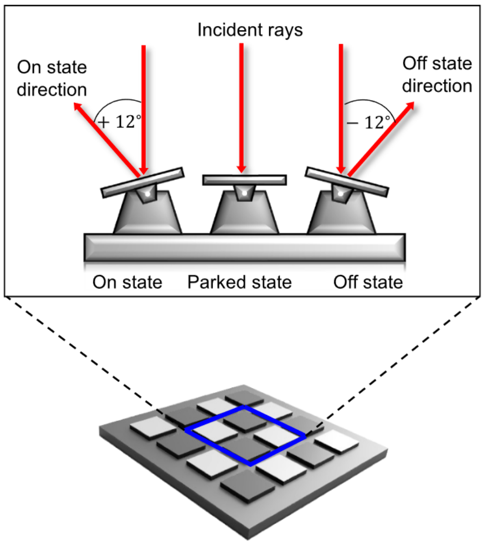


Figure 3: Three possible mirrorstates: On, Parked, Off for the DMD. The On state corresponds to a white pixel, the Off state to a black pixel. The Parked state is the defualt state when not using the DMD. Figure from [15].

The On(Off) state corresponds to a white(black) pixel, the Parked state is the default state, when the DMD is not used.[10]

Thanks to the Python API[9] it is possible to access the DMD. One can display a tailored image by loading an array onto the DMD. Due to the binarity of the states of the micromirrors each mirror is represented by an entry in the loaded array of either 1 or 0 (On or Off).

3.1 Diffraction

Due to the small size and the periodic arrangement of the mirrors the DMD acts as a diffraction grating. The DMD's intensity pattern is obtained by convolving it's grid pattern[25]

$$\mathrm{grid}(x, y) = \sum_{\substack{0 \leqslant i < 1920 \\ 0 \leqslant j < 1080}} \delta(x - ai, y - aj)\,, \tag{3.1}$$

where $\delta(x, y) = \delta_{x,y}$, with a rectangle in two dimensions [25]

$$\mathrm{rect}(x, y) = \begin{cases} 1 & \text{if } |x| \leqslant \frac{d_{\mathrm{eff}}}{2} \wedge |y| \leqslant \frac{d_{\mathrm{eff}}}{2}, \\ 0 & \text{if } |x| > \frac{d_{\mathrm{eff}}}{2} \vee |y| > \frac{d_{\mathrm{eff}}}{2} \end{cases} \tag{3.2}$$

since a single mirror of the DMD corresponds to a rectangle.

3.1.1 Diffraction of the grid

The DMD provides an additional degree of freedom beyond the standard configuration: the ability to adjust the tilt angle of the mirrors.[13] For constructive interference the condition

$$m\lambda = d_{\mathrm{eff}}(\sin\alpha + \sin\theta_0)\,, \quad m \in \mathbb{Z} \tag{3.3}$$

needs to be fulfilled. The blazing condition refers to the specific alignment of the mirror tilt angle, such that the reflected light is directed primarily into a desired diffraction order *m*. This maximizes the efficiency of light in that order, enhancing the intensity of the projected image.

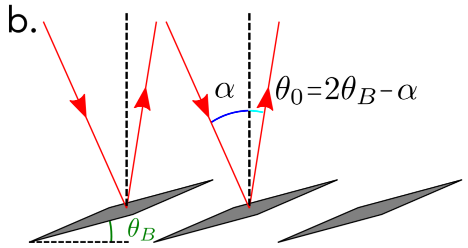


Figure 4: Schematic illustration of the blazed grating on the DMD in 2D. The incident angle is denoted by α, the mirror tilt is given by θ_B and the angle of for the zeroth order θ_0. Figure from [24]

The angles are defined according to Figure 4. Also, the light envelope is reflected by each mirror, leading to a peak in intensity when the reflection condition for the mirrors

$$\theta_0 = -\alpha + 2\theta_B \tag{3.4}$$

is met.[14] Using that, the intensity pattern of the DMD's [25] grid is given by

$$I_{grid} = \sum_{\substack{-\infty<m<\infty \\ -\infty<n<\infty}} \delta\left(\sin(\alpha_x) + \sin(\theta_{0,x}) - \frac{m\lambda}{d_{eff}}, \sin(\alpha_y) + \sin(\theta_{0,y}) - \frac{n\lambda}{d_{eff}}\right), \tag{3.5}$$

where $\alpha_{x(y)}$ and $\theta_{0,x(y)}$ are the *x(y)* components of α, θ_0 respectively. The diffraction order in 2D is denoted by *m* and *n*.

3.1.2 *Diffraction of one single mirror*

The diffraction of one single mirror is represented by (3.2). It's Fourier transform

$$\mathcal{F}[\text{rect}] = \text{Rect} \propto \text{sinc}\left(\frac{\pi d_{eff}}{\lambda}(\sin(\theta_x) - \sin(\theta_{sr,x}))\right) \text{sinc}\left(\frac{\pi d_{eff}}{\lambda}(\sin(\theta_y) - \sin(\theta_{sr,y}))\right) \tag{3.6}$$

is dependant on the angles $\theta_x = \arcsin\frac{x}{z}$ and $\theta_y = \arcsin\frac{x}{z}$ of the DMDs normal in the x, respectively y axis. [25]

The angles $\theta_{sr,x(y)}$ are between the specular reflection and the DMD's normal in the $x(y)$ axis.

Applying the *Frauenhofer* principle [18] the intensity pattern for a single mirror

$$\begin{aligned} I_{rect} &= \text{Rect}(x, y, z)^2 \\ &\propto \text{sinc}\left(\frac{\pi d_{eff}}{\lambda}\sin(\theta_x) - \sin(\theta_{sr,x})\right)^2 \text{sinc}\left(\frac{\pi d_{eff}}{\lambda}\sin(\theta_y) - \sin(\theta_{sr,y})\right)^2 \end{aligned} \tag{3.7}$$

is obtained.[25]

3.2 Diffraction pattern of the DMD

Finally, the pattern on the DMD can be calculated by multiplying the two obtained intensity patterns[25]

$$\begin{aligned} I_{DMD} = I_0 \text{sinc}\left(\frac{\pi d_{eff}}{\lambda}\sin(\theta_x) - \sin(\theta_{sr,x})\right)^2 \text{sinc}\left(\frac{\pi d_{eff}}{\lambda}\sin(\theta_y) - \sin(\theta_{sr,y})\right)^2 \\ \sum_{\substack{-\infty<m<\infty \\ -\infty<n<\infty}} \delta\left(\sin(\alpha_x) + \sin(\theta_{0,x}) - \frac{m\lambda}{d_{eff}}, \sin(\alpha_y) + \sin(\theta_{0,y}) - \frac{n\lambda}{d_{eff}}\right). \end{aligned} \tag{3.8}$$

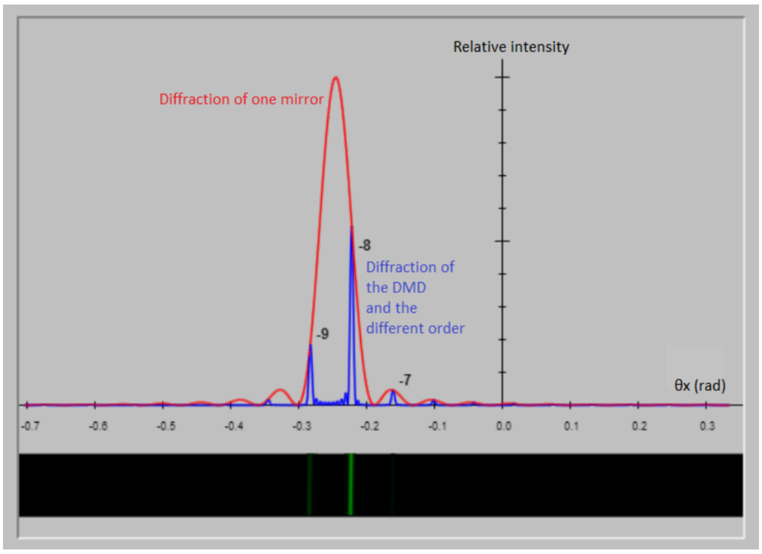


Figure 5: Relative intensity of the diffraction for the whole DMD as well as a single mirror. For the DMD different orders are highlighted. Figure from [25].

Figure 5 depicts the calculated intensity patterns (3.8), (3.7). From now on the main order is the order with the coefficent (m,n) with m=n.

3.3 Setup

The DMD is placed in the object plane, see Chapter 5 for experimental setup, the error due to the tilt of the mirrors is neglectable, since the mirrorsize is very small compared to the focal distance.

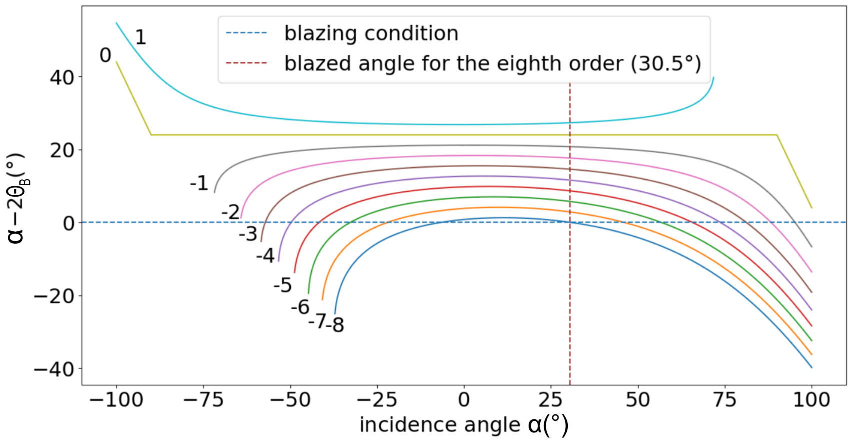


Figure 6: Blazed condition for different orders $m \in \{-8, -7, -6, ..., 0, 1\}$. The dashed vertical line shows the incidence angle α for m=8 according to 3.3. The horizontal dashed line shows the new diffraction condition (3.9). Figure from [25].

The goal is to maximize the intensity in one particular order, this can be done in the blazing angle [17], which in this case is found to be equal to $\alpha = 30.5\,°$ for the -8 th order, Figure 6 [25].

In order to reflect the light perpendicular to the DMD $\theta_0 = 0$ is chosen [14] meaning the new condition that has to be fulfilled is

$$m\lambda = d_{\text{eff}} \sin\alpha \,. \tag{3.9}$$

Since $\alpha = 30.5^\circ$ doesn't satisfy (3.9), the closest feasible angle is

$$\alpha = \arcsin \frac{8\lambda}{d_{\text{eff}}} = 24.66^\circ\,. \tag{3.10}$$

3.4 Point-Spread-Function

When light passes through an optical system with finite size, a single point on the object isn't images as a perfect point. Instead, it appears as a spread-out pattern called a point spread function (PSF), which is the diffraction pattern of a point object after the optical setup.

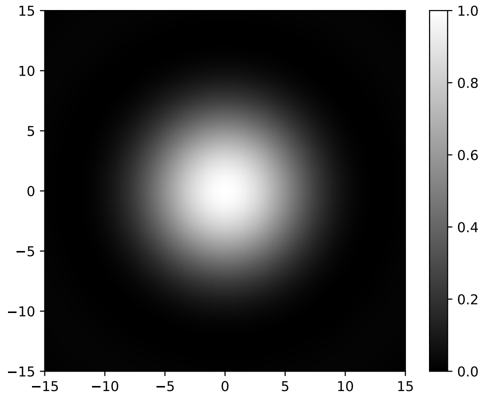


Figure 7: Simulated airy disk according to (3.11).

For spherical optics this spread-out pattern is well approximated by an airy disk

$$I(\theta) = I_0 \left(\frac{J_1(x)}{x} \right)^2 , \quad x = \frac{\pi D \sin\theta}{\lambda} , \tag{3.11}$$

see Figure 7 for a simulated example, where J_1 is the Bessel function of first kind of order and *D* is the diameter of the lens. [16]

4 FLOYD–STEINBERG ALGORITHM

Due to inhomogenities of the light source and lens aberations the final image of the DMD gets distorted. In order to reach the desired pattern it needs to be corrected, the DMD's binarity complicates this. It is necessary to find a suitable method to control the spatial intensity. This section introduces the method used in this thesis. In digital image processing, while images can be binarized based on a threshold to simplify intensity values, dithering offers a more nuanced approach. Dithering is a technique that strategically applies noise to the image in order to simulate graylevel beyond a device's color palette.
One of the most widely used algorithms for dithering is the *Floyd-Steinberg algorithm*[8]. This method propagates quantization errors from pixel to pixel throughout the image, effectively smoothing transitions and enhancing visual fidelity.

The effect is fascinating, Figure 8 shows a grayscale image in it's original version compared to the binarized and dithered version. The original and the dithered image appear to look the same, although the dithered image actually is binary, yet looks very different from the binarized image.

Only by looking very closely, one can see that the dithered image is a bit more grainy. Zooming in to the eye, the binar structure of the image reveals itself and one can get a feel for how the error is propagated along the image, see Figure 8 on the right.

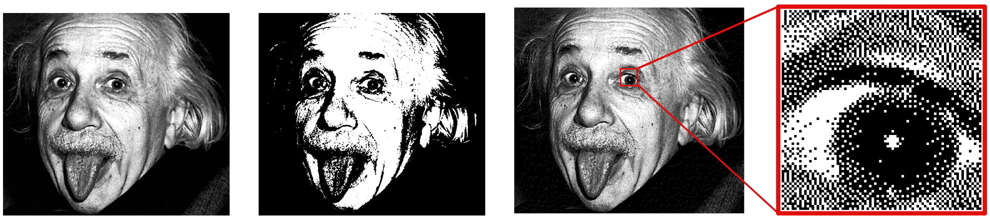


Figure 8: On the left a greyscale image[12] is given, in the middle the binarized image with a threshold of 128, and on the right the dithered image with the *Floyd-Steinberg* algorithm. The small square in red is the area that was zoomed in. On the right is the zoomed in part of the dithered *Floyd-Steinberg* image.

In mathematical terms the error is propagated with an error diffusion matrix [13]

$$\begin{bmatrix} I_{(i-1,j-1)} & I_{(i-1,j)} & I_{(i-1,j+1)} \\ I_{(i,j-1)} & I_{(i,j)} & I_{(i,j+1)} \\ I_{(i+1,j-1)} & I_{(i+1,j)} & I_{(i+1,j+1)} \end{bmatrix} = \tag{4.1}$$

$$\begin{bmatrix} I_{(i-1,j-1)} & I_{(i-1,j)} & I_{(i-1,j+1)} \\ I_{(i,j-1)} & I_{(i,j)} & I_{(i,j+1)} \\ I_{(i+1,j-1)} & I_{(i+1,j)} & I_{(i+1,j+1)} \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{7}{16} \\ \frac{3}{16} & \frac{5}{16} & \frac{1}{16} \end{bmatrix} \cdot \epsilon \,. \tag{4.2}$$

in row-major order, beginning at pixel (0,0). Here, ϵ is the calculated error

$$\epsilon = I_{(i,j)} - \text{round}[I_{(i,j)}] \tag{4.3}$$

based on the normalized image and the function $\text{round}[I_{(i,j)}]$, that rounds the value $I_{(i,j)}$ to either 1 or 0. The matrix *I* here is a snippet of the normalized image containing all the neighbouring pixels of the pixel *(i,j)* for some of which the error from the binarization is currently distributed. The entries of the error diffusion matrix are selected for a uniform intensity of 0.5 to produce a checkerboard pattern. A graphic representation of the error propagation is given in Figure 9.[13]

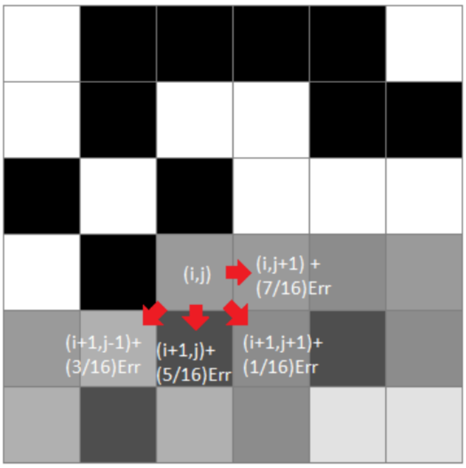


Figure 9: Graphic representation of the error propagation of the *Floyd-Steinberg* algorithm visualizing the error diffusion matrix (4.1). Figure from [13].

