Karlsruher Institut für Technologie

Institut für Thermische Verfahrenstechnik

Development and Optimization of an Optical System and an Image Processing Code to Investigate the Particles Motion in a Solar Particle Receiver

Master thesis

cand. M.Sc. Lana Fionnguala Liebl

November 20, 2019 - July 6, 2020

Advisor: **Prof. Dr.-Ing. Thomas Wetzel**
Second advisor: **Prof. Dr. Reinhard Rauch**
Supervisor: **Serdar Hicdurmaz, Dr.-Ing. Martina Neises-von Puttkamer**
Institute of Solar Research, German Aerospace Center, Stuttgart

Karlsruher Institut für Technologie

Institut für Thermische Verfahrenstechnik

Masterarbeit

für
Frau B.Sc. Lana Liebl

Entwicklung und Optimierung eines optischen Systems und eines Bildverarbeitungsalgorithmus zur Untersuchung der Partikelbewegung in einem solaren Partikelreceiver

Development and Optimization of an Optical System and an Image Processing Code to Investigate the Particles Motion in a Solar Particle Receiver

Solar Towers using sand-like granular particles as Heat Transfer Fluid (HTF) have various advantages over molten salt based solar receivers, which are state-of-the-art Concentrated Solar Tower technology, like higher maximum temperature in the receiver and potentially lower operational and component costs. The Centrifugal Particle Receiver is a promising design compared to other particle receiver concepts because it allows an adjustment of particle residence time and thus particle outlet temperature. In the centrifugal particle receiver, the ceramic particles with various diameters (0.3 – 1 mm), which are accelerated centrifugally and gravitationally, are moving in an inclined rotating drum while at the same time being exposed to solar radiation that enters through the aperture. On-sun tests have shown that the particles can be heated up to 965 °C in a 2.5 MWth prototype.

Depending on the receiver rotational speed and the inlet particle mass flow rate, an optically opaque but sufficiently thin particle film can be achieved near the receiver wall. However, the particles, forming a particle film altogether and following helical-like trajectories, move relative to neighboring particles and eventually mix in all directions (radial, tangential and axial) due to sliding and rolling over neighboring particles. For a better understanding of the movement of single particles in the film, a numerical DEM-model (Discrete Element Method) is being developed at the moment, which will be validated by experimental results.

The first main objective of this work is to design and optimize an optical system consisting of GoPro cameras and diffusive lamps that will be integrated to a lab scale Centrifugal Receiver with ~ 0.8 m of cavity diameter. By means of this optical set-up, the particle motion can be detected in the receiver. The second main objective is the development of a particle tracking code to be used in image analysis of recorded particle flows.
The main tasks are as follows:

- Literature review of particle-based solar receiver systems and their experimental test set-ups
- Geometric characterization of ceramic particles by using sieve analysis
- Conducting experiments with stationary particle-tracer mixture to optimize the optical design and check its compatibility to the main system
- Image Processing Code Development in Python for tracer (white particle) tracking
- Image Processing Code Development in Python for particle film thickness evaluation
- Conducting experiments to detect and calibrate the effects resulting from the nature of camera and development of Image Processing Code in Python in order to generate corrected data for post-processing

The written documentation should include a clear and concise presentation of the work carried out. The results of the work will be presented within the Institute Colloquium.

Ausgabe der Arbeit: 20.11.2019
Abgabe der Arbeit:

Aufgabensteller: Prof. Dr.-Ing. Thomas Wetzel
Betreuer: Serdar Hicdurmaz, Martina Neises-von Puttkamer

Declaration of originality

I hereby declare that I have composed this paper by myself and without any assistance other than the sources given in my list of works cited. This paper has not been submitted in the past or is currently being submitted to any other examination institution. It has not been published. All direct quotes as well as indirect quotes which in phrasing or original idea have been taken from a different text (written or otherwise) have been marked as such clearly and in each single instance under a precise specification of the source.

I am aware that any false claim made here results in failing the examination.

Karlsruhe, July 6, 2020

Lana Fionnghuala Liebl

Acknowledgement

I would like to express my sincere gratitude to Prof. Dr.-Ing. Thomas Wetzel (KIT) and to my supervisors Serdar Hicdurmaz (DLR) and Dr.-Ing. Martina Neises-von Puttkamer (DLR) for giving me the opportunity to examine this field of research for writing my master thesis. I want to say thank you to Prof. Dr.-Ing. Thomas Wetzel especially for the uncomplicated and quick communication.

I wish to expend my special thanks to Serdar Hicdurmaz for the technical discussions throughout the thesis as well as the support to finalize the thesis.

I wish to show my appreciation to all members of the research group for their straightforward support. I would like to express my special gratitude to Tamara Uhlig, Andreas Krause and Jens Rheinländer for the organizational assistance during the experiments in home office.

Last, but not least, I would like to thank my family and my boyfriend for their unlimited support during my studies and especially throughout the thesis period.

Contents

Symbols and Acronyms

Roman symbols

Symbol	Unit	Description
A		camera intrinsic matrix
a	m/s^2	acceleration
C		optical center
d	m	distance
f	$1/s$	frame frequency
f	m	focal length
FT	m	film thickness
g		gray level pixel value
k		radial distortion coefficient
l	m	length
M	m	point in world coordinate system
m		projected point on image plane
m	kg	mass
n		number
p		tangential distortion coefficient
$Q_3(x_\mathrm{i})$	$\%$	cumulative distribution
q	m	skew parameter
$q_3(x_\mathrm{i})$	$1/m$	mean distribution density
R	$1/m$	rotation matrix
s		scale factor
t		translation vector
u		tracer position
v	m/s	velocity
w		mass fraction
X, Y, Z	m	world coordinates
$\mathrm{X_c}$, $\mathrm{Y_c}$, $\mathrm{Z_c}$		camera coordinates
x, y, z		pixel coordinates
x	m	sieve size
Δt	s	time step
Δx, Δy		distortion correction function

Greek symbols

Symbol	Unit	Description
α	°	angle
σ		standard deviation

Subscripts

Symbol	Description
CW	camera and wall
d	distorted
FOV	field of view
id	ideal (distortion free)
len	lengthwise
max	maximum
maxdot	most upper detected particle
min	minimum
n	frame number
pixel	pixel dimensions
rad	radial
stat	static angle of repose
tan	tangential
vertical	camera's vertical angle of view
x, y, z	directions x, y, z

Acronyms

Acronym	Description
CentRec	centrifugal particle receiver
CSP	concentrated solar power
DEM	discrete element method
DLR	Deutsches Zentrum für Luft- und Raumfahrt
FOV	field of view
fps	frames per second
HSV values	hue saturation value of a pixel
HTM	heat transfer medium
HT storage	high temperature storage
LED	light emitting diode
LT storage	low temperature storage
PSD	particle size distribution
PV cells	photovoltaic cells
RGB values	red green blue value of a pixel
ROI	region of interest
SG	Saint-Gobain
SM	storage medium

1 Introduction

A growing population, climate change and an increasing demand of energy and electricity are the major challenges humanity has to face in the twenty-first century. In the world energy outlook 2019, the international energy agency predicts a rise in the global energy demand of 1.3 % each year until 2040 [1]. As reported by the international renewable energy agency in 2019, the energy-related CO_2 emissions need to be shortened by 3.5 % per year until 2050 and further reductions are necessary afterwards, in order to meet the aims of the Paris agreement [2]. To expedite the decarbonization and achieve the sustainability goals, replacing fossil fuels by renewable energy sources (sun, wind, water, biomass, geothermal) is inevitable. However, the share of the renewable energies in electricity generation was only 26 % in 2018, which is why the electricity production from renewables needs to be scaled up [3]. Drastically rising renewables, such as solar energy, as power source for electricity generation, while increasing the usage of electricity for heating and transportation, could reduce the global CO_2 emissions significantly [4].
Up to now, the generation of electricity using wind and solar photovoltaic (PV) are the cheapest options in many markets, but other renewable technologies, such as concentrated solar power (CSP), bioenergy or geothermal, are expected to be cost competitive within the next decade [5]. According to the international renewable energy agency, the cost of all commercially available renewable technologies for electricity generation decreased in 2018 [5]. Especially, the reduction of the global weighted-average cost of electricity for CSP is notable, as it was reduced by 26 % compared to the previous year [5]. Since the technology costs for CSP plants have remarkably fallen in the last decade, the global capacity of CSP plants enlarged over the years from 5.1 GW in 2016 [6] to predicted 9.6 GW in 2023 [7]. Mainly responsible for this growth are the falling prices and the fact, that most of the future CSP plants are equipped with a thermal storage with increasing size [7]. The international renewable energy agency estimates, that the share of CSP in the global electricity generation in 2050 will be 4 % [8] and, if the technology develops following the current trends, CSP offers the possibility of covering up to 25 % of the global energy demand [6]. As both, wind and solar PV, are not able to deliver continuous and constant power, energy storage capabilities, management based on the demand and flexible power generation installations are necessary in order to sustain grid stability. Therefore, CSP plants with thermal energy storage are an auspicious option for locations with high direct normal irradiation throughout the whole year [9]. To further improve and increase energy production using CSP, the plant components need to be optimized. Thus, research is done on innovative receiver technologies, such as particle receivers.

In the scope of this thesis, a solar tower plant with sand-like granular particles as heat transfer medium (HTM) is considered. This CSP technology offers many advantages, such as high maximum receiver temperature and the potential of relatively low operation and component costs. The investigated centrifugal particle receiver CentRec provides a good regulation of the receiver outlet temperature, enabling the system to achieve the desired temperature for the downstream power block.

In the centrifugal particle receiver the particles are fed into an inclined rotating drum, where they are accelerated by gravitational and centrifugal force and heated up by the solar radiation, which is inciding through the aperture of the receiver drum. To optimize the receiver efficiency, knowledge of the particle motion, such as the relative movement to their neighbors or potential radial, tangential and axial mixing, is necessary. Therefore, the main purpose of this thesis is the development and optimization of an optical system and corresponding image processing codes to investigate the particle motion in the solar particle receiver. In order to do so, the fundamentals of CSP, particle receivers and optical basics are presented in chapter 2. Chapter 3 provides an overview of the materials used in the experiments and presents the results of the geometric characterization of the ceramic particles and the tracers. Furthermore, the methodology and results of the design and optimization of the optical system to investigate the particle motion in the solar particle receiver are summarized. The developed image processing codes for tracer tracking and the determination of particle film thickness are explained in detail in chapter 4. Lastly, a summary and outlook can be found in chapter 5.

2 Fundamentals

In this chapter the fundamentals of concentrating solar power such as the potential of solar energy and the principal assembly of a concentrating solar power plant are described. A more detailed look into the receiver and especially into particle receiver is presented. Moreover, some optical basics and fundamentals of camera calibration are explained.

2.1 Utilization of solar energy

Solar radiation can be used for the generation of electricity or heat (high-temperature process heat or low-temperature heat). In flat plate collectors, the solar radiation is transformed into thermal energy and can be supplied for domestic use, e.g. for generation of hot water [10]. For the use of solar energy to produce electricity, there are two main technologies: PV cells and solar thermal plants. PV cells exploit the photoelectric effect to convert the solar radiation directly into electrical power [11]. Contrarily, the solar thermal plants generate process heat by conversion of the solar energy, which can be used in heat engines to generate electricity [12]. Following Carnot's theorem, the efficiency of a heat engine is increasing with increasing difference between hot and cold reservoir. Therefore, the temperature, generated by the solar radiation, is aimed to be as high as possible in order to maximize the temperature of the working fluid in the downstream process, which results in higher efficiency.

2.2 Concentrated solar power

Since the solar power inciding on the earth is relatively low, the solar radiation is optically concentrated first, in order to achieve the power density necessary for the generation of electricity in thermal solar power plants. For concentrating the solar radiation, mirrors in different shapes are utilized, as explained in the next section. The global solar-to-electricity efficiency of solar thermal plants is between 10 and 30 % [13; 14], depending on the collector shape, the material properties and the inclination angle. The solar-to-electricity efficiency includes the solar-to-heat efficiency, that describes the transformation of the energy from solar radiation to heat through collector, concentrator and receiver, and the heat-to-electricity efficiency of the conversion in turbines [14].

2.2.1 Collector types

The main part of solar power plants is similar to conventional power plants, except that the steam generators are not fired with fuel, but instead solar energy is used. The required high temperatures for the steam process imply the concentration of the solar radiation. [15]
To concentrate the solar radiation, mirrors are used to collect the incident solar rays and to focus them onto a smaller area. Depending on the collector shape, CSP systems are divided into line and point focus systems. In parabolic trough and linear Fresnel systems, the solar radiation is collected

with mirrors and focused onto a focal line. On the contrary, point focus collectors, such as dish Stirling (paraboloidal dish) or solar tower plants, focus the solar radiation collected with mirrors onto a single focal point. As the point focus systems focus all the collected solar energy onto a smaller area than the line focus ones, the achieved concentration is higher and hence, the reachable temperatures in the heat transfer medium are higher. Line focus systems reach concentration factors of around 8 - 80 (parabolic trough, [13; 16]) and 25 - 100 (Fresnel, [13]) and consequently, receiver temperatures up to around 400°C [16]. In contrast, point focus systems can reach concentration factors between 300 - 1000 (solar tower, [13]) and 1000 - 3000 (dish, [13]) and therefore, receiver temperatures up to 1000°C [13; 17; 18]. The annual solar efficiency of parabolic trough systems is around 10 - 16 % [13; 18]. Because of their simpler optical design, the annular solar efficiency of Fresnel collectors is 33 % less than the one of parabolic trough collectors (8 - 12 %, [13; 18]) [19]. Solar tower plants show an annual solar efficiency between 10 - 25 % [13; 18] and dish Stirling engines offer the highest efficiencies (16 - 29 %, [18]). The higher efficiencies of the point focus systems are derived directly from the higher temperatures reached in the focal points.
The parabolic trough collector technology is the most mature and the most often installed technology in solar fields. Fresnel collectors are similar to parabolic trough collectors, but instead of bowed mirrors, the mirrors are flat, aiming at a simpler design and consequently lower costs. The paraboloidal dish solar concentrator needs a stable supporting structure and two-axis tracking to evolve its high energy efficiency. Thus, the cost of generation is relatively high for this collector type and no storage is available. Solar towers provide high efficiencies as well and additionally, offer the possibility of increasing the annual solar efficiency to $\geq 65\,\%$ [17] by storing the thermal energy. [13; 17; 18]

2.2.2 Assembly of a concentrating solar power plant

Figure 2.1 shows a schematic figure of a concentrating solar power plant. The sun rays incide onto the mirrors, which form the so called solar field. The mirrors bundle the solar rays and direct them to the focal point, where the receiver is located. The main task of the receiver is to collect the solar energy and transfer it to the HTM, which is flowing through the receiver. There are various receiver concepts with different ways to heat up the HTM. A distinction is made between direct heating of the HTM by the solar radiation and indirect heating of the HTM. A detailed description of different receiver designs is given in section 2.3. In figure 2.1 a direct heating particle receiver is used to heat up the HTM. In this case the particles are not only used as HTM, but also as storage medium (SM).

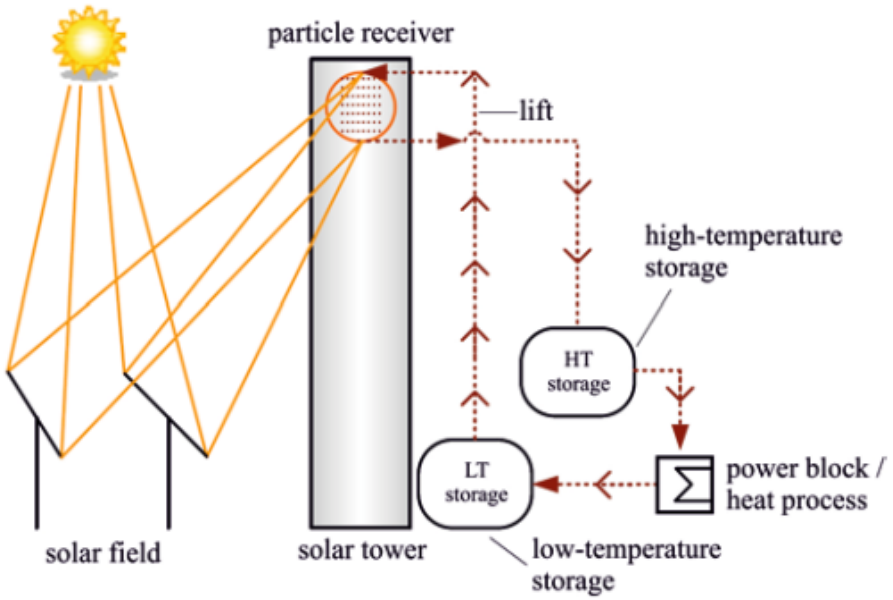


Figure 2.1: Schematic figure of a CSP plant with a particle receiver. Particles are both HTM and SM [20]

The particles are heated up in the receiver and transported to the heat exchanger, which is connected to a power block (generation of electricity) or to a process, in which the heat is used, e.g. in a chemical reaction. The cooled particles leave the heat exchanger towards the receiver, where they are heated up again. This circular flow can be complemented by a storage system, in order to extend the overall efficiency and the plant's capacity of utilization. If the available solar energy is higher than the demand, particles from the low temperature storage (LT storage) are fed into the circular particle flow, heated up in the receiver and stored in the high temperature storage (HT storage). If the solar radiation eases off, the stored hot particles are fed into the system and used in the heat exchanger as an additional heat source for the downstream process. Subsequently, the cooled particles are partly poured back into the LT storage. [20]

2.3 Receiver

The main task of the receiver is to absorb the concentrated solar radiation and transfer it to the HTM. Thus, the receiver can be considered as a heat exchanger. Increasing the thermal performance of the receiver, reduces the required size of the heliostat field and therefore, leads not only to higher efficiencies, but also to lower capital costs. Hence, many researchers put effort into the receiver optimization and the development of new technologies. In the literature many different receiver design options can be found and this section gives a rough overview of different receiver concepts.
One distinguishing characteristic is the geometrical configuration of the surface, which is absorbing the concentrated solar radiation. Moreover, receivers can be divided into direct and indirect absorption receivers, according to the manner of energy transfer to the HTM, or they can be classified by the HTM used. [21; 22]

2.3.1 Geometrical configuration

Depending on the design of the geometrical configuration, receivers can be constructed as external or cavity receivers. While external receivers have a surface that is completely exposed to the surrounding, cavity receivers are closed structures with an open or windowed aperture. Since the absorbing surface is exposed to the surrounding, the radiative and convective losses are higher in external receivers than in cavity receivers. Therefore, the reachable operating temperatures of external receiver applications are below approximately 730°C [22]. In cavity receivers, the solar radiation is absorbed inside the cavity, resulting in high absorption efficiency and therefore, in higher reachable temperatures of the HTM [22]. Generally, in cavity receivers the pressure is limited, commonly to below 15 bar, whereas in external receivers higher pressures can be applied. However, the temperature in external receivers is significantly lower than in cavity receivers.[23]

2.3.2 Absorption and heat transfer in the receiver

In indirect absorption receivers the HTM is not directly exposed to the solar radiation, but the solar radiation is absorbed by an absorber material. An example of indirect absorption receivers are tubular receivers. The external lateral surface of the receiver tubes absorbs the solar radiation, which is transformed into heat. By conduction through the tube wall, the heat is carried to the internal lateral surface of the receiver tubes and from there transferred to the HTM, which is passing in upward direction through various small vertically arranged tubes. The additional heat transfer step through tube wall and the high reflection losses are the biggest disadvantages of tubular receivers. In recent tubular receivers molten salt, sodium and water, which can be used for direct steam generation, have been investigated and applied as HTM. [24]
Other examples for indirect absorption receivers are volumetric receivers made of metal grids, reticulated wires or porous structures, such as honeycombs or porous ceramics. The structures are irradiated by concentrated sunlight and thereby heated up. The HTM, for example air, flows through the structure and is heated by forced convection. [25; 26]
In direct absorption receivers the HTM is directly exposed to solar irradiation. Thus, the heat transfer between an additional absorbing material and the HTM, as it is the case in indirect absorbing receivers, is eliminated. Direct absorption receivers also offer the advantage of avoiding technical restrictions by structural materials, that can not be heated up to temperatures as high as the HTM, e.g. particles, can be. [26]
An example for a direct absorption receiver is the falling particle receiver. Basically, the falling particle receiver in cavity shape is the simplest direct absorption receiver. The particles are poured into the receiver from the top and fall through the receiver, driven by the gravitational force. While falling, the particles are heated up by being irradiated directly with concentrated sunlight. The hot particles can be used for power generation in the power block or filled into the thermal storage tank. [27]

2.3.3 Heat transfer medium

Besides the receiver concept, the choice of the HTM is of crucial importance during the design of a CSP plant. Depending on the required process temperature an adequate and stable HTM should be chosen. Generally, typical HTMs are molten salt, water/steam (saturated and superheated), air or

other gases, thermal oils or organics, but research work is also put into liquid metals [28; 29], solid particles [30; 31; 32] and other gases, such as supercritical carbon dioxide [33] or helium [34].
Table 2.1 shows the range of working temperatures of different HTMs.

Table 2.1: Working temperature range of different HTMs

HTM	Temperature in °C	Literature
Organics	12 - 393	[35; 36]
Thermal oils	-20 - 400	[35; 37]
Water/steam	250 - 300 (saturated steam, 45bar) 249 - 565 (superheated, 160bar)	[38] [39]
Air	up to 680 - 800	[40; 41]
Liquid metals	98-883 (Na) up to 1533 (Pb–Bi)	[42] [35]
Molten salt	238 - 600 some up to 900	[43] [35]
Solid particles	more than 1000	[44]

Since the HTM is one of the most important components of CSP plants and needed in a large amount, the aim is to find the optimal combination of performance and price. Thereby, the desired characteristics of the HTM are a low melting and a high boiling point (liquid HTM) as well as thermal stability, high thermal conductivity and high heat capacity. Moreover, low vapor pressure at high temperatures and low corrosive properties with metal alloys as well as no toxicity and chemical stability are preferable. [35; 45]
Table 2.1 shows, that thermal oils and other organics are not suitable as HTMs in applications with high temperatures. The combination water/steam provides the advantage of efficiency increase by direct steam generation and its use as both the HTM and the working fluid in the power cycle, resulting in higher inlet temperatures of the turbine [46]. Air is a cheap, available and non-toxic HTM, that offers a wide working temperature range, but the limited heat transfer and the great power needed for pumping are disadvantages of air as HTM [25; 42]. Liquid metals are stable at high temperatures and offer the possibility of high receiver outlet temperatures [42]. Additionally, they are found to be able to reduce costs, due to their high heat transfer efficiency at high temperatures [47]. Molten salts offer the possibility to be used as HTM and SM. Relatively high temperures can be reached and they are cheap, compared to thermal oils or organics [35]. Furthermore, molten salts are utilized in many CSP plants, because of their high heat capacity as well as their good thermal and physical properties at high temperatures [48]. However, their corrosive properties have to be considered in the design of solar plants using molten salt as HTM [35].
Following Carnot's theorem, a higher temperature difference between hot and cold temperature leads to higher process efficiencies. Therefore, higher hot temperatures in the receivers are desirable to increase the overall thermal-to-electricity efficiency of the solar plant. Increasing temperatures not only restricts the availability of structural materials, but also affect the selection of the HTM. The commonly used molten salt ($NaNO_3$, KNO_3) can only be used up to 600 °C, cf. table 2.1. Higher temperatures lead to thermal decomposition of the molten salt [49]. For temperatures around 1000°C and higher, other HTMs, such as particles or liquid metals are investigated [50; 51].

2.4 Particle receiver

As described in the section above, particle receivers are currently in development for high temperature CSP plants. In this section the particle properties and the general design of falling particle receivers are explained. In particular, the centrifugal particle receiver CentRec, developed at the DLR, is presented.

2.4.1 Particle properties

As ceramic bauxite particles are used in the industry (gas and oil production), they are available in large quantities and are expected to have low costs [52; 53]. Moreover, these nearly black ceramic particles present good optical absorption properties over the solar radiation spectrum and are thermally stable up to 1000 °C [54]. This offers the possibility of high process temperatures and therefore, high process efficiencies. As the particles have high heat capacities and are suitable for storage, the use of the particles not only as HTM, but also as SM, gives the opportunity to produce electricity additionally during the night and reach higher efficiencies. In comparison to molten salt, the transport of particles is simpler and their usability does not depend on the temperature. The particles can be carried mechanically or pneumatically and there is no need for additional pipe heating to avoid solidification at lower temperatures.

2.4.2 Falling particle receiver

The most common particle receiver is the falling particle receiver, in which a falling particle film absorbs the concentrated solar radiation directly. An exemplary particle fall film receiver concept is shown in figure 2.2. Here, the receiver is built as a face-down cavity receiver, so that the aperture of the receiver faces the ground. With this receiver configuration, the convection losses are reduced without the necessity for covering the aperture with a window [55]. The cavity design also diminishes the receiver losses due to reflection, because part of the reflected radiation is absorbed by the particles inside the cavity receiver. Whereas the particle film in the centrifugal particle receiver CentRec, cf. section 2.4.3, is mostly optically opaque, the one in a falling particle receiver is often optically translucent. Consequently, the uncovered parts of the receiver are irradiated and heated up as well, leading to lower receiver efficiencies.
Recirculation and therefore, higher particle mass flows resulting in more opaque particle curtains are investigated to increase the receiver efficiency. Furthermore, the residence time of the particles in falling particle receivers is defined by their velocity of fall. This is the maximum time, that the particles are irradiated by the concentrated sunlight. In order to reach the requested outlet temperature, the particles might need to pass through the receiver several times. Due to reduced reflection losses in the face-down cavity receiver the recirculation is able to increases the receiver efficiency up to 92% [55].

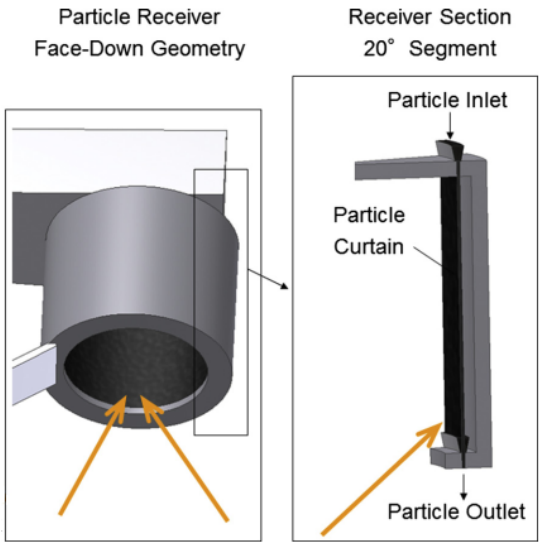


Figure 2.2: Particle receiver face-down geometry view from below (left) and 20° segment of receiver section with particle curtain (right) [55]

Besides the fall film receiver, there are other downflow particle receiver concepts, such as the choked-flow and the centrifugal particle receiver. Moreover, there are horizontal fluidized bed particle receivers and upflow particle receiver concepts, such as the spiral particle receiver, the upflow fluidized particle in tube receiver and the fluidized bed particle air receiver. Detailed descriptions of these concepts can be found in the comprehensive review of Jiang et al. [56]
In the scope of this thesis an optical system is developed and optimized to investigate the particle motion in the centrifugal particle receiver CentRec, whose principal function and set up is described in the following section.

2.4.3 Particle receiver CentRec

The centrifugal particle receiver CentRec, was developed at the DLR and patented in 2010 [57]. To demonstrate proof of concept of the CentRec, a prototype in laboratory scale (15 kW_{th}) was designed and tested. The feasibility was proven by experimental studies resulting in an outlet temperature of 900°C and a predicted receiver efficiency of > 85 % under full-load conditions [20; 58; 59]. Subsequently, the receiver was upscaled and a prototype (2.5 MW_{th}) was fabricated [60] and tested with infrared heaters [61]. First on-sun tests have been conducted in the Juelich Solar Tower. The maximum measured temperature on the receiver outlet was 775°C, while temperatures above 900°C were achieved inside the receiver. However, the heliostat field at the Solar Tower Juelich is not designed for small, high flux density receivers. Hence, the spillage is very high in experiments with the CentRec and the highest flux density reachable in tests is 500 kWm^{-2} [62]. Further tests targeting receiver outlet temperatures up to 900°C are conducted successfully resulting in a maximum average receiver outlet temperature of 965°C [63].
In the following, the principal assembly of the CentRec is explained. Furthermore, the particle motion in the receiver is described in detail.

