Technical University of Munich
Department of Informatics

Master's Thesis in Robotics, Cognition, Intelligence

# Vignette Calibration for Fisheye Lenses

**Maurice Hermwille**

Technical University of Munich
Department of Informatics

Master's Thesis in Robotics, Cognition, Intelligence

# Vignette Calibration for Fisheye Lenses

Vignettierungskalibrierung für Fischaugenobjektive

**Maurice Hermwille**

Supervisor: Prof. Dr. Daniel Cremers
Advisor: Nikolaus Demmel
September 28th, 2018

# Declaration of authorship

I confirm that this master's thesis is my own work and I have documented all sources and material used.

Garching, Maurice Hermwille

# Contents

# Abstract

Camera systems are one of the mainstays of contemporary computer vision and essential for a multitude of applications. Increasing accuracy of the employed hard- and software and ensuring the reproducibility and genuineness of the retrieved data is a never-ending process in this field. Fisheye lenses provide an easy tool to retrieve visual data at a wide opening angle in a single frame. These lenses however often suffer from strong vignetting in the edges, reducing the degree of veritableness towards these regions. Accurate photometric calibration is in particular important for algorithms assuming brightness-constancy for world points seen from different views, for example so-called direct methods for visual SLAM and 3D reconstruction. They rely on direct comparison of pixel-intensities and have recently demonstrated great performance in particular when carefully calibrated. This work proposes a vignette calibration procedure for fisheye lenses from a single image sequence. The main focus is the implementation of a radially symmetric vignetting model, utilizing camera projection models well-suited for fisheye lenses, and the integration of a robust marker detection for camera pose estimation under partial observations. Unlike the previous methods this work is based on, this does not rely on undistortion and 4-point-homography calculation of the detected markers, but instead avoids undistortion and uses the PnP algorithm with non-linear refinement to accurately estimate camera poses based on all detected marker points. This approach is evaluated on a variety of different lenses and conditions, which shows that the assumption of a radial vignette model fits these lenses well and that the used projection models are suitable. The evaluation moreover demonstrates that a radial model improves the resulting vignette compared to a full model by averaging out local inconsistencies as well as requiring less data for calibration. This has the potential to allow vignette calibration based on existing images for intrinsic camera calibration instead of requiring special vignette calibration sequences.

# Chapter 1

# Introduction

In modern computer science, computer vision has risen to an invaluable tool for retrieval, processing and analysis of visual data. Recording high quality images and videos within a wide range of various parameters makes great demands on the employed hardware as well. This includes the recording tools consisting of a variety of camera systems in different sizes with different sensors and lenses. A lens type allowing extra-ordinarily wide angle shots are so-called fisheye lenses exceeding an opening angle of 180 degrees retrieving image information even from behind the sensor plane. This type of lenses can be particularly useful in some computer vision fields like motion estimation (Rituerto et al., 2010; Zhang et al., 2016). To ensure reproducibility of results, one is reliant on high accuracy of the gathered data which requires hardware and software to be as error-free as possible.

A common issue for wide-angle lenses that fisheye lenses also suffer from is an effect called "vignetting" (see Figure 1.1) where the light reaching the camera sensor is the more attenuated the greater its angle relative to the optical axis is. This evokes the need of a software side calibration to compensate for these attenuations and produce realistic images where the image information is independent on the angle under which light from an object entered the camera. This is in particular true for algorithms assuming brightness-constancy for world points seen from different views, for example so-called direct methods for visual SLAM and 3D reconstruction. They rely on direct comparison of pixel-intensities and have recently demonstrated great performance in particular when carefully calibrated (Engel et al., 2018, 2016; Yang et al., 2017). This thesis seeks to improve upon previous work by implementing better suited camera models and removing the need to undistort the recorded images for pose estimation, which is problematic for wide-angle lenses approaching a field of view of 180° or more. A radially symmetric model is employed to reduce the amount of data needed for a decent vignette calibration as well as canceling out local inconsistencies. Furthermore, a more robust variant for marker detection has been implemented to make use of April marker grids instead of finding a homography based on four points of a single Aruco marker on undistorted images. The pose estimation was realized via the UPnP algorithm (Kneip et al., 2014) and subsequent non-linear refinement with pose-only bundle adjustment, utilizing all detected marker points.

Figure 1.1: *Image without (left) and with Vignetting effect (right). (Image from (Heeling, 2018))*

This chapter gave a short introduction to the matter. In the second chapter, this thesis will give a brief overview over different camera models and presents the aforementioned fisheye lenses whereas the Chapter 3 explains the vignetting effect. In the fourth and fifth chapter we describe the process of vignette calibration and present the results on different image sequences. The last chapter then gives a conclusion and short outlook.

# Chapter 2

# Camera models and lenses

## 2.1 Camera models

### 2.1.1 Simple Pinhole Camera model

The exact behavior and path of a light beam from its entrance at the front of the camera lens to its impact on the sensor chip is a complicated physical phenomenon. To simplify calculations and approximate the behavior, a simple camera model that assumes the camera works like a pinhole model has become used in most disciplines dealing with image recordings via cameras. The basic assumptions of this model is that Light beams enter the camera through a single point (pinhole) and travel in a straight line until hitting the sensor in the back of the camera case. The pinhole as well as the center point of the imaging sensor are assumed to be on the optical axis. This makes for a very simple geometric model that eliminates the complex lens behavior while sacrificing a small amount of accuracy. The first cameras before the usage of lenses for imaging used this pinhole effect to produce images. This model only has four parameters $f_x, f_y, c_x$ and $c_y$ where $\boldsymbol{f}$ is the focal length and $\boldsymbol{c}$ the principal point with their respective $x$ and $y$ components. These so-called intrinsic parameters are often accumulated in the camera matrix $K$ that can be used for projections:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

This camera model, while giving a good estimate for simple applications, is insufficient in the case of fisheye lens cameras since the field of view is limited to 180 degrees. Light rays originating from behind the pinhole plane cannot enter the camera through the pinhole. Another simple model that only employs a single additional parameter for distortion, the Field-of-view camera model (Devernay and Faugeras, 2001), was shown to not be accurate enough for fisheye lenses (Usenko et al., 2018).

Since accuracy is an important factor in producing reliable results in the computer vision domain, different models where proposed to better approximate the actual

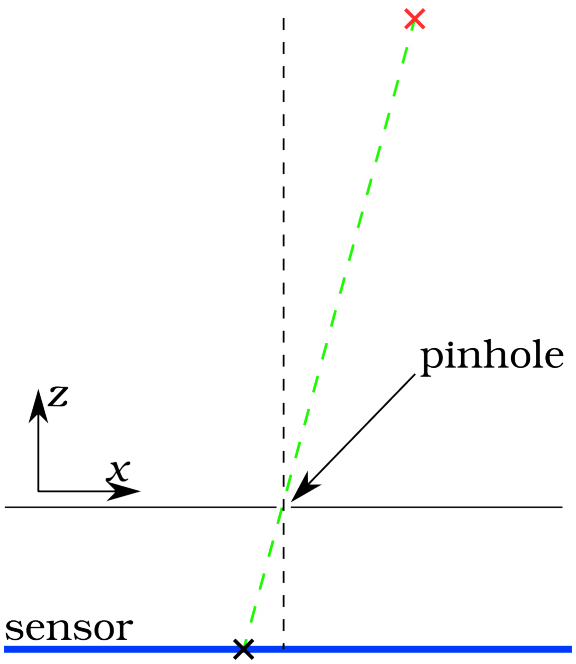


Figure 2.1: *Pinhole model concept image. This is the original variant of the pinhole camera without a lens. If the model is just a simplification for a lens camera, the image plane is technically between pinhole and object. Compare also to Figures 2.2 and 2.3*

behavior of light in the imaging process. Two models in particular will be presented here, the Extended Unified Camera Model and the Kannala-Brandt model. Both of these were found to describe fisheye lenses with a high accuracy (see e.g. (Usenko et al., 2018)).

### 2.1.2 Extended Unified Camera Model

The Unified Camera (or Projection) Model (Geyer and Daniilidis, 2000) employs a slightly different approach than the simple pinhole camera model. Instead of the light rays falling through a pinhole they are assumed to be projected twice, once onto a unit sphere and then again onto the imaging plane. The center of the unit sphere is positioned in front of the imaging plane by a distance $\xi$. For better numerical properties (Usenko et al., 2018) has shown that it is worthwile to rewrite the parameter

$$\xi = \frac{\alpha}{1-\alpha}.$$

With

$$d = \sqrt{(x^2 + y^2 + z^2)}$$

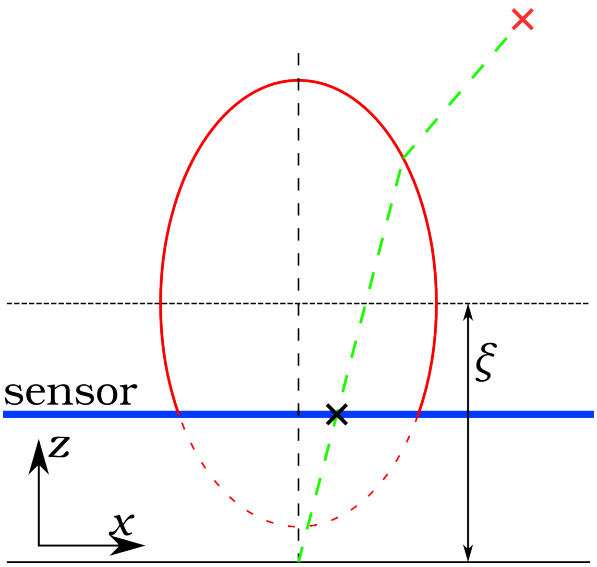


Figure 2.2: *EUCM concept image. The distortion of the ellipsoid is controlled by the parameter $\beta$, with $\beta = 1$ being the spherical UCM case. The distance between the center of the ellipsoid and the imaging plane is $\xi = \frac{\alpha}{1-\alpha}$.*