Thanks to the Python library pillow[27] the algorithm is easy to implement, but comes with a loss of information compared to the original image, as the second picture from the left in Figure 8 demonstrates.

4.1 Correction loop

Taking into account that the PSF can overlapp, Figure 12, grayscales can be achieved using the correction method with the *Floyd-Steinberg* dithering. The correction loop to iteratively obtain the desired potential is sketched in Figure 10. Iteration n produces the in the end of the loop the n th correction image. For iteration n+1 in the beginning the error is calculated as the difference between the target image and the n th recorded image_n multiplied by a constant Kp

$$\text{error image}_n = \text{Kp} \cdot (\text{target image} - \text{recorded image}_n)\,, \tag{4.4}$$

which directly yields the error of the pixel intensity values for the whole image. Next, to the error image the $(n-1)$ th old image is added

$$\text{new image}_n = \text{error image}_n + \text{old image}_{n-1}\,, \tag{4.5}$$

where the old image_{n-1} is the convolution of the (n-1) th recorded image_{n-1} with the point spread function.
For the first two iterations the old image is set equal to the target image, since there yet isn't an image to be able to compare it to. The obtained new image is passed to the convolution with the point spread function, resizing to fit the DMDs size, dithering and finally the Floyd-Steinberg algorithm.
Convolving old image_{n-1} with the Point-Spread-Function produces old image_n which is saved to use it in the next loop n+2.
It is necessary to convolve old image_{n-1} in order to propagate the effect of the DMD. The image after the Floyd-Steinberg (FS) algorithm, called FS image, is displayed onto the DMD.

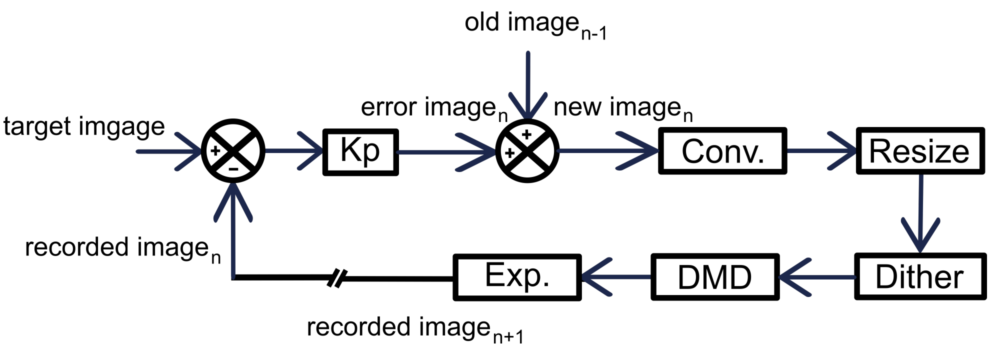


Figure 10: Block schematic diagram of the correction loop used. The calculated output is convolved with a point-spread-function, resized to fit the DMDs size and passed on to the dithering and the Floyd Steinberg algorithm where the image is led to the experiment.

The loop is automated and the code is uploaded to Github[11], there also the README.md is located for further information of installation and dependencies.

5 EXPERIMENTAL SETUP

The test setup is shown in Figure 11. In the Appendix 6 Figure 40 shows the final setup intended to implement into the experiment. It is also already set up, but wasn't used to obtain the results discussed in this thesis.

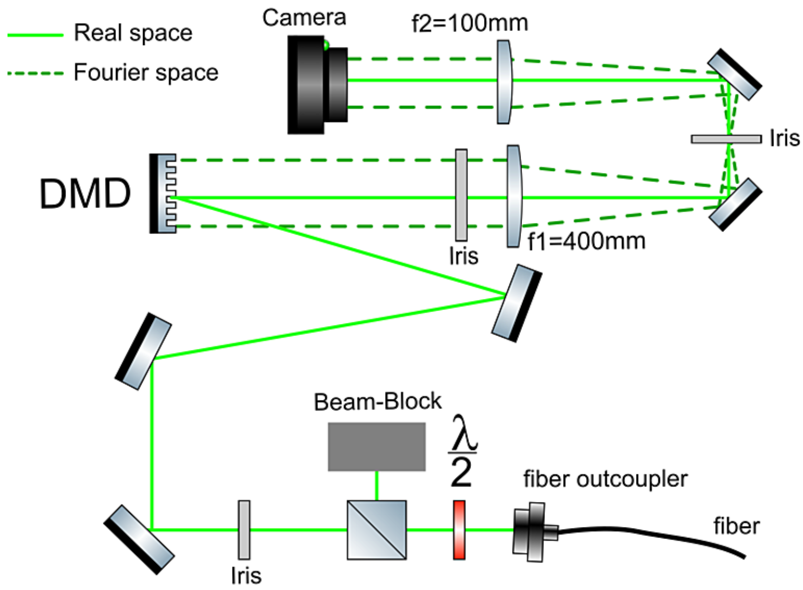


Figure 11: Optical setup used in this thesis

The laser used is a 18 W Coherent Verdi V 10 532 nm laser. The used DMD is the V-9501 from Vialux with a mirror size of $d = 10.8\,\mu m$ and an effective mirror size of $d_{eff} \approx 10.2\,\mu m$. The setup doesn't include an AOM for power stabilization, since it broke during the thesis. In order to create tailored optical traps, it is neccessary to use all micromirrors of the DMD, meaning the beam diameter must fit the DMDs short axis. It is important to make sure that the beam also isn't larger than the short axis, otherwise additional stray light will be scattered off the metallic frame of the DMD.

Furthermore the beam must be colliminated. The outcoming beam has a diameter of $d_{waist} \approx 6.5\,mm$.

The beam shows a hexagon shape, see Figure, which is produced by the cladding mode. It is not really possible to get rid of it[22]. The used outcoupler is a Schäfter+Kirchhoff 60FS-SMA-T-23-M200-04, it was chosen due to the large beam diameter after it. This way a telescope isn't needed to fully illiminate the DMDs short axis.

After the outcoupler the polarization of the beam is cleaned with a cube and a $\frac{\lambda}{2}$ waveplate. The reflected pattern from the DMD has multiple orders, the iris is positioned such that only the 8 th order, which is maximized in intensity, can pass through the f1=400 nm lens. The next iris in front of the f2=100 mm lens smoothes the edges, see Chapter 5.8.1. The telescope has a total demagnification of 4, in order to perform the optimization loop and record the images on the camera properly. The used camera is a FLIR Blackfly S BFS-U3-120S4M with a pixel size of $d_{c,pxl} = 1.85\,\mu m$ [33].
The setup planned to be implemented in the experiment is also already set up and can be found with further explanations and reasoning in the appendix 6.

The DMD's diffraction efficiency is measured to be 68 % in this setup, coming close to 70 % Vialux claims for the blazing angle [10] .

Directly after the DMD one can get an intensity of 3.0 $\frac{kW}{m^2}$. Considering the demagnification of 1/75, an intensity of 16.88 $\frac{MW}{m^2}$ is expected after the third demagnification stage in the final setup. Given the scalar polarizability at 532 nm for Dysprosium $\alpha = 350$ u.a.[28] the potential depth has the order of magnitude of $T = 1\,\mu K$.

5.1 Point-Spread-Function in framework of Digital Micromirror Devices

Taking into account geometrical optics, the displayed images from the DMD should be just as binary as the mirror states on the DMD itself. But this is not the case. Loading a chessboard to the DMD, Figure 12 on the left, the image in Figure 12 on the right is deflected, this has to do with the previous mentioned PSF, see Chapter 3.4.

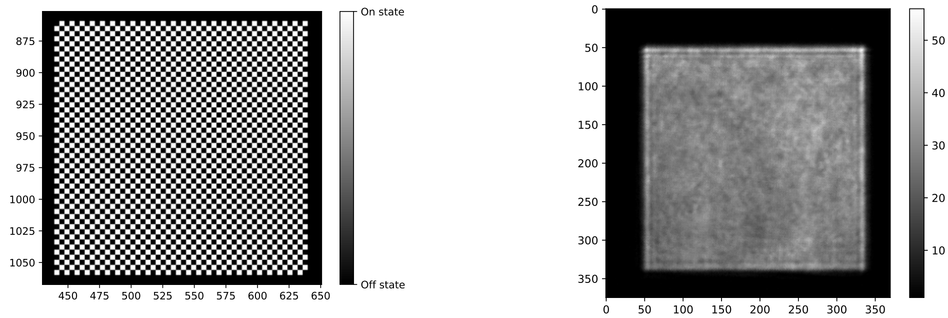


Figure 12: On the left the image loaded onto the DMD displaying a 200 x 200 px chessboard in the center of the DMD, where each black or white field consists of two pixels. On the right the corresponding recorded image is shown. It shows gray shades, instead of a binary structure.

One has also to take into account that the airy disk only considers diffraction due to finite aperture, but there is further due to aberrations resulting from the optics. Given that, the electric fields associated with the PSF of two distant mirrors can overlap and interfere together, which is why shades of grey can be seen in figure 12 on the right and not just black and white pixels. The overlap of multiple PSF is illustrated in Figure 13.

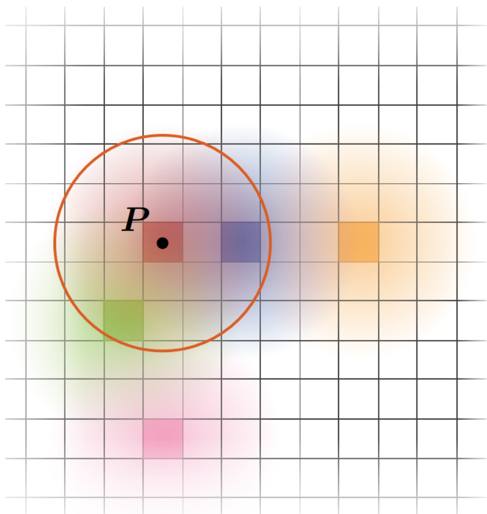


Figure 13: Illustration of the overlap of multiple PSF. When images of individual DMD pixels are projected onto the image plane, and the spot is much smaller than the pixel size, the DMD pixels are distinctly visible. For instance, among the five active pixels. Each is represented in a different color. Only the red one influences the intensity at point P. However, if the image is blurred due to the limited resolution of the imaging system, the pixels appear as large, overlapping spots. In this scenario, all pixels within the orange circle affect the intensity at point P. In the first case, where the PSF of the imaging system is symmetric, three out of the five active pixels (red, blue, and green) significantly contribute to the intensity at point P.[26] Figure from [26].

The same explanation for why the chessboard spreads out to a gray area is also applyable to why the Einstein dithered image, Figure 8, seems to have gray tones. It is because the human eye has finite optical resolution. It is important to point out that the finite PSF is the reason why the optimization loop using the *Floyd-Steinberg* dithering works. Only if the PSF is larger compared to the effective size of the mirrors after the demagnification it is possible to create different gray shades with the binar DMD.
Many illuminated pixels, as mentioned in the previous chapter, are desired not only for creating a large pattern, but also to have many pixels contributing to the PSF, resulting in a larger color palette in gray scale. This DMD has a relatively large pixel size, which is not the best, because this makes the pixels distinguishable as Figure 13 also demonstrates.

The resolution of the setup is determined by the minimal distance at which two pixels can be distinguished. Turning a single pixel in the center of the DMD On, the radius after which the intensity drops to zero is measured to be $r = (7 \pm 1)\,\text{px}$. The optical resolution is then found out to be $o = (7 \pm 1)\,\text{px} \cdot d_{c,pxl} = (13 \pm 1.5)\,\mu\text{m}$.

The image at the end of the optimization loop, Chapter 4.1, will be the convolution of the PSF with the pattern displayed onto the DMD. The calculation for that is shown in the Listing 1.

Listing 1: Snippet of the Python code from the Appendix, where the image passed to the *Floyd-Steinberg* dithering is created by convolving the PSF with new image. The new image is defined according to Chapter 4.1. The size in pixels of the recorded image is given by newsize in the code.

```

x=np.linspace(-newsize//2,newsize//2,newsize)
y=np.linspace(-newsize//2,newsize//2,newsize)
sigmax=1
sigmay=1

exp_convol=np.ones((newsize,newsize))
for i in range(exp_convol.shape[0]):
    for k in range(exp_convol.shape[0]):
        exp_convol[i,k]=1/2/np.pi/sigmax/sigmay*np.exp(-1/2*x[i]**2/sigmax**2-1/2*y[k
            ]**2/sigmay**2)
img_conv=scipy.signal.fftconvolve(new_img,exp_convol,mode="same")
img1=Image.fromarray(np.uint8((np.array(img_conv))),mode="L")
img1.save("previous.bmp")

img2=img1.resize((200, 200))
```

5.2 Setting up the optimization loop

In order to perform the optimization loop successfully one has to characterize the deflected pattern from the DMD first.

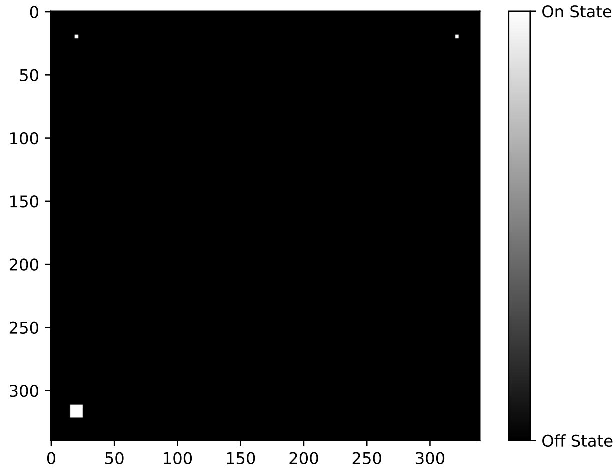


Figure 14: Three points (two 4 px large and one 28 px large) are loaded onto the DMD. The points are created by setting neighbouring pixels equal to one. In the bottom left the point is chosen to be larger by turning more pixels on compared to the other points. The zoomed in part is shown.

By choosing to display a pattern of three points with one selected to be larger than the rest, Figure 14, one can find out the total transformation. Since the DMD is mounted at 45 °, the original pattern appears rotated on the camera. The pattern is created by turning neighboring pixels on, meaning setting their value in the array equal to one. For the recorded image, Figure 15, it is evident that the image has undergone not just a rotation, but also a transposition.