Assembly of the receiver

Figure 2.3 shows the design principle of the receiver.

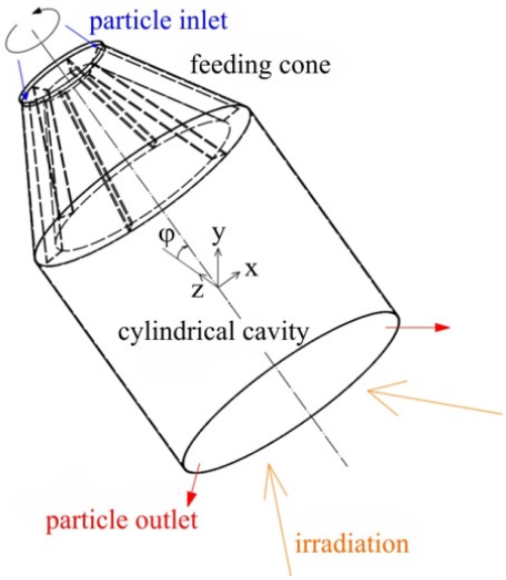


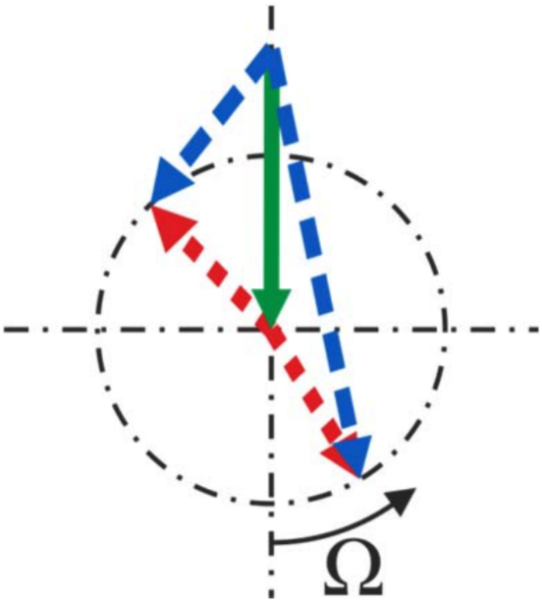
Figure 2.3: Design principal of the centrifugal particle receiver CentRec [20]

Principally, the centrifugal particle receiver consists of a cylindrical cavity and a feeding cone with particle inlet. The inclination angle of the rotating receiver can be varied. The cold particles are fed into the feeding cone, which is constituted of an inner and an outer cone and 32 fins [60]. Thus, the feeding cone has 32 chambers, where the particles are accelerated to the intended circumferential speed. Acceleration is necessary to secure, that the particles are pressed to the inner receiver wall and driven by gravitational forces, they slowly move downwards in axial direction instead of falling through the receiver. Other design options for the particle inlet, such as a bent pipe pouring the particles into the receiver, are currently under investigation. The aim of optimizing the particle inlet is to reach a uniformly and opaque particle film on the receiver wall and reduce the loss of particles. After entering the cylindrical cavity, the centrifugal force presses the particles to the inner receiver wall, where an opaque particle film is formed in such a way, that the entire circumference is covered by particles. Concentrated solar radiation incides into the aperture of the receiver and directly heats up the particles, resulting in an axial temperature gradient in the particle film. A collector ring at the bottom of the receiver collects the hot particles and afterwards, these particles are transported to either the particle heat exchanger connected the power block or the storage, cf. figure 2.1. The inclination angle of the rotating receiver can be varied from 0 to 90° to the horizontal plane, whereas 90° is related to the face-down receiver position with horizontal aperture and 0° corresponds to a horizontal receiver alignment with vertical aperture. [58]
The residence time and the particle film thickness and therefore, the particle outlet temperature is adjustable with the rotational speed and the mass flow. This is one of the major advantages of the centrifugal particle receiver compared to other particle receivers, e.g. falling particle film receivers. A controllable and constant outlet temperature is essential for the downstream processes, that use the captured solar energy. Inclination of the receiver reduces convective losses and the cavity effect

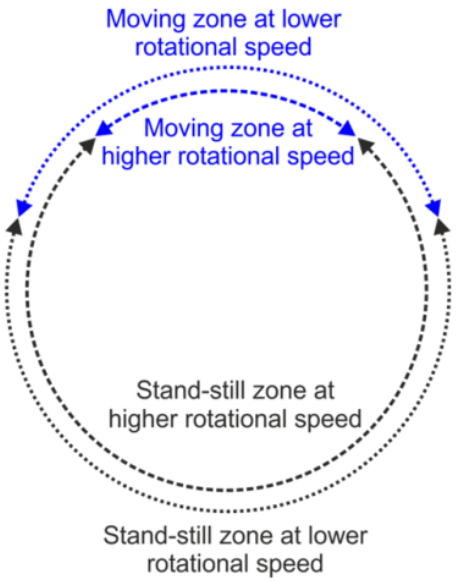
diminishes the radiation losses, resulting in possible thermal receiver efficiencies up to 90% [58].

Particle motion

As the properties of the particle film (shape, particle form and material, thickness) are crucial for the performance of the receiver, a more detailed look into the particle motion in the centrifugal particle receiver is presented. Furthermore, one main task of this thesis is the development of an image processing code to investigate the particle motion in the receiver in order to validate a numerical DEM (discrete element method) model in the future. For this purpose, possible validation parameters are proposed.
The particle motion in the rotating receiver is dominated by gravitational and centrifugal forces. If the inclination angle is set to 90° (face-down receiver position), resulting in a vertical rotational axis of the receiver, the alignment of both forces is constant over the whole circumference during one turn. On the contrary, other inclination angles of the centrifugal receiver lead to periodical changes in the vectorial force resulting from the superposition of gravitational and centrifugal force. [60; 61] Figure 2.4a shows this resulting force for two different particle positions on the circumference of the receiver.



(a) Superposition (blue) of gravitational force (green) and centrifugal force (red) for two different particle positions on the circumference [61]



(b) Width of the moving zone with changing rotational speed [61]

If the particle is located in the lower half of the receiver, a part of the centrifugal force and a part of the gravitational force are acting in the same direction and the particles are pushed against the receiver wall (long blue arrow). This collaboration of the forces results in the stop of the particle motion, why this part of the receiver wall is called stand-still zone, as shown in figure 2.4b. Conversely, in the upper part of the receiver, a part of the centrifugal force and a part of the gravitational force counteract and the particles are not pushed against the wall very strongly (short blue arrow in figure 2.4a). Consequently, the particles are able to move. The part of the receiver wall, in which the

particles move, is called moving zone, as shown in figure 2.4b. Additionally, the axial component of the gravitational force causes the particle movement towards the receiver aperture. The particle movement is observed to be stepwise during each turn. As shown in figure 2.4b, the width of the moving zone depends on the rotational speed. The higher the rotational speed is, the narrower the moving zone is and consequently, the smaller the axial step of the particles per receiver turn is. This results from the stronger centrifugal force due to higher rotational speed. The resistance time of the particles in the receiver can be controlled by the rotational speed and with the regulation of the mass flow entering the receiver, the required outlet temperatures can be achieved under any load condition. [60; 61]
One important parameter is the thickness of the particle film on the receiver wall. To avoid high temperature gradients between the different particle layers and reach optimum thermal efficiency, it should be as thin as possible, while still guaranteeing an opaque particle film to prevent the heating up of the receiver wall, respectively the stationary base layer of particles on the receiver wall. This base layer is constructed to achieve the same friction coefficient between the moving particles and the wall (base layer) as between the particles of two moving layers. Therefore, sliding of particles at lower rotational speed and accumulation, followed by avalanching of the particles at higher rotational speed, are avoided and a constant particle movement is enabled instead. The stationary base layer consists of particles, which are stuck in a wire mesh point-welded onto the inner receiver wall. [61]
Thicker films cause higher temperature gradients between the particle layers and consequently, lower average receiver outlet temperatures of the particles compared to the peak film temperature. Therefore, the residence time of the particles in the receiver is increased for thicker films leading to significantly higher temperatures of the surface particle layer. The resulting raised radiation losses cause lower receiver efficiencies. [58]
The particle motion directly impacts the receiver performance and it is important to understand and characterize the movement of a single particle in the particle film relatively to its neighbors, e.g. mixing in all directions (radial, tangential and axial). In order to do so, a numerical model, based on the discrete element method, is developed. For the validation of the developed DEM-model, experimental results are necessary. In the following, possible validation parameters are suggested and a concise description of them is given.

- **Axial step size**:
  How far do particles move in axial direction per turn depending on the mass flow and the rotational speed.
- **Width of the moving zone**:
  How wide is the moving zone depending on mass flow and the rotational speed.
- **Sampling rate**:
  How often do particles emerge and sink per turn or per square meter depending on the mass flow and the rotational speed.

Figure 2.5 shows one expected trajectory of a tracer during one turn of the receiver and illustrates the proposed validation parameters axial step size and width of the moving zone.

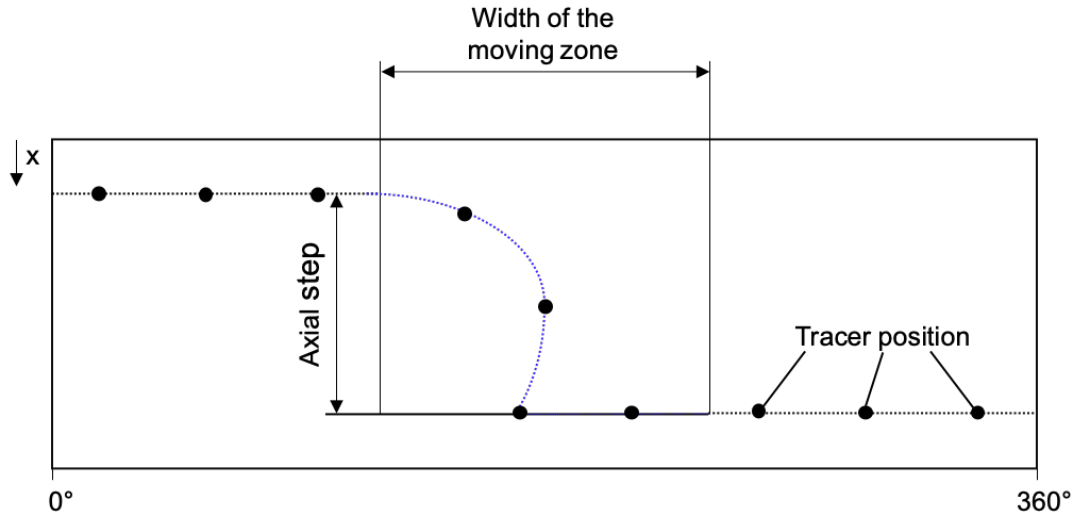


Figure 2.5: Expected tracer trajectory consisting of tracer positions detected during one turn of the receiver with illustration of the proposed parameters to validate DEM-model

The black circles illustrate the positions of detected tracers in frames of different time steps. The blue dotted part of the tracer trajectory describes the moving zone, cf. figure 2.4b
To obtain the necessary experimental results, an optical system and associated image processing codes are developed and optimized in the scope of this thesis. The methodology and results of stationary pre-tests to optimize the set up of the optical system, that in the future shall be used in the investigation of the particle motion in the receiver, are presented in section 3.3. In the following sections, optical basics and fundamentals of camera calibration are explained.

2.5 Optical basics

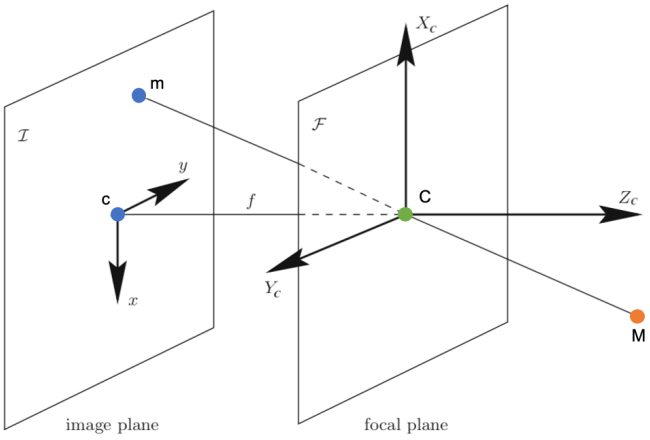
The **field of view (FOV)** of a camera is the total picture area of a picture taken with the camera. Its size depends on the camera's angle of view in horizontal and vertical direction. It can be indicated in pixel size or as area in terms of object size (metric system). In the second case, the distance between camera and object is of importance, because the metric size of one pixel depends on this distance and the selected resolution of the camera for recording videos, respectively taking pictures.
The so called **region of interest (ROI)** is the part of the camera's FOV, that contains the interesting information and therefore, is cut out for further examination in the image processing.
Color spaces are used to model colors in pictures quantitatively and represent them by specific tuplets of numbers, often triplets. In the scope of this thesis, two different color models are utilized, the RGB color model and the HSV color model.
The **RGB color model** is generally well known as it is used in most of the image processing and graphic programs. Moreover, it provides the basis for the color display in electronic devices, such as computers, digital cameras and scanners, as well as in the saving process of image files. The RBG color model is an additive color model, as each color is generated by a specific combination of the three primary colors red (R), green (G) and blue (B). Therefore, in the RGB color model colors are represented by triplets in the form of (R, G, B). The values for R, G and B are always positive and their upper limit is defined by a maximum value. As the most digital images are 8 bit images, this

maximum is set to 255. However, the components of the RBG triplets are often normalized, resulting in values between 0 and 1 for each of the component R,G and B. In the RGB color model, black is expressed by the triplet (0, 0, 0) and white is represented by the triplet (1,1,1). [64; 65]
Similar to the RGB color model, the **HSV color model** represents colors by triplets (H, S, V), meaning hue (H), saturation (S) and value (V). Thereby, hue describes the position in the color spectrum and is relatively independent of the brightness of the color. The normalized values of the H component are between 0 and 1. For all shades of gray, the hue component is not defined. The saturation can be indicated in terms of percent or normalized to values between 0 and 1. A saturation value of 1 corresponds to vibrant colors. Lower values of the saturation result in less intense colors. All shades of gray, including white and black, have saturation values of 0. The third component of the HSV color model represents the brightness of the color with normalized values between 0 (very bright) and 1 (dark). In consequence, the HSV color model describes the color white by the triplet (-,0,1) and the color black by (-,0,0). As in the HSV color model the color tone is only defined by the first component, its application is very advantageous for image segmentation by color. Compared to the RGB color model, the HSV model is based on the human color recognition and therefore more intuitive. [65]

2.6 Camera calibration

Camera models describe the correlation between coordinates of points in the three dimensional space and their projected points on the image plane of the camera [66]. The basis for mathematically describing cameras is the ideal pinhole camera model [67]. As the ideal pinhole camera has no lens, this model does not consider any distortion caused by camera lenses. To model the camera used in the scope of this thesis, the pinhole camera model has to be extended by modeling parameters for lens distortion [68]. Figure 2.6a illustrates the ideal pinhole camera model.

(a) Ideal pinhole camera model, adapted from [69]



(b) Different coordinate systems and their relation, adapted from [70]

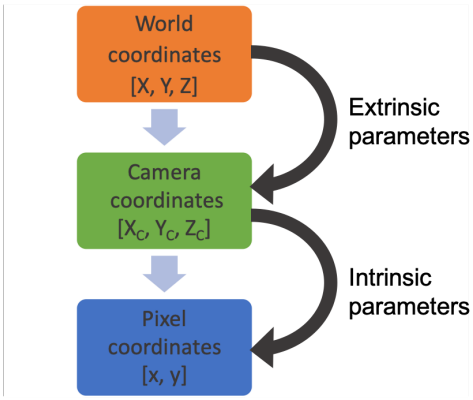


Figure 2.6: Fundamentals of camera modelling

In the ideal pinhole camera, a pinhole *C* in focal plane *F*, which is at a fixed distance *f*, called the focal length, allows light rays, reflected or emitted by an object, e.g. point *M* in figure 2.6a, to impinge on the image plane *I*. Thereby, it is assumed, that the light can reach the image plane only through the

pinhole. The optical axis is the line perpendicular to the image plane *I*. The intersection point of the optical axis with the focal plane is called optical center *C* and the point, where the optical axis intersects the image plane *I*, is called the principal point *c*. In a real camera the lens is located in the optical center and the housing prevents the image plane from light rays inciding from other directions, than through the lens. The result of the perspective projection, meaning the projection of points in the 3D space to an image plane without considering any effects of real cameras, such as blurring of unfocused objects or lens distortion, is an inverted image of the object on the image point. [69]
Camera models use three different coordinate systems: the world coordinates (3D), the camera coordinates (3D) and the pixel coordinates in the image plane (2D). Figure 2.6b shows the relation between these different coordinate systems. Any point in the world coordinate system is represented by a triplet of coordinates. Point *M* in figure 2.6a is therefore described with $[X, Y, Z]^T$. The extrinsic parameters correlate the world coordinates to the camera coordinates. The origin of the camera coordinate system is in the optical center *C* on the focal plane. Points in the camera coordinate system have the same structure as points in word coordinates, a triplet of coordinates to describe one point. With the intrinsic parameters of the camera, the camera coordinates are converted to pixel coordinates. The pixel coordinate system often originates in the upper left corner of the image. The projection of point *M* results in point *m* on the image plane, represented by $[x, y]^T$. [69; 70]
With equation 2.1 the position of one point $m = [x, y]^T$, projected onto the image plane in the pixel coordinate system, can be calculated from the position of the original point $M = [X, Y, Z]^T$ in the world coordinate system [67].

$$sm = A[R \quad t]M \tag{2.1}$$

A is the camera intrinsic matrix, which is explained in section 2.6.2. R and t are extrinsic parameters for rotation and translation, as discussed in section 2.6.1, and s is an arbitrary scale factor [67].
According to the definition of the perspective projection, all points in the world coordinate system are projected onto the image plane. Therefore, all points are located in the same image plane, resulting in a fixed value of the z component of the projected points. By defining s in equation 2.1 to $s = 1/z$ and assigning the value one to the constant z coordinate, the projected point *m* is defined as shown in equation 2.2. [71]

$$m = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2.2}$$

The developed Python code used in the calibration procedure to determine the camera parameters (cf. section 3.4), is based on the ideal pinhole camera model. Additional parameters to describe and minimize the distortion in the pictures [72], are explained in section 2.6.3. In the tracer tracking algorithm, cf. section 4.2, the determination of the centers of gravity of the detected tracers and the consequential neighbor search are important steps of the image processing, which need the capability of good measurement of distances in the image. Therefore, the correction of lens distortion in the pictures is of crucial importance to reduce errors in the tracking resulting from false determination of the tracers' position. In the scope of camera calibration, intrinsic and extrinsic parameters [73] and distortion coefficients are estimated. In the following each of these parameters is explained in detail.

2.6.1 Extrinsic parameters

As shown in figure 2.6b, the extrinsic parameters covert the three dimensional position and orientation of a point in the world coordinate system into the camera coordinate system and vice versa [73]. For a certain position of the camera in the world coordinate system, the camera coordinate system is fixed relatively to this position of the camera. Therefore, the extrinsic parameters have to be determined experimentally according to each experimental set up of the camera and the object to be recorded. In the ideal pinhole camera model, the extrinsic parameters consist of the 3×3 rotation matrix R and the translation vector $t = [t_\mathrm{x}, t_\mathrm{y}, t_\mathrm{z}]^\mathrm{T}$. The rigid body (Euclidean) transformation between the two cartesian coordinate systems (world and camera) considers rotation and translation and can be described mathematically with equation 2.3 [73].

$$\begin{bmatrix} X_\mathrm{c} \\ Y_\mathrm{c} \\ Z_\mathrm{c} \end{bmatrix} = R \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + t \tag{2.3}$$

2.6.2 Intrinsic parameters

The intrinsic camera matrix A includes the focal lengths f_x and f_y and the pixel coordinates of the optical center C_x and C_y, located in the image plane, cf. figure 2.6a. In the parameter q any possible skew between the sensor axes and the optical axis, arising from inaccurate mounting of the sensor perpendicular to the optical axis of the camera, can be considered in the camera model. In the scope of this thesis, this effect is negligible and the parameter is considered to be zero. Equation 2.4 shows the structure of the camera matrix used in the intrinsic model to transform the camera coordinates to pixel coordinates. [71]

$$A = \begin{bmatrix} f_\mathrm{x} & q & C_\mathrm{x} \\ 0 & f_\mathrm{y} & C_\mathrm{y} \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

With the defined extrinsic and intrinsic parameters and the ideal pinhole camera model (equation 2.1), the pixel coordinates of the ideal projected point *m* on the image plane can be calculated. As in this linear projection no distortion is considered, the camera model has to be extended by distortion coefficients, which describe and compensate the distortion. These coefficients are described in the next section.

2.6.3 Distortion coefficients

Using correction functions Δx and Δy, the distortion, leading to the deviation between the ideal projection point $[x_\mathrm{id}, y_\mathrm{id}]^T$ and the real (distorted) normalized pixel coordinates of the projected point on the image plane $[x_\mathrm{d}, y_\mathrm{d}]^T$, is added to the pinhole model explained before, resulting in equation 2.5 [67].

$$\begin{bmatrix} x_\mathrm{d} \\ y_\mathrm{d} \end{bmatrix} = \begin{bmatrix} x_\mathrm{id} \\ y_\mathrm{id} \end{bmatrix} + \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \tag{2.5}$$

Deviations between the image points and the ideal image points have different causes, such as radial-symmetric (short: radial) distortion, tangential distortion or affinity and shear [74]. The correction

functions for affinity and shear include the deviations resulting from non orthogonal coordinate axes or non uniform scaling of the coordinate axes to the distortion model. As the impact of affinity and shear is negligibly small compared to the radial and the tangential distortion, their correction function is not considered in the scope of this thesis. Therefore, a suitable accurate camera model is used, including the correction functions for radial and tangential distortion [68].
Figure 2.7 shows the displacement of image points caused by radial and tangential distortion (figure 2.7a) and illustrates the resulting distortion of an image due to radial (figure 2.7b) and tangential distortion (figure 2.7c).

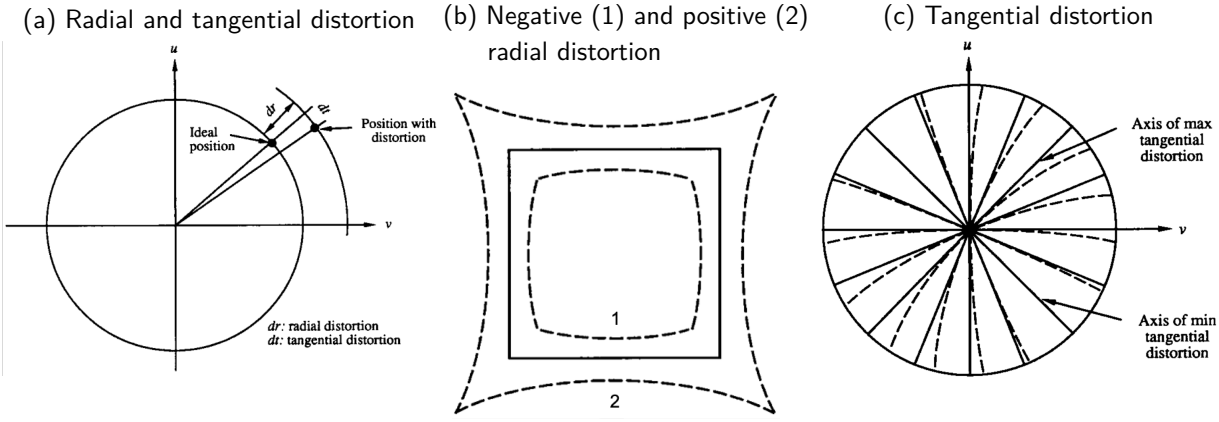


Figure 2.7: Illustration of radial and tangential distortion, in (b) and (c): no distortion (solid lines), distortion (dashed lines) [75]

In most of the cases, the radial distortion is more distinct than tangential distortion. For good quality lenses the impact of radial distortion can be up to 10 times greater than the one of tangential distortion [74]. The effect of distortion is expressed with the correction functions $\Delta x_{\mathrm{rad}}, \Delta y_{\mathrm{rad}}$ and $\Delta x_{\mathrm{tan}}, \Delta y_{\mathrm{tan}}$ for the radial and the tangential distortion each and shown in equation 2.6[71; 76] and equation 2.7 [74; 76].

$$\begin{aligned} \Delta x_{\mathrm{rad}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ \Delta y_{\mathrm{rad}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{2.6}$$

$$\begin{aligned} \Delta x_{\mathrm{tan}} &= p_1(r^2 + 2x^2) + 2p_2 xy \\ \Delta \mathrm{y}_{\mathrm{tan}} &= p_2(r^2 + 2y^2) + 2p_1 xy \end{aligned} \tag{2.7}$$

$$\text{with} \qquad r^2 = x^2 + y^2$$

Commonly, the radial distortion is approached by a polynomial series, in which k_{n} represent the radial distortion coefficients [76]. For most of the camera lenses, only the first two or three coefficients are needed to model the radial distortion sufficiently accurate [74].
Radial distortion is originated in imperfect lens shapes and is visible in the form of curved lines as projection of straight lines [75]. Decreasing magnification with increasing distance between the projection of an object and the principal point on the image plane, is called negative radial distortion

or barrel distortion (figure 2.7b, dashed line 1). Straight lines appear curved out in the image plane. On the contrary, increasing magnification with increasing distance between object and optical axis, is named positive radial distortion or pincushion (figure 2.7b, dashed line 2). The projection of straight lines occurs bowed inwards. [77]
The maximum of radial distortion occurs at the image edges and the resulting deviation is symmetric to the point in which the projection ray and the image plane are perpendicular to each other (principal point), cf. figure 2.6 [78].
Misalignment of the individual lens centers and decentering of the lenses within the camera objective result in tangential distortion, described by the parameters p_1 and p_2. In the determination of the tangential distortion coefficients the maximum tangential distortion is considered, cf. figure 2.7c.
In the scope of this thesis, the following distortion coefficients are determined experimentally during the calibration, cf. section 3.4, and are used to undistort the frames in the recorded videos to generate corrected data for the tracer tracking algorithm and the determination of the film thickness, cf. section 4.1. As explained above, k_1, k_2 and k_3 are the coefficients for radial distortion and p_1 and p_2 are the ones considering the tangential distortion, cf. equation 2.8. Because these parameters can be assigned to a physical meaning, they are called physical camera parameters [68].

$$\text{distortion coefficients} = \begin{pmatrix} k_1 & k_2 & p_1 & p_2 & k_3 \end{pmatrix} \tag{2.8}$$

In wide-angle or fisheye cameras, such as GoPros, the distortion arising from the lenses is extremely distinct. In these cameras the camera's FOV is enlarged by the use of multiple sequential lenses and therefore, they are often not described sufficiently accurate with a camera model based on the ideal pinhole camera. As in the scope of this thesis, the distance between the camera lens and an object is kept below 200 mm, the effect of the wide-angle effect is not very distinct. Thus, the resulting deviation in the object size, originating from the distortion, is smaller than in pictures taken from further distances. When taking pictures from close distances between the camera and the object, it is assumed, that the deviation between the undistorted pictures, using camera calibration parameters based on the pinhole camera model, and the pictures undistorted with camera parameters estimated for a wide-angle camera, is small. As the camera model for wide-angle cameras is much more complex and time consuming, the camera calibration for estimating the GoPros' camera parameters is done using a modified pinhole camera model including the distortion coefficients. This is an approach, which can also be found in other applications [79].

3 Experiments

This chapter contains a detailed description of the materials used in the experiments, conducted in the scope of this thesis, and explains the procedure of particle characterization, including particle size distribution (PSD) and static angle of repose and their results. Furthermore, the methods and results of the optical design optimization as well as the results of the camera calibration are presented and discussed.

3.1 Material

In this section the materials used in the experiments are listed and their properties are described in detail.