the projection $(x_\pi, y_\pi)$ of a point $(x, y, z)$ can then be calculated as follows:

$$x_\pi = f_x \frac{x}{\alpha d + (1-\alpha) z} + c_x, \tag{2.1}$$

$$y_\pi = f_y \frac{y}{\alpha d + (1-\alpha) z} + c_y, \tag{2.2}$$

and the unprojection:

$$x = \tau m_x \tag{2.3}$$

$$y = \tau m_y \tag{2.4}$$

$$z = \tau - \xi \tag{2.5}$$

with

$$\tau = \frac{\xi + \sqrt{1 + (1-\xi^2) r^2}}{1 + r^2} \tag{2.6}$$

$$m_x = \frac{x_\pi - c_x}{f_x} (1-\alpha) \tag{2.7}$$

$$m_y = \frac{y_\pi - c_y}{f_y} (1-\alpha) \tag{2.8}$$

$$r = \sqrt{m_x^2 + m_y^2} \tag{2.9}$$

$$\xi = \frac{\alpha}{1-\alpha} \tag{2.10}$$

The UCM can be expanded to the Extended Unified Camera Model (Khomutenko et al., 2016) where the first projection on a sphere is replaced by a projection on an ellipsoid. This leads to an additional parameter $\beta$ for the elliptic distortion, where $\beta = 1$ just represents the normal UCM. In this case, the projection becomes is just altered by the factor $\beta$ in the term $d$:

$$d = \sqrt{\beta(x^2 + y^2) + z^2}$$

The unprojection however becomes more complicated due to the ellipsoid which leads to the following formulae:

$$x = \frac{1}{\lambda} m_x \tag{2.11}$$

$$y = \frac{1}{\lambda} m_y \tag{2.12}$$

$$z = \frac{1}{\lambda} m_z \tag{2.13}$$

with

$$\lambda = \sqrt{m_x^2 + m_y^2 + m_z^2} \tag{2.14}$$

$$m_x = \frac{x_\pi - c_x}{f_x} \tag{2.15}$$

$$m_y = \frac{y_\pi - c_y}{f_y} \tag{2.16}$$

$$r = \sqrt{m_x^2 + m_y^2} \tag{2.17}$$

$$m_z = \frac{1 - \beta\alpha^2 r^2}{\alpha\sqrt{1 - (2\alpha - 1)\beta r^2 + (1 - \alpha)}} \tag{2.18}$$

### 2.1.3 Kannala-Brandt Camera model

The Kannala-Brandt camera model (Kannala and Brandt, 2006) that is able to well approach a variety of lens types including fisheye lenses, uses a slightly different approach in approximating the imaging process. The basic assumptions is that the distance of a projected point on the imaging plane is proportional to an (up to) ninth order polynomial $d(\theta)$ of the angle $\theta$ under which the object point is perceived. Kannala and Brandt also proposed a model with only six intrinsic camera parameters where the polynomial is cut off at the fifth order but we decided to employ the approach with four additional parameters.

With

$$d(\theta) = \theta + k_1\theta^3 + k_2\theta^5 + k_3\theta^7 + k_4\theta^9,$$

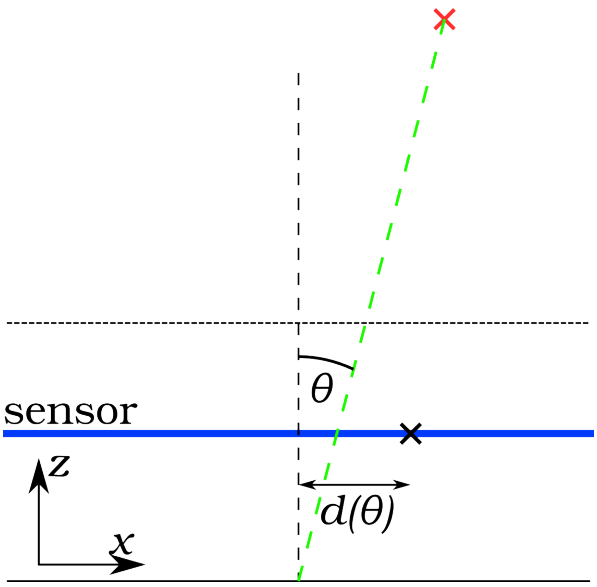


Figure 2.3: *KB concept image. The distance between the optical axis and the projection of the point is proportional to the polynomial function $d(\theta)$ which depends on the angle $\theta$ between the axis and the incident light ray.*

the projection can be expressed as follows:

$$x_\pi = f_x d(\theta)\frac{x}{r} + c_x \tag{2.19}$$

$$y_\pi = f_y d(\theta)\frac{y}{r} + c_y \tag{2.20}$$

with

$$r = \sqrt{x^2 + y^2} \tag{2.21}$$

$$\theta = \text{atan2}(r, z) \tag{2.22}$$

The unprojection function becomes more complicated since it requires solving the polynomial (via $d^{-1}(r) = \theta$) but we can define it as:

$$x = \sin(\theta^*)\frac{m_x}{r_u} \tag{2.23}$$

$$y = \sin(\theta^*)\frac{m_y}{r_u} \tag{2.24}$$

$$z = \cos(\theta^*) \tag{2.25}$$

with

$$m_x = \frac{x_\pi - c_x}{f_x} \tag{2.26}$$

$$m_y = \frac{y_\pi - c_y}{f_y} \tag{2.27}$$

$$r_x = \sqrt{m_x^2 + m_y^2} \tag{2.28}$$

$$\theta^* = d^{-1}(r_u) \tag{2.29}$$

Both, the EUCM and the Kannala-Brandt model, are radially symmetric and thus particularly suitable for the radial model envisaged by this thesis.

## 2.2 Fisheye lenses

Camera lenses are a photographer's—or computer visionist's—tool to capture a specific visual state of the real world and save it for later usage. Most conventional lenses are only capable of taking images from a rather narrow angle of view compared to what a human eye is able to see while other lenses were created for landscape photography and panoramic views. Throughout the years of camera lens development, some manufacturers pushed the limit of the widest angle one can capture with a lens further to reach higher and higher values. This in turn started to distort images when pictured on a flat 2D screen or paper resulting in a view similar to that expected of a fish's perception, thus the name 'fisheye lenses'. An example of a distorted image is shown in Figure 2.4. These lenses are often created with opening angles of 180 degrees and higher, some concepts reaching as far as 310 degrees, effectively imaging objects lying far behind the camera (and imaging plane) itself. Both of the aforementioned camera models are well-suited for wide angle and fisheye lenses (Usenko et al., 2018). An image of a fisheye lens can be seen in Figure 2.5

Figure 2.4: *Fisheye view of ESO's Very Large Telescope (VLT), taken from (Salgado, 2015) under Creative Commons license.*

Figure 2.5: *Image of a fisheye lens, Nikkor 6mm f/2.8, taken from (Commons, 2018) under Creative Commons license.*

# Chapter 3

# Vignetting

The imaging process of a camera consists of converting the incident light beam to usable data. Within a scene, it is usually enough to have data up to a scalar factor as long as one is able to compare the images with each other. Let us assume an image $B$ with coordinates $\boldsymbol{x}$, then the image information $I$ provided by the sensor at a certain pixel is:

$$I(\boldsymbol{x}) = G(B(\boldsymbol{x})t), \tag{3.1}$$

where $G$ is the response function of the camera and $t$ the exposure time. For simplicity let's define $U = G^{-1}$ as the inverse response function. A common phenomenon in camera imaging is the light attenuation in captured pictures towards the image edges. This effect is called 'vignetting' and while sometimes desired by photographers it is predominantly an issue for imaging techniques. Vignetting is the result of having more than one aperture (pinhole cameras don't suffer from vignetting) and the ray beam being partially restricted by one of them. The imaging model (3.1) then gains an additional, position dependent, attenuation factor $V(\boldsymbol{x})$ which leads to:

$$I(\boldsymbol{x}) = G(B(\boldsymbol{x})V(\boldsymbol{x})t) \tag{3.2}$$

As $V(\boldsymbol{x})$ in general is only known up to a scalar factor, unless the exact irradiance is known, this work always normalizes it to $V_{\max} = 1$. The image information that is retrieved in vignetted areas is not accurate since the attenuation will produce lower brightness values compared to the same point being captured by the lens center. To avoid this effect one can either only keep image information from within the area without vignetting or calibrate the camera lens to correct the images afterwards. The first approach is either costly (reduce the area of vignetting by changing the lens geometry) or wasteful (rejecting the erroneous image data), while the second one requires to take a calibration sequence. This sequence has to contain pictures accurately representing the vignetting affect in possibly all areas of the lens. This thesis seeks to show the latter approach for fisheye lenses. We employ the same mode as mentioned in (Engel et al., 2016).

# Chapter 4

# Vignette Calibration

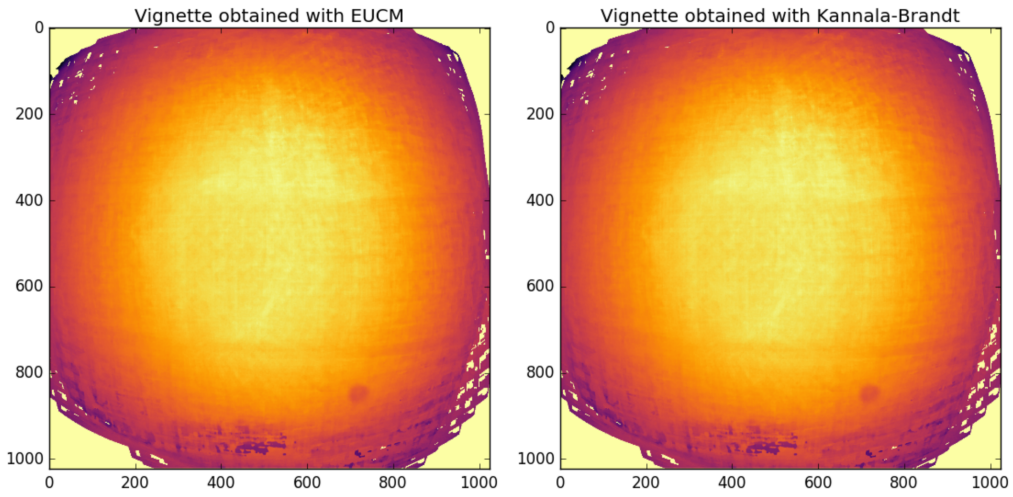


Figure 4.1: *Comparison between EUCM (left) and KB (right) full vignettes.*

The goal of this thesis is to provide vignette parameters allowing the calibration of fisheye lenses by recording a calibration sequence. There will be a comparison between a dense model where every pixel is treated seperately and a radial model where the attenuation is a function of the distance from the principal point.