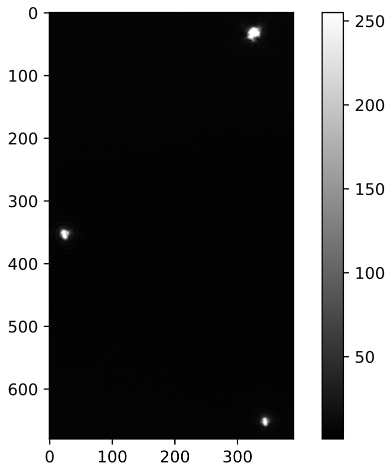


Figure 15: Zoomed image recorded from the camera by loading an array corresponding to Figure 14 to the DMD. The cameras position in the experimental setup is depicted in Figure 11.

To measure the rotation angle, the image is first loaded, transposed and then rotated such that the arrangement of the displayed image, Figure 14, is obtained.

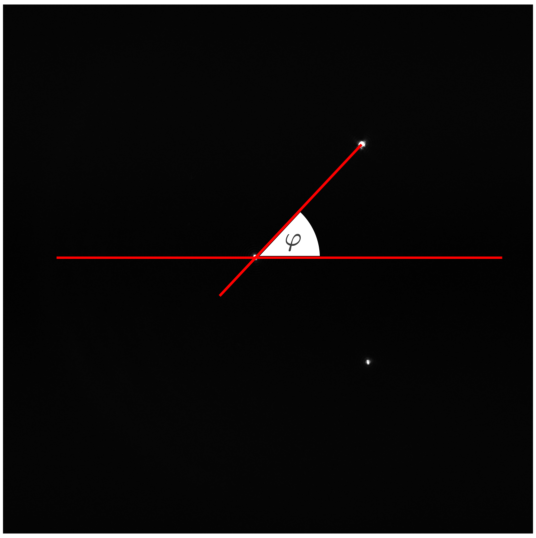


Figure 16: Illustration of how the rotation angle is calculated using for example Inkscape.

This angle can be measured using software such as for example Inkscape by loading the recorded image and measuring the angle between a horizontal line that goes through the second highest point and a line that goes through the highest and second highest point, see Figure 16.

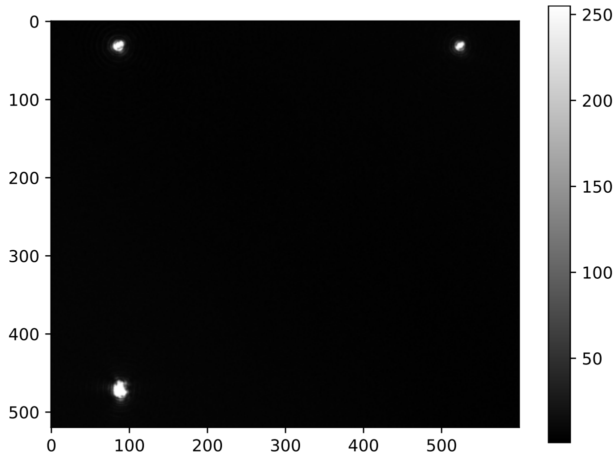


Figure 17: Figure 15 rotated by the measured rotation angle $-46.95\,°$ and cropped. The scale on the right shows the pixel intensity value.

The rotation angle is found to be $\varphi = (-46.95 \pm 0.1)\,°$, the rotated image is shown in Figure 17, also an additional crop has been performed. The precision of the rotation angle is limited by the extent of the points. The middle of the points was chosen as crossing points of the lines.
To measure the demagnification, the distance between the upper two points is measured in pixels for the image loaded to the DMD d_{DMD} and the image recorded by the camera d_{Cam}. The recorded image is already rotated and transposed of course. The distance for the DMD in pixels can be directly calculated from the passed array. Also the effective pixel size for the DMD is used with one pixel corresponding to one micro mirror. The measured distances are multiplied by it's corresponding pixel sizes, in order to get the distance in meters

$$d_{DMD} = (300 \pm 3)\,\text{px} \cdot 10.2\,\frac{\mu\text{m}}{\text{px}} = (3.06 \pm 0.03)\,\text{mm} \tag{5.1}$$

$$d_{Cam} = (424 \pm 3) \cdot 1.85\,\frac{\mu\text{m}}{\text{px}} = (0.784 \pm 0.006)\,\text{mm}\,. \tag{5.2}$$

The error for the distances is estimated to be around 3 px. The ratio between the two distances hold the demagnification

$$m = \frac{d_{Cam}}{d_{DMD}} = \frac{(0.784 \pm 0.006)\,\text{mm}}{(3.06 \pm 0.03)\,\text{mm}} = 0.256 \pm 0.004\,. \tag{5.3}$$

Compared to the ideal demagnification

$$m_{ideal} = \frac{f2}{f1} = \frac{100\,\text{mm}}{400\,\text{mm}} = 0.25\,, \tag{5.4}$$

the obtained value seems reasonable. With that measured, the first test of the loop can be performed.

5.3 Initial test - Optimization for a rectangle

A 200 x 200 px rectangle is displayed onto the DMD, Figure 18.

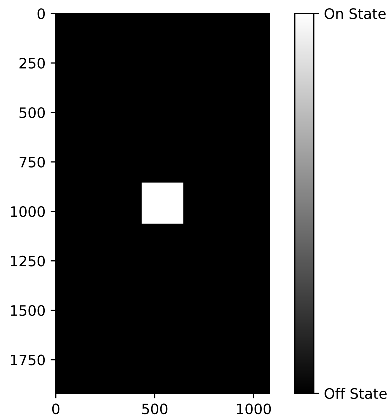


Figure 18: Image loaded onto the DMD displaying a 200 x 200 px rectangle in it's center.

A proper cropping of the transformed (transposed and rotated) recorded image is crucial for the image to be corrected in the right way. Cropping too tight on one edge will leave it uncorrected, see Figure 19, while cropping too loose the loop will also the background of the deflected pattern into account and also calculate an error for that. So the background will also be corrected, making the image larger. Also now is clear why the rotation angle has to be really precise, if it is off, for a tight crop it will cut the edges.

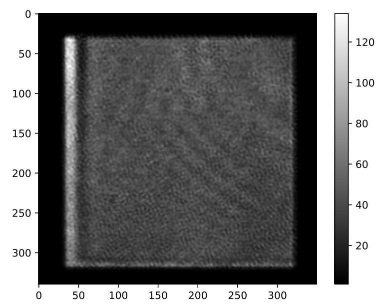


Figure 19: A 200 x 200 px rectangle is displayed onto the DMD. The optimization loop is performed, but the recorded image is cropped too hard on it's left and bottom edge. As a consequence the edge isn't corrected. The eight iteration is shown. The image is cropped, rotated and zoomed.

In order to find the best crop just the edges of a 200 x 200 px square are displayed, Figure 20. This is the easiest method to do so, since the edges are high in contrast.

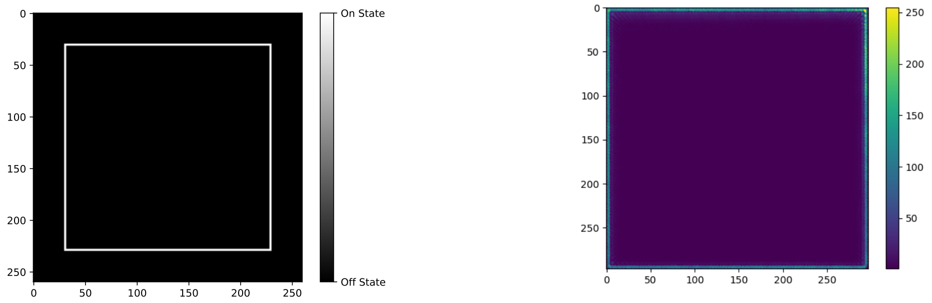


Figure 20: On the left the zoomed image loaded onto the DMD is shown. It is created by a 199 x 199 px square subtracted from a 200 x 200 px square. On the right the recorded image is shown. The image is cropped tight to it's edges, getting the best result for the onwards performed optimization. Instead of grayscale this colormap is chosen for higher contrast.

With that performed, the transformed recorded image, that will be further used in the algorithm is ready, Figure 21.

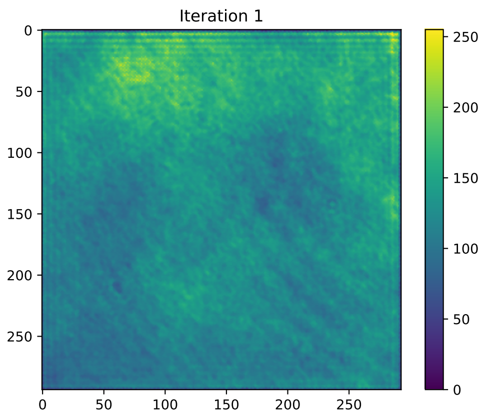


Figure 21: Cropped recorded image for the DMD deflecting Figure 18.

The cropped recorded image, Figure 21, shows a good crop. It needs to be adjusted over time a little bit, this is not automizable since the available methods like pythons edge detection aren't precise enough.

The recorded image is 293 x 293 px large, the target is automatically defined to have the same size in the code.

The optimization loop corrects the image to a defined intensity level. The error image (4.4) is calculated based on a defined target image. Since the DMD is binary and thus the loaded pattern has a maximum intensity of 1, it has to be scaled by a value called minP. This value scales the level to which the intensity of the image will be corrected. Besides scaling the loaded pattern it has to be cropped to the desired target pattern.

All images used in the correction loop in the Appendix shall be shown and discussed here. The recorded image, cropped, rotated and transposed, now is ready to use. To be able to compare the error image, the target image has to be defined, which is simply the image displayed onto the DMD scaled by a value minP.

The error image, see Figure 22 on the left, is obtained by subtracting the recorded image from the target image and multiplying the difference by a value Kp, in this case Kp=0.2 . The target image is obtained by cropping the pattern of the rectangle in Figure 18 and multiplying it with a value minP that scales the intensity to which the pattern shall be corrected, since the DMD is binary in it's intensity values. After that, it is used to calculate the new image, see the image on the left in Figure 23, by adding it to the old image, which is equal to the target image for the first correction. The new image then is convoluted, middle image in Figure 23 and resized to fit the DMD's size 23. After that a background of intensity value zero that matches the DMD's size is added below the new image and it is dithered, creating the final image, see Figure 22.

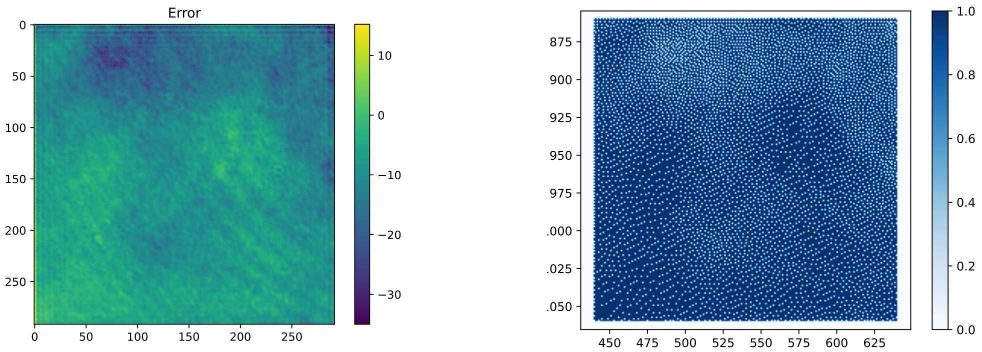


Figure 22: On the left: The error image obtained according to (4.4) with Kp=0.2 . The target image is obtained by cropping the pattern of the rectangle in Figure 18 and multiplying it with a value minP that scales the intensity to which the pattern shall be corrected, since the DMD is binary in it's intensity values. On the right the zoomed *Floyd-Steinberg* dithered image. It is placed on top of an array consisting of zeros, that matches the DMD's size. In the top the error is negative, the reaction to that can clearly be seen in the dithered image: In the top a lot of pixels are set equal to zero, removing intensity.

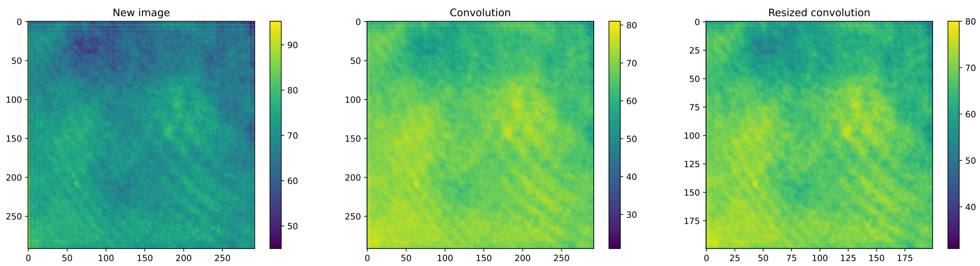


Figure 23: On the left the new image according to (4.5), where the old image is the target image. In the middle the convoluted image and on the right the convolulted image resized to fit the DMD's size.

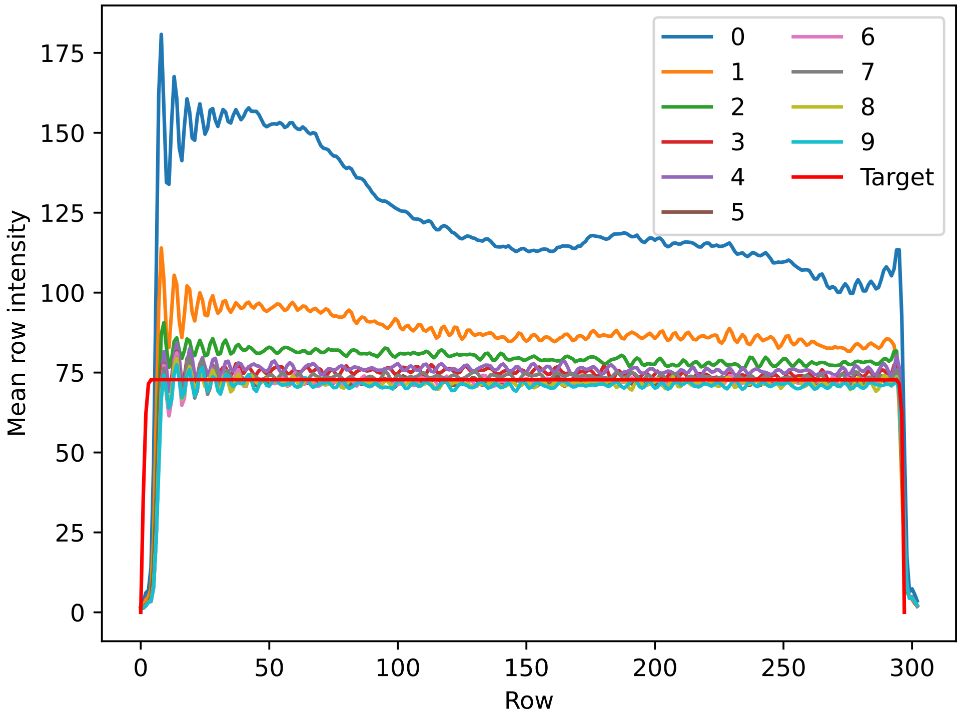


Figure 24: Average intensity of the rows of the displayed rectangle for the iterations zero (display) to nine (final corrected image). The intensity profile of the convoluted target pattern is also displayed as a reference.