3.1.1 Particles

The particles used in the experiments, Proppants 16/30 Sintered Bauxite, are distributed by Saint-Gobain (SG). According to the manufacturer the main diameter is 0.971 mm and the sphericity calculated following Krumbein & Sloss [80] is 0.9. The PSD provided by Saint-Gobain states that 86 % of the SG particle diameters are between 0.841 and 1.19 mm. The particles are made of Bauxite (CAS number: 1318-16-7), which is a mineral hydrated form of aluminum hydroxide. The color of the particles is tan to reddish-brown. [81] The initial boiling point is 2980 °C [81], the absorption coefficient lies between 0.8 and 0.9 [58; 82] and the specific heat capacity is about 1100 J / (kg K) [58]. Thus, the SG particles are an excellent HTM for CSP particle receivers. The characteristic parameters of the SG particles are summarized in table 3.1.

Table 3.1: Characteristic parameters of the SG particles [83] and the tracers [84]

	SG particle	Tracer
Name	Proppants 16/30 Sintered Bauxite	SAZ - ER120S
Manufacturer	Saint-Gobain	Mühlmeier Mahltechnik
Material	Aluminum hydroxide	Zirconium mixed oxide
Main diameter	0.971 mm	0.8 - 1.25 mm
Specific weight	3.49 g/cm^3	3.8 g/cm^3
Bulk density	2.02 g/cm^3	2.3 g/cm^3
Sphericity	0.9	$>$ 0.7 (95 % of the tracer)

The results of the determination of SG particles' PSD is presented and discussed in section 3.2.1. The experimentally determined angle of repose is 27 °, cf. section 3.2.2.

3.1.2 Tracers

The tracers used to track the particle movement in the centrifugal receiver are chosen in a way that the flow properties are as similar as possible to the ones of the SG particles. Thus, it is possible to conclude the flow regime of the continuum directly from the tracer movement. The tracers, SAZ-Perlen ER120 S 0.8 - 1.25 mm ZrO_2, are provided by Mühlmeier Mahltechnik [84] and the color of the tracers is white to achieve a high contrast compared to the brownish SG particles. As shown in table 3.1, the specific weight and the bulk density of the tracers are less than 15 % higher than the ones of the SG particles. The determined PSD is presented and discussed in section 3.2.1. The experimentally determined angle of repose is 22.2 °, cf. section 3.2.2.

3.1.3 Camera

The purpose of the camera is to record the moving particles in the receiver. Important parameters for the selection of the camera and the recording mode are the frame frequency, the resolution and the size of the field of view. The camera used in the experiments is a GoPro HERO3+ Silver. The camera's angle of view is 90 ° in horizontal and 100 ° in vertical direction, respectively. The camera takes color photos and has the ability to record videos or take photos with adjustable resolution, frame frequency and size of the FOV. For the purpose of the experiments in the scope of this thesis, the following camera settings are chosen:

- video resolution of 1080p, which corresponds to a screen resolution of 1920x1080 pixels
- frame frequency of 60 fps
- narrow FOV

To start and stop recording during future experiments in the receiver, the camera can be connected to a SmartRemote 2.0 via WiFi.

3.1.4 Illumination

The purpose of the illumination is to illuminate the region of interest in the FOV of the camera. In general, strobe lights offer brighter illumination than permanent ones. However, a strobe light can not be used, because the shutter signal of the GoPro camera is not accessible and therefore, it is not possible to synchronize the camera's exposure time and the flashing of the illumination. LED lights, which are often used in machine vision because of their multiple advantages like fast response, long lifetime, high output stability and their mechanical resistance [85], are selected as luminant. The illumination used for tracking the tracers is the FLDL-i70A white direct ring light from FALCON illumination. For the measurement of the film thickness the FLDL-i86x15 white bar light from FALCON illumination is used. The important characteristics of the illuminations are listed in table 3.2. The illumination is chosen after conducting several stationary tests, which are explained in section 3.3.1 along with the discussion of the results.

Table 3.2: Characteristic parameters of the illuminations [86; 87]

	Illumination tracer tracking	Illumination film thickness
Name	FLDL-i70A	FLDL-i86x15
Manufacturer	FALCON illumination	FALCON illumination
Shape	Direct ring light	Bar light
Color of light	White	White
Dimensions	Ø70.0x22.5 mm	17.5x94.5x20.0 mm (WxLxH)
Luminant	LED	LED
Illumination area	Ø65mm	86x15 mm

3.1.5 Laser

The laser will be used in the image processing to determine the angular position of the camera for each frame. The laser will be mounted onto the stationary part of the receiver outlet so that the laser always focuses the same spot on the receiver wall. The dot laser, FP-D-520-1-C-F, is provided by Laser Components GmbH. The wavelength of the light is 520 nm, which corresponds to green light. [88]

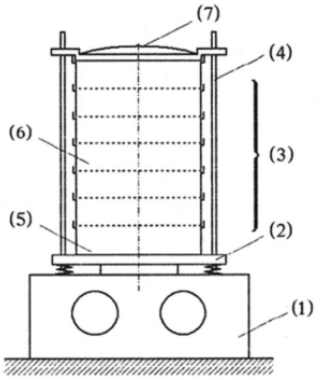
3.2 Particle characterization

This section includes the description and the results of the particle characterization. To characterize the particles, the PSD and static angle of repose is experimentally determined.

3.2.1 Particle size distribution: sieving analysis

There are different approaches to determine the PSD of a particle collective. It is distinguished between optical and mechanical methods [89]. The sieving analysis is an accurate, simple, cheap and frequently used mechanical method to determine the PSD of a particle collective [90]. The sieving analysis provides more accurate results for increasing sphericity of the particles examined. Alternatively, there are optical methods, such as the laser diffraction spectroscopy. However, in the scope of this thesis, the sieving analysis is selected for the geometric characterization of the SG particles and tracers, as the sphericity of the SG particles is very high [91].
The sieving analysis is an important characterization method for particle collectives. It determines the mass fraction of particular particle size intervals defined by two different mesh sizes of sieves. There are several procedures depending on the fracture opening of the sieves. Typical fracture openings of the sieves are between 5 µm and 125 mm. For small particle sizes (5 µm - 25 mm) wet sieving is advantageous. For particle mixtures with a narrow particle size distribution between 20 µm and 500 µm air jet sieving is a good option. [89]
As described in section 3.1.1 the particle used within the scope of this thesis are assumed to have a nominal diameter of around 1 mm [83]. Therefore, the vibration sieving, which is suitable for particles in the range of 90 µm - 125 mm, is used to determine the PSD. The result of the sieving analysis is a discrete mass distribution, whose resolution depends on the number of sieves and the differences in

mesh size between the sieves. The principal assembly of the vibration sieving analysis used in this thesis is shown in figure 3.1.



1: Mechanical or electromagnetic shaker
2: Spring-mounted plate
3: Sieve assembly
4: Holding structure
5: Sieve pan
6: Sieve
7: Cover

Figure 3.1: Principal assembly of a vibration sieving analysis apparatus [89]

The sieves (6) with different mesh size are stacked on top of each other, with increasing mesh size from the bottom to the top. All sieves are placed in a holding structure (4) and covered on top (7). Below the lowest sieve a sieve pan (5) is located to collect the small particles, which are not sorted out in the sieves. All described components are placed onto a plate (2), that is spring-mounted on a mechanical or electromagnetic shaker (1). The duration and the intensity in terms of amplitude and acceleration of the vibrational motion can be adjusted, e.g. when using more sieves and/or finer meshes in order to increase the resolution of the PSD. [89]
Before starting the sieving analysis, the total mass of the particles is measured. Subsequently, the particles are poured into the most upper sieve and the shaker is switched on. The particles are separated by their size of diameter. Whether a particle is able to move onto the lower next sieve, depends on the ration between their diameter and the mesh size of the sieve on which the separation takes place. If the particles are much smaller than the mesh size, they can easily pass through the sieve, independent of their rotational orientation. If the particles are much bigger than the mesh size, they stuck on the surface of the sieve. The closer the ratio between particle diameter and mesh size is to one, the more complicated and slower the separation on the surface of the sieve becomes. In this case, the success of the separation strongly depends on the shape of the particles. If they are not perfectly spherical, the orientation is of crucial importance and the duration and intensity of shaking has to be adjusted in a way, that the particles have the possibility to reach the sieve holes several times with different orientation. During the sieving analysis particles, having nearly the same diameter as the mesh size of the sieve, may cause plugging of the sieve holes. Thus, the shaking of the sieves must be intense and long enough to loosen the stuck particles out of the sieves. [89]
In the scope of this thesis, the vibrations sieving is carried out with 15 different sieve sizes from 1.8 mm to 0.071 mm. The sizes of the sieves used are the following: 1.8, 1.7, 1.18, 1.12, 1.0, 0.9, 0.85, 0.8, 0.71, 0.6, 0.5, 0.4, 0.2, 0.1, 0.071 mm. The duration of the vibration is set to ten minutes to enable the particles to accumulate in the adequate sieve [89]. As a result of some pre-tests the frequency of the vibration is set to 70 Hz and the amount of particles poured in is chosen as 500 g. The mass fraction for each sieve size w_n is calculated with equation 3.1.

$$w_\mathrm{n} = \frac{m_\mathrm{n}}{m} \qquad \text{with} \qquad m = \sum_{\mathrm{n}=1}^{N} m_\mathrm{n} \tag{3.1}$$

To reduce statistical errors the sieving analysis is conducted three times and the mean value is calculated for each sieve size. Assuming that the measured values are samples of the entire population, the standard deviation σ for each sieve size is determined with equation 3.2.

$$\sigma = \sqrt{\frac{\sum_{\mathrm{n}=1}^{N} (w_{\mathrm{n}} - \mu)^2}{N-1}} \qquad \text{with} \qquad \mu = \frac{\sum_{\mathrm{n}=1}^{N} w_{\mathrm{n}}}{N} \tag{3.2}$$

The PSD is obtained for the SG particles and for the tracers and compared to the specification of the manufacturer and to each other. The discrete particle size distribution is plotted as a histogram [92]. In order to do so, the mean distribution density $q_3(x_{\mathrm{i}})$ for each sieve size interval Δx_{i} is calculated with equation 3.3.

$$q_3(x_{\mathrm{i}}) = \frac{\bar{w_{\mathrm{i}}}}{\Delta x_{\mathrm{i}}} \qquad \text{with} \qquad \Delta x_{\mathrm{i}} = x_{\mathrm{i}} - x_{\mathrm{i}-1} \tag{3.3}$$

The cumulative distribution $Q_3(x_{\mathrm{i}})$ is determined with equation 3.4 and displays the percentage of SG particles, respectively tracers, with diameters smaller than x_{i}.

$$Q_3(x_{\mathrm{i}}) = \sum_{\mathrm{n}=1}^{i} w_{\mathrm{n}} * 100\,\% \tag{3.4}$$

To compare the particle size distribution of the SG particles and the tracers, the median diameter and its standard deviation are calculated as well as the $q_{90,3}$. This value represents the lower limit of the 90 wt% percentile, which implies that 10 wt% of the SG particles or tracers have a diameter that is smaller than this value.

PSD of the SG particles

The discrete particle size distribution of the SG particles is shown in figure 3.2 on the primary axis. The width of the bars is defined by the difference in mesh size of two consecutive sieves. The vertical red lines illustrate the standard deviations. On the secondary axis the cumulative distribution is displayed in green.

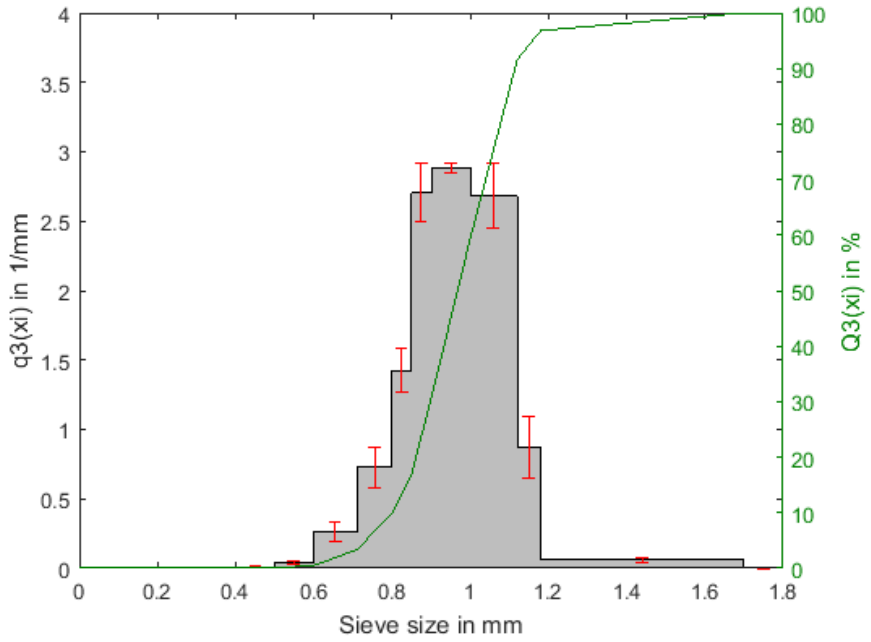


Figure 3.2: Particle size distribution of the SG particles determined by sieving analysis

The mean diameter of the SG particles is 0.967 mm and the calculated standard deviation is 0.013 mm. This value matches the specification of the manufacturer (main diameter 0.971 mm, [83]). The percental deviation between the measured mean diameter and the specification of the manufacturer is 0.36 %. 90 wt% of the particles have a diameter greater than 0.798 mm and 86.9 wt% of the particle diameters are between 0.8 and 1.18 mm. This result corresponds to the specification of the manufacturer, according to which 86 % of the particles have a diameter between 0.841 and 1.19 mm [83].
Since only 61 wt% of the particles have a diameter between 0.9 and 1.12 mm, the assumption of one single diameter in the numerical model should be revised. The results of the SG particles' PSD indicate, that a PSD instead of a single particle diameter should be used in the numerical simulation to improve the accuracy of the model.

PSD of the tracers

On the primary axis in figure 3.3 the discrete particle size distribution of the tracers is displayed. Analogue to figure 3.2, the width of the bars is defined by the difference in mesh size of two consecutive sieves and the vertical red lines illustrate the standard deviations. The cumulative distribution of the tracer diameters is shown on secondary axis in green.

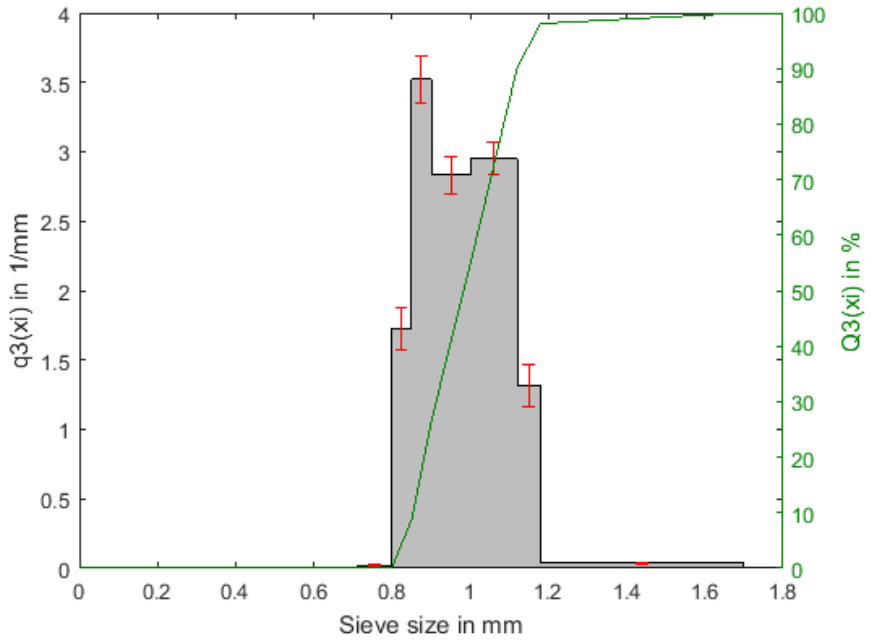


Figure 3.3: Particle size distribution of the tracers determined by sieving analysis

The mean diameter of the tracers is 0.983 mm and the calculated standard deviation is 0.008 mm. According to the manufacturer, the tracer diameter is between 0.8 and 1.25 mm [84]. The results of the sieving analysis show, that 97.91 wt% tracer diameters are between 0.8 and 1.18 mm and 90 wt% of the tracers have a diameter greater than 0.853 mm.
The mean diameter of the tracers is slightly greater than the one of the SG particles. The absolute deviation between the mean diameter of the tracers and the one of the SG particles is 0.016 mm, corresponding to 1.6 % of the mean SG particle diameter. This result justifies the assumption, that the tracers' effect on the flow characteristics of the particle film in the receiver is negligible. To strengthen this assumption, the angle of repose is measured for both the pure SG particles and a SG particle tracer mixture and then compared. The results are discussed in the following section.

3.2.2 Angle of repose

To characterize a bulk material, the angle of repose is an essential parameter, because it provides a good indication of flowability of granular material and it can be related to macroscopic bulk material phenomena, such as segregation, avalanching and stratification. Generally, the definition of the angle of repose depends on the application and the behavior of the material used [93]. One commonly used definition of the angle of repose is the angle between the surface of a pile under gravity and the horizontal plane at which no avalanches occur spontaneously [94]. There are two different types of angle of repose: static and dynamic. For the purpose of this thesis only the static angle of repose is determined. The aim of the determination of the angle of repose is to investigate the influence of the tracers on the fluid-like mechanical properties of the particle collective. The angle of repose of a SG particle tracer mixture with 5 wt% tracers is experimentally determined and compared to the one of the pure SG particle bulk material. Figure 3.4 shows the experimental assembly, that is used to determine the static angle of repose.

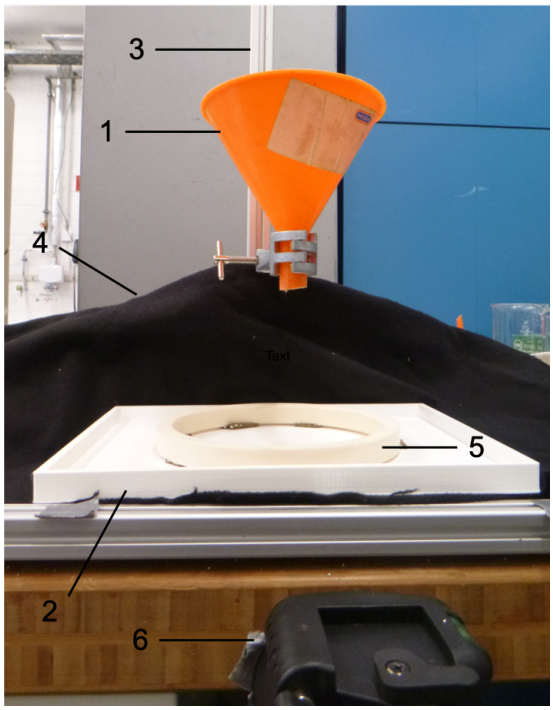


Figure 3.4: Assembly for the experimental determination of the static angle of repose

The material to be examined is filled into the fixed funnel (1). The funnel is mounted onto a holding structure (3), which places the outlet of the funnel directly above the center of a polyester ring (5). The polyester ring has a height of 20 mm and is placed on a white base plate (2). A digital camera, fixed to an adjustable tripod (6), is placed in front of the pile in a way, that the camera focuses the top of the pile. Black molton (4), respectively a white screen, is positioned behind the pile to increase the contrast and therefore, facilitate the image analysis. The images taken by the digital camera are evaluated by a online digitizer tool called WebPlotDigitizer [95] and a Matlab script. The WebPlotDigitizer is used to find and export the coordinates of points on the surface of the pile. Afterwards, the Matlab script reads the coordinates of the points describing the pile and divides the pile into two almost even parts. Subsequently, a polynomial fit of 4th order is used to describe the surface of the pile mathematically and a linear fit for each side of the pile is obtained. With the slope of the linear fit and the arctan function the static angle of repose is calculated for both sides of the pile. In figure 3.5 a cutout of a picture taken during the experiments with a SG particle tracer mixture (5 wt% tracer) is shown. Schematically the lines used in the calculation of the static angle are drawn.

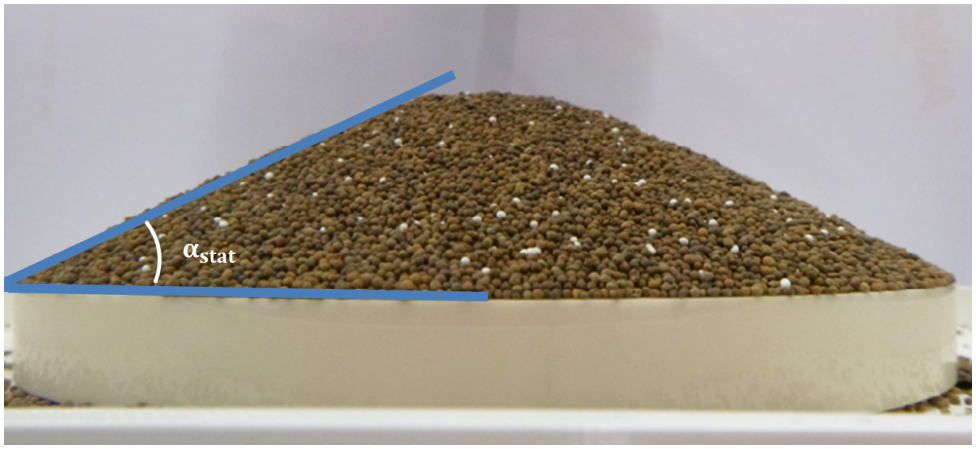


Figure 3.5: Cut out of a picture taken during the experiments with a SG particle tracer mixture (5 wt% tracer), schematic figure of the static angle of repose α_{stat}

Since avalanches do not happen symmetrically, but one side at a time, the top of the pile is not in the center of the pile. Hence, the calculated angles for the right and the left side do not coincide and the angle of repose is assumed to be the arithmetic average of both. To diminish the statistical error, the measurement is conducted ten times, resulting in a total number of values of 20. As well as the mean angle of repose for each side, the average of all 20 determined angles of repose is calculated and listed in table 3.3.

Table 3.3: Angle of repose for pure SG particles, pure tracers and a SG particle tracer mixture (5 wt% tracer), averaging over 10 measurements each left and right

Collective	Left angle	Right angle	Average angle
Pure SG particles	27.2875 ± 1.1869°	26.7572 ± 0.4981°	27.0224 ± 0.9025°
Pure tracers	21.7597 ± 0.4998°	22.5338 ± 0.6732°	22.1468 ± 0.6921°
5 wt % tracer	26.0577 ± 0.4009°	25.9923 ± 0.7663°	26.0250 ± 0.5776°

The mean angle of repose of the tracers is approximately 18 % lower than the one of the SG particles. Since the intended mass fraction of the tracers is not higher than 5 wt%, this deviation is not expected to be significant. To verify this assumption, the angle of repose is not only determined for pure SG particles and pure tracers, but additionally for a SG particle tracer mixture. The mass fraction of the tracers is chosen to be 5 wt%. In the mass fraction range between 0 and 5 wt% a sufficiently high probability of having enough tracers in the FOV of the camera is assumed, so that the recognition of tracers is feasible.
As the angle of repose does not change significantly when adding a tracer mass fraction of 5 wt%, the influence of the tracer mass fraction on the fluid-like mechanical properties of the SG particle tracer mixture is assumed to be nearly invariable in the range of tracer mass fraction up to 5 %. Thus, the tracer mass fraction is chosen in a way, that sufficient tracers are detected in the ROI of the picture. With a higher number of detected tracers, the computing time of picture processing increases and the tracking of the tracers becomes more complex due to a higher possibility of two or even more closely located tracers. If the tracers are too close to each other, the algorithm might merge them to one big tracer, which is then eliminated by the area filter in the tracking algorithm, cf. section 4.2. Because the mean angle of repose of the SG particle tracer mixture with 5 wt% tracers is only 3.7 %

lower than the one of the pure SG particles, tracer mass fractions up to 5 wt% are considered to not significantly impact the flow properties of the particle collective in the receiver.

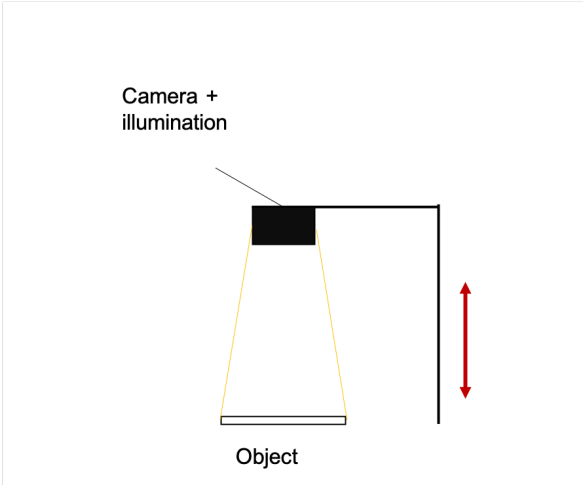
3.3 Optical design optimization

One purpose of this thesis is to select and optimize the illumination and the camera set up by conducting tests with a stationary SG particle tracer mixture. Furthermore, the image processing is examined and optimized during these tests.

3.3.1 Illumination

It is of crucial importance, that the illumination is bright and homogeneous. Overexposure, causing too bright pictures, and underexposure, resulting in too dark pictures, should be prevented, as objects in very bright or shadowy regions can not be recognised correctly. As luminant LED lights are chosen, as described in section 3.1.4. In the selection process of the illumination lights from various manufacturers with different shapes and light colors are compared to each other. The most important evaluation parameters are the homogeneity of the illumination, the mean pixel value of a white sheet and the difference between the pixel values of the tracers and SG particles. Moreover, the set up complexity and the costs are considered. The set up shown in figure 3.6 is used to compare the different illumination options.

(a) Schematic figure of the set up under the paper box



(b) Paper box covered with molton at the inner surface

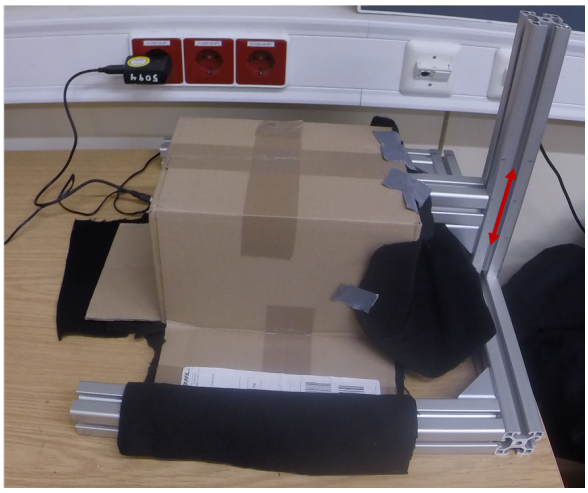


Figure 3.6: Assembly of the test set up to choose the most suitable illumination

The illumination and the camera are mounted onto the silver item structure, cf. figure 3.6a. Using the flexible item structure, the distance between the camera, respectively the illumination, and the bottom, where a flat white bowl is placed, can be adjusted. The inner surface of the paper box is covered with black molton to minimize the reflection of light, cf. figure 3.6b. For the homogeneity tests a white sheet is placed in the bowl. The camera and the illumination are located at the same height and the black box is imposed on the system. All edges and slots are covered with extra molton,

in order to prevent ambient light from affecting the measurement. While recording, the illumination is switched on and off three times. The time between each switch off and the switch on is 30 seconds, the same as each exposure time. This procedure is conducted to eliminate inhomogeneities, that could occur shortly after switching on or off the illumination. To check the calculations of the FOV, cf. table 3.6, the white sheet in the set up is replaced by a grid with squares of 5 mm x 5 mm and the procedure is repeated as described above. To compare the effect of the different illuminations on the recognition of the tracer, a SG particle tracer mixture (1 wt% tracers) is poured into the bowl on the ground in a way, that an opaque particle film covers the bottom of the bowl. The film thickness is a few particles (1 - 4 particles), similar to the film thickness expected in the experiments in the receiver. In order to find the most suitable illumination, five different options are tested, compared and rated. The tested illumination and their characteristics are listed in table 3.4. A photo of each illumination can be found in Appendix A.1. The results of the illumination tests are discussed below and table 3.5 summarizes the results of the rating criteria used for the selection of the illumination.