The calibration sequence contains a large flat surface with approximately uniformly colored regions as well as a (set of) marker(s) to extract the exact camera pose for every image. To this end, we used an April marker (Olson, 2011) grid whose markers are automatically detected in the images. The pose estimation is then performed with an initial guess from UPnP and non-linear refinement using the Gauss-Newton algorithm as presented in (Usenko et al., 2018). In the end this yields the intrinsic parameters of the camera (if not already known from pre-calibration) as well as the current camera pose for each image.

As a first step, the EUCM and the Kannala-Brandt model mentioned earlier where compared to each other to assess which of them may suit our needs best. For the most part both had very similar results (see Figure 4.1 and Figure 4.2) and since the

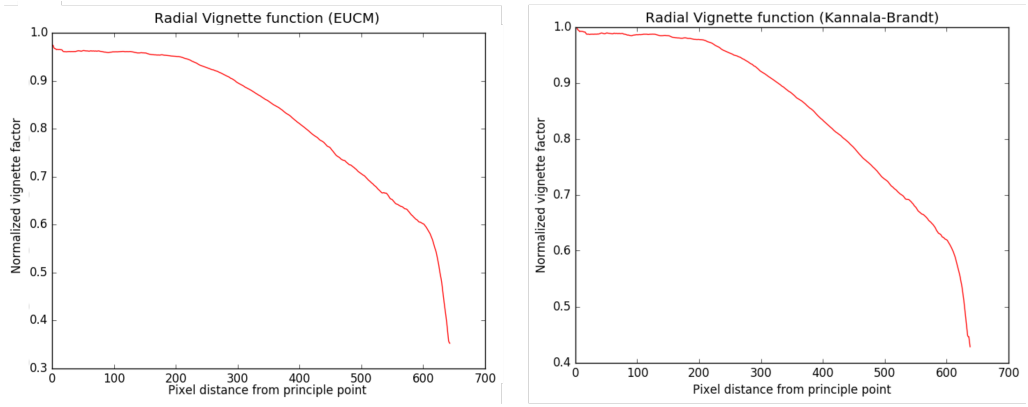


Figure 4.2: *Comparison between EUCM (left) and KB (right) radial vignette functions.*

Kannala-Brandt model can reach potentially higher accuracy (Usenko et al., 2018) we decided to use it for the remainder of the study.

The retrieved information from the image sequence is then used to generate the vignetting factor $V$ for every pixel of the sensor by comparing values of the same surface point when perceived by different different views. This assumes that the response function of the pixels is known (up to a scalar factor) and the exposure time is either constant over the sequence or factored into the calculations. In the case of Aruco data sets we used the calibrated response function from (Engel et al., 2016) whereas in the April grid sequences the camera was configured to have a linear response (the scalar factor of the response function can be ignored, since the vignette is determined only up to a scalar factor anyway).

The marker sets used in the observed sequences can be found in (Garrido-Jurado et al., 2014; Olson, 2011), examples can be seen in Figure 4.3, the camera lenses are of the type Lensagon BF2M2020S23 with a 195° field of view ($Lens_1$) and BM2420 with a 148° FOV ($Lens_2$), both from Lensation.

To create a relatively robust representation of $V$ we recorded a long sequence of images ($I_i$ with $i = 0, 1, \ldots$) in order to obtain image values for every pixel of the sensor. The method assumes the markers to be situated on a well-lit, planar surface $\mathcal{S} \subset \mathbb{R}^3$ exhibiting Lambertian reflectance and estimates the camera's position with respect to $\mathcal{S}$ via marker detection. The pixel values on the sensor are then calculated via a projection $\pi_i : \mathcal{S} \to \Omega$ based on one of the camera models presented in Chapter 2 and the current camera pose. Let $C : \mathcal{S} \to \mathbb{R}$ be the irradiance of the surface and $V : \mathcal{P} \to \mathbb{R}$ the vignetting factor where $\mathcal{P}$, the domain of $V$, is $\mathcal{P} = \Omega$ for the full model and $\mathcal{P} = [0, r_{max}]$ in the radial case. $r_{max}$ is the distance

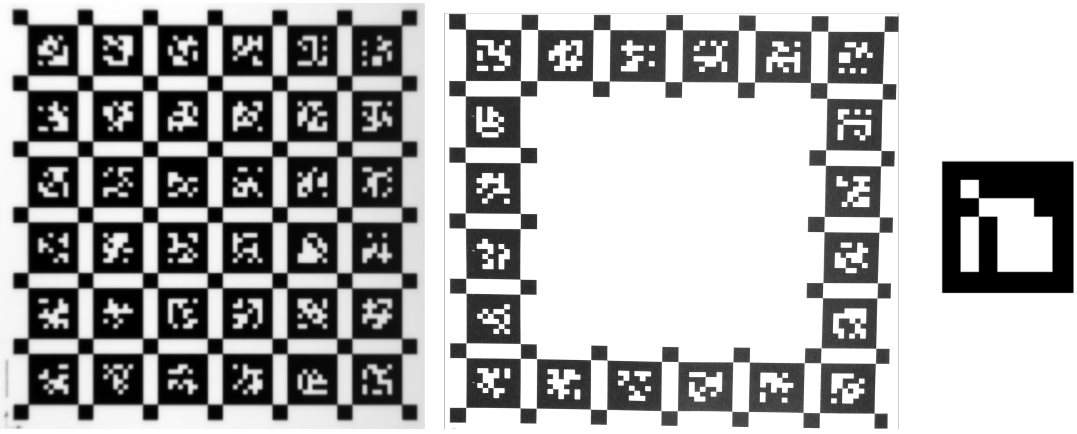

Figure 4.3: *Exemplary markers, April grid (left), April square (middle) and Aruco marker (right)*

from the principal point to the furthest corner of the image. We can then define a Maximum-Likelihood energy (Aldrich, 1997)

$$E(C,V) = \sum_{i,\boldsymbol{x}\in\mathcal{S}} \left(t_i V\left(m(\pi_i(\boldsymbol{x}))\right) C(\boldsymbol{x}) - U\left(I_i(\pi_i(\boldsymbol{x}))\right)\right)^2, \tag{4.1}$$

where $m$ depends on the employed model: It is the identity $\mathbb{I}$ in for the full model and in the radial case $m$ is a mapping: $m(\boldsymbol{y}) = ||\boldsymbol{y}-\boldsymbol{c}||$ from any point to its distance to the principal point, $||\cdot||$ being the euclidian norm.

In case of the full model, bilinear interpolation between the four nearest points is used to retrieve values in-between discrete positions whereas normal linear interpolation is employed in the case of the radial vignette. The interpolation of points in the camera images is still done bilinear even in the radial model.

In our case we kept $t_i = t$ constant for every frame and selected a $1000 \times 1000$ points large region ("grid") around the marker for the calibration (see Figure 4.4). This energy is subsequently minimized in a two-step-process that is iterated multiple times where we first estimate the image $C^*$ of the grid and in the second step calculate the optimal $V^*$:

$$\begin{aligned} C^*(\boldsymbol{x}) &= \arg\min_{C(\boldsymbol{x})} E(C,V) \\ &= \frac{\sum_i tV\left(m(\pi_i(\boldsymbol{x}))\right) U\left(I_i(\pi_i(\boldsymbol{x}))\right)}{\sum_i \left(tV(m(\pi_i(\boldsymbol{x})))\right)^2}. \end{aligned} \tag{4.2}$$

Figure 4.4: *One image of a calibration sequence showing the grid region that was taken into account for the calibration. The red lines show the same distortion as the grid itself, verifying correct calibration of the camera parameters.*

To calculate the optimal $V$ for a given $C$ we differentiate $E$ with respect to $V$ which gives us:

$$\frac{\partial E(C,V)}{\partial V} = \sum_{i,\boldsymbol{x}\in S} 2\big(t_i V\left(m(\pi_i(\boldsymbol{x}))\right) C(\boldsymbol{x}) - U\left(I_i(\pi_i(\boldsymbol{x}))\right)\big)(t_i C(\boldsymbol{x})) \tag{4.3}$$

We now introduce $p$ as the integer coordinates of the vignetting factor $V$, as well as defining a neighborhood $\mathcal{N}(q)$ that contains the closest four (or two in the radial case) integer coordinate points around a point $q$. One can define weights $w(a,b)$ that correspond to the (bi)linear weights of (bi)linear interpolation where $a$ and $b$ are the point to interpolate and the data point to be interpolated from respectively. The weights $w$ are subject to

$$\forall \boldsymbol{x}, i : \sum_{p\in\mathcal{N}(m(\pi_i(\boldsymbol{x})))} w(p, m(\pi_i(\boldsymbol{x}))) = 1.$$