The intensity profile clearly converges, Figure 24 shows the rows intensity average plotted over the rows number for different iteration numbers. It is also visible, that the profile smoothens a lot. In order to provide a more informative plot, because this method neglects the intensity distribution of the columns, a 2D plot for different iterations is shown in Figure 25. Iterations zero (the initial display), one and nine have been chosen, since the error and flatness, Figure 26, start to converge. Meaning for those iterations the evolution is visible best.

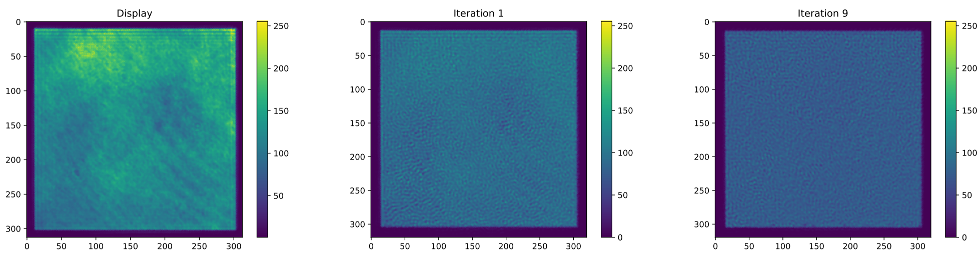


Figure 25: 2D plot of the intensity profile of the initially displayed 200 x 200 px rectangle, the initial display, first and the final correction image from left to right. The intensity level clearly converges, while also becoming more flat.

5.4 Characterizing the approximation of the potential

To characterize the approximation of the potential the error is calculated as rms

$$\epsilon_{RMS}[\%] = 100\sqrt{\frac{1}{IJ}\sum_{(i,j)}^{(I,J)}\left(\frac{T_{(i,j)} - D_{(i,j)}}{C}\right)^2}, \tag{5.5}$$

where T corresponds to the matrix of the target image, D corresponds to the data matrix of the recorded image, C corresponds to the difference between the minimum and the maximum intensity value of the recorded image. It is important here that just pixels contributing to the target are considered here, without the background of the image.

The length of the rows and columns of the turned on pixel values in the recorded image is given by I,J respectively.

The flatness

$$F[\%] = 100\left(1 - \sqrt{\frac{1}{IJ}\sum_{(i,j)}^{(I,J)}\left(\frac{P_{(i,j)} - M}{M}\right)^2}\right) \tag{5.6}$$

is a further tool to characterize how well the potential approximates the desired pattern, where

$$M = \sqrt{\frac{\sum_{(i,j)}^{(I,J)} P_{(i,j)}}{IJ}} \tag{5.7}$$

is the root mean intensity of the potential displayed and P corresponds to the matrix of the cropped target. The target is cropped in such a way that the edge's gradient isn't considered. [13]

Figure 26 shows the calculated error and flatness according to (5.5) and (5.6).

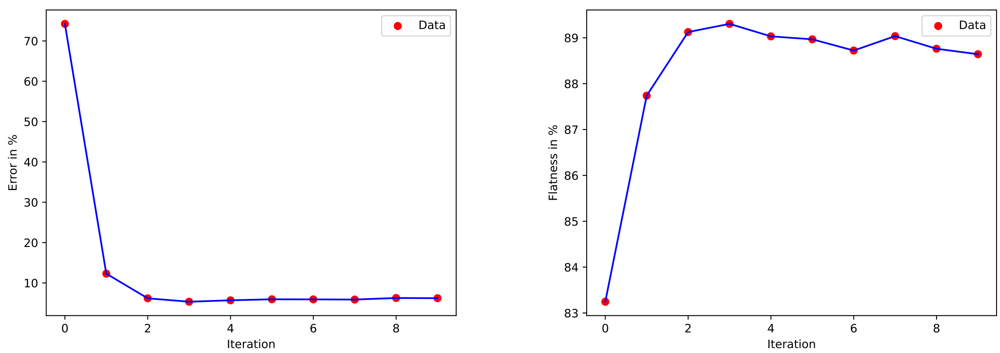


Figure 26: The error and flatness are calculated for each iteration according to (5.5) and (5.6) and plotted against the iteration at which the image was taken. The initial display is denoted with iteration zero, all in all ten iterations were performed. The error clearly converges, Kp=0.2 was chosen. The flatness also increases.

The error already starts to converge after iteration two, reaching a minimum value of $\epsilon_{min} = 5.34\,\%$ at iteration ten. The flatness doesn't reach as good values as the error, it's maximum is $F_{max} = 89.30\,\%$ at iteration three and decreases after that, converging to a value of $F = 88.5\,\%$. This is because the final array loaded onto the DMD has irregularly mirrors in the On and Off state compared to the initial loaded array where for the target pattern all mirrors are in the On state. This means the 'roughness' of the DMD's surface increases due to the correction, leading to speckles, Figure 27. This can be compensated a little bit with a larger PSF and will be further discussed in Chapter 5.8.1.

The error of the potentials can be treated as a perturbation, which has important consequences on the quantum gas regarding density modifications and energy scale changes, for further details see [32].

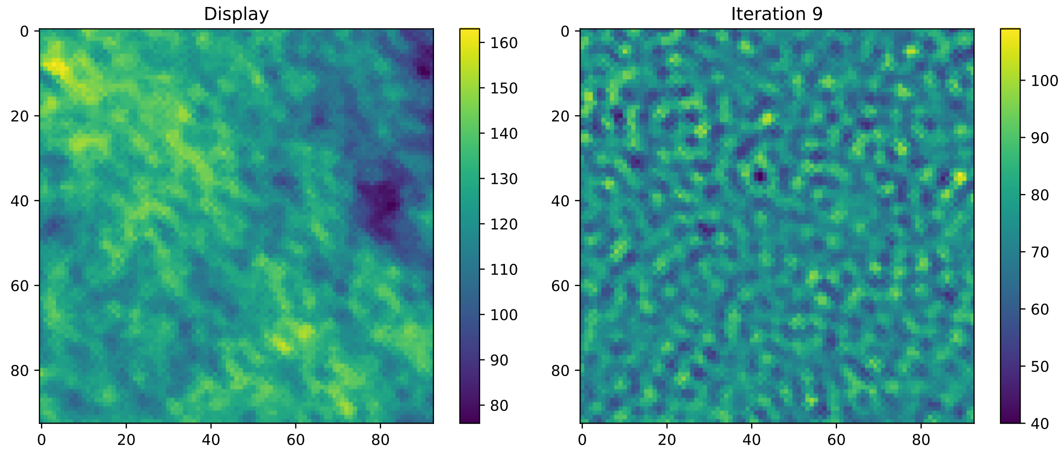


Figure 27: Zoom for the display and the final iteration of the 2D plot in Figure 25. On the left the zoomed display and on the right the zoomed final corrected image for iteration 9. The loop creates a speckle like image in the end, because the 'roughness' of the DMD increases due to mirrors being irregularly in the On and Off state in the corrected image.

General limitations of the optimization loop are dirt on the camera glass, mirrors, or other optical elements used, the crop and the precision of the rotation angle.

5.5 Optimization of Kp

The loop from section 4.1 yields the constant Kp that scales the calculated error.

The goal is to find the best possible Kp regarding the amount of iterations needed to achieve a good potential. Different Kp's result in varied reactions regarding the time needed to achieve a good corrected image as well as the convergence of the error and the flatness along the iterations.

This is the reason why the error is scaled and Kp is not just set equal to one, since a change on the DMD won't change the recorded image in the same way due to aberrations and PSF.

It is expected that higher Kp will be worse, because they will take more iterations to achieve a good image, since the error will likely oscillate.

The 200 x 200 px rectangle is displayed to the DMD, see Figure 21. The correction loop is performed for the values

$$\mathrm{Kp} \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0\}. \tag{5.8}$$

In the Appendix the Table 1 shows the error and flatness according to (5.5) and (5.6) for all the Kp values, the expected behaviour is verified.

When using lower Kps, the system converges slowly, while higher gains can lead to instability. It alternately overshoots (resulting in a density lower than the target) and undershoots (yielding a density higher than the target). The magnitude of these deviations from the target is roughly equal, which is why there isn't a significant impact on the flatness F.[30] The best compromise is to be found for Kp=0.2. This value will be used for coming measurements. Compared to the initial test in Figure 26 the intensity of the laser was turned a bit down to reduce the error of the initial display.

5.6 Optimization for a torus

In order to show that arbitrary traps can be displayed onto the DMD now a more complicated pattern is chosen to be optimized. A torus with outer radius r_{outer}=200 px and inner radius r_{inner}=130 px is loaded in the center of the DMD, Figure 28.

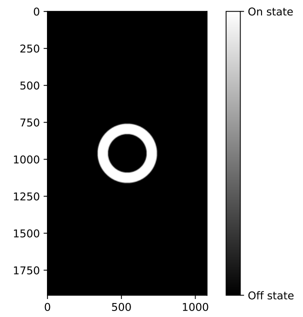


Figure 28: A torus with outer radius r=200 px and and inner radius r_{inner}=130 px is loaded in the center of the DMD.

The recorded image is shown in Figure 29.

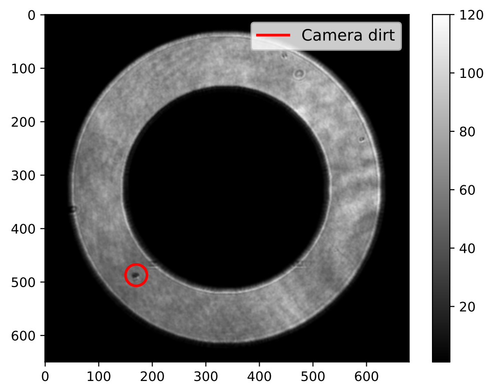


Figure 29: Recorded image for the initial display of the pattern in Figure 28. The red circle inside the area of the torrus marks a position where most likely camera dirt is located which is blocking off the light causing a higher error propagation since the optimization will constantly percieve this as an error in the actual image, rather than the recorded image limited by the glas infront of the camera sensor.

Like for the rectangle a good crop is crucial, the recorded image of the torus should ideally be cropped perfectly symmetrical. The torus brings some additional difficulties for the correction considering the noise in the background. Since it is not really possible to get rid of it long term by using the correction loop, the calculated error image has to be modified. A mask is used that sets all the values in the background outside of the torus shape to zero. The comparison between the normal error image and the cleaned error image is shown in Figure 30. For the two images on the right for the color map a clip has been performed, in order to see the effects of the mask more clearly.

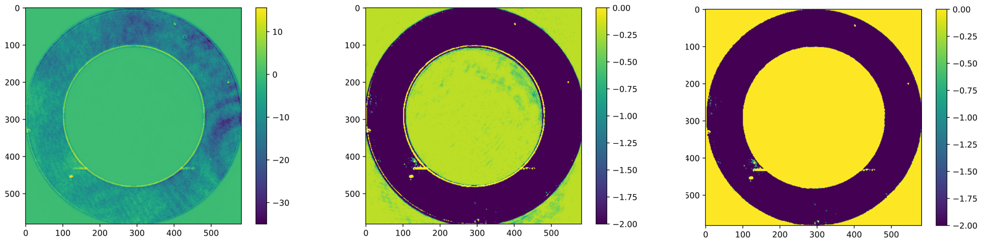


Figure 30: On the left: Calculated error image from the target image, and recorded image of the initial display, Figure 29. The error is scaled by Kp=0.2, since this is found out to be the best value in Chapter 5.5. The target image is the zoomed in part on the torus pattern in Figure 28 scaled by a value minP. In the middle the same error image is shown, this time with a changed colorbar ranging from -2 to 1 in order to see the background noise better. On the right the cleaned error image with the performed mask is shown. It uses the same colorbar as the image in the middle.

To prevent masking parts of the torus due to a slightly asymmetrical crop, the mask is performed for r_{outer}=201 px and r_{inner}=129 px.

The target has to match the recorded images size in pixels. Here the previous calculated demagnification, see Chapter 5.2, comes into play, since the target image has to match the recorded images size. To do so, the outer and inner radius of the torus for the target are multiplied with the magnification, it has to be multiplied by the pixel ratio of the DMD and the used camera. Alternatively one could also use r_{outer}=200 px and r_{inner}=130 px for the target and multiply the displayed pattern with the demagnification. Both methods are equal, one just has to make sure that the deflected image is large enough in order to prevent limitations due to the camera pixel size.

Figure 31 shows the intensity profile of the torus along each iteration.

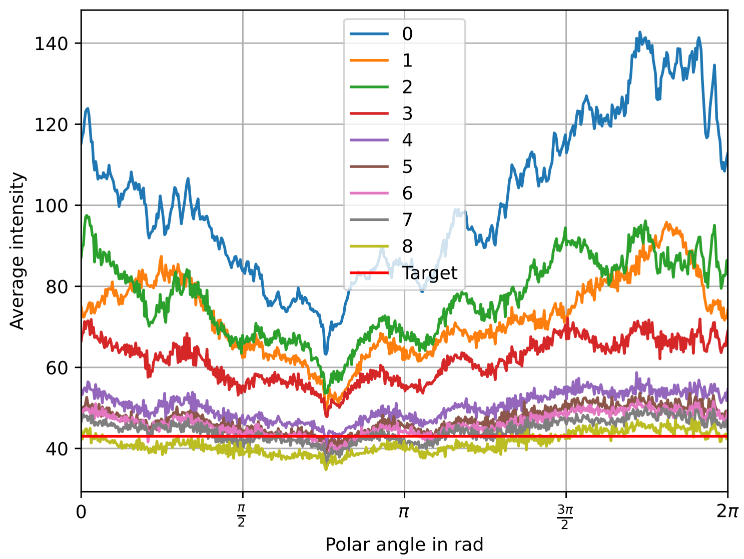


Figure 31: Intensity profile of the torus. For each iteration the average intensity is calculated along the polar angle of the torus. The intensity of the target is also plotted.

The error and flatness calculated according to (5.5) and (5.6) are shown in Figure 32. The converging error is very high, even at the 8 th iteration $\epsilon = 25\,\%$. The loop was stopped already at the 8 th iteration, because the image already started converging, but it looked like it wouldn't get significantly better.

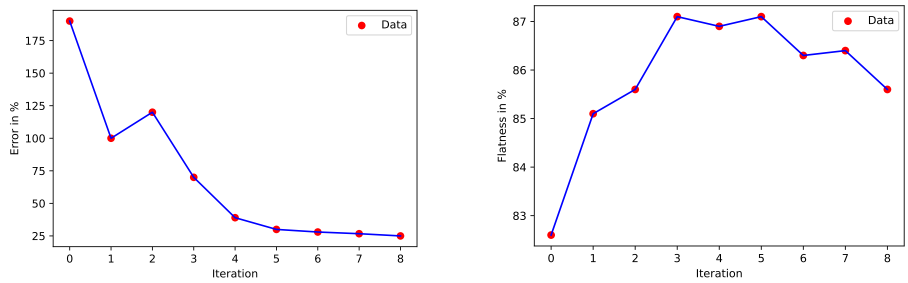


Figure 32: Error on the left and flatness on the right according to (5.5) and (5.6) for the torus, Figure 28. Kp=0.2.