Table 3.4: Tested illuminations and their specifications

Abbr.	Name	Manufacturer	Shape	Light color	Illumination area	Price	Data sheet
P1	14-14-Sled-2-Flat-VA-16w	planistar Lichttechnik GmbH	Area light	White	140x140 mm	659€	[96]
L1	LED Spot5W-W	iiM AG	Spot light	White	Adjustabe, up to Ø250 mm at 250 mm work distance	580€	[97]
F1	FLDL-i70A	FALCON illumination	Ring light	White	Ø65 mm	450€	[86]
F2	FLDL-i86x15	FALCON illumination	Bar light	White	86x15 mm	300€	[87]
F3	FFPR-Si100-RGB	FALCON illumination	Ring light	Blue (470 nm)	Ø94.6 mm	725€	[98]

The first image processing step of the videos taken during the tests is exporting each frame from the videos and selecting one representing frame for further analysis. As these tests are stationary, the position of the SG particles and tracers in the pictures should not change between different frames. To obtain a representative picture, a frame approximately 4 seconds after switching on the illumination is selected to avoid any initial illumination inhomogeneities. In the following, the selection parameters are discussed in detail. The parameters are: homogeneity of the illumination, brightness, difference between the pixel value of tracers and particles, set up complexity and price.

Homogeneity

To compare the homogeneity of the different illuminations, the procedure described in section 3.3.1 is conducted with a white sheet. A Python code, cf. Appendix A.2, is created to evaluate the color value of each pixel. From the selected pictures the center region with a size of 50 mm x 50 mm is cut out, if possible, cf. table 3.6. For distances smaller than 100 mm, the cut out area is chosen as close as possible to this size. Therefore, the size of the cut out depends on the distance between the camera and the object. To cut out the right area size, the pictures of the grid are used to evaluate the pixel size for each distance between the camera and the particles. Furthermore, the pictures of the grids are utilized to review the size of the cut out. Optimization results regarding the distance between the camera and the particle collective are discussed in section 3.3.2. With equation 3.5, which is based on the color coding of analog television, the RGB values of the pictures are converted to one gray level pixel value g [64].

$$g = 0.299\,R + 0.587\,G + 0.114\,B \tag{3.5}$$

Each pixel value is stored into a matrix and saved to a text file. Furthermore, the minimum and the maximum pixel value and their difference are calculated. Additionally, the mean pixel value and its standard deviation as well as the median pixel value are calculated. For the calculation of the homogeneity equation 3.6 is used, in which $\bar{g}$ represents the mean pixel value and σ the standard deviation of the mean pixel value.

$$\text{Homogeneity} = \frac{\bar{g} - \sigma}{\bar{g}} * 100\% \tag{3.6}$$

These values are saved into a text file for easier comparison between the different illuminations. Moreover, a histogram showing the number of pixels for each pixel value from 0 to 255 is created and saved as plot and as text file. Figure 3.7 shows the colored plot of the pixel value for the non cut pictures (whole FOV of the camera) taken with different illuminations. The magenta square illustrates the cut out ROI (50 mm x 50 mm). The distance between the camera and the white sheet is always 135 mm.

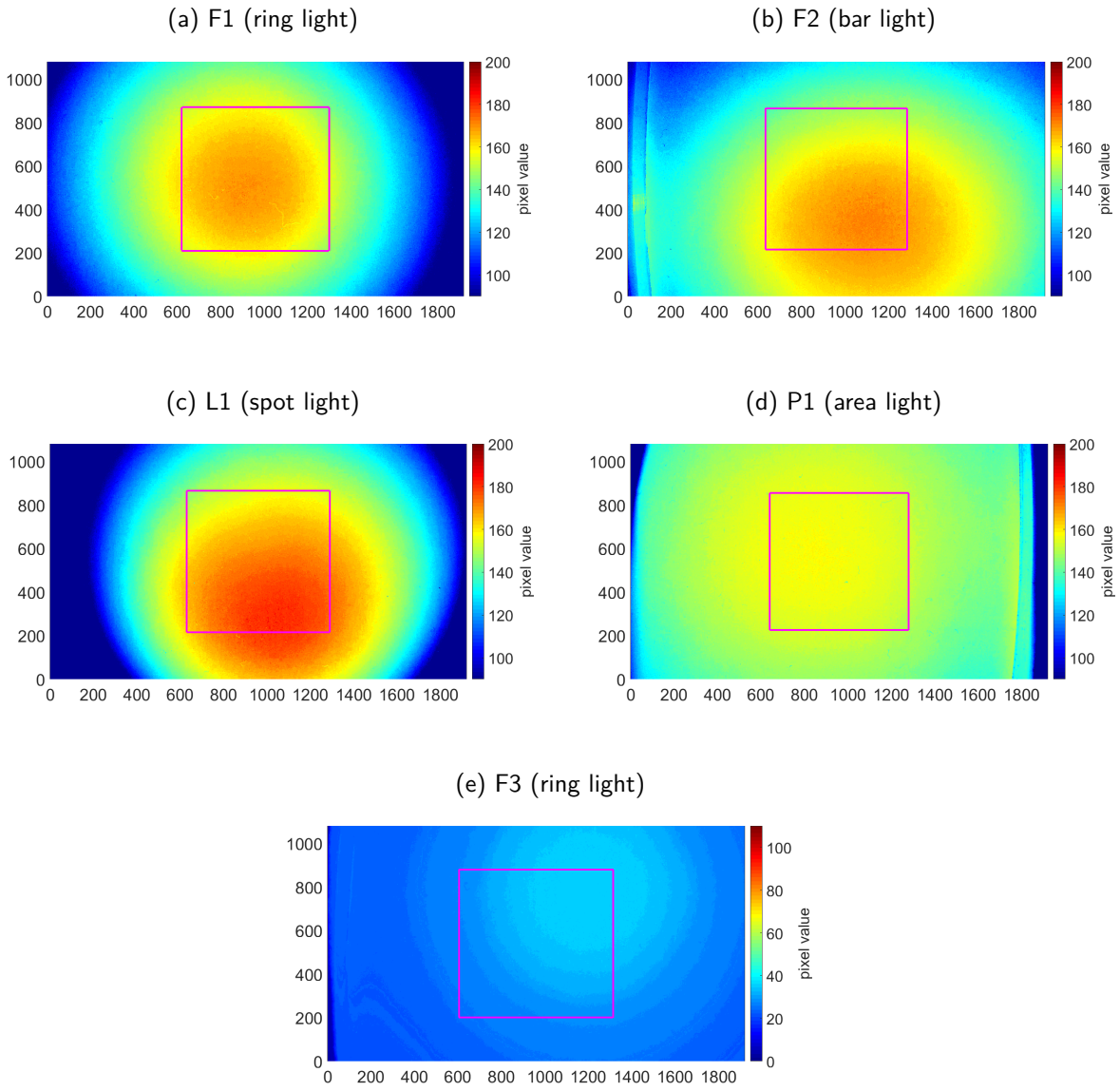


Figure 3.7: Pixel value illustration of a white sheet taken with different illuminations from 135 mm distance between the camera, respectively illumination, and the white sheet; the magenta square illustrates the cut out ROI (50 mm x 50 mm)

The color bar in figure 3.7a - 3.7d is set from 90 (dark blue) to 200 (dark red). As white is represented in the gray level pictures with the value 255, the more red the pixels are, the higher the pixel value is and hence, the brighter the illumination is at this point. While the illuminations (F1, F2, L1, P1) used to take the pictures in figure 3.7a - 3.7d are equipped with white light, the F3 illumination (figure 3.7e) supplies blue light. Thus, for illumination F3, the expected gray level pixel value of the white sheet is approximately 29, following equation 3.5. The color bar in figure 3.7e set from 0 (dark blue) to 110 (dark red), so that the range of the color bar is 110 in all five cases. The red area in the lower center of figure 3.7c shows, that illumination L1 is very bright in this area. The pixel value here is around 180. Considering the whole picture area, the outer edges are less bright (dark blue) and thus, the illumination, considering the total picture, is inhomogeneous. Cutting out the lower center as ROI for the picture processing would lead to higher homogeneity. However, mounting of

this illumination in a way, that the homogeneous and bright spot area is always in the center of the picture and thereby, cut out as ROI in the image processing, is a quite complex task. The ranking considering the set up is discussed in section 3.3.1. Illuminations F1 and F2 offer a less bright light than illumination L1, but the homogeneity of the total picture is higher. In figure 3.7a and 3.7b only the upper edges are dark blue and the color range is not fully exploited. Illumination F3 (figure 3.7e) provides a quite homogeneous light, but compared to F1 the center of the illumination does not fit as adequately to the camera's center. Illumination P1 (figure 3.7d) provides the most homogeneous light in the total FOV of the camera. Since only the ROI is cut out in the image processing, the homogeneity of the cut out area for each illumination is plotted against the distance between the camera and the white sheet, as shown in figure 3.8.

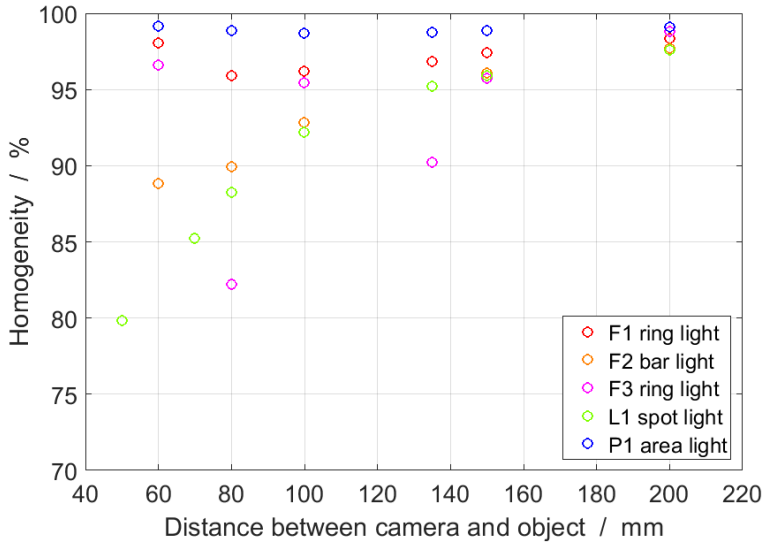


Figure 3.8: Calculated homogeneity plotted against the distance between camera and white sheet for five different illuminations

The plot shows that the area light P1 is the most homogeneous illumination. The homogeneity of this illumination is nearly independent of the distance between the camera and the white sheet and constantly above 98.7 %. Likewise, the ring light F1 provides high homogeneities between 95.92 % and 98.35 %. With exception of the smallest distance (60 mm), the homogeneity increases with the distance between camera and object. This behavior is also observed for the bar light F2 and the spot light L1, but even more distinct. Except for the distance 80 mm and 135 mm, illumination F3 offers homogeneities above 90 %. The deviation could result from misalignment of the centers of camera and illumination as seen in figure 3.7e. Illuminations L1 and F2 are equipped with smaller illumination areas, cf. table 3.4. Therefore, smaller distances between the illumination and the objects lead to less homogeneity in the ROI, because of the higher influence of less illuminated areas such as the edges. The further the distance between the illumination and the object is, the greater the illuminated area is and in conclusion, the higher the homogeneity becomes.
Figure 3.9 exemplary shows the FOV (total picture) and the cut out ROI (magenta square) for two different distances (50/60 mm and 200 mm) between the camera and a white sheet for illumination L1 (3.9a,3.9b) and P1 (3.9c,3.9d). In Appendix A.3 the analogue figures for illumination F1 (A.2a,A.2b), F2 (A.2c,A.2d) and F3 (A.2e,A.2f) are displayed.

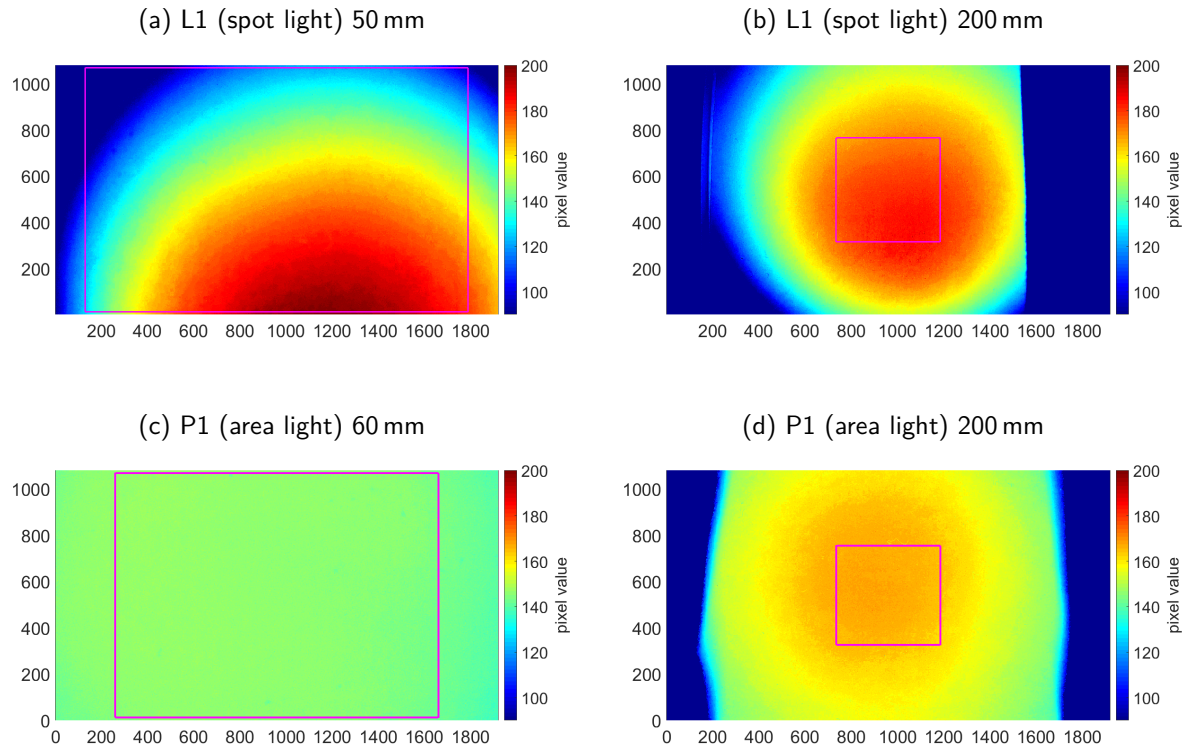


Figure 3.9: Pixel value illustration of a white sheet taken with different illuminations from 50/60 mm (left) and 200 mm (right) distance between the camera, respectively illumination, and the white sheet; the magenta square illustrates the cut out ROI

The influence of edge effects is not only depending on the distance between the illumination and the recorded object, but also on the shape and the size of the illuminated area of the illumination itself. The dark blue regions on the left and the right side of the pictures taken from 200 mm distance are the black molten inner lining of the box for the stationary test. The area light offers a relatively big illumination area and thus, even the edges of the picture taken from 60 mm are well illuminated, cf. figure 3.9c. Figure 3.9a shows, that for the spot light illumination L1 the distance between the camera and the recorded object is important due to the fact, that the illuminated region is divided in spheres with different intensity. With smaller distance, the ROI is a greater part of the FOV and therefore, the homogeneity of the illumination in the ROI is less. Further evaluation of the camera set up optimizing the distance between the camera and the particle film is discussed in section 3.3.2 and the ranking of the illuminations regarding the homogeneity is shown in table 3.5.
Since the total energy of the light is constant, with a greater illuminated area the mean brightness of the illuminated area decreases. The result of the stationary tests regarding the brightness are discussed in the next section.

Brightness

The brightness of the illumination is reviewed using the mean pixel value of the cut out from the pictures (ROI) of a white sheet, cf. figure 3.7. Figure 3.10 shows the mean pixel value (gray scale) plotted against the distance between the camera, respectively the illumination, and the white sheet for the five different illuminations.

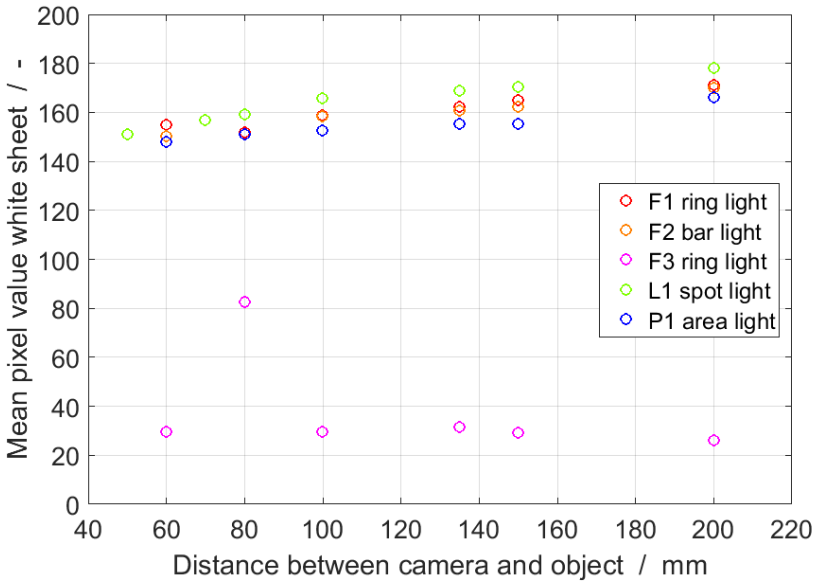


Figure 3.10: Mean pixel value (gray scale) plotted against the distance between camera and white sheet for five different illuminations

In the experiments a white sheet is used and thus, the expected values of the pixels in gray scale for white light illumination are around 200 - 255 and for the blue light illumination around 29. For illumination F3 (blue light) the calculated values correspond to the expected value, but for the other illuminations (white light), the calculated pixel values are between approximately 150 and 180 and therefore, lower than expected. As the same white sheet and the same experimental set up are used in all experiments, the changes in the mean pixel value is assumed to be attributable to the illumination and the distance between the illumination and the white sheet. Illuminations F1, F2, L1 and P1 all show increasing mean pixel values with increasing distance between the illumination and white sheet. This trend is expected due to the fact, that the cut out of the pictures is always the same size in terms of the object size (50 mm x 50 mm). With increasing distance, a larger part of the cut out area is located in the center of the illumination, which is the brightest region of the illuminated area. Thus, the closer the illumination and the object are to each other, the more important the shape and size of the illumination area are. The mean pixel value of a white sheet is depending on the fraction of less illuminated area (e.g. edges), analogue to the effect discusses in the section above and displayed in figure 3.9. The less bright the ROI is, the lower the mean pixel value of a white sheet is. In consequence of the shape, this effect is most distinct for the spot light L1 and less distinct for the area light P1 (except the rise between 150 and 200 mm). The percentual increase of the mean pixel value between 60 mm and 200 mm distance between illumination and object is 17.9 % for illumination L1, 13.4 % for illumination F2, 12.3 % for illumination P1 and 10.4 % for illumination F1. Illumination F3 shows a different graph. Except for the distance of 80 mm, the mean pixel value of the white sheet decreases with increasing distance between the illumination and the white sheet. The mean pixel value decreases 11.4 % between 60 mm and 200 mm distance. This behavior is explicable with the shape and size of the ring light F3. The inner diameter of the ring light F3 is 73 mm, which is quite wide compared to the inner diameter of the ring light F1 (30 mm) and the diameter of the camera lens (22 mm). Thus, the influence of less illuminated edges is reduced. Therefore, the mean pixel value decreases with increasing distance, resulting from the growing size of the illuminated area

while the illumination power remains constant.
As the results of the blue illumination are the closest to the expectations, it is ranked as the best illumination considering the brightness, followed by L1, F1 and F2. Figure 3.10 shows the lowest mean pixel value for white lights for illumination P1. The ranking regarding the brightness is shown in table 3.5.

Difference between SG particles and tracers and tracer recognition

For the tracking of the tracers it is of crucial importance, that the tracers are detected completely and correctly. For this purpose, the mean values of SG particles and tracers and the shape of the distribution function of the number of pixels with a particular pixel value are necessary. To evaluate the capability of the tracer recognition for each illumination, the procedure explained above is conducted with a SG particle tracer mixture (1 wt% tracers). For the cut out, the mean value of the pixels is calculated and, as the tracer mass fraction is low, assumed to be the mean pixel value of the SG particles. The tracer mean pixel value is obtained manually using the online webtool imagecolorpicker [99]. The webtool offers the possibility to upload an image and use a crosshair cursor to find the RGB value of a pixel. The cut outs of the pictures are uploaded to the website and for each picture three different tracers are detected manually and the RGB value of the center is read out with the webtool. Afterwards, the RGB values are converted into gray scale value using equation 3.5 and the mean pixel value is calculated. This procedure is repeated for six different distances for each of the five illuminations. The results are plotted in figure 3.11.

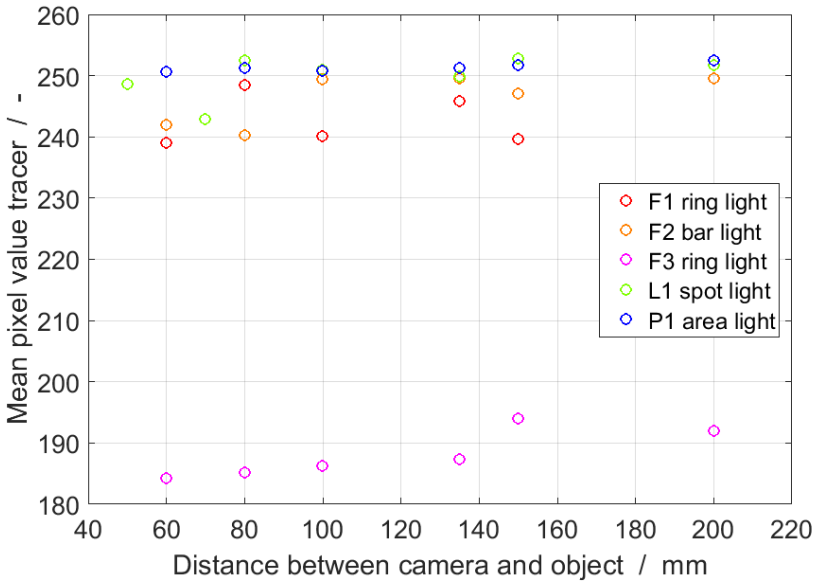


Figure 3.11: Mean pixel value of the tracers for different illuminations plotted against the distances between illumination and SG particle tracer mixture (1 wt% tracers) collected with the webtool imagecolorpicker [99];

The expected value for the tracers using white light illumination with no interference, caused by incident ambient light through a perfectly obscured set up, is 255. As seen in figure 3.11, illuminations L1 and P1 offer a mean pixel value of the tracer around 250 over the total range of tested distances, which is close to the expected value. The mean pixel value of the tracers using illuminations F1 and

F2 is lower than the ones of the other two white illuminations at all tested distances. For illumination F2 the mean tracer value is around 240 for distances lower than 100 mm and around 250 for further distances. Illumination F1 shows mean pixel values of the tracers between 230 and 248 for all tested distances, decreasing with increasing distance. However, the recognition of the tracers using color filter is possible for all four white illuminations, since the pixel value of the SG particles is lower than 200. The tracers' mean pixel values using the blue light illumination F3 is lower than the ones using the white light illuminations over the whole range of tested distances. The values are between 184 and 194 and increase slightly with increasing distance.
For the tracer recognition the difference between the pixel value of the tracers and the one of the SG particles is essential. Thus, for each combination of distance and illumination, the difference between the mean pixel values of tracers and SG particles is calculated and in figure 3.12 plotted against the distance between the camera and the particle collective.

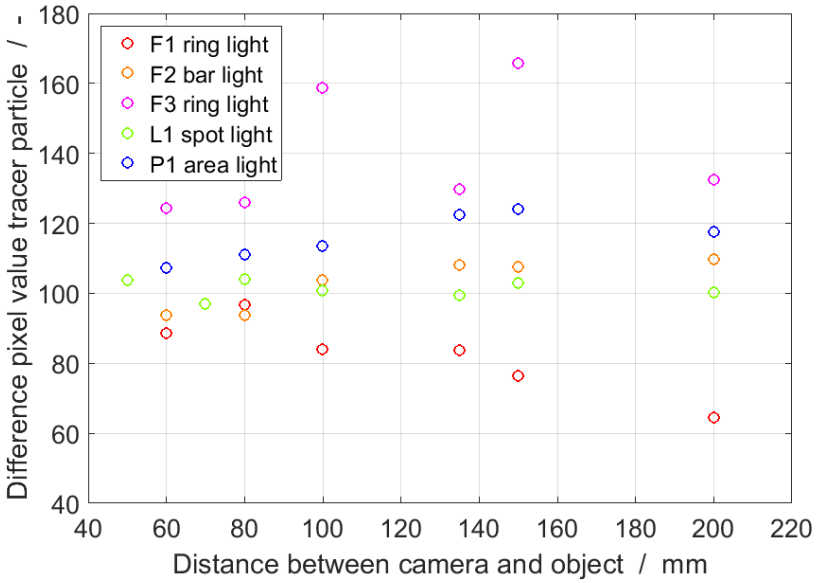
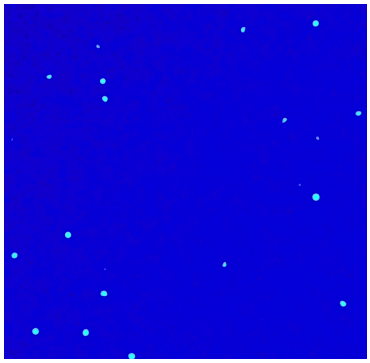


Figure 3.12: Difference between the pixel value (gray scale) of SG particles and tracers plotted for different illuminations against the distance between the camera and the SG particle tracer mixture (1 wt%)

The plot shows that illumination F3 offers the greatest difference in the pixel value between the tracers and the SG particles and hence, the easiest distinction between them. As the pixel values of the SG particles, vary between approximately 27 and 60, the difference of the pixel values is between approximately 125 and 165. These variations are not linearly connected to the distance between the camera and the particle collective. The resulting pixel value differences for 100 and 150 mm distance are about 30 pixels higher than the ones from the other distances. Figure 3.11 indicates, that the deviations for these two distances result from the smaller pixel values of the SG particles, as there is no anomaly in the tracers' mean pixel values at this distances. Figure 3.13 exemplary shows the cut out of a picture of the particle collective, taken with illumination F3 from 100 mm distance on the left side and one from 135 mm distance on the right side.

(a) Distance 100 mm



(b) Distance 135 mm

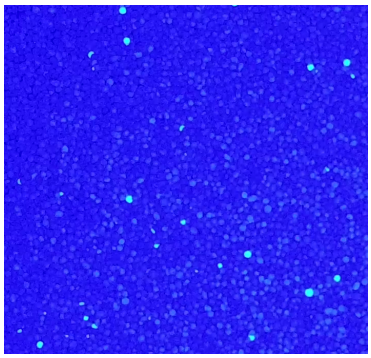


Figure 3.13: SG particle tracer mixture (1 wt% tracers), picture taken from different distances between the camera and the particle collective using illumination F3, cut out 50 mm x 50 mm

While in the right picture (135 mm distance) the single SG particles are clearly visible, in the left picture (100 mm distance) only the tracers are identifiable as bright dots. The picture taken from 150 mm looks similar to the one taken from 100 mm and the pictures taken from the 60, 80 and 200 mm distance appear like the one taken from 135 mm. This difference in the pictures explains the variation in the difference of the pixel value of tracers and SG particles observed in figure 3.12 for illumination F3. Considering the experimental set up and procedure, there is no explanation, why the pictures taken from 100 and 150 mm distance look different. The difference in the pixel value between the SG particles and the tracers for illumination P1 is increasing with the distance between the camera and the particle collective and is the second highest. The increase of the difference between the pixel values of the tracers and the SG particles, with increasing distance, results from the decreasing value of the SG particles, with increasing distance. The further away illumination and object are, the less intense the illumination is and the smaller the pixel value of the red-brownish SG particles becomes. Illumination F2 shows the same behavior. For illumination L1, the difference between the pixel value of the tracers and the SG particles is nearly independent of distance between the camera and the SG particle tracer mixture. Its value is about 100. Using illumination F1, the difference between the pixel value of the tracers and the SG particles is greater with less distance between the camera and the particle collective. For this illumination, the highest difference value is approximately 97 for a distance of 80 mm. At a distance of 200 mm this difference is 64, which corresponds to only 66 % of the maximum difference. In addition to the difference between the mean pixel value of tracers and SG particles, the range of the pixel values for tracers and SG particles is important as well for the evaluation of the tracer recognition. In order to rate the illuminations according to the detection of the white tracers, not only the tracers' mean pixel value and the difference between the mean pixel value of the tracers and the SG particles are considered, but also the shape of the number distribution of the SG particles' pixel value.