Equation (4.3) then can be written as

$$\frac{\partial E(C,V)}{\partial V} = \sum_{i,\boldsymbol{x}\in S} \sum_{p\in\mathcal{N}(m(\pi_i(\boldsymbol{x})))} 2w(p, m(\pi_i(\boldsymbol{x})))\big(t_i V\left(m(\pi_i(\boldsymbol{x}))\right) C(\boldsymbol{x}) - U\left(I_i(\pi_i(\boldsymbol{x}))\right)\big)(t_i C(\boldsymbol{x})). \tag{4.4}$$

We can now switch the order of summation and set the term to zero since we are interested in minimizing the energy:

$$\frac{\partial E(C,V)}{\partial V} = \sum_{p} \sum_{i,\boldsymbol{x}\in S:\, p\in\mathcal{N}(m(\pi_i(\boldsymbol{x})))} 2w(p, m(\pi_i(\boldsymbol{x})))\big(t_i V\left(m(\pi_i(\boldsymbol{x}))\right) C(\boldsymbol{x}) - U\left(I_i(\pi_i(\boldsymbol{x}))\right)\big)(t_i C(\boldsymbol{x})) \stackrel{!}{=} 0 \tag{4.5}$$

Since this has to hold true for every $p$, the calculation of $V^*$ becomes:

$$\begin{aligned} V^*(p) &= \arg\min_{V(p)} \sum_{i,x:\, p\in\mathcal{N}(m(\pi_i(\boldsymbol{x})))} \left(t_i V\left(m(\pi_i(\boldsymbol{x}))\right) C(\boldsymbol{x}) - U\left(I_i(\pi_i(\boldsymbol{x}))\right)\right)^2 \\ &= \frac{\sum_{i,\boldsymbol{x}\in S:\, p\in\mathcal{N}(m(\pi_i(\boldsymbol{x})))} w(p, m(\pi_i(\boldsymbol{x})))(U(I_i(\pi_i(\boldsymbol{x}))))(t_i C(\boldsymbol{x}))}{\sum_{i,\boldsymbol{x}\in S:\, p\in\mathcal{N}(m(\pi_i(\boldsymbol{x})))} w(p, m(\pi_i(\boldsymbol{x})))(t_i C(\boldsymbol{x}))^2} \end{aligned} \tag{4.6}$$

Note that $V$ is still defined on non-integer coordinates via (bi)linear interpolation.

The full approach essentially assumes that the vignetting that happens for every single pixel is independent of every other pixel in the image. Due to the radially symmetric nature of lenses, we want to use a model that exploits this symmetry for a parametric approach of the vignette calibration. This also seems natural as the geometric calibration already employs a radial model. Our model assumes that the vignetting factor of the lens is not individual for each point but instead a radial function with its center in the principal point of the camera system.

In a real, non-ideal calibration environment surfaces often do not fulfill the assumption of lambertian reflectance and can thus produce different brightness values under different angles. Shadows cast by the camera itself or the user handling the recording cannot always be fully avoided as well. The radial method reduces the amount of parameters of the calibrated vignette and allows to smoothen out local inconsistencies since those are compensated for by the remaining values at the same radial distance. Another advantage is the need for less data and thus shorter calibration sequences since the averaging nature of the radial model results in multiple data points used for each radial value. A comparison of a dense and a radial vignette can be seen in Chapter 5, Figures 5.1, 5.4 and 5.7.

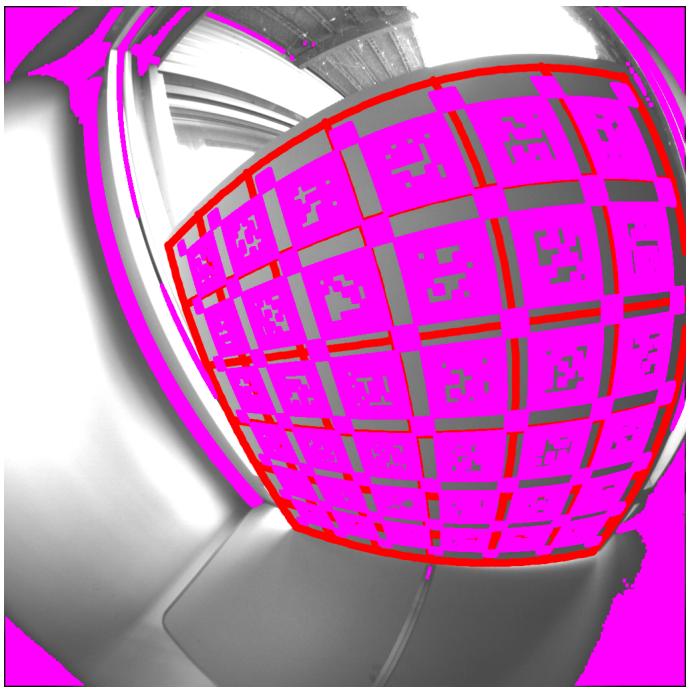

Figure 4.5: *Image from an April grid sequence. Purple pixels weren't considered for the vignette calibration.*

While (Engel et al., 2016) only employed an Aruco marker (Garrido-Jurado et al., 2014) we also used an April marker set (Olson, 2011) for calibration. The additional markers provided more points to obtain a higher accuracy position estimation. In addition to higher robustness it also allows pose detection directly on the distorted images as the undistortion performed in (Engel et al., 2016) is not possible for fields of view larger than 180 degrees.

In the radial approach, $V(\boldsymbol{x})$ will be simplified to a one-dimensional function $V(r)$ that only depends on the distance from the principal point.

We decided to exclude the black markers themselves for the calibration (and a small border surrounding them) since at the edge of a marker the light rays from dark and bright object points will fall on the same pixel on the sensor and thus produce a mixed grey value that might otherwise be interpreted as attenuation by vignetting. The dark marker parts are also inherently less valuable for the vignette estimation since their low brightness values leave less range (and thus less precise gradation steps) for attenuation. An example of which parts where excluded can be seen in figure 4.5.

In order to have a way to compare our radial approach to the the dense vignette model, we performed 360 cuts in radial direction through the full vignette, starting at the principal point of the camera (see Figure 4.6. These cuts are spaced out by angles of one degree and are then combined into a single data line by taking the mean value of all cuts and presenting error bars for the $1\sigma$ standard deviation. Both

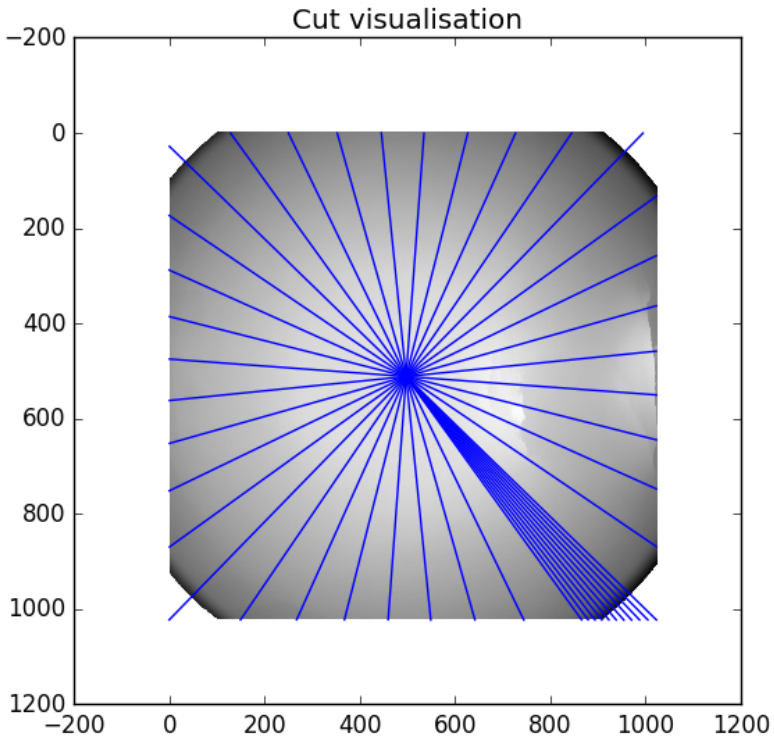


Figure 4.6: *Visualisation of the 360 cuts. With the exception of the bottom right corner, only every 10th cut is depicted for better visibility.*

the radial cut and the mean cut of the dense model are then normalized to have a mean deviation of 1 for better alignment of the unknown scale factor. This can be done without loss of information since the vignette already comprises an unknown scale factor.

# Chapter 5

# Results

This chapter presents the results obtained during the work of this thesis. Six different setups where used for datasets:

1. Aruco marker on the wall, with $Lens_2$ (Engel et al., 2016)
2. April grid board, artificial lighting, $Lens_1$, existing dataset
3. April grid board, artificial lighting, $Lens_1$, recorded dataset
4. April grid board, natural lighting, $Lens_1$
5. April square, mixed lighting $Lens_1$
6. April square, natural lighting $Lens_1$

We will reference these as $setup_1$ to $setup_6$ in the thesis.

As a first step, we compared our radial model with the dense vignette obtained from a large, previously recorded dataset under good conditions (large, flat, unicolored surface; relatively even lit). The difference between the dense model and the radial vignette are rather minor and hard to spot even in false color images (Figure 5.1). The radial function is a smooth, concave function (with the exception of the very edge region where little to no data is available) with a typical shape for vignettes (Figure 5.2). When comparing the 360 radial cuts with the radial model (Figure 5.3)

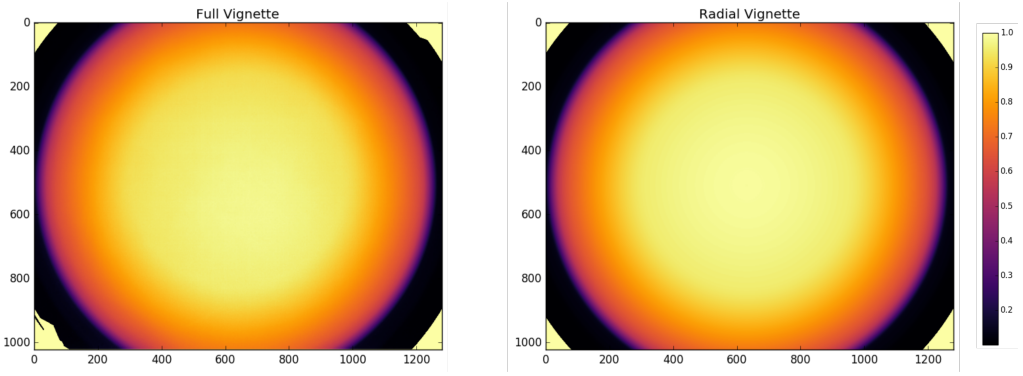


Figure 5.1: *Comparison between full model (left) and radial model (right). False colors for better visualisation of graduations. Data from* $setup_1$.