5.7 Fluctuations

The error for the torus is most likely that high due to intensity fluctuations. Due to the lacking laser-power stabilization, there are large fluctuations regarding the intensity of the displayed image that can even be seen even by eye. To quantify the intensity fluctuations a deflected rectangle from the DMD is recorded with a camera for 5 minutes, because that corresponds to the order of magnitude of a measurement and this way also temperature fluctuations are taken into account. Approximately every 0.05 s the camera took a picture.

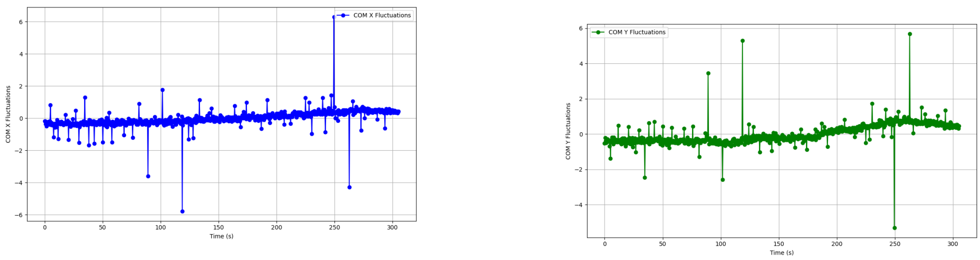


Figure 33: Fluctuation of the mean value of the first recorded image. The fluctuation takes into account the center of mass for the x and y coordinate according to (5.9).

The graph is shown in figure 33. The center of mass is calculated for the recorded images using the function `scipy.ndimage.center_of_mass()` from the Python library scipy.ndimage. It is calculated as

$$x_{com} = \frac{\sum_x \sum_y x \cdot I(x,y)}{\sum_x \sum_y I(x,y)} \quad y_{com} = \frac{\sum_x \sum_y y \cdot I(x,y)}{\sum_x \sum_y I(x,y)} \tag{5.9}$$

where *x(y)* represents the pixels along the columns(rows) and I represents the pixel intensity value. The images are cropped before calculating the center of mass to possibly disregard the effects of noise. Furthermore the deviation of the center of mass (COM) is plotted. As a reference value the average intensity of all of the COM values was used. A fluctuation of 6 px is very high, which explains why the loop converges to such a high error.

5.8 Intensity compensation

In an attempt to minimize the error a time averaging of the images recorded by the camera was tried, this way the intensity would be averaged over time and small fluctuations wouldn't come into weight that much. Since this method is not really promising, the best method found out is presented here.

To perform the optimization loop properly, an intensity compensation is implemented into it.

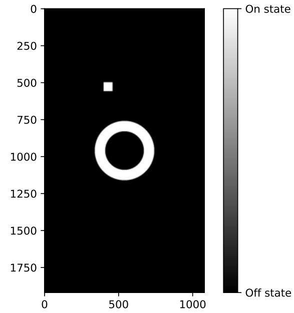


Figure 34: Torus with outer radius $r_{outer} = 200$ px and inner radius $r_{inner} = 130$ px in the center of the DMD. Next to the torus a small 60 x 60 px rectangle is displayed that will not be considered by the correction loop and acts as a reference for how the intensity changed.

This is done by adding a small rectangle to the displayed pattern that acts as a reference of the intensity fluctuation, see Figure 34. The torus will still be cropped out as usual to perform the optimization. The desired converging intensity is still defined by the minP value that scales the target patter.

The small rectangle is cropped seperately. The crop goes about 5 px beyond the rectangles edges because this way it is possible to leave the crop the same over the whole loop. Even for small position fluctuations the tight crop will work.

To compensate the intensity fluctuations the minP value has to be changed for each iteration. The average intensity of the small square in the inital display defines the reference intensity. If in the next iteration the intensity changed compared to the initial display, the initially defined minP value is also changes by the same amount. So for example the recorded images shall converge to an intensity of 80, meaning $minP_{display}$=80. For the display the average intensity of the small square is 90. For the first iteration it is 96, meaning a change in intensity of 6.7 %. So in the first iteration $minP_{iteration1} = 80 \cdot 1.067 = 85.36$. If for the second iteration the mean intensity of the small square is 90, $minP_{iteration2}$=$minP_{display}$.
This way the calculated error neglects the intensity fluctuation.

With that done, the loop as explained in Figure 10 can be performed.

The mean intensity converges more nicely, as depicted in Figure 35.

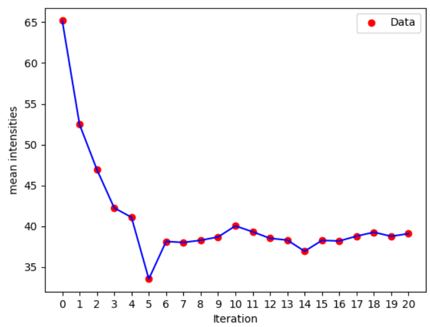


Figure 35: Mean intensities of the torus recorded for each iteration of the optimization loop. This plot shows the converging intensity using the compensation method.

The error and flatness obtained with the intensity compensation are depicted in Figure 36. The converging minimal error value now is 6.3 %, which is definetely an improvement compared to 25 % with the method without any compensation.

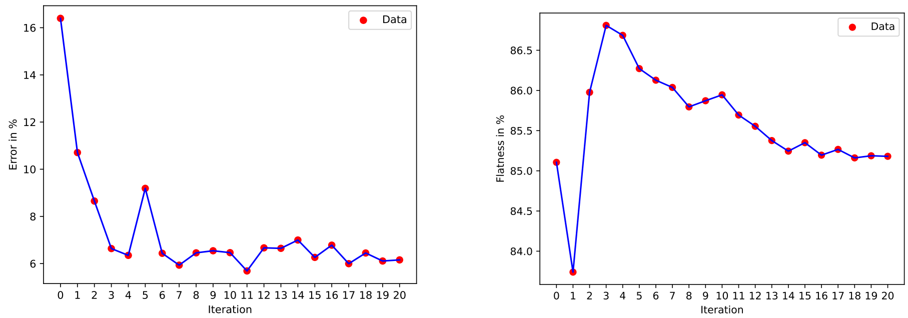


Figure 36: Error and flatness according to (5.5) and (5.6) for the torus, Figure 34, with implemented intensity compensation. Kp=0.2

But this method comes with a slight loss of flatness.

5.8.1 Optimization for the iris Aperture

The flatness of the image can be increased using an iris. Figure 37 shows the size of the PSF for different iris apertures.

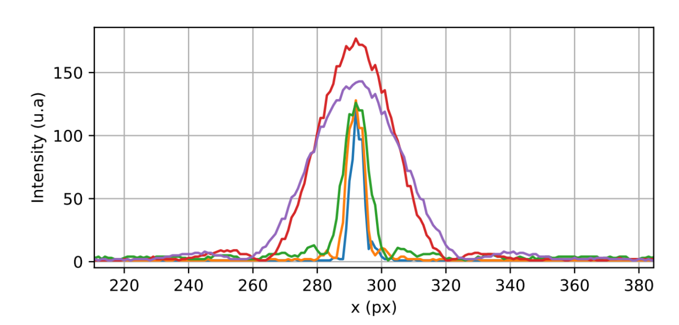


Figure 37: One pixel in the center of the DMD is turned On. The deflected image is recorded for different aperture sizes. The PSF extracted from the images is plotted for different iris aperture sizes. The PSF gets larger with smaller iris size, resulting in a more blurred and thus flatened image. For the blue curve the iris diameter is 12 mm. For the orange curve 9.2 mm, for the green 6 mm, for the purple 2.7 mm and for the red 0.2 mm.

This works, because the iris is positioned in the fourier plane. By closing the iris the high frequencies are removed and the image gets more blurry, increasing the flatness. Figure 38 shows the image at iteration 11 of the optimization loop using the intensity compensation with open, quarter, half and three quarter open iris. The effect is visible. The smoothness increases with the closing of the iris, but the blur also causes less sharp defined edges making it harder to correct the image, Figure 39.

The flatness for the 2D plot is calculated like this

$$\text{flatness image} = 1 - \frac{\text{recorded image} - \text{mean(polar image)}}{\text{mean(polar image)}}, \tag{5.10}$$

using Pythons library numpy [31]. An additional mask setting the background to zero after that calculation is performed.

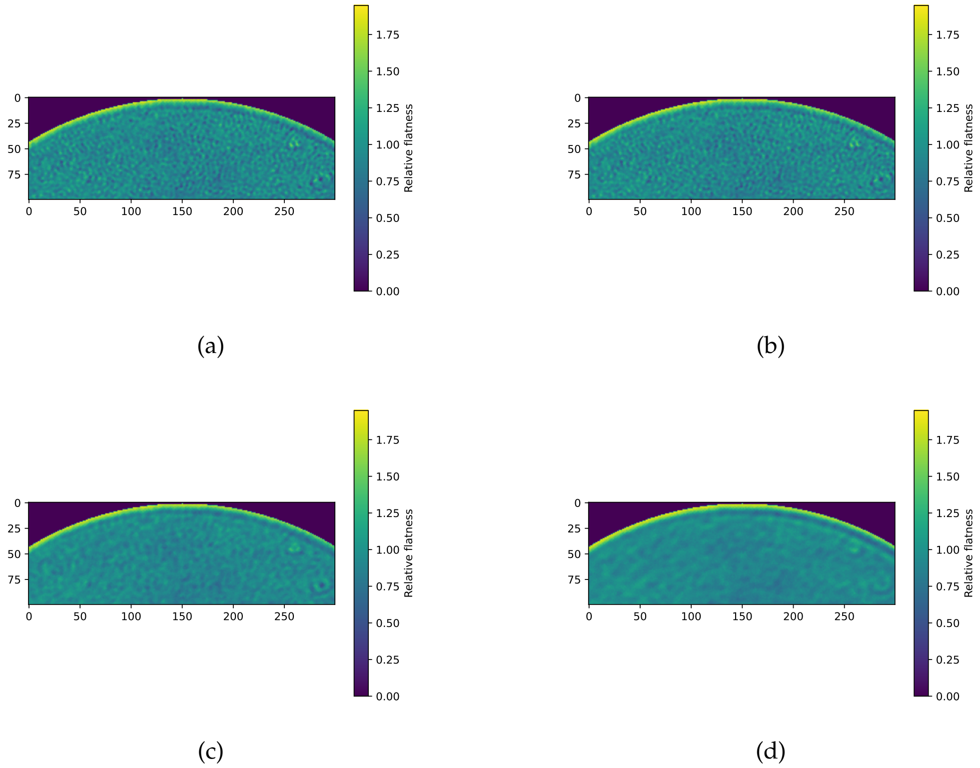


Figure 38: Zoomed in part of the tours. From top left to bottom right relative flatness according to (5.10) for iteration 11 of the optimization loop for fully open (a), quarter (b), half (c) and three quarter (d) open iris. Plots show the relative flatness, meaning an intensity of 1 is perfect.

A half closed iris is found to be the best, since the image is noticable less grainy compared to a fully open iris but it still offers a sufficient sharpness at the edges.

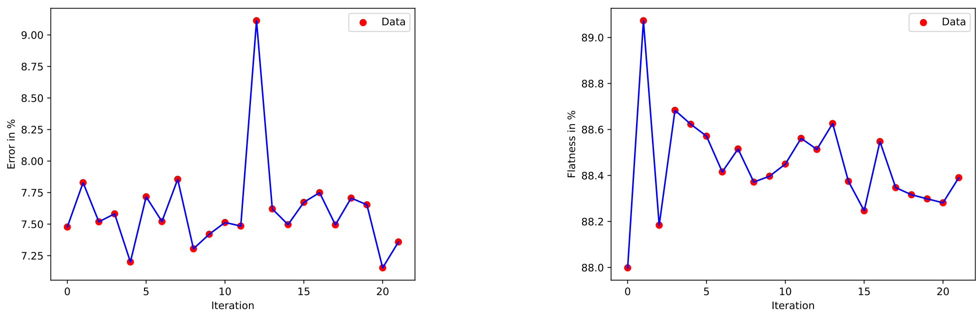


Figure 39: Error and flatness for a torrus for different iterations of the optimization loop with implemented intensity compensation and a quarter closed iris.

Regarding the flatness there is a small improvement, Figure 39, regarding the error it got slightly worse. Compared to Figure 36 the error is a bit lower at the start, because the laser intensity was scaled a bit lower and is converging to a value of ≈ 7 %, which is a bit higher than for the method without the iris, where the error was converging to ≈ 6 % , Figure 36. The raised error is due to the fact that because of the more closed iris the edges of the torus are also more blurred, causing a higher error.

It stands out, that the flatness is already better from the start with F=88 %, compared to F=85.1 % without the irirs, Figure 36. It oscillates between the value of 88 % and 89 %, overall converging to a higher value of 88.3 % compared to 85.4 % .

The tradeoff of the slightly higher error, but improved flatness seems good.

6 CONCLUSION AND OUTLOOK

In this thesis the first steps towards creating tailorable optical potentials using a digital micromirror (DMD) device were performed. The DMD was characterized. It's diffraction efficiency in this setup is ≈ 68 %, the estimated potential depth that can be reached is $T = 0.1\,\mu K$ An optimization loop was implemented in Python and it was shown that the loop works. The loop was tested first for a rectangle and optimized for the gain of the error Kp. The best value was found out to be Kp=0.2. The loop for the rectangle is converging to an error of $\epsilon \approx 6\,\%$ and flatness of $F\approx$ 88.7 %. After that it was performed for a torus, reaching a minimal error of $\epsilon \approx 25\,\%$ and a converging flatness of $F\approx$ 86 %. The fluctuations in the lab were quantified reaching 6 px of deviation from the center of mass. An intensity compensation for the torus was implemented to reduce the error. The new minimal error for the torus is $\epsilon \approx 6.3\,\%$ and a converging flatness of $F\approx$ 85.4 %. After that the setup was optimized for iris aperture, the best was found to be a half closed, which improved the flatness to a converging value of $F\approx$ 88.3 %. The iris improved the flatness, because it is placed in the Fourier plane, making the Point-Spread-Function larger and removing high frequencies, blurring the image more. The error is worse, because the aperture causes the edges of the torus to be less sharp.

The values are still not at the limit of achievable precision, regarding that errors of $\epsilon \approx 3\,\%$ and flatnesses of $F \approx 95\,\%$ can be reached [13]. For the flatness there is the most amount of possible improvement.

Nevertheless, the achieved values in this thesis come close. An AOM for better power stabilization will likely improve the results. Also an isolation to reduce air flow or temperature fluctuations will help get better results.

REFERENCES

[1] Axel Griesmaier et al. Bose-Einstein Condensation of Chromium. Physical Review Letters (2005). DOI https://doi.org/https://doi.org/10.1103/PhysRevLett.94.160401

[2] Matthias Wenzel. Macroscopic States of Dipolar Quantum Gases. PhD thesis. Universität Stuttgart, 2018.

[3] Fabian Böttcher et al. Transient supersolid properties in an array of dipolar quantum droplets. Physical Review X (2019). DOI https://doi.org/https://doi.org/10.1103/PhysRevX.9.011051

[4] L. Chomaz, I. Ferrier-Barbut, F. Ferlaino, B. Laburthe-Tolra, B. L. Lev, and T. Pfau. Dipolar physics: A review of experiments with magnetic quantum gases. Nov 2022

[5] Ashkin, A. (1997). Optical trapping and manipulation of neutral particles using lasers.