Figure 3.14 displays the number distribution of the pixel values in the cut out of the pictures taken of a SG particle tracer mixture (1 wt % tracers) for the five different illuminations. The distance between the illumination, respectively the camera, and the particle collective is always 135 mm.

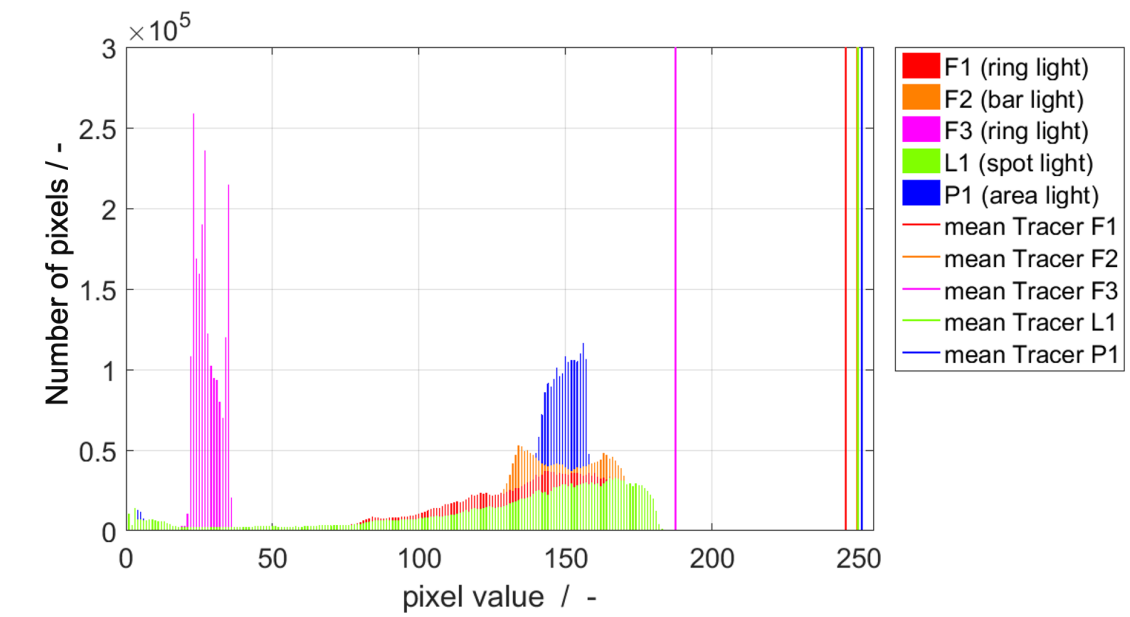


Figure 3.14: Histograms of the SG particles' pixel value and the mean tracer pixel value (vertical lines) for five different illuminations at 135 mm distance

As the range of the tracers' pixel values is quite narrow, the mean pixel value of the tracers is marked in figure 3.14 with a vertical line. The greater the difference between the mean pixel value of the tracer and the right limit of the SG particles pixel value is, the easier the distinction between the SG particles and the tracers becomes.
The histograms confirm the results of the calculated homogeneity, cf. section 3.3.1. The narrower the peaks are, the more homogeneous the illumination is. The blue light illumination F3 has much higher peaks than the white light illuminations. The absolute range of the pixel values is narrower, and since the absolute number of pixels in the cut out of the pictures is always the same at one distance, the peak is higher. But as equation 3.6 states, the homogeneity of the illuminations is normalized by referring to the mean pixel value in order to compare blue and white light illuminations. The shape of the peak is also important for the distinction between SG particle and tracer. The narrower the peaks of SG particles and tracers are and the greater the difference between the top of the peaks is, the easier the distinction between the SG particles and the tracers becomes. Moreover, figure 3.14 shows, that there is not only sufficient difference between the mean pixel value of the SG particles and the tracers, but also between the upper limit of the SG particles' pixel value distribution and the tracers. With the vertical lines in figure 3.14 representing the tracers' mean pixel value, illumination F3 is the best illumination regarding the tracer recognition, respectively the distinction between tracers and SG particles, followed by P1. However, since the distinction between the tracers and the SG particles is possible and sufficiently good with all illuminations, the ranking is not considered to be crucial in the decision process of the illumination for tracking the tracers.

Set up complexity

The complexity of the set up depends on the shape and the size of the illumination. F1 and F3 are ring lights and therefore, the camera can easily be placed on top of them, in a way, that the horizontal alignment of the camera and the illumination is achieved. As the inner diameter of illumination F1 is

30 mm, the width of the camera is 59 mm and the height of the camera is 41 mm, the center of the camera lens and the center of the illuminated area are easily arranged to coincide. For illumination F3 this is also achievable, but since the inner diameter (73 mm) is greater than the dimensions of the camera, the alignment of the centers is more complex. The area illumination P1 also has the advantage of easy horizontal alignment of camera and illumination, but the size and weight of this illumination are much greater compared to the others. This can be a disadvantage in the rotating receiver, because it causes more vibrations in the experimental set up, which can have a negative impact on the cameras' stability and therefore, on the tracking of the tracers and the determination of the film thickness. Illumination F2 is a bar light, which has to be inclined as well as the spot light illumination L1. The inclination angle has to be optimized in order to guarantee a homogeneous and bright illumination in the center of the picture. As seen in figure 3.7b and 3.7c the center of the illuminated area and the center of the camera lens do not coincide and thus, the illumination is less homogeneous. For these two illuminations, not only the inclination angle, but also the horizontal alignment of the camera and the illumination has to be considered additionally in the set up, because it is not feasible to place the camera on top of the illumination. The ranking of the illuminations regarding the set up complexity is shown in table 3.5.

Result

Table 3.5 summarizes the selection parameters and the evaluation of them for the five tested illuminations. The ratings discussed above are illustrated on a scale from ++ (best) to -- (worst).

Table 3.5: Summary of the optical design optimization results regarding the illumination

Illumination	Homogeneity	Brightness	Difference pixel value tracer and particle	Set up	Price	Total
P1	++	--	+	o	-	3
L1	-	+	o	-	o	5
F1	+	o	- /--	++	+	1
F2	o	-	o	-	++	4
F3	--	++	++	+	--	2

In conclusion of the stationary tests, illumination F1 is chosen for the tracking of the tracers. It offers good homogeneity and brightness. Even though the difference between the pixel value of the tracers and the SG particles is the lowest one, it is still sufficient to distinguish between tracers and SG particles. Combined with the easiest set up and an acceptable price, this illumination appears to be the best option for the tracking experiments. Moreover, the thickness of the particle film in the receiver is determined experimentally using an optical method, which is why a second illumination is needed. In this case, the requirements are less strict and the area of the picture is smaller, why the smallest and cheapest illumination (F2) is chosen. As the pictures will be taken of the particle film, the illumination and the camera will be nearly in parallel with the vertically inclined receiver wall. Thus, the small bar light F2 offers the easiest mounting inside the receiver.

3.3.2 Camera set up

The most important parameter in the optimization of the camera set up for tracking the tracers and determining the film thickness is the resolution of the pictures. Hence, the distance between the camera and the tracers, respectively the receiver wall, is optimized. If this distance is too small, the pictures become blurry, if it is too big, the recognition of the tracers is affected and flawed. To investigate the influence of the distance between the camera and the receiver wall, stationary tests with different distances are conducted. The distance between the camera and the receiver wall is determined as a parameter of the length and width of the camera's FOV and the relative size of one pixel to the average diameter of the particles. Figure 3.15 illustrates the characteristic dimensions of the camera set up.

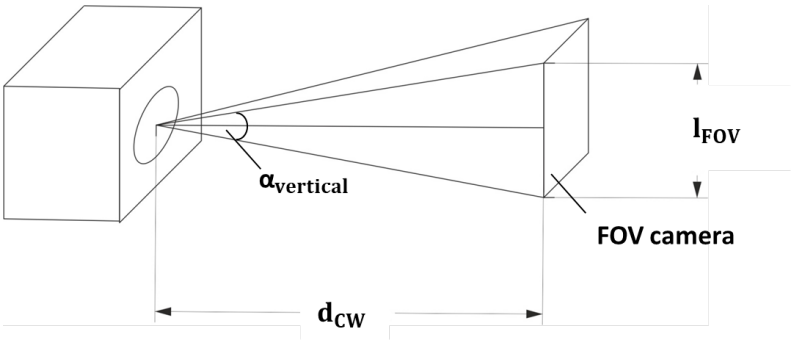


Figure 3.15: Schematic draft of the camera set up illustrating the characteristic dimensions

The minimum area of the picture required is set to 40 mm x 40 mm. This size is defined by the mean axial step of one particle in the receiver per rotation. From results of prior tests this distance is estimated as 4 cm (60 rpm, particle diameter: 1 mm). Consequently, the minimum FOV of the camera is 40 mm x 40 mm. As the camera's horizontal angle of view (100 °) is not much greater than the vertical one (90 °), and the aspect ratio is 16:9, the length of the pixels is the crucial dimension for a sufficient tracer detection. In conclusion, the minimum distance $d_{\mathrm{CW,min}}$ is calculated with equation 3.7 to 20 mm considering the camera's vertical angle of view $\alpha_{\mathrm{vertical}}$ and the minimum length of the FOV l_{min}, cf. figure 3.15.

$$d_{\mathrm{CW,min}} = \frac{l_{\mathrm{FOV,min}}}{2\tan\left(\frac{\alpha_{\mathrm{vertical}}}{2}\right)} \tag{3.7}$$

To decrease the number of particles hitting the camera lens, the minimum distance between the camera and the receiver wall is set to 50 mm. The pixel size has to be smaller than the particle size in order to secure the sufficient resolution of a single particle and thus, the accuracy of the measurement. Hence, the maximum length of one pixel is the diameter of one particle and, to ensure a good recognition of the tracers, a factor of 4 is applied. With the screen resolution of the picture (1920x1080 pixels, cf. section 3.1.3) the maximum length l_{max} of the FOV is calculated as 261.09 mm, using equation 3.8, where $l_{\mathrm{pixel,max}}$ is the maximum length of one pixel (here 1/4 of the median SG particle diameter) and $n_{\mathrm{pixel,len}}$ is the number of pixels lengthwise (here 1080).

$$l_{\mathrm{max}} = l_{\mathrm{pixel,max}}\, n_{\mathrm{pixel,len}} \tag{3.8}$$

For this case, the maximum distance between the camera and the receiver wall $d_{\mathrm{CW,max}}$ is calculated with equation 3.7 to 130.55 mm. In stationary tests six different distances between the camera and

the particle film are examined: 60, 80, 100, 135, 150 and 200 mm. The distances of 150 and 200 mm are evaluated and compared to the smaller distances to justify the calculated maximum distance between the camera and the particle film of 130.55 mm. For each of the five illuminations videos of a grid (5 mm x 5 mm) are recorded from the six different distances. The square in the center of the exported pictures is used to determine the length and the width of one pixel. Thereby, it is assumed, that the distortion of the camera lens is negligible small in the center of the picture. From the dimensions of the pictures and the known screen resolution (1920 x 1080), the sizes of the FOV of the camera for the different distances between the camera and the receiver wall are calculated and displayed in table 3.6. Additionally, the length and width of the pixels and the size of the cut out area are listed in table 3.6.

Table 3.6: Sizes of the FOVs, the pixels, the cut out area and the maximum distance between the positions of one tracer in two consecutive frames for the different distances between the camera and an object (axial (x) and tangential (tan)), videos taken with illumination F1

d_{CW} in mm	Length FOV in mm	Width FOV in mm	Length pixel in mm	Width pixel in mm	Cut out area in mm ^{2}	$d_{\mathrm{max,x}}$ in pixel	$d_{\mathrm{max,tan}}$ in pixel
60	40.3	70.6	0.037	0.037	40 x 50	5.36	5.44
80	50.0	85.7	0.046	0.045	49 x 50	4.32	4.48
100	63.5	110.3	0.059	0.057	50 x 50	3.40	3.48
135	81.8	141.2	0.076	0.074	50 x 50	2.64	2.72
150	91.5	160.0	0.085	0.083	50 x 50	2.36	2.40
200	122.7	208.7	0.114	0.109	50 x 50	1.76	1.84

The frame frequency is set to 60 fps. Pre-tests indicate, that the average velocity in axial direction is approximately 4 cm/s. Considering that the whole circumference is divisible into a moving and a stand-still zone, cf. section 2.4.3, and that the width of the moving zone is approximately 1/3 of the whole circumference, the average axial speed in the moving zone is about 12 cm/s. As those parameters strongly depend on the width of the moving zone and the mass flow, respectively on the residence time of the particles in the receiver, their actual value has to be determined in pre-tests. The maximum axial speed $v_{\mathrm{max,x}}$, the frame frequency f and the maximum length of one pixel $l_{\mathrm{pixel,max}}$ determine the maximum distance between the positions of one tracer in two consecutive frames in axial direction $d_{\mathrm{max,x}}$. The calculation of the tangential speed is done analogously, using the maximum width instead of the maximum length. The maximum distances in pixel are calculated with equation 3.9.

$$d_{\mathrm{max,x}} = \frac{v_{\mathrm{max,x}}}{\mathrm{f}\, l_{\mathrm{pixel,max}}} \tag{3.9}$$

The distance between the positions of one tracer in two consecutive frames is an important parameter in the tracking algorithm, cf. section 4.2. Considering the different distances between the camera and the receiver wall in table 3.6, the maximum axial and tangential distance between the positions of one tracer in two consecutive frames are calculated and listed in table 3.6. As the orientation of the camera is adjustable, length and width of the FOV are interchangeable by rotating the camera.

Moreover, table 3.6 lists the size of the cut out area, the ROI of the picture, depending on the distance between the camera and the particle tracer mixture. For the distances 60 mm and 80 mm, the desired cut out area of 50 mm x 50 mm is not achievable. In these cases, the FOV of the camera is smaller than the demanded ROI and the greatest area possible is cut out. As discussed before, the smaller the distance between the camera and an object is, the closer to one is the ration between ROI and FOV and consequently, the more influencing edge effects of the picture are. On the contrary, the further away the camera is from an object, the smaller the ratio between chosen ROI and FOV is. Consequently, the less part of the picture is cut out and because the center of the cut out and the center of the picture always coincide, the less influencing the edge effects are, as discussed in the evaluation of the homogeneity in section 3.3.1 (cf. figure 3.9).
Figure 3.16 shows the cut out of the pictures taken with the experimental set up explained above using illumination F1, figure 3.17 the analogue pictures using illumination F2. The distance between the camera and the particles varies between the pictures. The pictures on the left are taken with 80 mm distance between the camera and the particles (3.16a, 3.17a), the ones in the center with 135 mm (3.16b, 3.17b) and the ones on the right with 200 mm (3.16c, 3.17c).

(a) $d_{\mathrm{CW}} =$ 80 mm (b) $d_{\mathrm{CW}} =$ 135 mm (c) $d_{\mathrm{CW}} =$ 200 mm

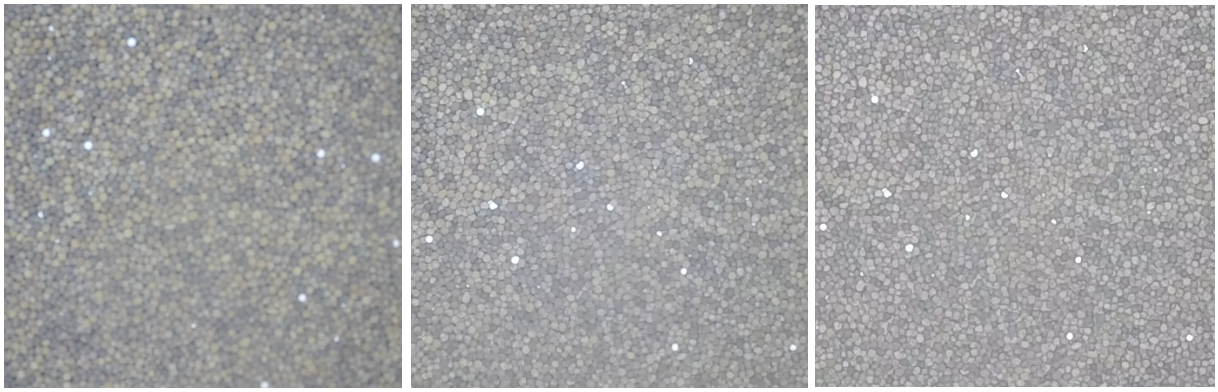


Figure 3.16: Picture of a SG particle tracer mixture (1 wt%) taken from different camera particle distances d_{CW} (80, 135 and 200 mm), illumination: F1

(a) $d_{\mathrm{CW}} =$ 80 mm (b) $d_{\mathrm{CW}} =$ 135 mm (c) $d_{\mathrm{CW}} =$ 200 mm

Figure 3.17: Picture of a SG particle tracer mixture (1 wt%) taken from different camera particle distances $\mathrm{d_{CW}}$ (80, 135 and 200 mm), illumination: F2

The pictures show that for both illuminations a distance of 80 mm between the camera and the SG particle tracer mixture results in blurry pictures. At this short distance, the camera is not able to focus and display the single particles clear and sharp. For the same reason the pictures taken from 60 mm appear blurry as well. Increasing the distance to 100 mm results in only slightly blurry pictures. The pictures taken from further distances (135, 150 and 200 mm) are all sharp. As one important step of tracking the tracers is the detection of the white tracers, blurry pictures might cause errors in their recognition. The further away the camera is from the SG particle tracer mixture, the sharper the pictures are for both illuminations. From this point of view, the maximum possible distance would be the best option. However, not only the sharpness of the picture, but also its resolution is important. With greater distance between the camera and the SG particle tracer mixture, the length and the width of the pixels increases, as listed in table 3.6. Thus, the number of pixels representing one tracer decreases with increasing distance between the camera and the SG particle tracer mixture. In conclusion, the algorithm for detecting the tracers, cf. section 4.2, becomes more error-prone. Regarding the resolution, the distance between the camera and the SG particle tracer mixture should be as short as possible, but the mean SG particle, respectively tracer, diameter should be at least four times bigger than the maximum size of one pixel side. This factor guarantees, that a tracer is represented by at least four complete pixels.

As discussed in the selection of the best possible illumination (section 3.3.1), the intensity of the illumination depends on the distance between the illumination and an object. Beside the optimization of the set up for camera and illumination, it is important to avoid under or overexposure of the SG particle tracer mixture. If the SG particle tracer mixture is overexposed, the range of pixel values in the picture is shifted to higher values and the peak is often cropped at the upper limit of the pixel value range [100; 101]. Underexposure is the opposite and the pixel value range is shifted to the lower limit of the pixel values with nearly no pixels having high values. Pictures of a black and white chessboard using a good illumination should result in a histogram with two clearly distinguishable peaks and a wide range of pixel values, at best from 0 to 255. To review the exposure of illumination F1, chosen to be used in future receiver experiments for tracking the tracers, videos of a chessboard, cf. figure 3.19, are recorded from different distances and the experimental set up explained in section 3.3.1. In figure 3.18 the histograms of pictures exported from these videos are displayed. The maximum number of pixels corresponds to the video resolution, here 1080p, resulting in 1920x1080=2.0736*10^6 pixels.

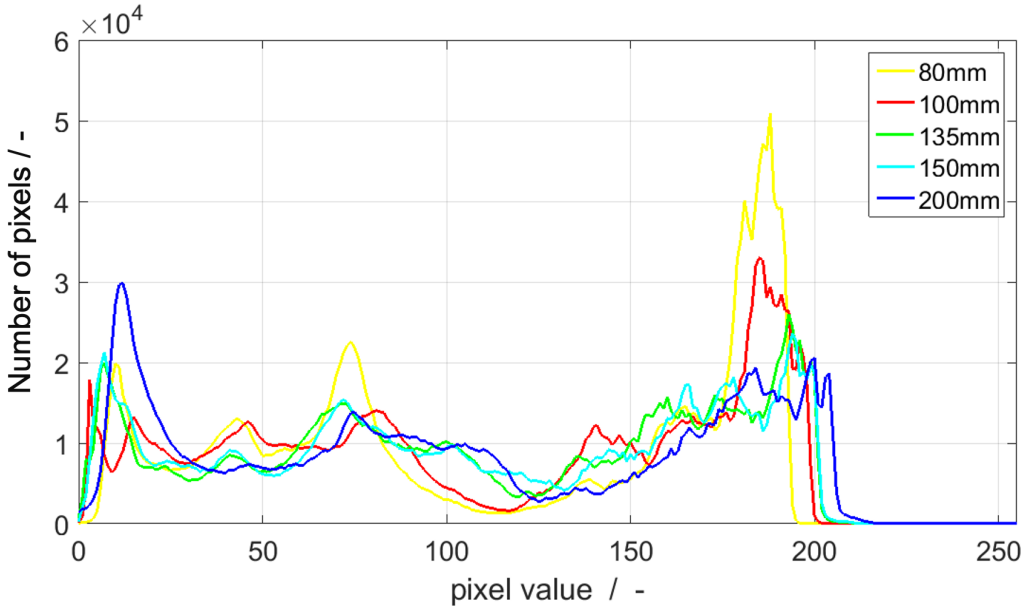


Figure 3.18: Histograms of a black and white chessboard picture for illumination F1 at different distances

All histograms do not show signs of over or underexposure, as the pixel value range for all distances is around 200. Further distance leads to slightly greater pixel value range, but also to less distinctive peaks in the range of bright colors (upper range of pixel values) and in the range of dark tones (lower range of pixel values). Regarding the exposure, from all distances tested in the scope of this thesis, illumination F1 does not over or under expose the object placed below and therefore, can be used in the receiver experiments for tracking the tracers with all tested distances between the camera and the receiver wall.
Considering all criteria explained above, 135 mm is chosen as the most adequate distance between the camera and the SG particle tracer mixture for tracking the tracers and the determination of the film thickness.

3.4 Camera calibration

The cameras used in the experiments are not ideal pinhole cameras. Lenses in real cameras cause distortions in the pictures, that need to be minimized in order to generate corrected data for tracking the tracers in the image processing. To detect and evaluate these errors arising from the nature of the cameras' lenses a calibration is conducted. The theory of geometric camera calibration is explained in section 2.6.
The intrinsic camera calibration parameters, as discussed in section 2.6.2, are determined experimentally with the set up used in the stationary tests and explained in section 3.3.1. Instead of a white paper sheet a chessboard, which is shown in figure 3.19, is placed below the camera. The chessboard is symmetrical and consists of 10 x 7 squares.

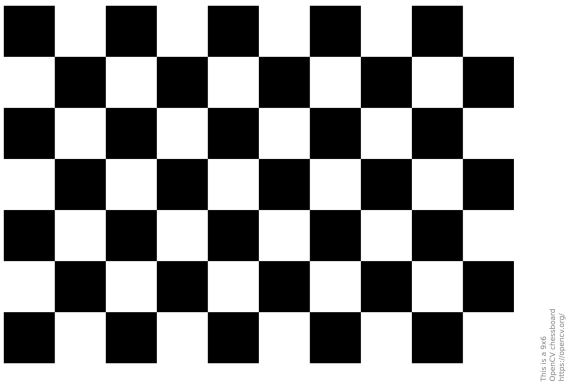


Figure 3.19: Calibration object for camera calibration: symmetrical 10 x 7 chessboard [102]

The chessboard pattern is printed out on a white sheet and glued to a rigid cardboard, so that the chessboard can be moved without being deformed. The maximum distance between the camera and the calibration object is set to 135 mm and illumination F1 or F2 is used during the calibration. For the simultaneous video recording for tracer tracking and film thickness determination, two cameras are needed. In conclusion, the calibration is done for both cameras. In order to do so, illumination F1 is used for camera A2 (tracer tracking) and illumination F2 is used for camera B3 (film thickness determination). The camera and the illumination are mounted onto the item structure used in the stationary tests (cf. figure 3.6) and thus, the position of both remains the same during the recording. The video recording settings of the camera are equal to the ones in the stationary tests, cf. section 3.1.3. While recording, the chessboard pattern is moved up, down, left and right and is also inclined in all directions. Thus, the camera takes images of the chessboard pattern from many different angles of view. As the movement of the chessboard pattern takes around 15 seconds and the camera records 60 frames per second, approximately 900 frames are recorded. To achieve high accuracy, the number of frames used in the calibration should be as high as possible and at least 4 - 5 different orientations are recommended [67]. For the calibration in this thesis 84 frames for camera A2 with illumination F1 and 121 frames for camera B3 with illumination F2 are used to calculate the camera parameters. The complete Python code to determine the camera parameters can be found in Appendix A.4. A detailed description of the functions used in the programming code for the calibration can be found in [72].
In the first step, the Python code reads the pictures considered for the calibration and searches for the chessboard. For this purpose, criteria for the iterative optimization algorithm and the dimensions of the chessboard are needed. After reading the pictures, they are converted into gray scale pictures. Function **cv2.findChessboardCorners()** searches for the positions of the chessboard's internal corners and function **cv2.cornerSubPix()** refines the locations of the found corners using the criteria of the iterative process defined before. The found corners are saved as image points with pixel coordinates and the corresponding object points are saved in world coordinates using the defined dimensions of the chessboard's squares and the assumption of a stationary XY plane with Z=0. The stationary camera and the fact that the chessboard is glued onto a rigid paper, permit this assumption for the sake of simplicity. Figure 3.20 shows the chessboard corners found with the procedure explained above.

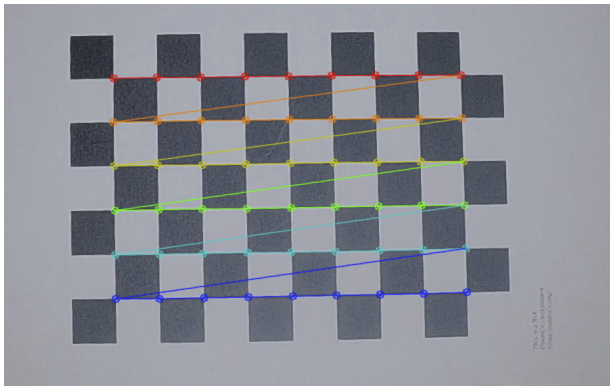


Figure 3.20: Chessboard corners found with the Python code for calibration (cf. Appendix A.4), picture taken with camera A2 using illumination F1

Function **cv2.calibrateCamera()** estimates the intrinsic and extrinsic camera parameters, using several pictures of the chessboard from different angles of view, and returns them. The function is based on algorithms described in [67] and [103]. The algorithm models the lens distortion mathematically and has the advantage of flexible use without the need of an expensive and elaborate calibration set up [67]. In addition to the camera calibration parameters, the Python code calculates the reprojection error. With function **cv2.projectPoints()** the object points (world coordinates) are converted into image points (pixel coordinates), using the determined camera calibration parameters. Consequently, the distances between the found image points and the projected object points are calculated and summarized in order to determine the arithmetical mean of the distances, the reprojection error. The closer this value is to zero, the better the calculated camera parameters are.
The results of the camera calibration for camera A2 and camera B3 are listed in table 3.7 (camera matrix) and in table 3.8 (distortion coefficients). The reprojection error of the camera calibration for camera A2 is 0.0523 pixel and the one for camera B3 is 0.0539 pixel. Thus, the calibrations are considered to be sufficiently precise.

Table 3.7: Calculated camera parameters and their standard deviation: Camera matrix, camera A2 with illumination F1 and camera B3 with illumination F2

		F1		F2	
Parameter	Unit	Value	Standard deviation	Value	Standard deviation
f_{x}	mm	1750.4106	167.1707	1774.5508	102.8925
f_{y}	mm	1756.3095	173.7426	1776.2548	102.4685
C_{x}	-	962.9112	66.9707	978.2048	21.6237
C_{y}	-	469.4679	103.5527	553.3701	58.6115

Table 3.8: Calculated camera parameters and their standard deviation: Distortion coefficients, camera A2 with illumination F1 and camera B3 with illumination F2

		F1		F2	
Parameter	Unit	Value	Error	Value	Error
k_1	-	-0.2732	0.0435	-0.2833	0.0411
k_2	-	0.1802	0.1343	-0.1606	0.1824
p_1	-	0.0028	0.0157	0.0006	0.0088
p_2	-	-0.0048	0.0117	-0.00008	0.0015
k_3	-	-0.2629	0.4660	-0.1409	0.7106

4 Image Processing Code

In this chapter the image processing codes developed in the scope of this thesis are explained in detail. The general video processing, including the correction of lens distortion of the camera, is described in the first section. Subsequently, the steps of the image processing code for tracer tracking are presented. Finally, the image processing code developed to evaluate the thickness of the particle film is explained. The image processing codes are implemented in Python using the OpenCV library [104]. All parameters, which have to be adjusted regarding the experimental set up in the receiver and the operation conditions in the experiments with the receiver, are explained in this chapter. Additionally, start values for the iterative determination of these parameters, resulting from pre-test with the experimental set up explained in section 3.3.1, are given.