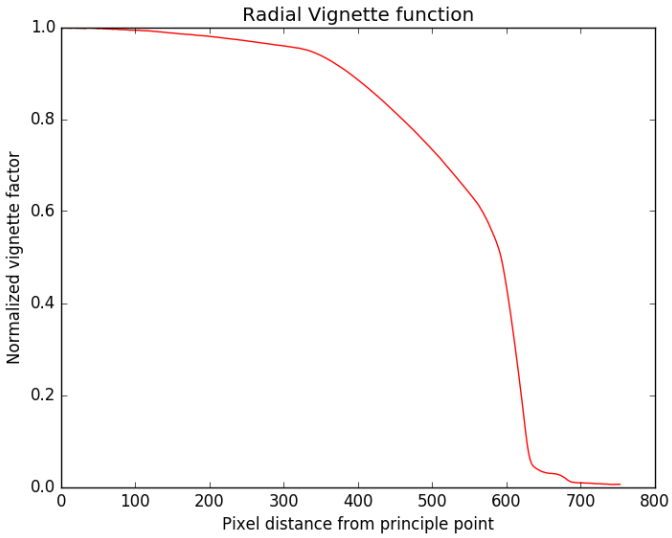


Figure 5.2: *Radial vignette function of the vignette shown in Figure 5.1. Data from* $setup_1$.

one can see the good correspondence between the two models. The radial model provides a well-suited alternative to the full model.

When the calibration sequence data is less ideal (Figures 5.4 to 5.6) the radial model already shows it's strength of smoothing local inconsistencies and averaging for areas where the dense vignette model contains artifcats. The radial vignette functions still provide a rather smooth curve and mostly have the expected concave shape of a vignette. The comparison with the 360 radial cuts already shows how much more the full model varies as soon as the points lie far from the center.

When being presented with very little usable data points, the full model is heavily compromised by artifacts (Figure 5.7). The radial model, while also suffering from local peaks and dips (Figure 5.8) still preserves its vignette-like shape while at the same time mapping the full data relatively well (Figure 5.9).

Figure 5.10 shows what happens with local anomalies. Probably due to lens flares, this dataset contained a very bright spot in the full model that affected the entire vignette due to it's high value in relation to which all other pixels are scaled. The radial model shows no anomaly at the same spot. While thresholding might also solve this particular issue for the full vignette, it shows the strength of the radial model dealing with aberrations as long as they are limited to a local area.

Figures 5.11 and 5.12 show the original and corrected image of one of the Aruco marker sequences. The more uniform brightness of the (supposedly) equally bright wall is especially noticeable in the false color image. A circular cut was performed on the image to exclude data with little value in the far edge regions where the calibration provides little to no data for the vignette.

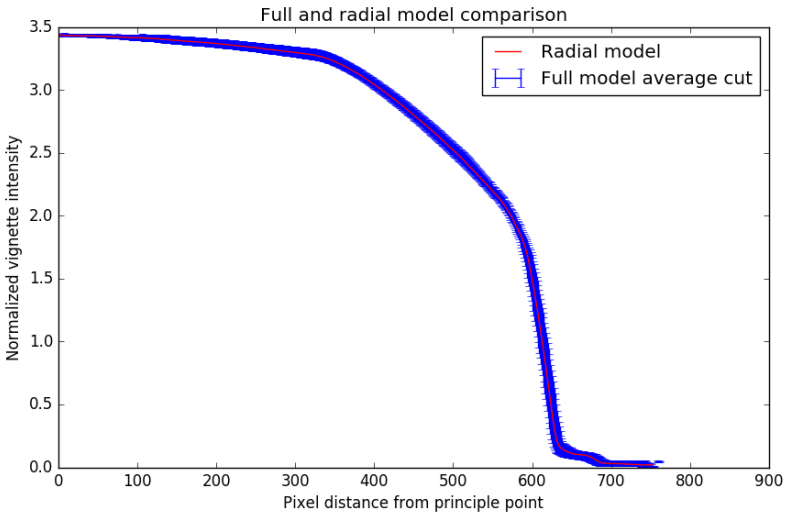


Figure 5.3: *Comparison between full model (averaged) and radial model. Data from* $setup_1$.

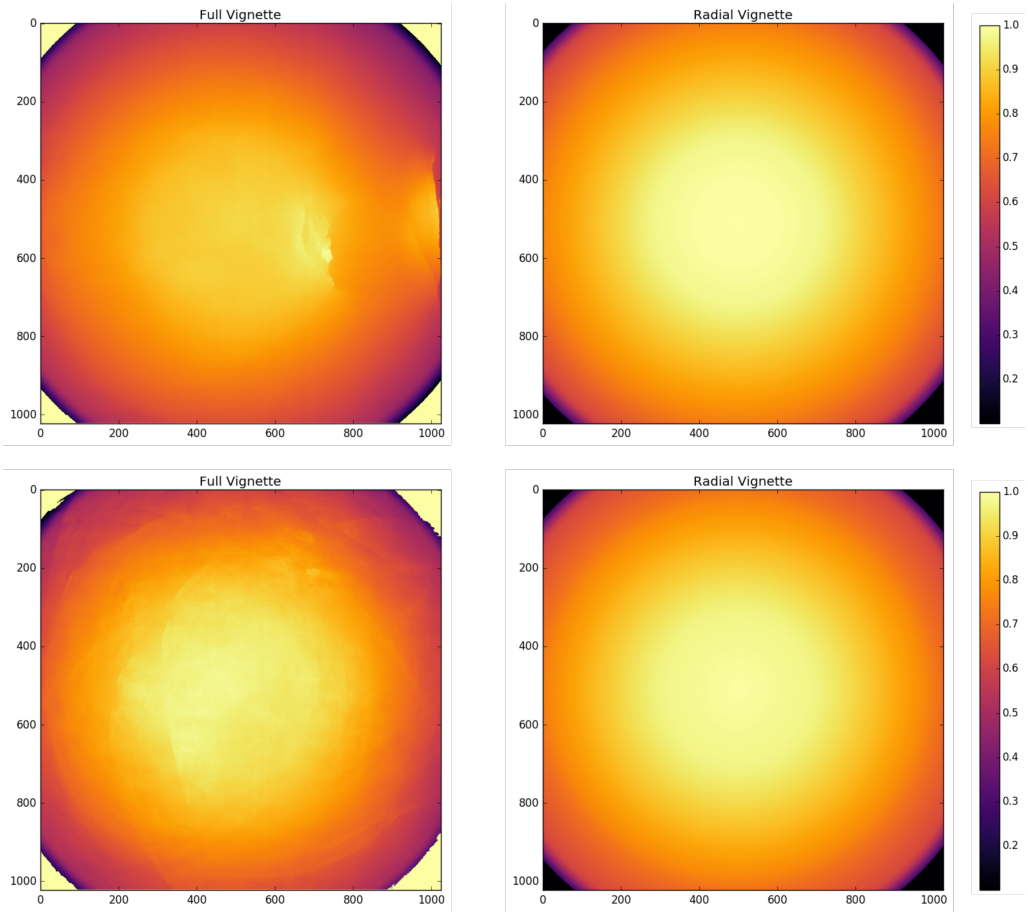


Figure 5.4: *Comparison between full model (left) and radial model (right) for datasets of mediocre quality. Data from* $setup_6$ *(top) and* $setup_4$ *(bottom).*

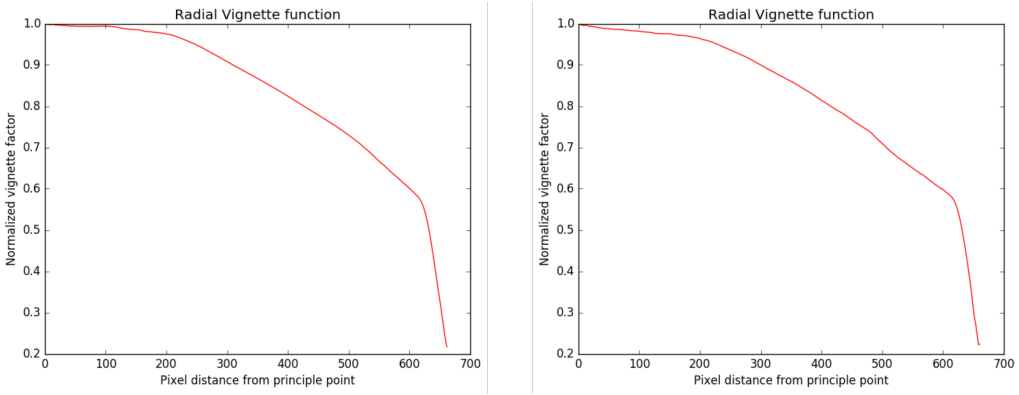


Figure 5.5: *Radial vignette functions of the vignettes shown in Figure 5.4. Data from $setup_6$ (left) and $setup_4$ (right).*