[6] Rudolf Grimm and Matthias Weidemüller, Yurii B. Ovchinnikov. Optical dipole traps for neutral atoms. DOI https://arxiv.org/pdf/physics/9902072

[7] Prof. Jean Dalibard. Collège de France lectures, Optical lattices, 2013. DOI https://pro.college-de-france.fr/jean.dalibard/index_en.html

[8] R.W. Floyd, L. Steinberg. An Adaptive Algorithm for Spatial Gray Scale. In Proceedings of SID (Society for Information Displays) – Digest of technical papers, pp. 36–37. 1975.

[9] Gitproject Python control module for Vialux DMDs based on ALP4.X API. DOI https://github.com/wavefrontshaping/ALP4lib, 16.07.24

[10] Vialux. Superspeed V models productdescription. DOI https://www.vialux.de/en/superspeed-v-modules.html

[11] Gitproject to the code for the optimazation loop https://github.com/natschoes/DMD/tree/main

[12] Einstein picture, Badische Zeitung https://www.badische-zeitung.de/panorama/einstein-archiv-nun-im-internet--57242520.html 04.07.2024

[13] Elia Perego. Generation of arbitrary optical potentials for atomic physics experiments using a Digital Micromirror Device. University of Florence. Master thesis.

[14] Thibault Bourgeois. Tailorable optical trap with Digital Micromirror Device for Dysprosium experiment. Physikalisches Institut Universität Heidelberg. Bachelor's thesis.

[15] Leerin Perumal, Andrew Forbes. J. Opt. 25 074003. 2023. DOI https://iopscience.iop.org/article/10.1088/2040-8986/acd563/pdf

[16] Smith, W. J. (2007). Modern Optical Engineering (4th ed.). McGraw-Hill Professional.

[17] Thorlabs https://www.thorlabs.com/newgrouppage9.cfm?objectgroupid=9026

[18] Robert K Tyson. Fresnel and Fraunhofer diffraction and wave optics. IOP Publishing, 2014. DOI https://doi.org/10.1088/978-0-750-31056-7ch3.

[19] C. J. Foot, et al. Optical Dipole Traps for Neutral Atoms. Reports on Progress in Physics

[20] J. H. Becher, S. Baier, K. Aikawa, M. Lepers, J.-F. Wyart, O. Dulieu, and F. Ferlaino. Anisotropic polarizability of erbium atoms. Phys. Rev. A, 97 :012509, Jan 2018

[21] Hui Li, Jean-François Wyart, Olivier Dulieu, and Maxence Lepers. Anisotropic optical trapping as a manifestation of the complex electronic structure of ultracold lanthanide atoms : The example of holmium. Phys. Rev. A, 95 :062508, Jun 2017.

[22] Xiuli Jiang, Zhengtian Gu, Li Zheng. Cladding modes in photonic crystal fiber: characteristics and sensitivity to surrounding refractive index. DOI https://www.spiedigitallibrary.org/journals/optical-engineering/volume-55/issue-1/017106/Cladding-modes-in-photonic-crystal-fiber--characteristics-and-sensitivity/10.1117/1.OE.55.1.017106.full, 12.07.24

[23] Jackson, J. D. (1999). Classical Electrodynamics (3rd ed.). Wiley

[24] Popoff, S. M., Bromberg, Y., Matthès, M. W., Gutiérrez-Cuevas, R. (2023). A Practical Guide to Digital Micro-mirror Devices (DMDs) for Wavefront Shaping. arXiv Preprint, arXiv:2311.17496. DOI https://arxiv.org/pdf/2311.17496

[25] Gaetan Clément. Installation and characterization of a digital micromirror device on a quantum gas experiment. Physikalisches Institut 5, Universität Stuttgart. Internship report.

[26] Mohammadamin Tajik, Bernhard Rauer, Thomas Schweigler, Federica Cataldini, João Sabino, et al.. Designing arbitrary one-dimensional potentials on an atom chip. Optics Express, 2019, 27 (23), pp.33474-33487. 10.1364/OE.27.033474. hal-02402473. DOI https://hal.sorbonne-universite.fr/hal-02402473v1/file/oe-27-23-33474.pdf

[27] Python. Pillow library https://pypi.org/project/pillow/

[28] Philipp Ilzhöfer. Creation ofDipolar Quantum Mixturesof Erbium and Dysprosium. PhD thesis. University of Innsbruck, 2020.

[29] Damien Bloch, Britton Hofer, Sam R. Cohen, Maxence Lepers, Antoine Browaeys, and Igor Ferrier-Barbut1. Université Paris-Saclay, Institut d'Optique Graduate School, CNRS, Laboratoire Charles Fabry, 91127, Palaiseau, France & Laboratoire Interdisciplinaire Carnot de Bourgogne, CNRS, Université de Bourgogne, 21078 Dijon, France. Anisotropic polarizability of Dy at 532 nm on the intercombination transition. DOI https://arxiv.org/pdf/2404.10480

[30] Édouard Le Cerf. Demixing phenomena in 2D Bose gases. Physics [physics]. Sorbonne Université, 2020. English. NNT : tel-03019926v2

[31] Python. Numpy library. DOI https://numpy.org/

[32] Thibault Bourgeois, Lauriane Chomaz. Physikalisches Institut, Universität Heidelberg & Département de Physique, Ecole Normale Supérieure, Paris. How is the density of quasi-two-dimensional uniform dipolar quantum Bose gases affected by trap imperfections? March 8, 2024

[33] Edmund optics. BFS-U3-120S4M-CS USB 3.1 Blackfly® S, Monochrome Camera. https://www.edmundoptics.com/p/BFS-U3-120S4M-CS-USB3-Blackflyreg-S-Monochrome-Camera/40171

[34] xgimi. Understanding the Functions and Importance of DMDs in Smart Projectors. https://us.xgimi.com/blogs/projectors-101/dmd-chips-in-dlp-smart-projectors

[35] Nir Navon, Robert P. Smith, Zoran Hadzibabic. Department of Physics, Yale University, Clarendon Laboratory, University of Oxford, Cavendish Laboratory, University of Cambridge. Quantum Gases in Optical Boxes. DOI https://arxiv.org/pdf/2106.09716

APPENDIX

Finished setup

The setup planned to be implemented in the experiment, Figure 40, is set up already. The beginning is the same as in Figure 11. The camera is placed after another telescope, consisting of f3=500 mm and f4=100 mm. Due to space issues it is not possible to place the camera after f2=100 mm by using a beam shaper for example. The next demagnification stage for the experiment consists of the lenses f3=500 mm and f4=100 mm. The demagnification for the camera was chosen smaller in order to avoid limitations due to the cameras pixelsize and still be able to get a good resolution, which leads to better results regarding the potentials optimization. A lower demagnification would be better, but was again not possible due to space issues. Because the beam is splitted with a cube after the f3=500 mm lens and lead to the camera another $\frac{\lambda}{2}$ waveplate was added to maintain polarization. The total demagnification then is 1/75.

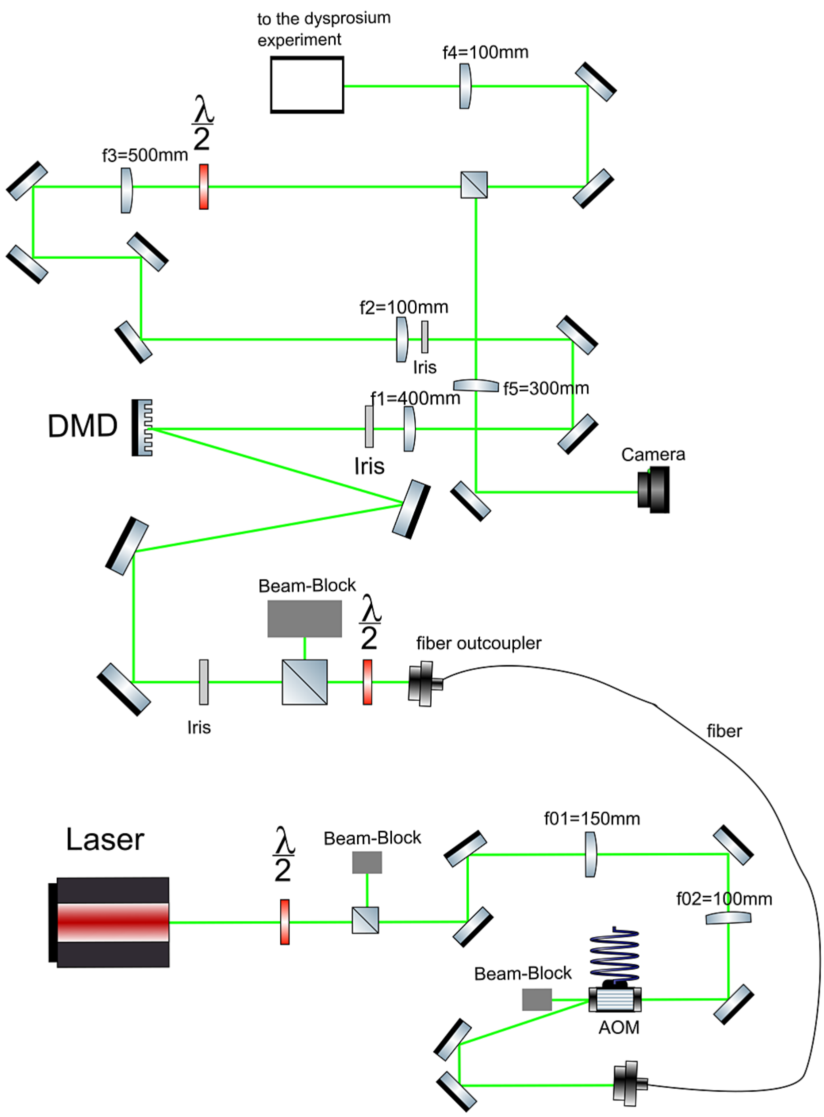
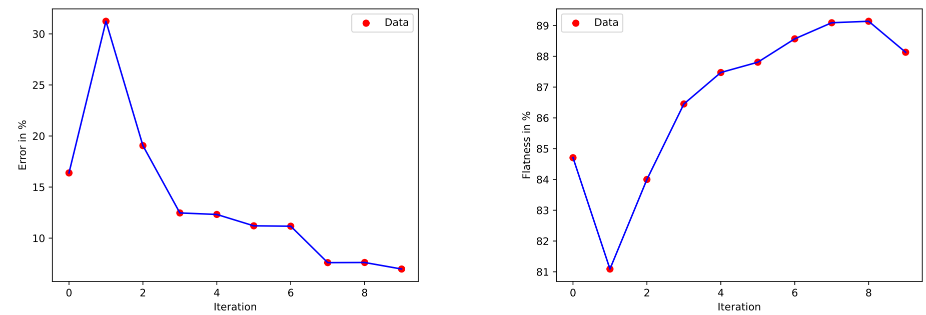


Figure 40: Final setup intended to be implemented into the experiment.

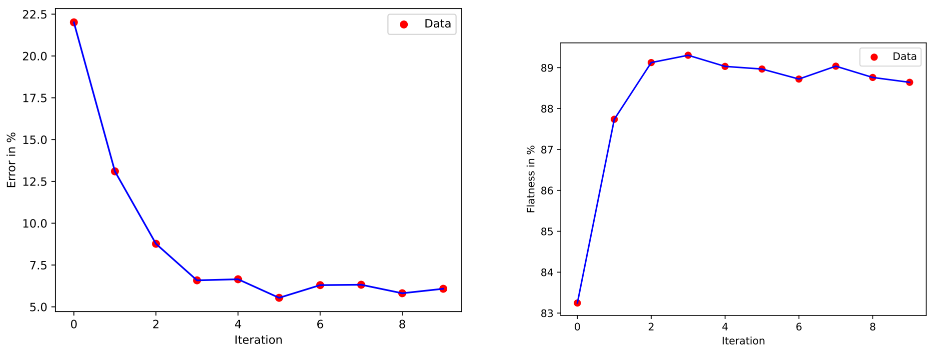
Plots for optimization of Kp

Table 1: The error and flatness are calculated for each iteration according to (5.5) and (5.6) and plotted against the iteration at which the image was taken. The initial display, Figure 21, is denoted with iteration zero, all in all ten iterations are performed. This is done for different Kp values (5.8) and a constand minP value. The Kp value corresponding is written above each pair of figures.

Kp=0.1



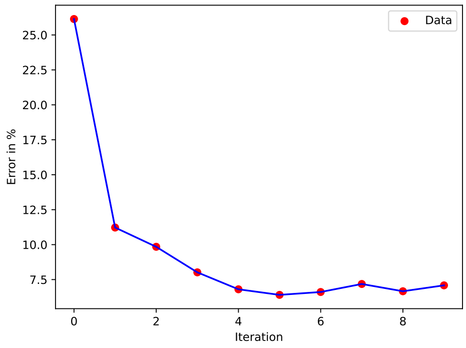
Kp=0.2

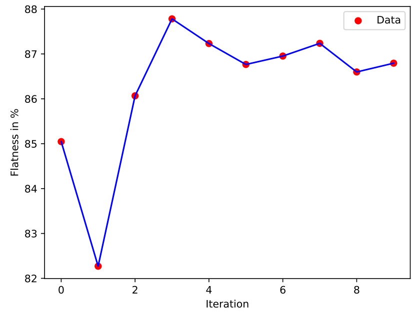


Continued on next page

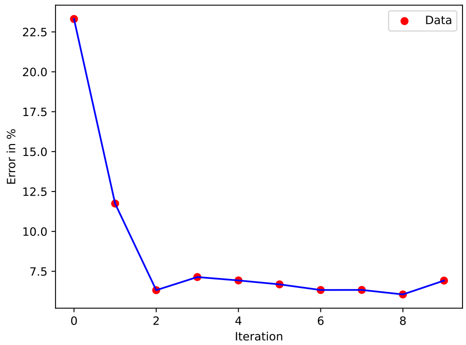
Table 1: (Continued)