4.1 General video processing

The evaluation of the recorded videos is done by means of a Python code using each frame of the video one by one. Thus, the processing of the videos is explained exemplary for one frame.
After reading the videos frame by frame, the calibration parameters, cf. section 3.4, are used in the first video processing step to correct the recorded frames. The Python code for the generation of corrected data can be found in Appendix A.5. For the purpose of correction, function **cv2.undistort()** is used and the camera's FOV is cropped to the new ROI. To define the new ROI, function **cv2.getOptimalNewCameraMatrix()** is utilized. Thereby, the width and the length of the actual frame, the camera matrix and the distortion coefficients imported previously as well as one cropping parameter are the input parameters. The cropping parameter can have values between 0 and 1 and allows to adjust the size of the cropped frame manually. If this value is set to 0, the whole undistorted FOV of the camera is taken. This can result in black regions at the edges of the pictures, originated in the undistortion of the frame. If the value of pixels at the edges of the original image are shifted to the center in the corrected image by the undistortion function, these edge pixels are assigned with zeros as pixel value in the corrected image, illustrated by black color. If the cropping parameter is set to 1, the frame is cropped maximum, so that no black regions at the edges in the corrected image are left. Some tests with pictures of the chessboard, taken during the calibration, and different values of the cropping parameter are conducted. As a result of these tests, the cropping parameter is set to 0, but it has to be confirmed when changing the experimental set up.
Figure 4.1 shows pictures of a chessboard taken from 135 mm distance with camera A2 using illumination F1 and the set up from the stationary tests, as explained in section 3.3.1. On the left side (figure 4.1a) the original image is displayed and on the right side (figure 4.1b) the resulting picture applying the calibration parameters listed in table 3.7 and 3.8.

(a) Picture before calibration (b) Picture after applying the calibration results

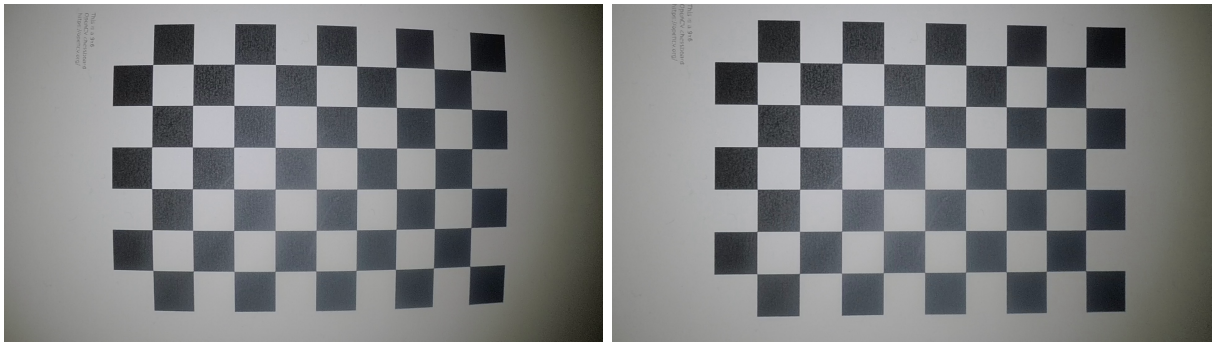


Figure 4.1: Symmetrical 10 x 7 chessboard, picture taken from 135 mm distance between camera A2 and the chessboard using illumination F1

The original picture (figure 4.1a) shows barrel distortion as the straight lines from the chessboard are visibly curved outwards. After undistorting the picture using the calculated camera parameters, the chessboard's lines appear straight, as seen in figure 4.1b. Thus, the distance between the corners of the chessboard is nearly constant. Table 4.1 lists the mean horizontal and vertical distances between two points in the grid of the chessboard.

Table 4.1: Mean horizontal and vertical distances between two grid points for the original and the corrected image and their standard deviation, picture taken from 135 mm distance between camera A2 and the chessboard using illumination F1

Frame	Horizontal distance in pixel	Standard deviation in pixel	Vertical distance in pixel	Standard deviation in pixel
Original	127.7567	4.9242	130.2236	2.6053
Corrected	131.4423	3.0725	132.8104	1.6895

As the chessboard is symmetrical, all the distances, vertical and horizontal ones, are equal in the object plane. Therefore, the mean horizontal and vertical distances between two close grid points should be as similar as possible. Table 4.1 shows, that the deviation between the mean horizontal and the mean vertical distance in the original image is greater than in the corrected one. Whereas the mean vertical distance in the original image is approximately 1.9 % greater than the horizontal one, it is only 1 % wider in the corrected image. Considering the standard deviation of the mean distances, the distances in both directions are more uniformly distributed in the corrected frame than in the original frame. In consequence, the correction of the frames using the calibration parameters and the Python code explained above is considered to be successful.

4.2 Tracer tracking

The general steps of the Python code for tracking the tracers are the following:

1. Segmentation
2. Filter
3. Determination of the center of gravity
4. Tracking algorithm

A schematic diagram of the procedure for detection of the tracers (step 1 - 3) can be found in Appendix A.6. In this section each step is explained in detail. The whole Python code for detecting and tracking the tracers can be found in Appendix A.7. The figures shown in this section to describe the steps of the Python code are all taken with camera A2 using illumination F1 and a distance between the camera, respectively the illumination, and the SG particle tracer mixture (1 wt% tracers) of 135 mm. For better visibility, an area of 50 mm x 50 mm is cut out from the frames.

Segmentation

The first step in the tracer tracking procedure is the segmentation of the white tracers from the particles in the pictures. Figure 4.2 shows a picture of the SG particle tracer mixture (1 wt% tracers) before (figure 4.2a) and after (figure 4.2b) the segmentation procedure. Furthermore, the result after applying the filters, cf. section 4.2, is displayed in figure 4.2c.

(a) Before segmentation (b) After segmentation (c) After applying filter

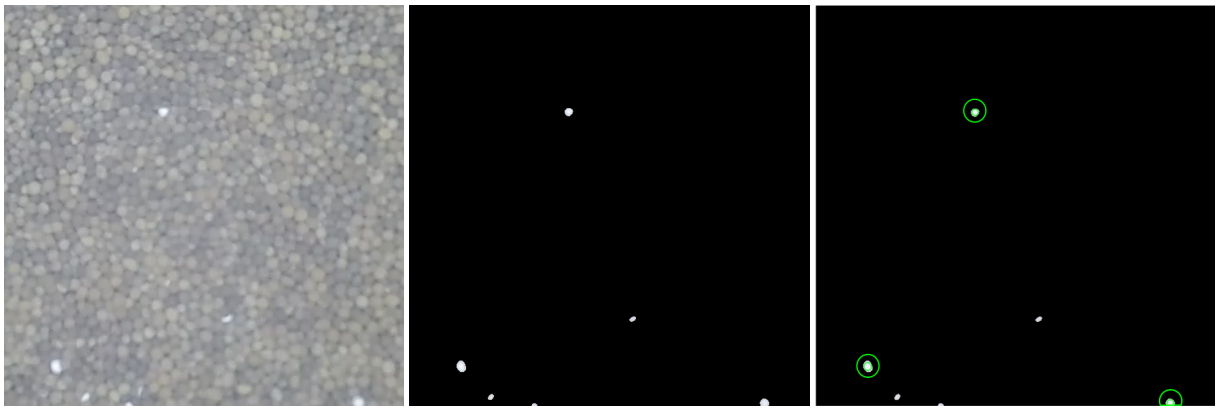


Figure 4.2: First two steps of the tracer tracking (Segmentation and Filter), detected tracers (green circles), picture taken from 135 mm distance between camera A2 and the SG particle tracer mixture (1 wt% tracers) using illumination F1

In order to distinguish the tracers from the SG particles, the pixel values of each frame are converted to HSV values. Furthermore, the HSV pixel values, representing the upper and lower limit for the tracer color, are defined depending on a sensitivity parameter. This parameter has to be adjusted to the pixel values of the tracers in the recorded videos with the experimental set up in the receiver. It can have values between 0 (only pure white considered as tracer color) and 100 (no segmentation).

Based on stationary tests, described in chapter 3, the start value of the sensitivity parameter is set to 50, resulting in a good segmentation between the tracers and the rest of the picture, cf. figure 4.2b. In the next step, a mask thresholds the image using function **cv2.inRange()**. In the resulting mask each pixel has either the value 255, if its value in the original image is in between the defined pixel value range, or, in case the pixel value in the original image is not in the defined pixel value range, its assigned value is 0. Afterwards, function **cv2.bitwise_and()** creates a new image by filling the pixel values either with the pixel value of the original image (in case the pixel value of the mask is 255) or with the value 0. To smooth the image, function **cv2.medianBlur()** can be used optionally. The input parameters of this function are the picture to be smoothed and the size of the aperture area (blursize x blursize) in which the median filter is used. Figure 4.2b shows the resulting picture of the SG particle tracer mixture (1 wt% tracers) at the end of the segmentation step.

Filter

In the segmentation step, all pixels with a pixel value outside of a specified value range are blanked out and set to 0. All other pixels keep their original pixel value. In the filter step, some characteristics of the tracers, such as area size or circularity, are used to detect possible tracers in the frame. Function **cv2.SimpleBlobDetector_Params()** is used to define the parameters for the tracer detection. The Python code offers the possibility of filtering the detectable tracers in the frame by color, area, circularity, convexity and inertia. Moreover, thresholds and minimum values for the distance between two tracers as well as the repeatability can be selected. The minimum threshold is set to 200 and the maximum to 255. In the scope of this thesis, filtering by one single color or by convexity is not performed for detecting the tracers. The following settings are results of iterative tests with the set up explained in section 3.1.4 and proposed as start values for the estimation of the values to track the tracers in the future experimental set up:

- **filterByArea:**
  (distance between camera and object: 135 mm)
  params.minArea = 83 (pixel^2)
  params.maxArea = 180 (pixel^2)
- **filterByCircularity:**
  params.minCircularity = 0.7
- **filterByInertia:**
  params.minInertiaRatio = 0.01
- params.minDistBetweenBlobs = 1.0 (pixel)
- params.minRepeatability = 2

The limits for the area filter are calculated with the mean tracer diameter (0.983 mm, cf. section 3.2.1) and the minimum (0.074 mm/pixel, cf. table 3.6) and maximum (0.076 mm/pixel, cf. table 3.6) dimension of a pixel at the distance of 135 mm between the camera and the SG particle tracer mixture. The area is determined as a circle with the mean tracer diameter and transformed from metric scale to pixel. The parameter of the minimum area *params.minArea* is set to 60 % of the calculated one and the one of the maximum area *params.maxArea* to 137 % of the calculated maximum area to

minimize errors originating from deviations in the difference between the camera and the particle film. The minimum value of the circularity *params.minCircularity* filter is set to 0.7, as the manufacturer states in the data sheet, that at least 95 % of the tracers show a sphericity greater than 0.7 [84]. The inertia ratio (*params.minInertiaRatio*) is the ratio between the distance of the closest point from the center and the furthermost one. For a line this value is 0, for a circle it is 1 and for elliptical geometries this value lies between 0 and 1. The minimum distance between two detected tracers *params.minDistBetweenBlob* is set to 1 pixel. If two detected tracers are located closer than this distance, they are merged. For each threshold from the minimum threshold to the maximum one, the SimpleBlobDetector finds possible tracers. The repeatability value *params.minRepeatability* is the lower limit of the amount, how often one tracer needs to be found in the different threshold steps to be detected. The higher the repeatability value is chosen, the more stable a detected tracer is across the different thresholds. In figure 4.2c the detected tracers are illustrated with green circles. After the segmentation step bright areas, that do not match the filters, are sorted out and not considered in the detection of the tracers using function **detector.detect()**, in which **detector** is the SimpleBlobDetector created using the parameters defined by the filters explained above.

Determination of the center of gravity

Important information, such as the coordinates of the center of gravity and the diameter of the detected tracers, is stored in the list **keypoints**. For each frame the coordinates (x,y) of each detected tracer are saved to a text file named *ExperimentName_4BE_framenumber.txt* and stored in the *doPTV* folder of the experiment. With the list of the detected tracers, the minimum and maximum diameter of the detected tracers in one frame as well as the mean diameter and its standard deviation are calculated. The results are exported to a text file named *ExperimentName_keydata_framenumber.txt*

Tracking algorithm

The fourth step of the tracer tracking is to connect matching positions of tracers in order to monitor their trajectories. To do so, an algorithm based on the 4BestEstimate algorithm [105] is implemented. In the following, the principle idea of this algorithm is explained and the created Python code is presented in detail.
At first some parameters for the tracking algorithm have to be selected. The most important parameters are the radii of the areas, in which the algorithm searches for possible candidates of sequential tracers. As discussed in section 3.3.2, the sizes of the radii depend on the maximum distance between the positions of one tracer in two consecutive frames and on the distance between camera and object. In consequence, these values have to be adapted to the characteristics of the experimental set up. Possible start values for the radii are listed in the following:

- **Search candidates for F1**
  radiusF1 = 30 (pixel)
- **Search candidates for F2**
  radiusF2 = 30 (pixel)
- **Search candidates for F3**
  radiusF3 = 30 (pixel)

The purpose of the Python code for tracking the tracers is to obtain the positions of the tracers and the length of their trajectories. In order to do so, the detected tracers, represented by the pixel coordinates of their center of gravity, are stored into matrices, from which possible trajectories (tracks) are extractable afterwards. Therefore, a matrix for the x coordinates of the centers of gravity (**trackingx**) and a matrix for the y coordinates of the centers of gravity (**trackingy**) are created. They have the same size (**dots_total_2 x i**), which is defined by the total number of detected dots (**dots_total_2**) in all frames of the recorded video (**i**). After defining the parameters and creating the storage matrices, the main loop, which repeats the procedure explained in the following for each frame of the recorded video, is executed. The algorithm needs at least four frames, which are referred to the numbers $0, 1, 2$ and 3, to detect trajectories. For each step in the loop, the actual frame is represented by number 0 and the following frames accordingly by $1, 2$ and 3. The code can be divided into nine steps repeated in the main loop:

1. Import positions of detected tracers in frames F0 - F3
2. Find matching tracers in the first two frames (**couplesF0F1**)
3. Calculate velocities **v1_x** and **v1_y**
4. Estimate tracers' positions in third frame (**center2_g**)
5. Find possible trajectories in the first three frames (**couplesF0F1F2c**)
6. Estimate tracers' positions in fourth frame (**center3_g**) and find matching tracers in the first four frames (**couplesF0F1F2cF3c**)
7. Find positions of tracers in third frame, which are part of a trajectory (**foundF1F2**)
8. Find positions of tracers in the third frame, which are not part of a trajectory (**center2_not**)
9. Append positions of tracers in the third frame (found in step 7 and 8) to **trackingx** and **trackingy**

In the first step, the Python code reads the detected tracers in the actual frame (step 1.1) and in the three subsequent frames (step 1.2 - step 1.4) by opening the text files, which are created in the determination of the center of gravity step (cf. section 4.2) and stored in the *doPTV* folder of the experiment. In each of the four frames used in the actual loop step, the coordinates of the centers of gravity of the detected tracers are imported and stored in lists (e.g. **x_0**, **y_0**, **center0**). Additionally, if the actual frame is the first frame, the coordinates of the detected tracers are saved into the first column of the storage matrices **trackingx** and **trackingy**.
The second step of the Python code for tracking the tracers is to search for matching sequential positions of the centers of gravity in frame 0 (actual) and in frame 1. Found couples are saved into the list **couplesF0F1**. Figure 4.3 schematically shows the second step of the tracking algorithm.

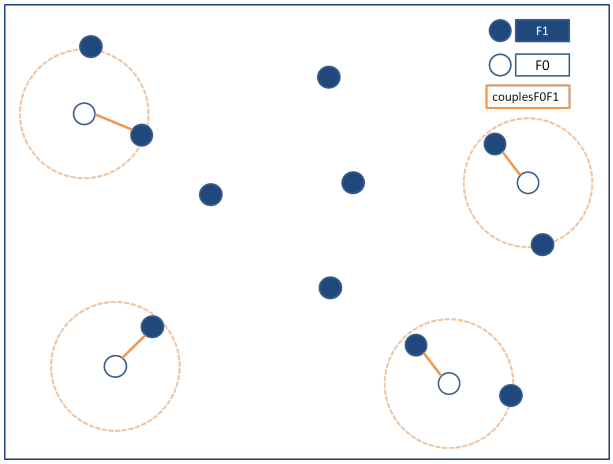


Figure 4.3: Schematic diagram of the tracer tacking algorithm
Step 2: Find matching tracers in the first two frames (**couplesF0F1**)

The empty circles represent the center of detected tracers in frame 0 and the filled dark blue circles illustrate the detected tracers in frame 1. The dotted orange circles depict the search volume with radius **radiusF1** surrounding the detected tracers in frame 0. For each detected tracer in frame 0, all detected tracers in frame 1, whose position lies within the circular search volume, are considered to be possible candidates for creating a trajectory. With the Nearest Neighbor approach, the tracer in frame 1, which yields the minimum distance to the position of the tracer in frame 0, is selected as the best matching option [105]. In figure 4.3 the resulting couples (**couplesF0F1**) are illustrated with an orange line. In case that the actual frame is the first one, the detected tracers in the second frame (F1) are saved to the storage matrices **trackingx** and **trackingy**. Thereby, **couplesF0F1** is used to find the index of the adequate row in the storage matrices, to add the tracers' positions in F1 to the storage matrices corresponding to the positions of the matching tracers in F0. Additionally, for F0 being the first frame of the video, all tracers detected in the second frame, which are not assigned to a suitable tracer in the first frame (**center1_not**), are appended to the storage matrices **trackingx** and **trackingy** in rows below the previously saved tracers.
In the third step of the tracer tracking algorithm, the velocities in both directions (**v_x,1** and **v_y,1**) are calculated with equation 4.1, within which $u_{\mathrm{i,n}}$ represents the x or y value of the detected tracer in frame $\mathrm{F_n}$ and $\Delta \mathrm{t}$ is the time step between two consecutive frames $\mathrm{F_n}$ and $\mathrm{F_{n-1}}$.

$$v_{\mathrm{i,n}} = \frac{u_{\mathrm{i,n}} - u_{\mathrm{i,n-1}}}{\Delta t} \tag{4.1}$$

In the fourth step of the tracer tracking algorithm, the calculated velocities $v_{\mathrm{i,n}}$ are used to determine the estimated position of the tracers in the third frame (F2) $u_{\mathrm{i,n+1}}$ for each couple in **couplesF0F1**, based on the position of the tracers detected in frame F1 $u_{\mathrm{i,n}}$. The pixel coordinates of these estimated positions are calculated with equation 4.2 [105] and saved in the list **center2_g**.

$$u_{\mathrm{i,n+1}} = u_{\mathrm{i,n}} + v_{\mathrm{i,n}} \Delta t \tag{4.2}$$

Consequently, in the fifth step, candidates of detected tracers in frame F2, which are located within a circular search volume with the radius **radiusF2** surrounding the estimated positions in frame F2

(**center2_g**), are found and saved in the list **cand_2**. Figure 4.4 schematically shows the fifth step of the tracer tracking procedure.

(a) Find **cand_2** (b) Found possible tracks **couplesF0F1F2c**

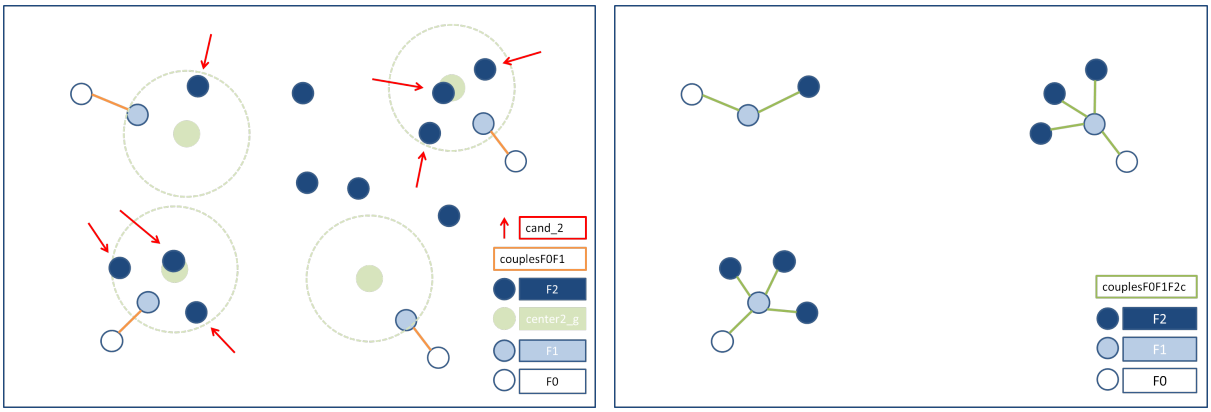


Figure 4.4: Schematic diagram of the tracer tacking algorithm
Step 5: Find possible trajectories in the first three frames (**couplesF0F1F2c**)

Analogue to figure 4.3 the orange connections in figure 4.4a illustrate **couplesF0F1** found in previous steps. The green dots represent the estimated positions of the tracers in F2 for each couple of F0 and F1 in **couplesF0F1**. The surrounding search volume is marked with a green dotted circle. All tracers detected in F2 are represented by filled dark blue circles. The ones located inside the search volume are the candidates, stored in the list **cand_2** and marked with a red arrow in figure 4.4a. To find matching triplets, stored in the list **couplesF0F1F2c**, the algorithm distinguishes between two different cases, which are shown in figure 4.4a. If there are no candidates for tracers in F2 (cf. bottom right corner in figure 4.4a), the track is ended. If there are candidates (cf. upper left, upper right and bottom left corner in figure 4.4a), the position of these tracers is added to the list **couplesF1F2F2c**, containing the coordinates of the tracers in F1 (**center1**), the matching estimated positions in F2 (**center2_g**) and the possible candidates in F2 (**cand_2**).
After the determination of the possible matching tracers detected in F2, the list **couplesF1F2F2c** is converted to the list **couplesF0F1F2c** by adding the matching tracers detected in F0 and removing the estimated position of the tracers in frame F2. The resulting list contains three sequential points of each possible track (F0, F1 and F2c). In figure 4.4b the resulting **couplesF0F1F2c** are illustrated with green connection lines.
To decide, if the triplets stored in **couplesF0F1F2c** are considered as tracks, the positions of the tracers detected in the fourth frame (F3) are examined in the sixth step, as shown schematically in figure 4.5.

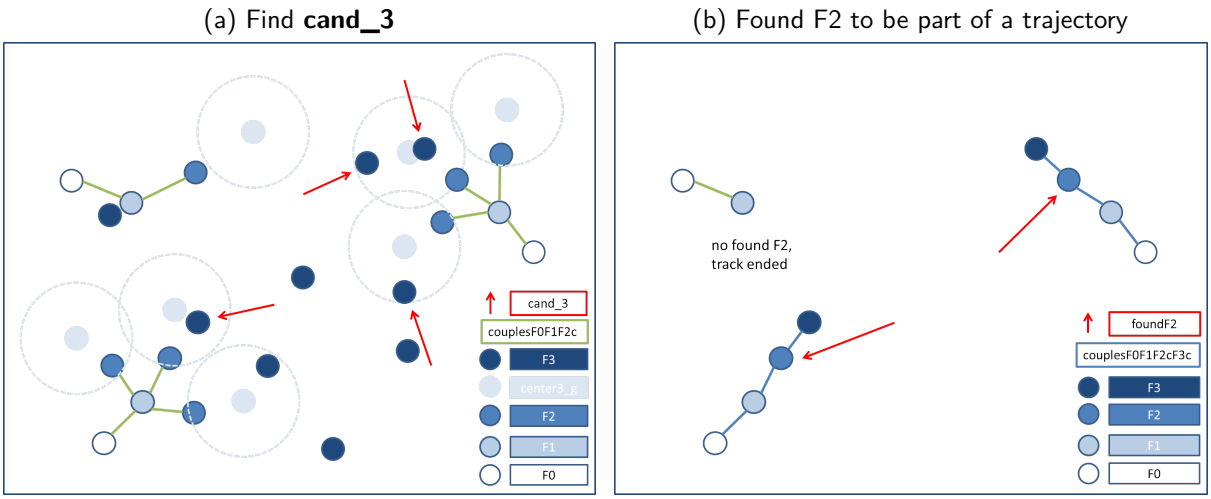


Figure 4.5: Schematic diagram of the tracer tacking algorithm
Step 6: Estimate tracers' positions in fourth frame (**center3_g**) and find matching tracers in the first four frames (**couplesF0F1F2cF3c**)

With the positions of the tracers in each triplet (**couplesF0F1F2c**), the velocities $v_{\mathrm{i,n}}$ (equation 4.1) and the acceleration $a_{\mathrm{i,n}}$ (equation 4.3) are calculated and consequently, the associated positions of tracers in frame F3 (**center3_g**) $u_{\mathrm{i,n+2}}$ are estimated with equation 4.4 [105].

$$a_{\mathrm{i,n}} = \frac{v_{\mathrm{i,n}} - v_{\mathrm{i,n-1}}}{\Delta t} \tag{4.3}$$

$$u_{\mathrm{i,n+2}} = u_{\mathrm{i,n}} + v_{\mathrm{i,n}}(2\Delta t) + a_{\mathrm{i,n}}(2\Delta t)^2 \tag{4.4}$$

Within equations 4.3 and 4.4 $u_{\mathrm{i,n}}$ represents the x or y value of the detected tracer in frame $\mathrm{F_n}$ and Δt the time step between two consecutive frames.
In figure 4.5 these estimated points are represented by light blue dots. The tracers detected in frame F3 are illustrated with filled dark blue circles. Those tracers, which are located within the circular search volume with radius **radiusF3** surrounding the estimated point (**center3_g**), as illustrated with a light blue dotted circle, are considered to be potential candidates for the previous found **couplesF0F1F2c**. These candidates are stored in the list **cand_3** and, in figure 4.5a, marked with red arrows.
To find the tracers in F3, which are possible extensions of already existing tracks, stored in list **couplesF0F1F2cF3c**, the algorithm distinguishes between three different cases, as shown in figure 4.5a. If there are no candidates for tracers in F3 (cf. upper left corner in figure 4.5a), no matching tracer can be found in F3. Consequently, according to the 4 Best Estimate approach, no tracer detected in F2 is appended to the track and track is ended. If there are candidates in F3 (cf. upper right and bottom left corner in figure 4.5a), their positions are added to the list with candidates of tracers detected in frame F3 **cand_3**. If this list contains only one argument, the coordinates of the tracers' positions in F1 and F2 in this possible track together with the distance between the estimated tracer position in frame F3 (**center3_g**) and the candidate found in frame F3 (**cand_3**) are saved into the list **foundcouplesF1F2c**. If the list **cand_3** consists of more than one argument,

the candidate with the smallest distance between the estimated tracer position in frame F3 and the position of the candidate in F3 is selected. For this candidate, the positions of the matching tracers in frame F1 and F2 together with the minimum distance are appended to list **foundcouplesF1F2c**. In the 4BestEstimate algorithm, the tracer detected in frame F2 resulting in the minimum distance between the estimated position in frame F3 and the position of one tracer found in F3, is selected as the most suitable option [105]. Therefore, in the seventh step the algorithm searches for the track, consisting of the tracers' position in F1 and the one in F2, selected to be the best match and saves this couple to the list **foundF1F2**.
Afterwards, all tracers detected in frame F2 with no matching tracer in frame F1, are saved to the list **center2_not** in the eighth step of the tracer tracking algorithm.
In the last step of the algorithm, the coordinates of the tracers detected in frame F2 and found to be part of one track, are saved to the storage matrices **trackingx** and **trackingy**. Thereby, list **foundF1F2** is used to append the coordinates of the tracers found in frame F2 to the adequate row in the storage matrices, according to the matching tracers found in frame F1. Additionally, all tracers detected in frame F2, which are not considered to be part of an already existing track (**center2_not**), are saved in the storage matrices **trackingx** and **trackingy** in rows below the already found tracks. Finally, the frame number is increased by one and the steps of the main loop are repeated.
After closing the main loop, the storage matrices **trackingx** and **trackingy** are reduced to the maximum necessary size, saved to text files (*ExperimentName_trackingx_final.txt* and *ExperimentName_trackingy_final.txt*) and stored in the *Tracking* folder of the experiment. Additionally, every track is exported (*ExperimentName_trackingx_tracknumber.txt*) and saved in the *Tracking/Tracks* folder of the experiment.