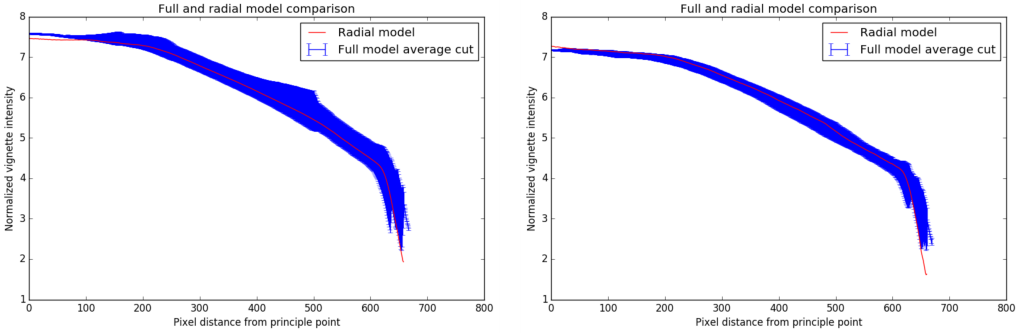


Figure 5.6: *Comparison between full model (averaged) and radial model. Data from $setup_6$ (left) and $setup_4$ (right).*

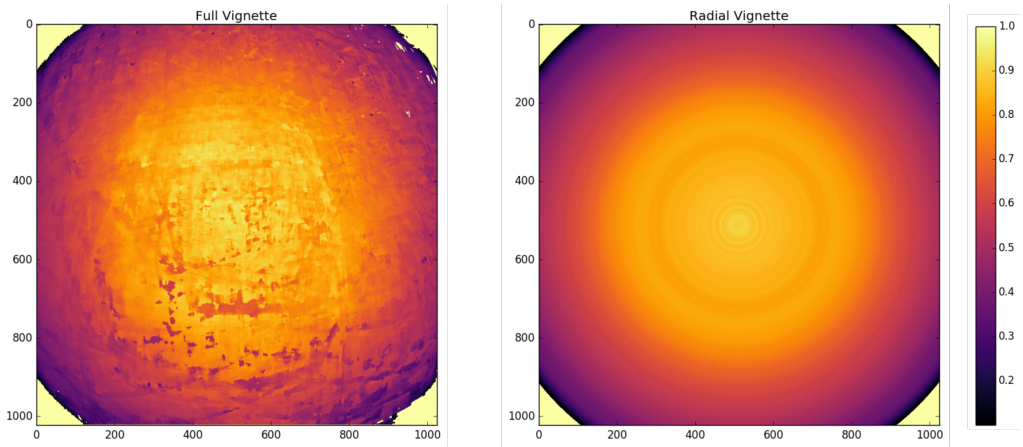


Figure 5.7: *Comparison between full model (left) and radial model (right) for a dataset of bad quality. Data from $setup_2$.*

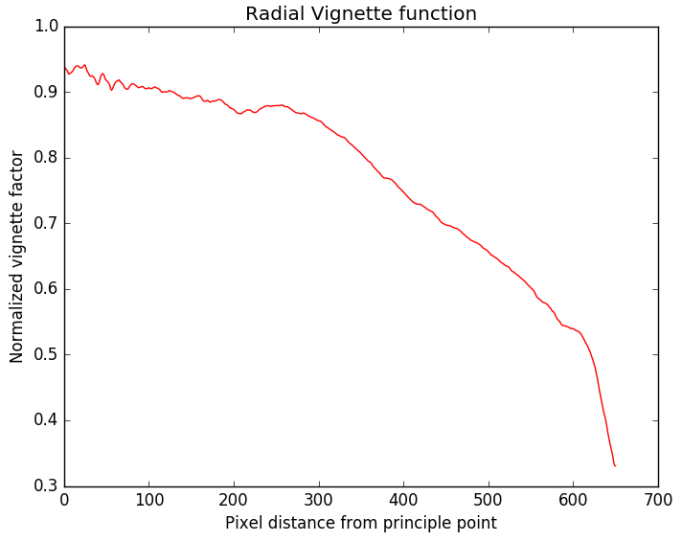


Figure 5.8: *Radial vignette function of the vignette shown in Figure 5.7. Data from* $setup_2$.

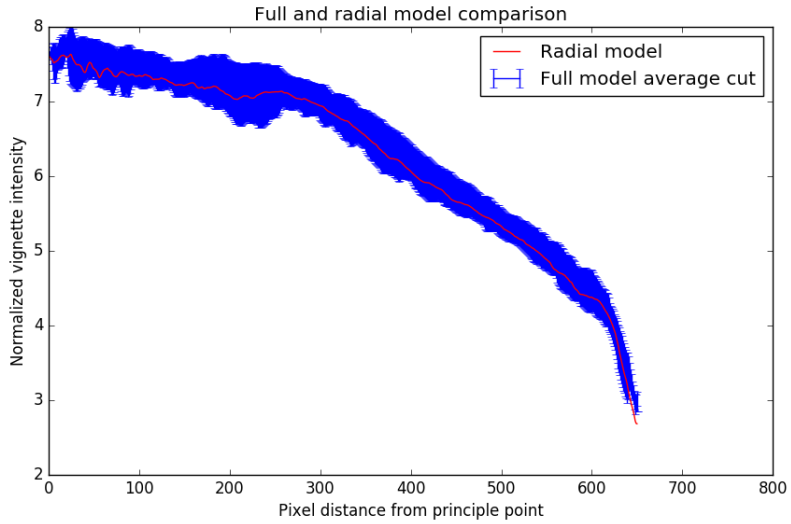


Figure 5.9: *Comparison between full model (averaged) and radial model. Data from* $setup_2$.

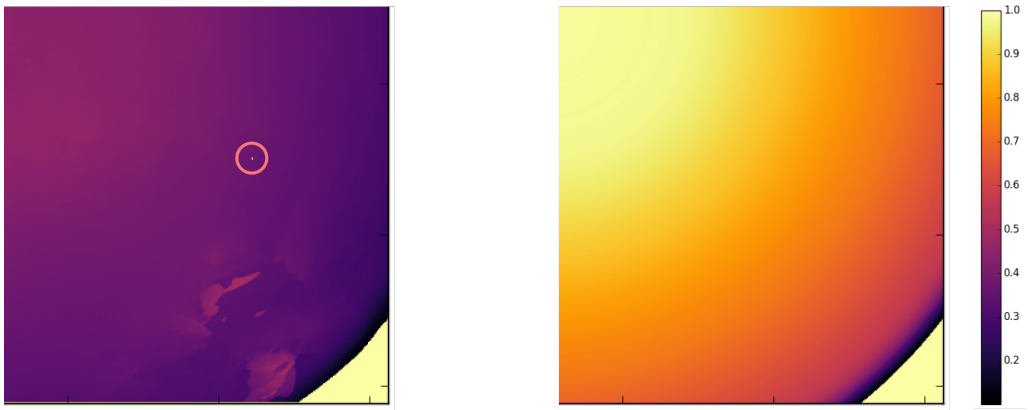


Figure 5.10: *Detailed comparison of a local anomaly in full and radial model. The full model has an overall much darker shade due to the abnormal pixel's high value and the relative scaling of the entire image. Data from* $setup_5$.

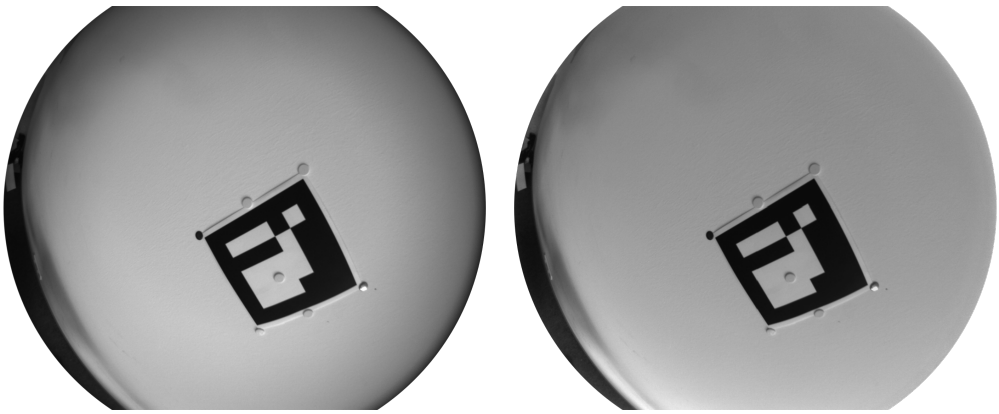

Figure 5.11: *Comparison of original (left) and corrected image (right) from one of the Aruco sequences. Circular cut to restrain the image region to usable data. Data from* $setup_1$.

Some of the data sets we recorded lead to mostly unusable data. Reflections especially (as seen in Figure 5.13) can lead to undesired results as the model unexpectedly receives increased instead of attenuated brightness values. Another issue that posed a problem was artificial lighting with fixed frequencies as present in flourescent tubes. When the frequency of the camera sensor is not properly aligned with the frequency of the illumination the resulting images' brightness is no longer independent of the time the images were recorded. The perceived attenuation in the middle of two light flashes of these tubes is considerably lower than at their peak, preventing proper vignette calibration.