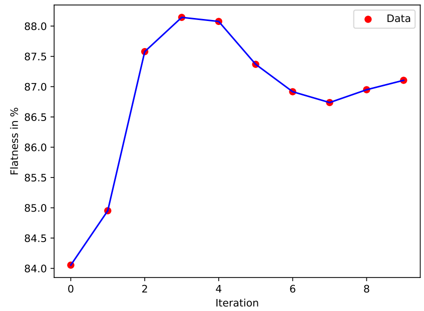
Kp=0.3



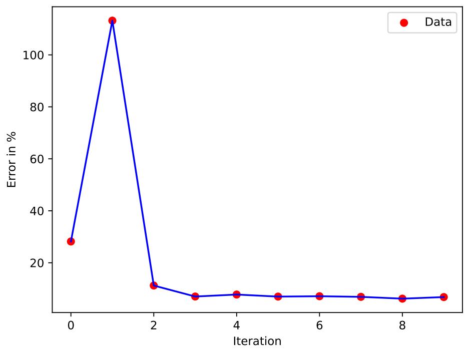


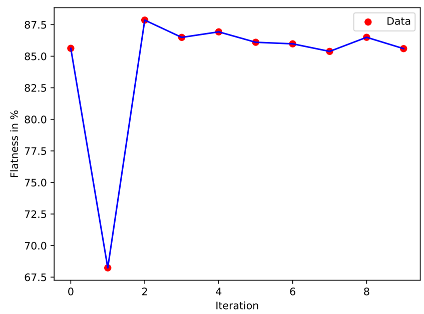
Kp=0.4



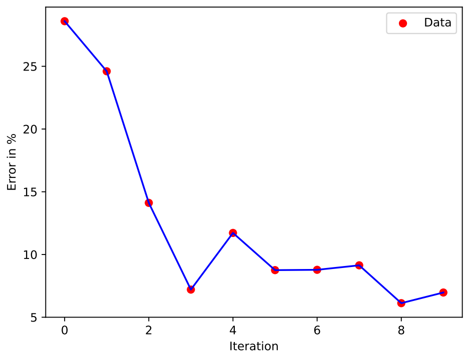


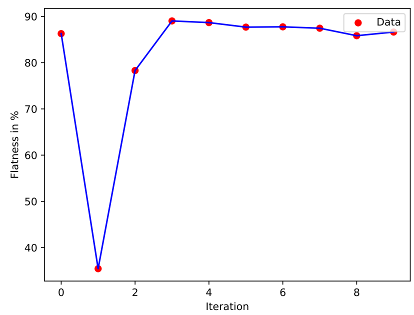
Kp=0.5





Kp=0.6

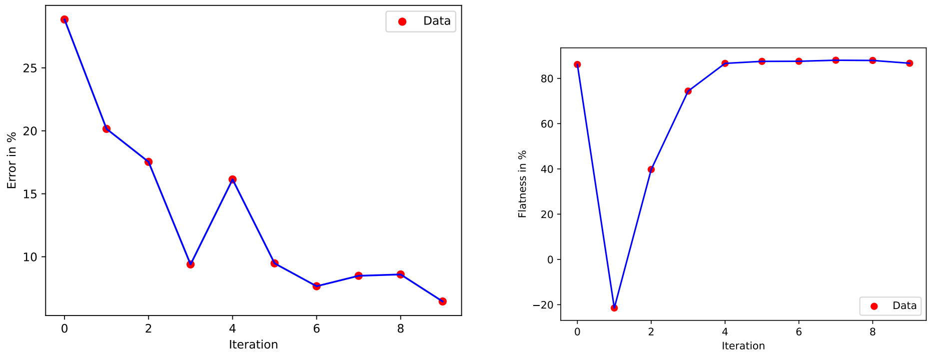




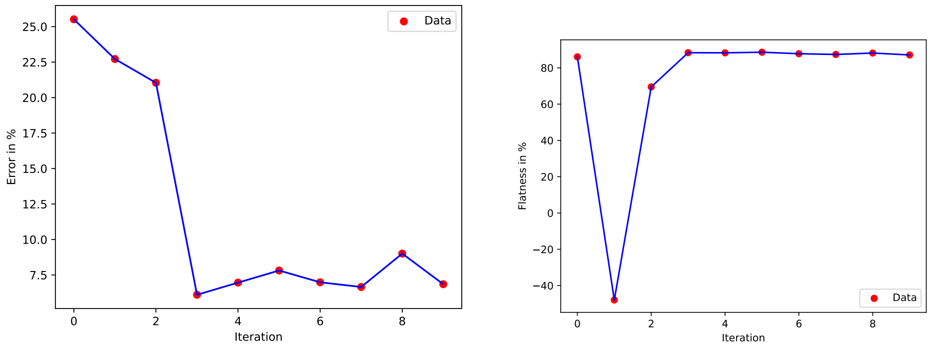
Continued on next page

Table 1: (Continued)

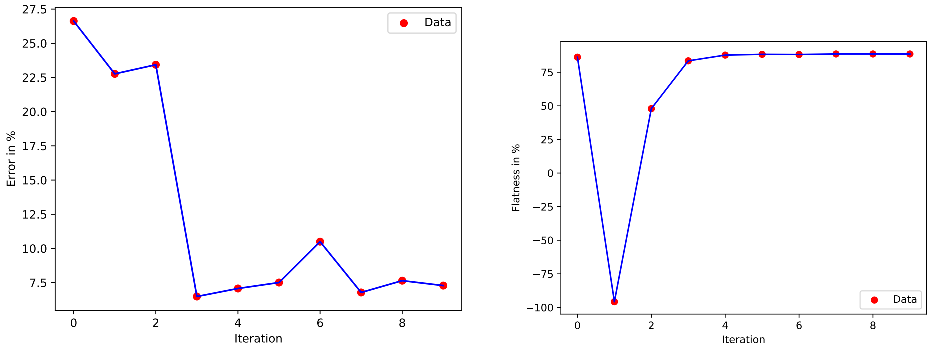
Kp=0.7



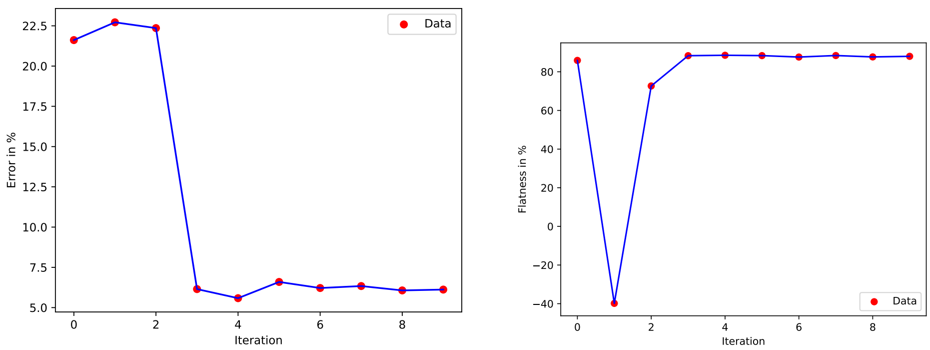
Kp=0.8



Kp=0.9



Kp=1.0



Python Code

In the following the python code of the main file, see Listing 1 as well as the defined functions, see Listing 2 that are imported in the main file are shown.

Listing 2: Main file DMD.py. Python code that was used to display and correct the different patterns. This version of the code isn't automated, since the setup used for this thesis isn't very stable in position meaning the crop of the images has to be adjusted by hand over time since automated methods lack in precision.

```

############HEADER##################################################
import PIL
import numpy as np
import scipy
from scipy.ndimage import rotate
import matplotlib.pyplot as plt
import PIL.Image as Image
import time
import sys
import matplotlib as mpl
from ALP4 import *
from PIL import Image
from PIL import Image, ImageDraw
import os
from scipy.optimize import curve_fit
import scipy
import cv2
import threading
import time

#package needed to access camera. the package is distributed by FLIR. please look into
     README for installation guidance
from pyspin import PySpin
from Definition import Constants as const
from Definition import Functions as func
from Definition import access_camera as access_c



# Main code

if __name__ == "__main__":

    ######IMPORT CONSTANTS############################################
    #If you want to adjust them, please look into Definition\Constants.py

    #constants for DMD
    display_time = const.display_time
    display_time_iterate = const.display_time_iterate
    mirror_state= const.mirror_state
    mirror_size = const.mirror_size        # = a
    Nx = const.Nx
    Ny = const.Ny
    pixel_number = (Nx, Ny)

    Demagnification=const.Demagnification
    magnification = const.magnification
    Rotation_angle=const.Rotation_Angle
    ratio_px=const.ratio_px

    #import images
    image_path = const.image_path
    img_path_old = const.img_path_old

    #constants for calculation of the error
    minP=const.minP
    Kp=const.Kp

    #constants for calculation of convolution with the PSF
```

```
newsize=const.newsize
sigmax=const.sigmax
sigmay=const.sigmay
size_of_resize=const.size_of_resize

#constants for torrus
outer_radius = const.outer_radius
inner_radius = const.inner_radius

#constants to save plots
save_folder = const.save_folder
folder_path = const.folder_path

#constants to calculate plots
minP_scaled=const.minP_scaled
num_angles = const.num_angles
num_samples = const.num_samples

#constants for cropping boundaries of the recorded image
top_exp = const.top_exp
bottom_exp = const.bottom_exp
left_exp = const.left_exp
right_exp = const.right_exp

#constants for cropping boundaries of the target image
top_target = const.top_target
bottom_target = const.bottom_target
left_target = const.left_target
right_target = const.right_target

#constants to crop polar transformed image [:,269:413]
min_r = const.min_r
max_r = const.max_r

#####constants for the camera
exposure = const.exposure
gain = const.gain

#constant for amount of iterations
number_of_iterations = const.number_of_iterations

######START #################################################################################

####Creating of the target pattern###################
img_rect = func.rectangle(pixel_number, (960,540), 200, 200, a=mirror_state)
img_torrus = func.torrus(outer_radius, inner_radius)

#####LOAD IMAGES, crop and rotate them accordingly#####
im_array_cropped = func.preprocess_recorded_image(image_path, top_exp, bottom_exp, left_exp, right_exp, Rotation_angle)
img_exp=im_array_cropped
#im_array_cropped=im_array[1320:1780,1175:1635] #for points magni
intensity_value = im_array[1144:1215,1444:1512]
#outcomment for display
intensity_value_previous = Image.open(r"D:\dmd thesis\FS testing\torrus\Iris\use filter\0.bmp").rotate(Rotation_angle)
im_array_intensity=np.transpose(np.array(intensity_value_previous))
intensity_value_previous = im_array_intensity[1144:1215,1444:1512]


def calculate_minP():
    minP=80
    reference_intensity_value_previous = np.mean(intensity_value_previous)
    reference_intensity_value = np.mean(intensity_value)
    print("previous: ", reference_intensity_value_previous, "now: ", reference_intensity_value)
    if reference_intensity_value==reference_intensity_value_previous:
        print("No intensity fluctuation")
    else:
        ratio = reference_intensity_value/reference_intensity_value_previous
```

```
            minP=minP*ratio
            print("fluctuation of: ", ratio)
        return minP

    minP=calculate_minP()
    print("minP: ", minP)

    img_old=Image.open(img_path_old)


    ######DEFINE REFERENCE PATTERN the error calculation is based on#########
    #use this for rectangle
    '''
    rect_size=im_array_cropped.shape[0]
    pixel_number_target=[rect_size,rect_size]
    print("pixel_number_target", pixel_number_target)
    center=[rect_size//2,rect_size//2]
    target_img=minP*func.rectangle(pixel_number_target,center, rect_size//2, rect_size//2, a=mirror_state)
    '''
    torrus_resized = func.torrus(outer_radius*magnification, inner_radius*magnification)
    target_img = minP*torrus_resized[top_target:bottom_target,left_target:right_target]#[669:1251,249:831]

    #for display and 1st iteration img_old=target_img, then please outcomment.
    img_old=target_img

    ##################### PID LOOP and CREATION OF THE NEW IMAGE ################
    error=Kp*(target_img-img_exp)
    #use this only for torrus
    error = func.filter_background_noise_for_torrus(error, inner_radius, outer_radius, magnification)

    new_img=img_old+error
    #img_close() needs to be OUTCOMMENTED for display and first iteration. Otherwise you will get an error.
    #img_old.close()

    ###################### CONVOLUTION WITH THE PSF ##################
    #convolution of image with added error with psf
    newarray=func.convolve_and_save_new_image(new_img, newsize, sigmax, sigmay, size_of_resize)

    ############## CREATION OF THE NEW FLOYD STEINBERG IMAGE ####################
    img_FS=func.create_floyd_steinberg_image(newarray,Nx,Ny)

    #####DISPLAY ON THE DMD###############################################
    #func.display_DMD([img_torrus], nbImg = 1, display_time = display_time)
    #func.display_DMD([img_rect], nbImg = 1, display_time = display_time)
    func.display_DMD([img_FS], nbImg = 1, display_time = display_time)
```

Listing 3: Functions.py file. Python code where all functions are defined for importation in the DMD.py file. Furthermore the functions used to plot the data, calculate the errors and more are also defined here.

```

import PIL
import numpy as np
import scipy
from scipy.ndimage import rotate
import matplotlib.pyplot as plt
import PIL.Image as Image
import time
import sys
#from PIL import image
import matplotlib as mpl
from ALP4 import *
from PIL import Image
import cv2
import os
```

```python
import os
from scipy.optimize import curve_fit
import scipy
import cv2

#Functions
def plot_intensity(image_paths,Rotation_angle):
    plt.figure()
    i=0
    for image_path in image_paths:
        img = Image.open(image_path).rotate(Rotation_angle)
        img_gray = img.convert("L")
        img_array = np.array(img_gray)
        row_intensity = np.mean(img_array, axis=1)
        plt.plot(row_intensity, label=i)
        i+=1
    plt.xlabel("Rows")
    plt.ylabel("Intensity")
    plt.title("Intensity of Rows")
    plt.legend()
    plt.show()


def display_DMD(img, nbImg, display_time):
    # Load the Vialux .dll
    DMD = ALP4(version = '4.3',libDir="./ALP-4.3 API")
    # initialize the device
    DMD.Initialize()
    # Binary amplitude image

    # Quantization of the image between 1 (on/off) and 8 (256 pwm grayscale levels).
    bitDepth = 1
    # WARNING even for a boolean quantization, the values readed are 0 and 255.
    # Array generation

    imgAff = img[0].ravel(order='F')*255 # Modification to a 1D array for the DMD

    if nbImg>1: #Array generation for a sequence with more than 1 image
        for i in range(nbImg-1):
            np.concatenate([imgAff,(img[i+1].ravel(order='F')*255)])

    # Allocate the onboard memory for the image sequence
    # you can load many pictures change only nbImg
    seq1 = DMD.SeqAlloc(nbImg, bitDepth = bitDepth)

    # Send the image sequence as a 1D list/array/numpy array
    # enter the images of the sequences

    DMD.SeqPut(imgData =imgAff.ravel(order='F'))

    DMD.SeqControl(controlType=2104 , value=2106) # activation off the uninterrupted mode

    # set image rate to 50 Hz so period of 2e4  s
    DMD.SetTiming(pictureTime = 2000)

    # Run the sequence in an infinite loop
    DMD.Run(loop=True)

    ###there is two method for stopping the DMD: after a chosen time of after pressing enter. Only one method work when the programm is running ( not in comment )
    time.sleep(display_time) # stop the programm for the display time (in seconds) but let the DMD running his sequence

    #input("Press enter to stop the DMD") # stop the sequence display after pressing enter
    # stop the sequence display
    DMD.Halt()
    # Free the sequence from the onboard memory
    DMD.FreeSeq()
    #De-allocate the device
    DMD.Free()
    print("END of display")
```

```
    return


### Functions creating different images

def cross(NumberPixel, ligne_croix, colonne_croix, epaisseur_ligne, epaisseur_colonne):
    ##creates a cross on the DMD
    img = np.zeros([NumberPixel[0], NumberPixel[1]], dtype=int)

    # horizontal line
    img[int(ligne_croix - epaisseur_ligne/2) : int(ligne_croix + epaisseur_ligne/2), :] =
        1

    # vertical line
    img[: , int(colonne_croix - epaisseur_colonne/2) : int(colonne_croix +
        epaisseur_colonne/2)] = 1
    return img



def uniform(NumberPixel, a):
    """ Creates a uniform image on the DMD (a = 0 or 1) """
    img = np.zeros([NumberPixel[0], NumberPixel[1]], dtype=int)
    img[:,:] = a
    return img



def k_pixels(NumberPixel, k, a, position_ligne, debut_trait):

    """
    Creates an image with k pixels equal to a (a = 0 or 1)

    line_position is the line on which the few pixels will be changed

    start_trait is the column where the stroke begins
    """

    # Line smaller than DMD

    assert debut_trait + k <= NumberPixel[1]

    # Some black pixels on a white background

    if a==0:
        img = np.ones([NumberPixel[0], NumberPixel[1]], dtype=int)
        img[position_ligne, debut_trait : debut_trait + k] = 0

    # A few white pixels on a black background

    else:
        img = np.zeros([NumberPixel[0], NumberPixel[1]], dtype=int)
        img[position_ligne, debut_trait : debut_trait + k] = 1

    return img


def point(NumberPixel):
    img = np.zeros([NumberPixel[0], NumberPixel[1]], dtype=int)
    img[960,540]=1
    return img


def points_magnification(NombrePixel):
    img = np.zeros([NombrePixel[0], NombrePixel[1]], dtype=int)
    img[960, 540] = 1
    img[961, 540] = 1
    img[960, 541] = 1
    img[961, 541] = 1