4.3 Film thickness

The film thickness is determined experimentally by image processing of videos recorded with a GoPro camera. A white pin, similar to the one shown in figure 4.6, with black marked lines is used as scale. The width of the white and black lines is 1 mm. For a good recognition the lines on the pin should be painted with non reflecting black color.

Figure 4.6: Pin used as scale in the determination of the film thickness

The pin will be mounted onto an item structure inside the receiver in a way, that the small end is pressed against the receiver wall. Before carrying out first experiments with this set up, a calibration measurement has to be conducted. Therefore, the distance between the receiver wall and the first line of the pin has to be measured. This value is an important parameter used in the evaluation of the calibration for the film thickness determination. The Python code for the calibration can be found in Appendix A.8.1.

After reading the frames of the video, the calibration parameters of the camera are used to undistort the picture, cf. section 4.1. In the segmentation step, the pin is detected and the rest is blended out, as shown in figure 4.7a. The procedure is analogue to the segmentation step in the tracer tracking algorithm (cf. section 4.2). Additionally, function **cv2.Canny()** is used to detect the edges of the pin and the scale. The result of this function is shown in figure 4.7b.

(a) Segmentation (b) Detected edges (c) Detected Lines

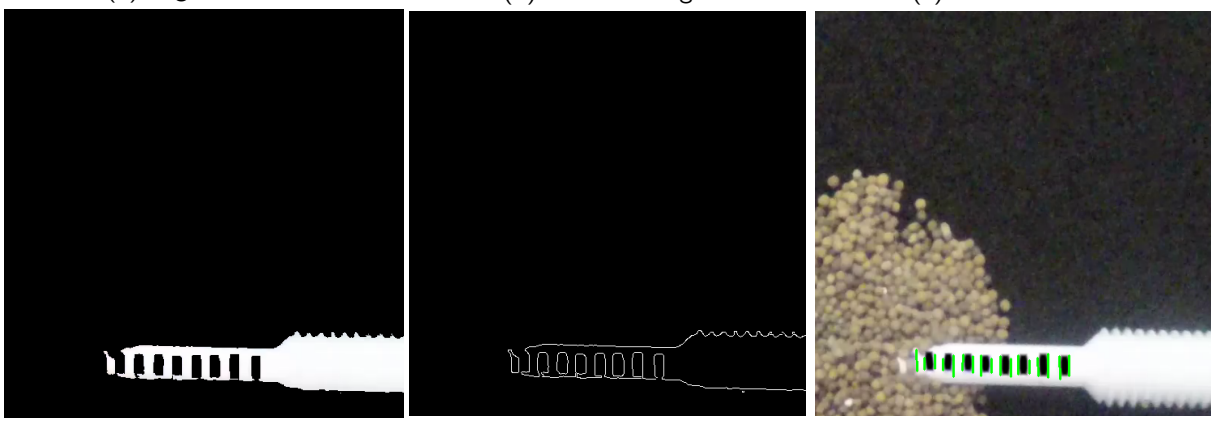


Figure 4.7: Results of the steps in the calibration for the determination of the film thickness

Within the edges found before, function **cv2.HoughLines()** searches for lines that meet specified criteria. Depending on the experimental set up (e.g. distance between camera/illumination and the particle film, inclination of the camera and the illumination, size of the cut out region and color of the lines on the pin), the values for the two parameters **minLineLength** and **maxLineGap** have to be set. **minLineLength** is the minimum length of lines, which means, that detected lines shorter than this value are sorted out. **maxLineGap** represents the maximum distance between two line segments to treat them as segments of the same line. If the gap between two line segments, each fulfilling the other restrictions, is wider than this value, the segments are considered to be two different lines. To prevent the detection of horizontal lines, **minangle** is the minimum angle between a line and the horizontal plane. If this angle is smaller, resulting in a line inclined to the horizontal, the detected line is excluded. As a result of pre-tests, the following values are found and proposed as start values while adjusting the parameters explained before for experiments in the receiver.

- minLineLength = 4 (pixel)
- maxLineGap = 5 (pixel)
- minangle =85 (degree)

As the detected lines are nearly vertical, the mean value of the x coordinates of the the two final points of each detected line is calculated. If the algorithm detects two lines, whose mean x values are closer than 5 pixels to each other, they are considered to be representing one line in the scale on the pin. Therefore, the mean of the two mean x values is calculated. In the next step, the selected lines, represented by their mean x value, are sorted by value and correlated with the film thickness, represented by the scale on the pin. At this point, the measured distance between the receiver

wall and the scale on the pin is used to associate the pixel coordinates with real coordinates of the experimental set up. With a linear fit, the film thickness can be estimated from the x value of one point in the picture. Therefore, the slope and the intercept of the linear fit from the calibration are exported to a text file (*CalibrationFT.txt*). This text file can afterwards be used in the determination of the film thickness in experiments with a particle film. In figure 4.8 the green lines illustrate different film thicknesses. The lines are calculated using the linear parameters and equation 4.5.

$$FT = slope \cdot x_{\mathrm{maxdot}} + intercept \tag{4.5}$$

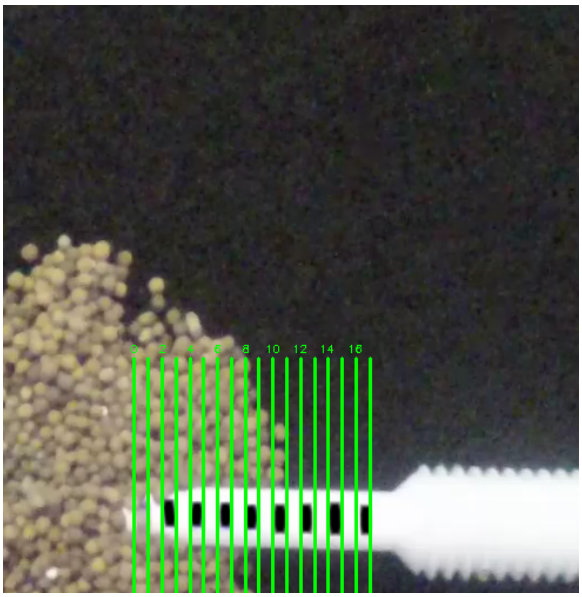


Figure 4.8: Determination of the film thickness, scale resulting from the calibration

The Python code for the determination of the film thickness consists of the following steps, which are repeated in a main loop for every frame of the analyzed video:

1. Undistortion of the frame
2. Detection of particles with the SimpleBlobDetector
3. Determination of the centers of gravity
4. Find coordinates of the most upper particle of the particle film
5. Calculation of the film thickness using the linear fit parameters of the calibration

The related Python code can be found in Appendix A.8.2. The first steps of the algorithm are the same ones as in the tracer tracking algorithm: Correction of the data (cf. section 4.1), detection of the particles (cf. section 4.2) and determination of the centers of gravity (cf. section 4.2). A detailed description of these steps can be found in the referred sections. Only the parameters in the SimpleBlobDetector, described in section 4.2, are adjusted, in order to not only detect the white tracers, but rather all particles in the particle tracer mixture. These parameters have to be checked with videos recorded in future experiments and adjusted accordingly. As a result of pre-tests, the following values are proposed as start values:

- **Thresholds:**
  params.thresholdStep = 5
  params.minThreshold = 60
  params.maxThreshold = 255
- **filterByArea:**
  (distance between camera and object: 135 mm)
  params.minArea = 40 (pixel^2)
  params.minArea = 175 (pixel^2)
- **filterByCircularity:**
  params.minCircularity = 0.7
- **filterByInertia:**
  params.minInertiaRatio = 0.01
  params.maxInertiaRatio = 1
- params.minDistBetweenBlobs = 1.0 (pixel)
- params.minRepeatability = 2

After detecting all particles, the one with the greatest x value, representing the uppermost particle on the particle film surface, is selected, as shown in figure 4.9b.

(a) Picture before detection of the particles

(b) Detected particles, the center of gravity of the highest detected center (red line) and highest point of the particle film (blue line)

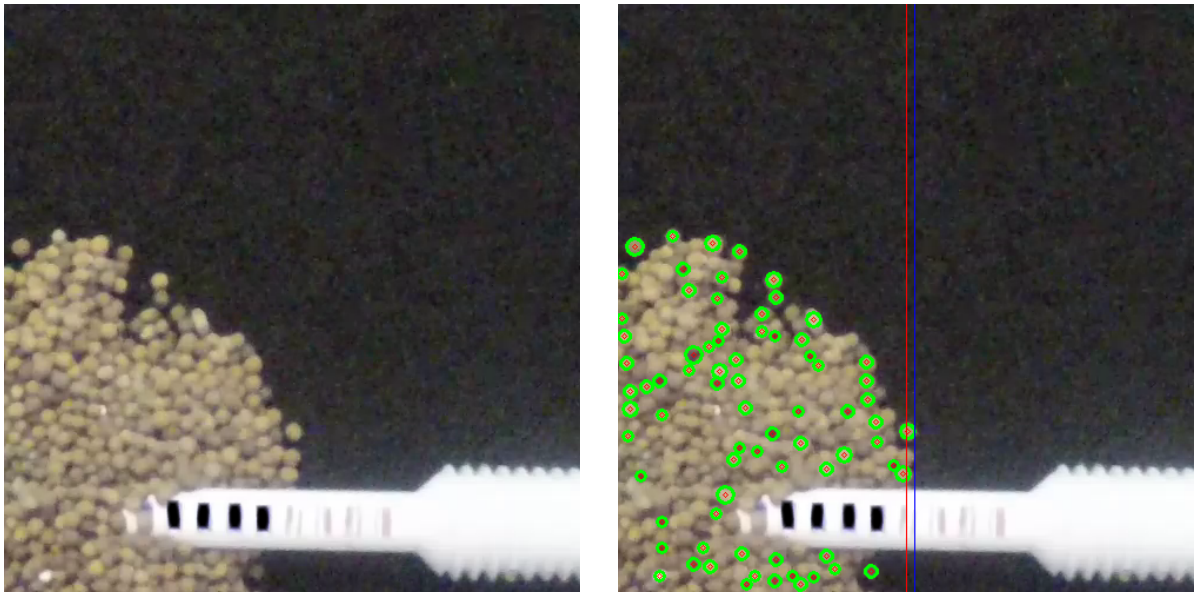


Figure 4.9: Determination of the film thickness; comparison before (a) and after detection of the particles (b)

With the results of the calibration the particle film thickness is calculated from the x value of the highest particle, using equation 4.5. The conversion between the x value of the highest particle and

the corresponding film thickness is done by linear regression of the calibration results, as explained before. Thus, the accuracy can be improved by detecting the scale of the pin in every frame. To do so, the black lines of the pin have to be painted very accurately and in such a way, that the lines are detected correctly in every frame. Furthermore, pre-tests have to be conducted to check the detection of the lines and to adjust the detection parameters of the lines (cf. section 4.3) and the particles (cf. section 4.3) accordingly.

5 Conclusion and Outlook

The aim of this thesis was to develop and optimize an optical system and an associated image processing code to investigate the particle motion in a solar particle receiver. Additionally, an image processing code to evaluate the particle film thickness had to be created and camera calibration had to be conducted.

In preparation, it was necessary to geometrically characterize the SG particles and the tracers, which will be used in future experiments in the prototype receiver at DLR Stuttgart, by sieving analysis. Additionally, the static angle of repose of pure SG particles, pure tracers as well as a SG particle tracer mixture was determined experimentally. These properties are important parameters in the numerical modelling of the particle motion in the receiver and they confirm the assumption, that the tracers do not significantly influence the flow properties of the particle mixture. Thus, the particle motion in the receiver can be investigated by tracking the tracers. Furthermore, an optical system for the investigation of the particle motion in the solar particle receiver, that is compatible to the existing experimental set up of the receiver prototype, was designed and optimized. Stationary experiments were conducted in order to select the best illumination and the optimum distance between the camera and the particle film, flowing down the receiver wall. Five different illumination options with different characteristics, such as shape and color of light, are compared. As selection parameters of the illumination the homogeneity, the brightness and the ability of tracer recognition as well as the set up complexity and the costs were chosen and the illuminations were ranked accordingly. To find the optimum distance between the camera and the receiver wall, pictures taken with the selected illuminations and the cameras from different distances were evaluated. Subsequently, an image processing code for tracer tracking was developed in Python within the scope of this thesis. This code detects the tracers in each frame of the video and it includes an algorithm, based on the 4 Best Estimate algorithm, which finds matching tracers in sequential frames and assembles their positions in order to find the tracers' trajectories. To evaluate the particle film thickness during the experiments and thus, be able to correlate the characteristics of the particle motion to the thickness of the particle film, an image processing code for the determination of the film thickness was developed in Python. Additionally, the optical system for evaluating the film thickness was designed in agreement with the existing experimental set up, and it was optimized regarding the illumination and the camera set up. To improve the results of the image processing codes for tracer tracking and determination of the film thickness, camera calibration was accomplished for both cameras, which will be used in future experiments in the receiver prototype. For this purpose, effects resulting from the nature of the cameras' lenses, such as distortion, were investigated. Moreover, an image processing code in Python was developed to determine the camera parameters (intrinsic, extrinsic and distortion coefficients) and generate corrected data, which subsequently will be used in the image processing codes for evaluation of the particle motion and the film thickness.

In future experiments with the receiver prototype, parameters of the image processing codes for tracer tracking and determination of the film thickness have to be adjusted accordingly, in order to achieve sufficiently good recognition of the tracers, respectively the particles and lines. These parameters are explained in chapter 4 and additionally, start values are given, which were experimentally obtained in the scope of this thesis. In order to be able to determine the film thickness, a calibration of the pin scale has to be conducted in the experimental set up inside the receiver prototype. Moreover, the results of the experiments with the receiver prototype can be used to calculate the proposed validation parameters and can be compared to simulation results in order to see, if these parameters are suitable for the validation of the numerical model. Furthermore, the length of the trajectories, found with the tracking algorithm, could be extended by considering the velocity of the tracers to be constant, if no matching tracer is found in the consecutive frame. The subsequent search with this constant velocity in the frame after next could give the possibility to obtain longer trajectories, even if the tracer is not visible in all of the frames. Otherwise, the found and exported trajectories can be searched for matching extensions to generate longer trajectories with tracers sinking and emerging through the particle film surface.

Bibliography

[1] IEA. World energy outlook 2019 executive summary. Technical report, International Energy Agency, IEA, Paris, 2019. [Online]. [Accessed 22.06.20]. Available from: `https://www.iea.org/reports/world-energy-outlook-2019`.

[2] IRENA. Global energy transformation: A roadmap to 2050 (2019 edition). Technical report, International Renewable Energy Agency, Abu Dhabi, 2019. ISBN 978-92-9260-121-8.

[3] IRENA. Global renewables outlook: Energy transformation 2050. Technical report, International Renewable Energy Agency, Abu Dhabi, 2020. ISBN 978-92-9260-238-3.

[4] MIT. The future of solar energy an interdisciplinary MIT study. Technical report, Massachusetts Institute of Technology, 2015. ISBN 978-0-928008-9-8.

[5] IRENA. Renewable power generation costs in 2018. Technical report, International Renewable Energy Agency, Abu Dhabi, 2019. ISBN 978-92-9260-126-3.

[6] Mayorkinos Papaelias, Fausto Pedro García Márquez, and Isaac Segovia Ramirez. *Concentrated Solar Power: Present and Future*, pages 51–61. Springer International Publishing, Cham, 2018.

[7] IEA. Solar energy: Mapping the road ahead. Technical report, International Energy Agency, IEA, Paris, 2019. [Online]. [Accessed 22.06.20]. Available from: `https://www.iea.org/reports/solar-energy-mapping-the-road-ahead`.

[8] IRENA. Global energy transformation: A roadmap to 2050 (2018 edition). Technical report, International Renewable Energy Agency, Abu Dhabi, 2018. ISBN 978-92-9260-059-4.

[9] J. Forrester. The value of csp with thermal energy storage in providing grid stability. *Energy Procedia*, 49:1632 – 1641, 2014. Proceedings of the SolarPACES 2013 International Conference.

[10] Martin Kaltschmitt, Marina Stegelmeier, and Wolfgang Streicher. *Solarthermische Wärmenutzung*, pages 181–262. Springer-Verlag, Berlin, Heidelberg, 2013.

[11] Martin Kaltschmitt, Kornelia Lippitsch, Jörg Müller, Stefan Reichert, Detlef Schulz, and Simon Schwunk. *Photovoltaische Stromerzeugung*, pages 353–452. Springer-Verlag, Berlin, Heidelberg, 2013.

[12] Martin Kaltschmitt, Gerhard Weinrebe, and Christina Wulf. *Solarthermische Stromerzeugung*, pages 263–351. Springer-Verlag, Berlin, Heidelberg, 2013.

[13] Pere Mir-Artigues, Pablo del Río, and Natàlia Caldés. *Short History, Recent Facts, and the Prospects of Concentrating Solar Power Generation*, pages 23–84. Springer International Publishing, Cham, 2019.

[14] Peter Viebahn, Stefan Kronshage, Franz Trieb, and Yolanda Lechon. Final report on technical data, costs, and life cycle inventories of solar thermal power plants. Technical report, 2008. [Online]. [Accessed 15.01.20]. Available from: `https://www.solarthermalworld.org/`.

[15] Robert Stieglitz and Volker Heinzel. *Solarthermische Hochtemperatursysteme*, pages 487–594. Springer-Verlag, Berlin, Heidelberg, 2012.

[16] Keith Lovegrove and Mike Dennis. Solar thermal energy systems in australia. *International Journal of Environmental Studies*, 63(6):791–802, 2006.

[17] Patricia Palenzuela, Diego-César Alarcón-Padilla, and Guillermo Zaragoza. *Combined Fresh Water and Power Production: State of the Art*, pages 27–60. Springer International Publishing, Cham, 2015.

[18] Fulya Verdier. Mena regional water outlook, part ii, desalination using renewable energy. Final report, Fichtner and DLR, 2011. [Online]. [Accessed 22.06.20]. Available from: `https://www.dlr.de/tt/Portaldata/41/Resources/dokumente/institut/system/projects/MENA_REGIONAL_WATER_OUTLOOK.pdf`.

[19] Franz Trieb. Concentrating solar power for seawater desalination. Executive summary, German Aerospace Center (DLR) Institute of Technical Thermodynamics Section Systems Analysis and Technology Assessment, 2007. [Online]. [Accessed 22.06.20]. Available from: `https://www.dlr.de/tt/Portaldata/41/Resources/dokumente/institut/system/publications/Trieb_AQUA-CSP-Full-Report-Final.pdf`.

[20] W. Wu, L. Amsbeck, R. Buck, R. Uhlig, and R. Ritz-Paal. Proof of concept test of a centrifugal particle receiver. *Energy Procedia*, 49:560 – 568, 2014.

[21] Clifford K. Ho and Brian D. Iverson. Review of high-temperature central receiver designs for concentrating solar power. *Renewable and Sustainable Energy Reviews*, 29:835–846, 2014.

[22] Lifeng Li, Joe Coventry, Roman Bader, John Pye, and Wojciech Lipiński. Optics of solar central receiver systems: a review. *Optics Express*, 24(14):A985–A1007, 2016.

[23] Sotirios Karellas and Tryfon C. Roumpedakis. Chapter 7 - solar thermal power plants. In Francesco Calise, Massimo Dentice D'Accadia, Massimo Santarelli, Andrea Lanzini, and Domenico Ferrero, editors, *Solar Hydrogen Production*, pages 179 – 235. Academic Press, 2019.

[24] Spiros Alexopoulos and Bernhard Hoffschmidt. *Concentrating Receiver Systems (Solar Power Tower)*, pages 29–71. Springer-Verlag, New York, NY, 2013.

[25] Antonio L. Ávila Marín. Volumetric receivers in solar thermal power plants with central receiver system technology: A review. *Solar Energy*, 85(5):891 – 910, 2011.

[26] Csaba Singer. *Stand des Wissens*, pages 5–22. Springer Fachmedien, Wiesbaden, 2013.

[27] Apurv Kumar, Wojciech Lipiński, and Jin-Soo Kim. Numerical modelling of radiation absorption in a novel multi-stage free-falling particle receiver. *International Journal of Heat and Mass Transfer*, 146:118821, 2020.

[28] Thomas Wetzel, Julio Pacio, Luca Marocco, A. Weisenburger, A. Heinzel, Wolfgang Hering, Carsten Schroer, Georg Müller, Jürgen Konys, Robert Stieglitz, Joachim Fuchs, Joachim Knebel, C. Fazio, M. Daubner, and Frank Fellmoser. Liquid metal technology for concentrated solar power systems: Contribution by the german research program. *AIMS Energy*, 2:89–98, 2014.

[29] Annette Heinzel, Wolfgang Hering, Jürgen Konys, Luca Marocco, Karsten Litfin, Georg Müller, Julio Pacio, Carsten Schroer, Robert Stieglitz, Leonid Stoppel, Alfons Weisenburger, and Thomas Wetzel. Liquid metals as efficient high-temperature heat-transport fluids. *Energy Technology*, 5(7):1026–1036, 2017.

[30] J Martin and Vitko J Jr. Ascuas: a solar central receiver utilizing a solid thermal carrier. Technical report, Sandia National Labs., Livermore, CA (USA), 1982. [Online]. [Accessed 21.06.20]. Available from: `https://www.osti.gov/biblio/5663779/`.

[31] J M Hruby. Technical feasibility study of a solid particle solar central receiver for high temperature applications. Technical report, Sandia National Labs., Livermore, CA (USA), 1986. [Online]. [Accessed 21.06.20]. Available from: `https://www.osti.gov/biblio/5905477-technical-feasibility-study-solid-particle-solar-central-receiver-high-temperature-applications`.

[32] C. Ho, J. Christian, D. Gill, A. Moya, S. Jeter, S. Abdel-Khalik, D. Sadowski, N. Siegel, H. Al-Ansary, L. Amsbeck, B. Gobereit, and R. Buck. Technology advancements for next generation falling particle receivers. *Energy Procedia*, 49:398 – 407, 2014. Proceedings of the SolarPACES 2013 International Conference.

[33] Jinping Wang, Jun Wang, P. Lund, and Hongxia Zhu. Thermal performance analysis of a direct-heated recompression supercritical carbon dioxide brayton cycle using solar concentrators. *Energies*, 12:4358, 2019.

[34] Luca Massidda and Alberto Varone. A numerical analysis of a high temperature solar collecting tube, using helium as an heat transfer fluid. Technical report, CRS4 Centro di Ricerca, Sviluppo e Studi Superiori in Sardegna, 06 2007.

[35] K. Vignarooban, Xinhai Xu, A. Arvay, K. Hsu, and A.M. Kannan. Heat transfer fluids for concentrating solar power systems – a review. *Applied Energy*, 146:383 – 396, 2015.

[36] National Renewable Energy Laboratory (NREL). *La Dehesa*. [Online]. [Accessed: 23.06.20]. Available from: `https://solarpaces.nrel.gov/la-dehesa`, 2011.

[37] National Renewable Energy Laboratory (NREL). *Solacor 2*. [Online]. [Accessed: 23.06.20]. Available from: `https://solarpaces.nrel.gov/solacor-2`, 2012.

[38] National Renewable Energy Laboratory (NREL). *Planta Solar 20 (PS20)*. [Online]. [Accessed: 23.06.20]. Available from: `https://solarpaces.nrel.gov/planta-solar-20`, 2009.

[39] National Renewable Energy Laboratory (NREL). *Ivanpah Solar Electric Generating System*. [Online]. [Accessed: 23.06.20]. Available from: `https://solarpaces.nrel.gov/ivanpah-solar-electric-generating-system`, 2014.

[40] National Renewable Energy Laboratory (NREL). *Solar Tower Juelich*. [Online]. [Accessed: 23.06.20]. Available from: `https://solarpaces.nrel.gov/julich-solar-tower`, 2008.

[41] Ming Liu, Martin Belusko, N.H. Steven Tay, and Frank Bruno. Impact of the heat transfer fluid in a flat plate phase change thermal storage unit for concentrated solar tower plants. *Solar Energy*, 101:220 – 231, 2014.

[42] J. Pacio and Th. Wetzel. Assessment of liquid metal technology status and research paths for their use as efficient heat transfer fluids in solar central receiver systems. *Solar Energy*, 93:11 – 22, 2013.

[43] National Renewable Energy Laboratory (NREL). *Crescent Dunes Solar Energy Project*. [Online]. [Accessed: 23.06.20]. Available from: `https://solarpaces.nrel.gov/crescent-dunes-solar-energy-project`, 2015.

[44] Clifford K. Ho. A review of high-temperature particle receivers for concentrating solar power. *Applied Thermal Engineering*, 109:958 – 969, 2016. Special Issue: Solar Energy Research Institute for India and the United States (SERIIUS) – Concentrated Solar Power.

[45] M.J. Bignon. The influence of the heat transfer fluid on the receiver design. *Electric Power Systems Research*, 3(1-2):99–109, 1980.

[46] Anish Modi and Fredrik Haglind. Performance analysis of a kalina cycle for a central receiver solar thermal power plant with direct steam generation. *Applied Thermal Engineering*, 65(1):201 – 208, 2014.

[47] Csaba Singer, Reiner Buck, Robert Pitz-Paal, and Hans Müller-Steinhagen. Assessment of Solar Power Tower Driven Ultrasupercritical Steam Cycles Applying Tubular Central Receivers With Varied Heat Transfer Media. *Journal of Solar Energy Engineering*, 132(4), 10 2010.

[48] Qiang Peng, Jing Ding, Xiaolan Wei, Jianping Yang, and Xiaoxi Yang. The preparation and properties of multi-component molten salts. *Applied Energy*, 87(9):2812 – 2817, 2010. Special Issue of the IGEC-IV, the 4th International Green Energy Conference (IGEC-IV), Beijing, China, October 20–22, 2008 Special Issue of the First International Conference on Applied Energy, ICAE'09, Hong Kong, January 5–7, 2009.

[49] Rene I. Olivares. The thermal stability of molten nitrite/nitrates salt for solar thermal energy storage in different atmospheres. *Solar Energy*, 86(9):2576–2583, 2012.

[50] Gregory J. Kolb, Richard B. Diver, and Nathan Siegel. Central-Station Solar Hydrogen Power Plant. *Journal of Solar Energy Engineering*, 129(2):179–183, 04 2006.

[51] J. Flesch, A. Fritsch, G. Cammi, L. Marocco, F. Fellmoser, J. Pacio, and Th. Wetzel. Construction of a test facility for demonstration of a liquid lead-bismuth-cooled 10kw thermal receiver in a solar furnace arrangement - sommer. *Energy Procedia*, 69:1259 – 1268, 2015. International Conference on Concentrating Solar Power and Chemical Energy Systems, SolarPACES 2014.

[52] Alejandro Calderón, Camila Barreneche, Anabel Palacios Trujillo, Mercè Segarra, Cristina Prieto, Alfonso Rodriguez-Sanchez, and Ana Fernández. Review of solid particle materials for heat

transfer fluid and thermal energy storage in solar thermal power plants. *Energy Storage*, page e63, 05 2019.

[53] Nathan P. Siegel, Michael D. Gross, and Robert Coury. The Development of Direct Absorption and Storage Media for Falling Particle Solar Central Receivers. *Journal of Solar Energy Engineering*, 137(4), 08 2015.

[54] J. W. Griffin, K. A. Stahl, and R. B. Pettit. Optical properties of solid particle receiver materials: I. angular scattering and extinction characteristics of norton masterbeads®. *Solar Energy Materials*, 14(3):395–416, 1986.

[55] Birgit Gobereit, Lars Amsbeck, Reiner Buck, Robert Pitz-Paal, Marc Röger, and Hans Müller-Steinhagen. Assessment of a falling solid particle receiver with numerical simulation. *Solar Energy*, 115:505–517, 2015.