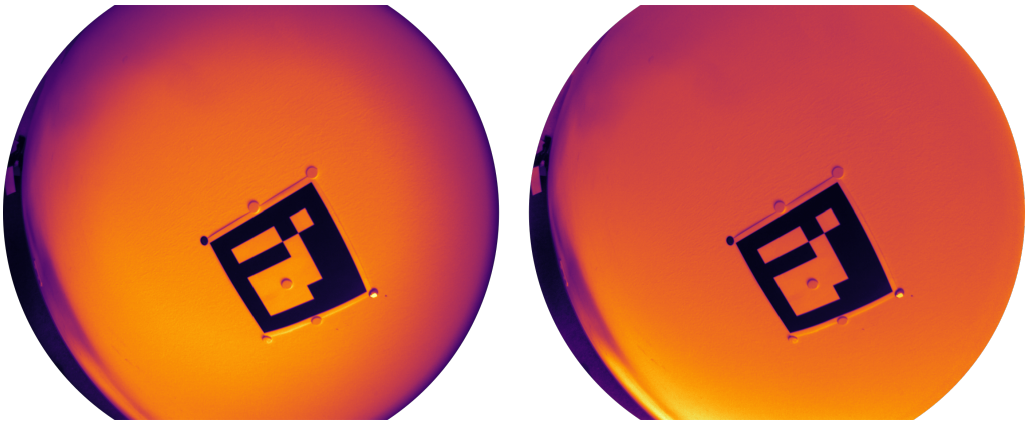

Figure 5.12: *Same images as Figure 5.11 but in false colors. The right picture shows a much more uniform distribution over the flat wall's surface.*

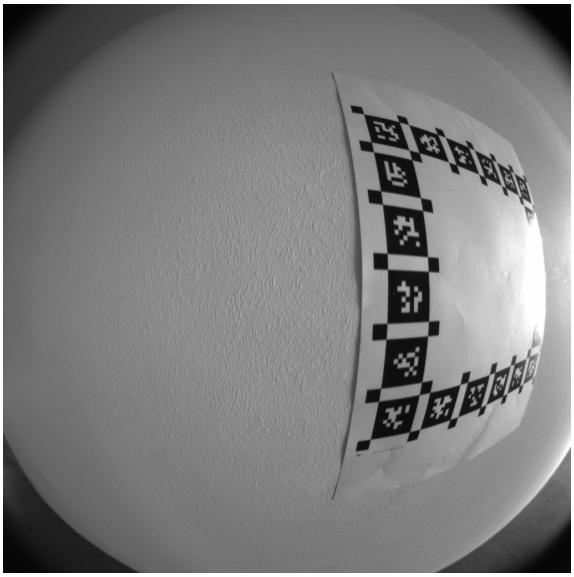

Figure 5.13: *Image from a sequence with natural lighting. The reflections present at steep angles lead to unusable values in that region. Data from* $setup_6$.

# Chapter 6

# Conclusion and Outlook

In this thesis we present an alternative model for vignette calibration of fisheye lenses. We implemented two suited camera models suited for these lenses and tested the calibration algorithm on multiple image sequences with two different marker sets. In comparison, the results showed a good accordance to the full vignette model while at the same time removing drawbacks like sensitivity to local inconsistencies and lack of data in few regions. This allows to also reduce the impact of few reflections in the data or shadows cast by the camera itself or the camera handler. The low amount of data needed for a decent vignette potentially offers the means to perform the calibration on already existing data (e.g. from intrinsic camera calibration) instead of recording explicit sequences for vignette calibration.

Further work could include parametrizing the radial function in a way to imitate common shapes of lens vignettes. This could make it even more robust at the cost of degrees of freedom provided by the current model.

# Appendices

# Appendix A

# Code

The calibration code is too extensive to fully list here and also contains previous work (see also (Engel et al., 2016)). It is however available in its entirety in my branch of the Mono-data-set-git at `https://gitlab9.in.tum.de/slam/mono_dataset_code/tree/maurice-devel`

## A.1 Evaluation code

Python evaluation code used for the comparison between the 360 cuts of the full model with the radial model:

```
1  import matplotlib.pyplot as  plt
2  import numpy as np
3  import os
4  import sys
5  import scipy
6  from PIL import Image
7  from mpl_toolkits.mplot3d import Axes3D
8  from optparse import OptionParser
9
10 def main():
11     parser = OptionParser()
12     parser.add_option("-f", "--folder", dest="folder", help="Specify save folder and path relative to which other file names are defined",default="./")
13     parser.add_option("-v","--vignettename",dest="filename", help="name (and location relative to FOLDER) of the vignette file",default="/vignette.png")
14     parser.add_option("-c","--camfile",dest="camFile",help="name (and location relative to FOLDER) of the camera intrinsics file",default="/camera.txt")
15     parser.add_option("-r","--radfile",dest="radFile",help="name (and location relative to FOLDER) of the radial vignette cut file (radFile.txt))",default="./radFile.txt")
16     parser.add_option("--centerarea",dest="area",type=float,nargs=4,help="pixel indices delimiting the area where to search for the maximum value, defining the center of the vignette. format: x_Start x_End y_Start y_End")
17     (options, args) = parser.parse_args()
18     folder = options.folder; filename = options.filename; camFileName = options.camFile; radFileName = options.radFile;
19
20     fileToOpen = folder+"/"+filename
21     center = np.array([1.,1.]);
22     img = plt.imread(fileToOpen)
```

```
23      s = img.shape
24      t = tilesPerSide = 7. #default value
25      if (not 'area' in locals() or not isinstance(area,tuple)):
26        area = np.round([np.floor(t/2)*(s[0]/t),np.floor(t/2+1)*(s[0]/t),np.floor(t↩
              /2)*(s[1]/t),np.floor(t/2+1)*(s[1]/t)])
27
28      radFile = open(folder + radFileName,'r')
29      radCut = np.array(radFile.readline().rstrip().split(' ')).astype(float)
30      radFile.close()
31      if(os.path.isfile(folder+camFileName)):
32        cameraFile = open(folder+camFileName,'r')
33        line = cameraFile.readline().split('\t')
34        cx = float(line[2]); cy = float(line[3])
35        if (cx<1.):
36          cx = cx*s[0]-.5; cy=cy*s[1]-.5
37        center = np.array((cx,cy))
38        print("Vignette center taken as principle point from camera file.")
39        cameraFile.close()
40      else:
41        center = findPeakInArea(img,area)
42        print("Vignette center estimated from values.")
43      print("Vignette center: {},{}".format(*center))
44      print("Image size: {},{}".format(*img.shape))
45
46      img = set1toNaN(img) #edges/corners without values will be set to NaN
47      cutArray = (cuts360Degrees(img,center))
48      means = np.array([]);
49      devs = np.array([]);
50      radCut = radCut[:cutArray.shape[1]]
51      alignmentStart = 20
52      alignmentEnd = -75
53      radCut = radCut[:]
54      cutArray = cutArray[:,:]
55      radMean = np.nanmean(radCut[:])
56      radCut = (radCut-radMean)/getMeanAbsoluteDeviation(radCut[alignmentStart:↩
            alignmentEnd]) #normalize
57
58      for cut in cutArray:
59          means = np.append(means,np.nanmean(cut[:]))
60          devs = np.append(devs,getMeanAbsoluteDeviation(cut[:]))
61      stDevs = np.zeros(cutArray.shape[1])
62      meanCut = np.zeros_like(stDevs)
63
64      for i in range(cutArray.shape[1]):
65        stDevs[i] = np.nanstd(cutArray[:,i])
66        meanCut[i] = np.nanmean(cutArray[:,i])
67      normalizerMean = np.nanmean(meanCut[:])
68      normalizerMean = 0
69      normalizerFactor = getMeanAbsoluteDeviation(meanCut[alignmentStart:↩
            alignmentEnd])
70      meanCut = (meanCut-normalizerMean)/normalizerFactor
71      stDevs = stDevs/normalizerFactor
72      print("radMean: {}, CutsMean: {}".format(np.nanmean(radCut),np.nanmean(↩
            meanCut)))
73      print("radDev: {}, CutsDev: {}".format(getMeanAbsoluteDeviation(radCut),↩
            getMeanAbsoluteDeviation(meanCut)))
74
75      fig=plt.figure(figsize=(10,6)); plt.title('Full and radial model comparison')↩
            ; plt.xlabel('Pixel distance from principle point'); plt.ylabel('↩
            Normalized vignette intensity')
76      x = range(meanCut.shape[0]); print(len(x));plt.errorbar(x,meanCut[:],stDevs↩
            [:],label='Full model average cut'); plt.plot(x,radCut,'r',label='Radial ↩
            model');plt.legend();#plt.show()
77
78      fig.savefig(folder+"meanCutWithEBarsFull.png",bbox_inches='tight'); plt.close↩
            (fig)
```

```
79      np.savetxt(folder+"/meanCutWithStDevs.txt",np.stack([meanCut,stDevs]),'%2.8f'↩
            )
80      np.savetxt(folder+"/360radialCuts.txt",cutArray,'%2.8f')
81      np.savetxt(folder+"/meansAndDevs.txt",np.stack([means,devs]),'%2.8f')
82
83  main()
84
85  def set1toNaN(array):
86      for i in range(array.shape[0]):
87          for j in range(array.shape[1]):
88              if (array[i,j] == 1):
89                  array[i,j] = np.nan
90      return array
91
92  def showSurfaceplot(dataArray):
93      fig = plt.figure()
94      ax = Axes3D(fig)
95      X = np.arange(0, dataArray.shape[1])
96      Y = np.arange(0, dataArray.shape[0])
97      X, Y = np.meshgrid(X, Y)
98      surf = ax.plot_surface(X,Y,dataArray,cmap='magma',linewidth=0, antialiased=↩
            False)
99      plt.show()
100
101 def get360Endpoints(center,length):
102     x = center[0]; y = center[1]
103     endpoints = np.zeros((360,2))
104     for i in range(360):
105         angle = i/180.*np.pi
106         Dx = length*np.sin(angle)
107         Dy = length*np.cos(angle)
108         endpoints[i,:] = [Dx+x,Dy+y]
109     return endpoints
110
111 def get360EndpointsNew(center,size):
112     x = center[0]; y = center[1];
113     w = size[0]-1; h = size[1]-1;
114     firstEndpoint = furthestCorner(center,size)
115     Ex = firstEndpoint[0]; Ey = firstEndpoint[1];
116     endpoints = np.zeros((360,2))
117     firstangle = np.arctan2(Ey-y,Ex-x)*180/np.pi
118     for i in range(360):
119         angle = (firstangle+i)%360
120         if (0 < angle and angle < 90):
121             Dx = w-x; Dy = h-y;
122             Ny = np.tan(angle/180.*np.pi)*Dx
123             if (Ny <= Dy):
124                 endpoints[i] = (w,y+Ny)
125             else:
126                 Nx = Dy/np.tan(angle/180*np.pi)
127                 endpoints[i] = (x+Nx,h)
128         elif (90 < angle and angle < 180):
129             Dx = x; Dy = h-y;
130             Ny = np.tan(angle/180.*np.pi)*(-Dx)
131             if (Ny <= Dy):
132                 endpoints[i] = (0,y+Ny)
133             else:
134                 Nx = (Dy/np.tan(angle/180*np.pi))
135                 endpoints[i] = (x+Nx,h)
136         elif (180 < angle and angle < 270):
137             Dx = x; Dy = y;
138             Ny = np.tan(angle/180.*np.pi)*(Dx)
139             if (Ny <= Dy):
140                 endpoints[i] = (0,y-Ny)
141             else:
142                 Nx = (Dy/np.tan(angle/180*np.pi))
```

```
143                 endpoints[i] = (x-Nx,0)
144         elif (270 < angle and angle < 360):
145             Dx = w-x; Dy = y;
146             Ny = np.tan(angle/180.*np.pi)*(-Dx)
147             if (Ny <= Dy):
148                 endpoints[i] = (w,y-Ny)
149             else:
150                 Nx = (-Dy/np.tan(angle/180*np.pi));
151                 endpoints[i] = (x+Nx,0)
152         else:
153             if (angle == 0):
154                 endpoints[i] = (w,y)
155             elif(angle == 90):
156                 endpoints[i] = (x,h)
157             elif(angle == 180):
158                 endpoints[i] = (0,y)
159             elif(angle == 270):
160                 endpoints[i] = (x,0)
161     return endpoints
162
163
164 def calculateEndpoint(Dx,Dy,corner,angle):
165     factor = (np.tan(np.arctan2(Dy,Dx)+(angle%180)/180.*np.pi))
166     if (factor==0):
167         xN = Dx
168     else:
169         xN = Dy/factor*(1-corner)       + Dx*(corner)
170     yN = Dx*factor*(corner)             + Dy*(1-corner)
171     return([xN,yN])
172
173 def interpolate(image, coordinates):
174     x = coordinates[0]; y = coordinates[1]
175     floorX = int(x);    floorY = int(y)
176     dx = x - floorX;    dy = y - floorY
177     dxdy = dx*dy
178     if((x<=0) or (y<=0) or (x>=image.shape[0]-1) or (y>=image.shape[1]-1)):
179         print("Point to interpolate outside of image")
180         return None
181     interpolated = (1 + dxdy - dx - dy) * image[floorX,floorY] \
182                    + (dx - dxdy)        * image[floorX+1,floorY] \
183                    + (dy - dxdy)        * image[floorX,floorY+1] \
184                    + dxdy               * image[floorX+1,floorY+1]
185     return interpolated
186
187 def furthestCorner(center,size):
188     x = center[0]; y = center[1];
189     w = size[0]; h = size[1]
190     return np.array([(0. if (x>(w-1)/2.) else w-1.),(0. if (y>(h-1)/2.) else h↩
        -1.)])
191
192 def centerToEndpointCoordinates(center,endpoint):
193     x = center[0]; y = center[1];
194     radialLength = np.sqrt(sum((endpoint-center)**2))
195     epsilon = 1e-8
196     pixels = int(np.ceil(radialLength-epsilon))
197     coordinates = np.zeros((pixels,2))
198     for i in range(pixels):
199         coordinates[i,0] = (x+i*(endpoint[0]-x)/radialLength)
200         coordinates[i,1] = (y+i*(endpoint[1]-y)/radialLength)
201     return coordinates
202
203 def findPeakInArea(img,area = [0,-1,0,-1]):
204     temp = np.zeros_like(img)
205     temp.fill(np.min(img))
206     temp[area[0]:area[1],area[2]:area[3]] = img[area[0]:area[1],area[2]:area[3]]
207     return np.unravel_index(np.argmax(temp,axis=None),img.shape)
```

```
208
209 def getMeanAbsoluteDeviation(array):
210     mean = 0;
211     mean = np.nanmean(array)
212     size = (~np.isnan(array).flatten()).sum()
213     mAbsDev = np.nansum((abs(array-mean)).flatten())/size
214     np.where(mAbsDev == 0, np.nan, mAbsDev)
215     return mAbsDev
216
217 def cuts360Degrees(img,center):
218     #cut off all cuts after the legnth of the minimal cut
219     w = img.shape[0]; h = img.shape[1]
220     x = center[0]; y = center[1]
221     minimalLength = (np.min([x,w-1-x,y,h-1-y]))
222     maximalLength = np.linalg.norm(furthestCorner(center,img.shape)-center)
223     cutArray = np.full((360,int(np.ceil(maximalLength))),np.nan)      #set to NaN ↩
            to exclude areas without values
224     print("Cutarray size: {}".format(cutArray.shape))
225     cutEndPointCoordinates = get360EndpointsNew(center,img.shape)
226     for i in range(cutEndPointCoordinates.shape[0]):
227         pixelCoordinates = centerToEndpointCoordinates(center,↩
                cutEndPointCoordinates[i])
228         temp = getRadialCut(img,pixelCoordinates)
229         cutArray[i,:temp.size] = temp
230         #plt.plot(pixelCoordinates[:,0],pixelCoordinates[:,1])
231     return cutArray
232
233 def getRadialCut(image,pixelCoordinates):
234     radialCut = np.zeros(pixelCoordinates.shape[0])
235     for i in range(pixelCoordinates.shape[0]):
236         interpolated = interpolate(image,pixelCoordinates[i,:])
237         radialCut[i] = interpolated if (interpolated.size==1) else interpolated↩
                [0]
238     return radialCut
```