```

```python

    # Create a 14x14 rectangle centered around (1256, 540)
    start_x = 1256 - 4  # 7 pixels left of 1256
    end_x = 1256 + 4  # 7 pixels right of 1256
    start_y = 540 - 4  # 7 pixels above 540
    end_y = 540 + 4  # 7 pixels below 540

    img[start_x:end_x + 1, start_y:end_y + 1] = 1

    img[960, 840] = 1
    img[961, 840] = 1
    img[961, 841] = 1
    img[960, 841] = 1
    return img



def rectangle(NumberPixel, centre, widthdividedbytwo, lengthdividedbytwo, a):

    """Creates a rectangle with the value a (a = 0 or 1) on the DMD.

    center must be of the form (center_row, center_column)

    Attention, NumberPixel = (number of column, number of row)

    """
    # Rectangle smaller than the DMD

    assert centre[1] - widthdividedbytwo >= 0

    assert centre[1] + widthdividedbytwo <= NumberPixel[1]

    assert centre[0] - lengthdividedbytwo >=0

    assert centre[0] + lengthdividedbytwo <= NumberPixel[0]

    if a == 0:
        img = np.ones([NumberPixel[0], NumberPixel[1]], dtype=int)
        for ligne in range(centre[0] - lengthdividedbytwo, centre[0] + lengthdividedbytwo):
            img[ligne, centre[1] - widthdividedbytwo : centre[1] + widthdividedbytwo ] = 0

    else :
        img = np.zeros([NumberPixel[0], NumberPixel[1]], dtype=int)
        for ligne in range(centre[0] - lengthdividedbytwo, centre[0] + lengthdividedbytwo):
            img[ligne, centre[1] - widthdividedbytwo : centre[1] + widthdividedbytwo ] = 1

    return img




def display_torus_and_rectangle(NumberPixel, outer_radius, inner_radius, rectangle_size):
    # Create a torus
    torus_img = torrus(outer_radius, inner_radius)

    # Define the rectangle parameters
    centre = (rectangle_size[0] // 2 + 500, rectangle_size[1] // 2 + 400)  # Center of rectangle
    widthdividedbytwo = rectangle_size[1] // 2
    lengthdividedbytwo = rectangle_size[0] // 2
    a = 1  # Rectangle value

    # Create a rectangle in the top-left corner
    rectangle_img = rectangle(NumberPixel, centre, widthdividedbytwo, lengthdividedbytwo, a)

    # Combine the images (together, torus + rectangle)
    combined_img = torus_img + rectangle_img

```

```
    # Display the combined image
    plt.imshow(combined_img, cmap='gray')
    plt.title('Torus and Rectangle in Top-Left Corner')
    plt.axis('off')
    plt.show()

    return combined_img



def circle(r):
    img = np.zeros([1920, 1080], dtype=int)
    for x in range(1920):
        for y in range(1080):
            if((x-960)**2+(y-540)**2)<(r*r):
               img[x,y]=1
    return img


def torrus(outer_radius, inner_radius):
    return (circle(outer_radius)-circle(inner_radius))>0.5


def sort_bmp_files(folder_path):
    # List all BMP files in the directory
    bmp_files = [f for f in os.listdir(folder_path) if f.endswith('.bmp')]

    # Sort the files based on the integer part of the filename
    sorted_bmp_files = sorted(bmp_files, key=lambda x: int(os.path.splitext(x)[0]))

    return sorted_bmp_files


def plot_intensities_torrus(bmp_files, folder_path, Rotation_angle, min_r,max_r, top_exp, bottom_exp, left_exp, right_exp):
    plt.figure()
    i=0
    for bmp_file in bmp_files:
        # Load the image
        img_path = os.path.join(folder_path, bmp_file)
        image = Image.open(img_path).convert('L').rotate(Rotation_angle)  # Convert to grayscale and rotate
        img_array = np.transpose(np.array(image))[top_exp:bottom_exp,left_exp:right_exp]

        #--- the following holds the square root of the sum of squares of the image dimensions ---
        #--- this is done so that the entire width/height of the original image is used to express the complete circular range of the resulting polar image ---
        value = np.sqrt(((img_array.shape[0]/2.0)**2.0)+((img_array.shape[1]/2.0)**2.0))
        polar_image = cv2.linearPolar(img_array,(img_array.shape[0]/2, img_array.shape[1]/2), value, cv2.WARP_FILL_OUTLIERS)
        polar_image = polar_image.astype(np.uint8)[:,min_r:max_r]
        '''
        plt.figure(1)
        plt.imshow(polar_image)
        plt.colorbar()
        plt.title(f"polar transformed image for {bmp_file}")
        plt.show()

        polar_image_converted=Image.fromarray(np.uint8((np.array(polar_image))),mode="L")
        polar_image_converted.save(f"polar_image{bmp_file}")
        '''
        row_intensity = np.mean(polar_image, axis=1)
        num_rows = polar_image.shape[0]
        angles = np.linspace(0, 2 * np.pi, num_rows)
        plt.plot(angles, row_intensity, label=bmp_file)
        i+=1
    plt.xticks([0, np.pi/2, np.pi, 3*np.pi/2, 2*np.pi], ['0', r'$\frac{\pi}{2}$', r'$\pi$', r'$\frac{3\pi}{2}$', r'$2\pi$'])
    plt.ylabel('Average intensity')
    #plt.title(f'Average Intensity')
```

```
    plt.grid(True)
    plt.legend()
    plt.show()


def plot_from_conv(errors,iterations,ylabel):
    # Ensure iterations and errors_final have the same length
    assert len(iterations) == len(errors), "Lengths of iterations and errors_final do not match"

    min_index = np.argmin(errors)
    max_index = np.argmax(errors)
    print(f"Minimum of {ylabel}: ", np.min(errors), f"for iteration {min_index}.", f"\nMaximum of {ylabel}: ", np.max(errors), f"for iteration {max_index}.")
    plt.scatter(iterations, errors, color="red", label='Data')
    plt.plot(iterations, errors, 'b-', label='')
    plt.xlabel('Iteration')
    plt.ylabel(f'{ylabel}')
    plt.title(f'')
    plt.legend()
    plt.show()


def convolve_target(target_exp, sigmax, sigmay):
    newsize = target_exp.shape[0]
    x=np.linspace(-newsize//2,newsize//2,newsize)
    y=np.linspace(-newsize//2,newsize//2,newsize)
    exp_convol=np.ones((newsize,newsize))
    for i in range(exp_convol.shape[0]):
        for k in range(exp_convol.shape[0]):
            exp_convol[i,k]=1/2/np.pi/sigmax/sigmay*np.exp(-1/2*x[i]**2/sigmax**2-1/2*y[k]**2/sigmay**2)
    target_exp=scipy.signal.fftconvolve(target_exp,exp_convol,mode="same")  #this is the image the camera should record in a theoretical and ideal world
    target_exp_cropped=target_exp#[869:1051,449:631]    #crop out just the target
    target_exp2_resized = Image.fromarray(np.uint8(target_exp_cropped), mode="L").resize((newsize, newsize))
    '''
    plt.figure(1)
    plt.imshow(target_exp2_resized)
    plt.title("")
    plt.colorbar()
    plt.show()
    '''
    return target_exp_cropped

def calculate_error_from_convolution(bmp_files, target_exp, sigmax, sigmay, folder_path, Rotation_angle, minP_scaled, inner_radius, outer_radius, magnification, top_exp, bottom_exp, left_exp, right_exp, min_r,max_r):
    print("Calculating error from convolution...")

    errors_calculated = []
    flatness_calculated = []
    mean_intensities = []

    convolved_target = convolve_target(target_exp,sigmax, sigmay)

    for bmp_file in bmp_files:
        # Load, rotate, transpose and crop the image
        img_path = os.path.join(folder_path, bmp_file)
        image = Image.open(img_path).convert('L').rotate(Rotation_angle)  # Convert to grayscale and rotate
        image = np.transpose(np.array(image))
        image_cropped = image[top_exp:bottom_exp,left_exp:right_exp]#[1560:2142,1416:1998]

        len_rows, len_cols = image_cropped.shape

        plt.figure(1)
        plt.imshow(image_cropped, cmap='viridis')
        plt.title("image experiment")
        plt.colorbar()
```

```

        plt.figure(2)
        plt.imshow(minP_scaled*target_exp, cmap='viridis')
        plt.title("target experiment")
        plt.colorbar()

        plt.show()
        #exit(0)

        ###ERROR###################
        #calculate the error and save the file to later on save pixel values in initialized lits and plot
        error =  minP_scaled*convolved_target - image_cropped

        # Get the dimensions of the image
        height, width = error.shape

        center = (width // 2, height // 2)
        inner_radius_temp = inner_radius*magnification-1
        outer_radius_temp = outer_radius*magnification+1

        # Create a grid of coordinates
        y, x = np.ogrid[:height, :width]

        # Calculate the distance from the center
        distance_from_center = np.sqrt((x - center[0])**2 + (y - center[1])**2)
        print("inner radius: ", inner_radius_temp, "outer radius: ", outer_radius_temp)
        # Create the mask for the torus
        mask = (distance_from_center >= inner_radius_temp) & (distance_from_center <= outer_radius_temp)

        #first transform the cropped image to polar coordinates to calculate the mean intensity of the torrus
        value = np.sqrt(((image_cropped.shape[0]/2.0)**2.0)+((image_cropped.shape[1]/2.0)**2.0))
        polar_image = cv2.linearPolar(image_cropped,(image_cropped.shape[0]/2, image_cropped.shape[1]/2), value, cv2.WARP_FILL_OUTLIERS)
        image_cropped_polar = polar_image.astype(np.uint8)[:,min_r:max_r]

        len_rows, len_cols = image_cropped_polar.shape


        # Apply the mask to the error image, setting values outside the torus to zero
        error_cleaned = np.where(mask, error, 0)

        error=error_cleaned

        error_calculated = 100*(np.sqrt(1/(len_cols * len_rows)*np.sum((error/(np.max(image_cropped_polar)-np.min(image_cropped_polar)))**2)))
        errors_calculated.append(error_calculated)

        ###FLATNESS#########################################

        mean_intensity = np.sqrt(np.sum(image_cropped_polar)/(len_rows * len_cols))
        print("mean intensity: ", mean_intensity)

        flatness = mean_intensity*target_exp - image_cropped

        real_mean_intensity = np.mean(image_cropped_polar)
        mean_intensities.append(real_mean_intensity)

        flatness_calc = 100*(1-np.sqrt(1/(len_cols * len_rows)*np.sum(((flatness)/mean_intensity)**2)))

        flatness_calculated.append(flatness_calc)

    return errors_calculated, flatness_calculated, mean_intensities


def plot_mean_intensity_of_each_iteration(iterations, mean_intensities):
    plt.scatter(iterations, mean_intensities, color="red", label='Data')
```

```
    plt.plot(iterations, mean_intensities, 'b-', label='')
    plt.xlabel('Iteration')
    plt.ylabel('mean intensities')
    #plt.title(f'')
    plt.legend()
    plt.show()


def convolve_and_save_new_image(new_img, newsize, sigmax, sigmay, size_of_resize):
    #convolution of image with added error with psf
    x=np.linspace(-newsize//2,newsize//2,newsize)
    y=np.linspace(-newsize//2,newsize//2,newsize)

    exp_convol=np.ones((newsize,newsize))
    for i in range(exp_convol.shape[0]):
        for k in range(exp_convol.shape[0]):
            exp_convol[i,k]=1/2/np.pi/sigmax/sigmay*np.exp(-1/2*x[i]**2/sigmax**2-1/2*y[k
                ]**2/sigmay**2)
    img_conv=scipy.signal.fftconvolve(new_img,exp_convol,mode="same")
    img1=Image.fromarray(np.uint8((np.array(img_conv))),mode="L")
    img1.save("previous.bmp")

    img2=img1.resize((size_of_resize, size_of_resize))

    newarray=np.array(img2)
    return newarray


def create_floyd_steinberg_image(newarray,Nx,Ny):
    #RESCALING to necessary array size
    background=PIL.Image.new(mode="L",size=(Ny,Nx)) #creates array with zeros in size of
        dmd
    background_array=np.array(background)

    dim_img=newarray.shape
    print("Dimension convolved image: ", dim_img)
    for i in range(dim_img[0]):
        for k in range(dim_img[1]):
            background_array[(960-dim_img[1]//2)+i, (540-dim_img[0]//2)+k]+=newarray[i,k]

    #for normalization of dithering
    #print("max background array pre setting pixel values >255=255: ", np.max(
        background_array))
    background_array[background_array>255]=255
    #print("max background array pre 255*np.max(): ", np.max(background_array))
    background_array=np.array((255/np.max(background_array))*background_array)
    #print("max background array: ", np.max(background_array))


    #DITHERING
    imgfinal=Image.fromarray(np.uint8(background_array),mode="L")
    img_FS=np.array(imgfinal.convert("1",dither=Image.Dither.FLOYDSTEINBERG))#>0.5

    return img_FS


def preprocess_recorded_image(image_path, top_exp, bottom_exp, left_exp, right_exp,
     Rotation_angle):
    imgbmp=Image.open(image_path).rotate(Rotation_angle)
    im_array=np.transpose(np.array(imgbmp))
    im_array_cropped=im_array[top_exp:bottom_exp,left_exp:right_exp]#[1560:2142,1416:1998]

    return im_array_cropped

def preprocess_old_image(img_path_old, Rotation_angle):
    previous_image=Image.open(img_path_old).rotate(Rotation_angle)
    previous_array=np.transpose(np.array(previous_image))

    return previous_array

```

```
def filter_background_noise_for_torrus(error, inner_radius, outer_radius, magnification):
    ###filter out errors caused by noise of the background###
    height, width = error.shape

    center = (width // 2, height // 2)
    inner_radius_1 = inner_radius*magnification
    outer_radius_1 = outer_radius*magnification

    # Create a grid of coordinates
    y, x = np.ogrid[:height, :width]

    # Calculate the distance from the center
    distance_from_center = np.sqrt((x - center[0])**2 + (y - center[1])**2)

    # Create the mask for the torus
    mask = (distance_from_center >= inner_radius_1) & (distance_from_center <=
        outer_radius_1)

    # Apply the mask to the error image, setting values outside the torus to zero
    error_cleaned = np.where(mask, error, 0)
    error1=error_cleaned

    return error1
```