Python code used for the radial plots:

```
import matplotlib.pyplot as  plt
import os
import sys

filename = sys.argv[1]+"/radFile.txt"
radFile = open(filename,"r")
print("Read file {0}".format(filename))
radData = [float(i) for i in radFile.read().split()]
plt.plot(radData[:],'r');plt.title('Radial Vignette function'); plt.xlabel('Pixel distance from principle point'); plt.ylabel('Normalized vignette factor')
wdir = os.getcwd()
print("Python working directory: {0}".format(wdir))
destination = sys.argv[1]+'/2DradVignetteCut.png'
plt.savefig('./{0}'.format(destination),bbox_inches='tight') #' /usr/stud/hermwill/Documents/project/python/radialVignette.png',bbox_inches = 'tight')
print("Saving file to {0}/{1}".format(wdir,destination))
```

# Bibliography


Aldrich, J. (1997). R. a. fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 12(3):162–176.

Commons, W. (2018). File:fisheye-nikkor auto 6mm f2.8 lens 2015 nikon museum.jpg — wikimedia commons, the free media repository. [Online; accessed 27-September-2018].

Devernay, F. and Faugeras, O. (2001). Straight lines have to be straight: Automatic calibration and removal of distortion from scenes of structured enviroments. *Mach. Vision Appl.*, 13(1):14–24.

Engel, J., Koltun, V., and Cremers, D. (2018). Direct sparse odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625.

Engel, J., Usenko, V. C., and Cremers, D. (2016). A photometrically calibrated benchmark for monocular visual odometry. *CoRR*, abs/1607.02555.

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F. J., and Marín-Jiménez, M. J. (2014). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47:2280–2292.

Geyer, C. and Daniilidis, K. (2000). A unifying theory for central panoramic systems and practical applications. In *Proceedings of the 6th European Conference on Computer Vision-Part II*, ECCV '00, pages 445–461, London, UK, UK. Springer-Verlag.

Heeling, P. (2018). *Rain pool*. `https://skitterphoto.com/photos/4240/rain-pool`[Accessed:20.09.2018].

Kannala, J. and Brandt, S. S. (2006). A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1335–1340.

Khomutenko, B., Garcia, G., and Martinet, P. (2016). An enhanced unified camera model. *IEEE Robotics and Automation Letters*, 1(1):137–144.

Kneip, L., Li, H., and Seo, Y. (2014). Upnp: An optimal o(n) solution to the absolute pose problem with universal applicability. In *ECCV*.

Olson, E. (2011). Apriltag: A robust and flexible visual fiducial system. In *ICRA*, pages 3400–3407. IEEE.

Rituerto, A., Puig, L., and Guerrero, J. J. (2010). Comparison of omnidirectional and conventional monocular systems for visual slam.

Salgado, J. F. (2015). File:imagine being a fly vlt.jpg — wikimedia commons, the free media repository. [Online; accessed 27-September-2018; By ESO/José Francisco Salgado (josefrancisco.org) (http://www.eso.org/public/images/potw1049a/) [CC BY 4.0 (https://creativecommons.org/licenses/by/4.0) or CC BY 3.0 (https://creativecommons.org/licenses/by/3.0)], via Wikimedia Commons].

Usenko, V. C., Demmel, N., and Cremers, D. (2018). The double sphere camera model. *CoRR*, abs/1807.08957.

Yang, N., Wang, R., and Cremers, D. (2017). Feature-based or direct: An evaluation of monocular visual odometry. *CoRR*, abs/1705.04300.

Zhang, Z., Rebecq, H., Forster, C., and Scaramuzza, D. (2016). Benefit of large field-of-view cameras for visual odometry. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 801–808.

# Acknowledgments

This thesis and all its contents would not have been possible without the support of numerous people that I would like to thank:

- **Prof. Dr. Daniel Cremers** for supervising the thesis and his very well taught lectures
- **Nikolaus Demmel** as advisor for great patience and advice as well as being insightful and helpful overall
- **Alexandra Kornhaas** for comfort, motivation and understanding even in difficult situations
- My **friends and family** for constant support and help throughout my studies