[56] Kaijun Jiang, Xiaoze Du, Yanqiang Kong, Chao Xu, and Xing Ju. A comprehensive review on solid particle receivers of concentrated solar power. *Renewable and Sustainable Energy Reviews*, 116:109463, 2019.

[57] Lars Amsbeck, Reiner Buck, Birgit Gobereit, Marc Röger, and Wei Wu. Patent DE10 2010 062 367.9: Solarstrahlungsempfängervorrichtung und Verfahren zur solaren Erhitzung von Wärmeträgermedium, 2010.

[58] Wei Wu. *A Centrifugal Particle Receiver for High-Temperature Solar Applications*. Ph.d. thesis, RWTH Aachen University, 01 2014.

[59] W. Wu, R. Uhlig, R. Buck, and R. Pitz-Paal. Numerical simulation of a centrifugal particle receiver for high-temperature concentrating solar applications. *Numerical Heat Transfer, Part A: Applications*, 68(2):133–149, 2015.

[60] Miriam Ebert, Lars Amsbeck, Andrea Jensch, Johannes Hertel, Jens Rheinländer, David Trebing, Ralf Uhlig, and Reiner Buck. Upscaling, manufacturing and test of a centrifugal particle receiver. Energy Sustainability, 06 2016.

[61] Lars Amsbeck, Reiner Buck, Miriam Ebert, Birgit Gobereit, Johannes Hertel, Andrea Jensch, Jens Rheinländer, David Trebing, and Ralf Uhlig. First tests of a centrifugal particle receiver with a 1m2 aperture. *AIP Conference Proceedings*, 2033(1):040004, 2018.

[62] Miriam Ebert, Lars Amsbeck, Reiner Buck, Jens Rheinländer, Bärbel Schlögl-Knothe, Stefan Schmitz, Marcel Sibum, Hannes Stadler, and Ralf Uhlig. First on-sun tests of a centrifugal particle receiver system. volume ASME 2018 12th International Conference on Energy Sustainability of *Energy Sustainability*, page V001T11A002, 06 2018.

[63] Miriam Ebert, Lars Amsbeck, Jens Rheinländer, Bärbel Schlögl-Knothe, Stefan Schmitz, Marcel Sibum, Ralf Uhlig, and Reiner Buck. Operational experience of a centrifugal particle receiver prototype. *AIP Conference Proceedings*, 2126(1):030018, 2019.

[64] Wilhelm Burger and Mark James Burge. *Farbbilder*, pages 233–298. Springer-Verlag, Berlin, Heidelberg, 2006.

[65] Alfred Nischwitz, Max Fischer, Peter Haberäcker, and Gudrun Socher. *Digitale Bilddaten*, pages 25–90. Springer Fachmedien, Wiesbaden, 2020.

[66] O. Scheer. *Stereoanalyse und Bildsynthese*, chapter Das Kameramodell, pages 37–63. Springer-Verlag, Berlin, Heidelberg, 2005.

[67] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, Nov 2000.

[68] J. Heikkila and O. Silven. A four-step camera calibration procedure with implicit image correction. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1106–1112, June 1997.

[69] Zhengyou Zhang. *Perspective Camera*, pages 590–592. Springer US, Boston, MA, 2014.

[70] The MathWorks. *Camera Calibration Parameters*. [Online]. [Accessed: 10.06.20]. Available from: `https://de.mathworks.com/help/vision/ug/camera-calibration.html#buvr2qb-2`, 2020.

[71] Richard Szeliski. *Computer Vision : Algorithms and Applications*. Texts in Computer Science-SpringerLink. Springer London, 2011.

[72] opencv dev team. *Camera Calibration and 3D Reconstruction in OpenCV 2.4.13.7 documentation*. [Online]. [Accessed: 20.05.20]. Available from: `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`, 2019.

[73] R. Tsai. A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses. *IEEE Journal on Robotics and Automation*, 3(4):323–344, August 1987.

[74] Thomas Luhman, Stuart Robson, Stephen Kyle, and Ian Harley. *Close Range Photogrammetry: Principles, Techniques and Applications*. Whittles Publishing, Dunbeath, Scotland, UK, 2011. ISBN 978-1870325-50-9.

[75] J. Weng, P. Cohen, and M. Herniou. Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(10):965–980, 1992.

[76] Duane C. Brown. Close-range camera calibration. *PHOTOGRAMMETRIC ENGINEERING*, 37(8):855–866, 1971.

[77] Arcangelo Distante and Cosimo Distante. *Optical System*, pages 177–209. Springer International Publishing, Cham, 2020.

[78] Wilfried Linder. *Lens Distortion and Calibration*, pages 109–114. Springer-Verlag, Berlin, Heidelberg, 2016.

[79] C. Balletti, F. Guerra, V. Tsioukas, and P. Vernier. Calibration of action cameras for photogrammetric purposes. *Sensors*, no. 9:17471–17490, 2014.

[80] William Christian Krumbein and Laurence L. Sloss. *Stratigraphy and sedimentation*. Freeman, San Francisco, 1958.

[81] Saint Gobain. Safety data sheet bauxite, 2019.

[82] Birgit Gobereit, Lars Amsbeck, Christoph Happich, and Martin Schmücker. Assessment and improvement of optical properties of particles for solid particle receiver. *Solar Energy*, 199:844 – 851, 2020.

[83] Saint Gobain Proppants. *Data sheet Sintered Bauxite, CER-0009-XM-0504-SGCS*, 2004.

[84] Mühlmeier Mahltechnik. *Produktinformation SAZ-ER120S, 070617 D-saz 02.06.2017*. [Online]. [Accessed: 24.06.20]. Available from: `https://www.muehlmeier.de/fileadmin/user_upload/DE/mahltechnik/produktinformation/muehlmeier_de_saz.pdf`, 2017.

[85] opto engineering. *lighting*. [Online]. [Accessed: 18.03.20]. Available from: `https://www.opto-e.com/basics/lighting`, 2020.

[86] Falcon illumination. *FLDR-i70A-W Form No: SOP-7.3.3 / Rev 1.8*. [Online]. [Accessed: 18.03.20]. Available from: `https://falconillumination.de/en/product-database/lightings/fldr-direct-ringlight.html`, 2018.

[87] Falcon illumination. *FLDL-i86x15-W Form No: SOP-7.3.3 / Rev 1.7*. [Online]. [Accessed: 17.03.20]. Available from: `https://falconillumination.de/de/produktdatenbank/beleuchtungen/fldl-fhdl.html`, 2015.

[88] Laser components. *Data sheet FLEXPOINT Dot Laser Modules, 09/17 / V13 / IF / laserlaserm/fp-dot-lasers*, 2017.

[89] Matthias Stieß, editor. *Mechanische Verfahrenstechnik - Partikeltechnologie 1*, chapter Partikelmesstechnik, pages 161–260. Spinger-Lehrbuch. Springer Berlin Heidelberg, 3 edition, 2009.

[90] RETSCH GmbH Haan. *Sieve Analysis Taking a close look at quality An expert guide to particle size analysis*. [Online]. [Accessed: 13.05.20]. Available from: `https://www.retsch.com/dltmp/www/53e4b562-5294-4711-9111-636500000000-b8e580d34c65/expert_guide_sieving_en.pdf`, 2015.

[91] Johannes Grobbel. *Modeling Solar Particle Receivers with the Discrete Element Method*. PhD thesis, RWTH Aachen University, 2019.

[92] Uwe Feuerriegel. *Verfahrenstechnik mit EXCEL: Verfahrenstechnische Berechnungen effektiv durchführen und professionell dokumentieren*, chapter Partikelsysteme, pages 333–356. Springer Fachmedien, Wiesbaden, 2016.

[93] Beakawi Al-Hashemi Hamzah M. and Baghabra Al-Amoudi Omar S. A review on the angle of repose of granular materials. *Powder Technology*, 330:397 – 417, 2018.

[94] A. Mehta and G. C. Barker. The dynamics of sand. *Reports on Progress in Physics*, 57(4):383–416, apr 1994.

[95] Ankit Rohatgi. *WebPlotDigitizer*. [Online]. [Accessed: 13.05.20]. Available from: https://automeris.io/WebPlotDigitizer/, 2019. version 4.2.

[96] planistar Lichttechnik GmbH. *data sheet & manual 14-14-Sled-2-Flat-VA-16w*. [Online]. [Accessed: 27.03.20]. Available from: https://downloads.datasheets-planistar.de/fileadmin/downloads/Datasheets/flaechenleuchten-va/sled-2-va-flat/datasheet_14-14-Sled-2-Flat-VA-16w.pdf, 2018.

[97] iiM AG. *data sheet LED High Power Spots LED Spot5W-W, Version:V052019*. [Online]. [Accessed: 27.03.20]. Available from: https://www.iimag.de/fileadmin/user_upload/lumimax/produkte/artikel/datenblatt/101.0028.00.03.00.pdf, 2019.

[98] Falcon illumination. *FFPR-Si100-RGB Form No: SOP-7.3.3 / Rev 1.0*. [Online]. [Accessed: 27.03.20]. Available from: https://falconillumination.de/en/product-database/lightings/ffpr-ffpq-darkfield.html, 2011.

[99] ImageColorPicker.com. *Imagecolorpicker*. [Online]. [Accessed: 03.03.20]. Available from: https://imagecolorpicker.com/de/, 2020.

[100] Anselm F. Wunderer. *Schwarz-Weiß-Fotografie*. Mitp Edition Profifoto. Verlagsgruppe Hüthig Jehle Rehm, 2014.

[101] Elizabeth Allen and Sophie Triantaphillidou. *The manual of photography*. Focal Press, Oxford, UK, 10th ed edition, 2011.

[102] catree at openCV git hub. *pattern.png*. [Online]. [Accessed: 29.04.20]. Available from: https://github.com/opencv/opencv/blob/master/doc/pattern.png, 2008.

[103] J.Y.Bouguet. *Camera Calibration Toolbox for Matlab*. [Online]. [Accessed: 20.05.20]. Available from: http://www.vision.caltech.edu/bouguetj/calib_doc/, 2015.

[104] Alexander Mordvintsev and Abid Rahman K. *Image Processing in OpenCV*. [Online]. [Accessed: 09.06.20]. Available from: https://docs.opencv.org/master/d0/de3/tutorial_py_intro.html, 2013. version 4.3.0.

[105] Nicholas T. Ouellette, Haitao Xu, and Eberhard Bodenschatz. A quantitative study of three-dimensional lagrangian particle tracking algorithms. *Experiments in Fluids*, 40(2):301–313, 2006.

List of Figures

List of Tables

Appendix

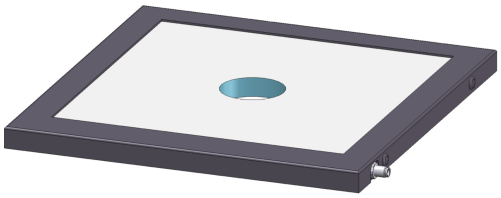
A.1 Pictures of tested illuminations

(a) F1 (ring light)

(b) F2 (bar light)

(c) F3 (ring light)

(d) P1 (area light)



(e) L1 (spot light)

Figure A.1: Pictures of the five different illuminations, which are tested in the selection of illumination during the optical design optimization procedure, cf. section 3.3.1

A.2 Python code: Pixelcolor

```
import cv2
import numpy as np
from numpy import *
import matplotlib
import matplotlib.pyplot as plt
from PIL import Image
from matplotlib.colors import LogNorm
from matplotlib.colors import BoundaryNorm
from matplotlib.ticker import MaxNLocator
import pathlib

# Analysed Frame
framename='F3-200226-15'
distance='150' #in mm
number='P-3'
framenumber='_250'
fileextension='.png'
path=str(pathlib.Path(__file__).parent.absolute()).replace("\\","/")

# Values for the cropbox depending on the distance between the camera and the
particles
###cropbox 60mm:50x46mm
##a1=376
##a2=1544
##b1=2
##b2=1078
###cropbox 80mm: 50x50mm
##a1=420
##a2=1500
##b1=15
##b2=1065
###cropbox 100mm: 50x50mm
##a1=525
##a2=1395
##b1=115
##b2=965
###cropbox 135mm: 50x50mm
##a1=605
##a2=1315
##b1=200
##b2=880
#cropbox 150mm: 50x50mm
a1=675
a2=1245
b1=260
b2=820
###cropbox 200mm: 50x50mm
##a1=730
##a2=1190
##b1=320
##b2=760


##
-------------------------------------------------------------------------------------
-------------------------------
## Generation of a colored image of the pixel color matrix (binary) to evaluate the
homogenity of the illumination
##


#L = R * 299/1000 + G * 587/1000 + B * 114/1000
image1 =
Image.open(path+"/"+framename+'/'+framename+number+'/Frames/ForAnalysis/'+framename+nu
mber+framenumber+fileextension)
cropbox=(a1,b1,a2,b2)
cut=image1.crop(cropbox)
cut.save(path+"/"+framename+'/'+framename+number+'/Frames/ForAnalysis/'+framename+numb
er+framenumber+'_cut1'+fileextension)# just to check the evaluated picture part
image =
Image.open(path+"/"+framename+'/'+framename+number+'/Frames/ForAnalysis/'+framename+nu
mber+framenumber+'_cut1'+fileextension).convert('L')
```

```
64
65
66  #get pixel number of cut image
67  widthcut=a2-a1
68  heightcut=b2-b1
69  print('Width_cut:'+str(widthcut))
70  print('Height_cut:'+str(heightcut))
71
72
73  # save the value of each pixel in the matrix a and generation of a txt-file with the
    values of each pixel to evaluate the homogenity of the illumination
74  a = zeros((widthcut,heightcut), dtype=int)
75  colorrow=list()
76  colors=list()
77  f=open(path+"/"+framename+'/'+framename+number+'/Pixelcolor/'+framename+number+framenu
    mber+'cut1_pixelcolor_gray.txt',"w+")
78
79  for y in range(heightcut):
80      for x in range(widthcut):
81          color=image.getpixel((x,y))
82          a[x,y]=color
83          colorrow.append(color)
84          f.write(str(color)+ ' ')
85      f.write("\n")
86      colors.append(colorrow)
87  f.close()
88
89  # plot and save the matrix a with colorbar
90  plt.pcolormesh(a,cmap=plt.cm.Greys)
91  plt.title('Pixel intensity image')
92  plt.xlabel('Pixel in x-direction')
93  plt.ylabel('Pixel in y direction ')
94  plt.grid(True)
95  plt.colorbar()
96  plt.savefig(path+"/"+framename+'/'+framename+number+'/Pixelcolor/'+framename+number+fr
    amenumber+'cut1_pixelcolor_gray.png')
97  plt.close()
98
99
100 ##
    ------------------------------------------------------------------------------------
    -------------------------------
101 ## Generation of a histogramm of the pixel values to evaluate the illumination
    (over/underexposure)
102 ##
103
104 maxi=np.amax(image)
105 print('Max:'+str(maxi))
106 mini=np.amin(image)
107 print('Min:'+str(mini))
108 delta=maxi-mini
109 print('Delta pixelcolor:'+str(delta))
110 medi=np.mean(image)
111 print('Mean:'+str(medi))
112 medi3=np.median(image)
113 print('Median:'+str(medi3))
114 stab=np.std(image)
115 print('Standard deviation:'+str(stab))
116 homog=100*(1-(stab/medi))
117 print('Homogenity in % '+str(homog))
118
119
120 # Save keydata of the grey level picture as txt file
121 f=open(path+"/"+framename+'/'+framename+number+'/Pixelcolor/'+framename+number+framenu
    mber+'cut1_keydata.txt',"w+")
122 f.write('Name:'+str(framename+number)+ "\n"+'Distance in mm:'+str(distance)+
    "\n"+'Max:'+str(maxi)+ "\n"+'Min:'+str(mini)+ "\n"+'Delta pixelcolor:'+str(delta)+
    "\n"+'Mean:'+str(medi) + "\n"+'Median:'+str(medi3)+ "\n"+'Homogenity in %:'+
    str(homog)+ "\n"+ 'width: '+ str(widthcut)+ "\n"+'height:'+ str(heightcut))
123 f.close()
124
125 # Save grey level pixel colors to csv file
```

```
126   np.savetxt(path+"/"+framename+'/'+framename+number+'/Pixelcolor/'+framename+number+fra
      menumber+"cut1_pixelcolor_gray.csv", image, delimiter=' ')
127
128   # Histogram
129   counts2,vals2 = np.histogram(image, bins=range(2 ** 8))
130   plt.plot(range(0, (2 ** 8) - 1), counts2)
131   plt.title('Greyscale image histogram')
132   plt.xlabel('Pixel value')
133   plt.ylabel('Count')
134   plt.savefig(path+"/"+framename+'/'+framename+number+'/Pixelcolor/'+framename+number+fr
      amenumber+'cut1_histogram_grey.png')
135   plt.close()
136
137   gray=cv2.cvtColor(cv2.imread(path+"/"+framename+'/'+framename+number+'/Frames/ForAnaly
      sis/'+framename+number+framenumber+'_cut1'+fileextension), cv2.COLOR_BGR2GRAY)
138   hist2=cv2.calcHist([gray],[0],None,[256],[0,256])
139   f=open(path+"/"+framename+'/'+framename+number+'/Pixelcolor/'+framename+number+framenu
      mber+'cut1_histogram_gray.txt',"w+")
140   f.write(str(hist2).replace("[","").replace("]",""))
141   f.close()
142
143   cv2.waitKey(0)
144   cv2.destroyAllWindows()
145
```

A.3 ROI of the camera

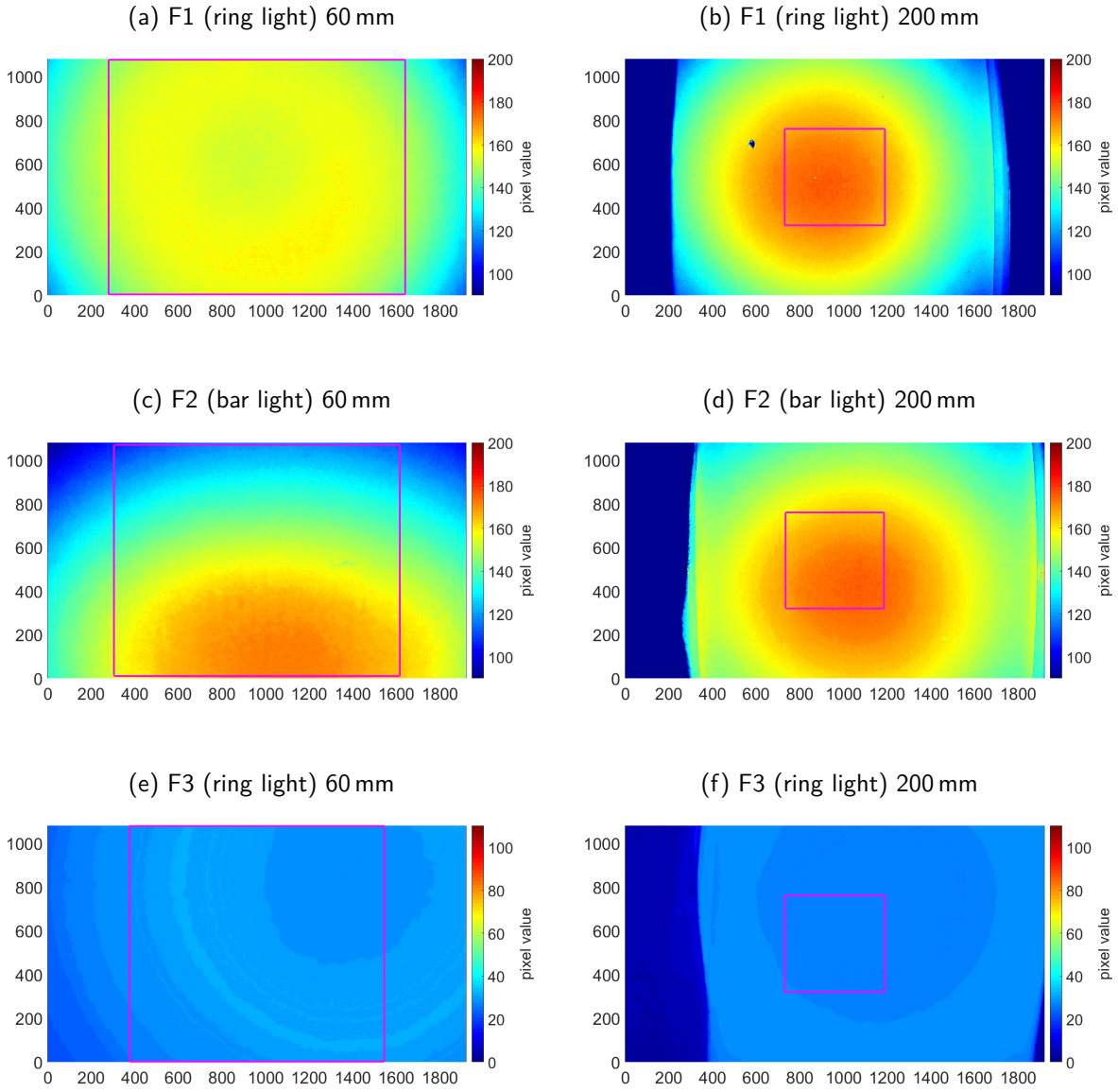


Figure A.2: Pixel value illustration of a white sheet taken with different illuminations from 60 mm (left) and 200 mm (right) distance between the camera, respectively illumination, and the white sheet; the magenta square illustrates the cut out ROI

A.4 Python code: Camera calibration

```
import numpy as np
import cv2
import glob
import pathlib

path=str(pathlib.Path(__file__).parent.absolute()).replace("\\","/")
date='20200509'
name='A2-F1-CB-1'

#-------------------------------------------------------------------------------------
# Find chessboard
#-------------------------------------------------------------------------------------


# termination criteria
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

# prepare object points based on the actual dimensions of the calibration board,
like (0,0,0), (50,0,0), (100,0,0) ....,(400,250,0)
boarddim=50     # size of a square in mm
numberofrows=9
numberoflines=6
objp = np.zeros((numberoflines*numberofrows,3), np.float32)
objp[:,:2] =
np.mgrid[0:(numberofrows*boarddim):boarddim,0:(numberoflines*boarddim):boarddim].T.res
hape(-1,2)

# Arrays to store object points and image points from all the images.
objpoints = [] # 3d point in real world space
imgpoints = [] # 2d points in image plane.

# Lists to store the calibration parameters of all pictures
fxs=list()
fys=list()
cxs=list()
cys=list()
k1s=list()
k2s=list()
k3s=list()
p1s=list()
p2s=list()
errors=list()

# Pictures used for the calibration
images = glob.glob(path+'/'+date+'/'+name+'/Frames/ForCalibration/'+'*.png')
i=1
j=0
for fname in images:
    # Read frame
    img = cv2.imread(fname)

    # Convert to grayscale
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

    # Find the chessboard corners
    ret, corners = cv2.findChessboardCorners(gray, (numberofrows,numberoflines),None)

    if ret==False:
        print('Frame '+str(fname)+ ': not usable')

    # If found, add object points, image points (after refining them)
    if ret == True:
        #Add the chessboard corner points
        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria) #
        increase the accuracy
        imgpoints.append(corners2)

        # Draw and display the corners
        #cv2.namedWindow("img", cv2.WINDOW_NORMAL)
        #img = cv2.drawChessboardCorners(img, (numberofrows,numberoflines),
```

```
            corners2,ret)
69          #cv2.imshow('img',img)
70          #cv2.waitKey()
71
72      #-------------------------------------------------------------------------------
73      # Find calibration parameters (fx, fy, cx, cy, k1, k2, p1, p2,k3) and add them
        to arrays
74      #-------------------------------------------------------------------------------
75
76          ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
            gray.shape[::-1],None,None)
77
78          fx=mtx[0][0]
79          fxs.append(fx)
80          if fx>2200:
81              print('delete: '+str(fname)+' in i: '+str(i))
82          fy=mtx[1][1]
83          fys.append(fy)
84          cx=mtx[0][2]
85          cxs.append(cx)
86          cy=mtx[1][2]
87          cys.append(cy)
88
89          k1=dist[0][0]
90          k1s.append(k1)
91          k2=dist[0][1]
92          k2s.append(k2)
93          p1=dist[0][2]
94          p1s.append(p1)
95          p2=dist[0][3]
96          p2s.append(p2)
97          k3=dist[0][4]
98          k3s.append(k3)
99
100
101         #-------------------------------------------------------------------------------
102         # Export calibration parameters for each picture
103         #-------------------------------------------------------------------------------
104
105
            f=open(path+'/'+date+'/'+name+'/Results/All/Keydata/'+name+'_keydata_4_'+str(i
            )+'.txt',"w+")
106         f.write('ret:'+str(ret)+ "\n"+'mtx:'+str(mtx)+ "\n"+'dist:'+str(dist)+
            "\n"+'rvecs:'+str(rvecs)+ "\n"+'tvecs:'+str(tvecs))
107         f.close()
108
            g=open(path+'/'+date+'/'+name+'/Results/All/Cameramatrix/'+name+'_cameramatrix
            _4_'+str(i)+'.txt',"w+")
109         g.write('fx: \n'+str(fx)+"\n"+'fy: \n'+str(fy)+"\n"+'cx:
            \n'+str(cx)+"\n"+'cy: \n'+str(cy)+"\n")
110         g.close()
111
112
            h=open(path+'/'+date+'/'+name+'/Results/All/Distortioncoefficients/'+name+'_di
            stortioncoeff_4_'+str(i)+'.txt',"w+")
113         h.write('k1: \n'+str(k1)+"\n"+'k2: \n'+str(k2)+"\n"+'p1:
            \n'+str(p1)+"\n"+'p2: \n'+str(p2)+"\n"+'k3: \n'+str(k3)+"\n")
114         h.close()
115         j+=1
116
117
118     i+=1
119
120
121 #-------------------------------------------------------------------------------
122 # re-projection error
123 #-------------------------------------------------------------------------------
124 mean_error = 0
125 for k in range(len(objpoints)):
126     imgpoints2, _ = cv2.projectPoints(objpoints[k], rvecs[k], tvecs[k], mtx, dist)
127     error = cv2.norm(imgpoints[k],imgpoints2, cv2.NORM_L2)/len(imgpoints2)
128     mean_error += error
```

```
129
130 #-------------------------------------------------------------------------
131 # Export re-projection error for each picture
132 #-------------------------------------------------------------------------
133 k=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_reprojectionerror_4.txt',"w+")
134 k.write('Error: \n'+str(mean_error/len(objpoints))+"\n")
135 k.close()
136 totalerror=mean_error/len(objpoints)
137 print ("total error: ", totalerror)
138 errors.append(totalerror)
139
140 #-------------------------------------------------------------------------------------
141 # Export calibration parameters for all calibration pictures in one txt-file
142 #-------------------------------------------------------------------------------------
143
144 g=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_cameramatrix_4_all.txt',"w+")
145 g.write('date: '+str(date)+"\n"+'name: '+str(name)+"\n")
146 g.write('fxs: \n'+str(fxs).replace("[","").replace("]","")+"\n"+'fys:
    \n'+str(fys).replace("[","").replace("]","")+"\n"+'cxs:
    \n'+str(cxs).replace("[","").replace("]","")+"\n"+'cys:
    \n'+str(cys).replace("[","").replace("]","")+"\n")
147 g.close()
148
149 h=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_distortioncoeff_4_all.txt',"w+"
    )
150 h.write('date: '+str(date)+"\n"+'name: '+str(name)+"\n")
151 h.write('k1s: \n'+str(k1s).replace("[","").replace("]","")+"\n"+'k2s:
    \n'+str(k2s).replace("[","").replace("]","")+"\n"+'p1s:
    \n'+str(p1s).replace("[","").replace("]","")+"\n"+'p2s:
    \n'+str(p2s).replace("[","").replace("]","")+"\n"+'k3s:
    \n'+str(k3s).replace("[","").replace("]","")+"\n")
152 h.close()
153
154
155 #-------------------------------------------------------------------------------------
156 # Calcluate mean calibration parameters and their standard deviation
157 #-------------------------------------------------------------------------------------
158
159 # Camera matrix
160 meanfx=np.mean(fxs)
161 stabfx=np.std(fxs)
162
163 meanfy=np.mean(fys)
164 stabfy=np.std(fys)
165
166 meancx=np.mean(cxs)
167 stabcx=np.std(cxs)
168
169 meancy=np.mean(cys)
170 stabcy=np.std(cys)
171
172 # Distortion coefficients
173 meank1=np.mean(k1s)
174 stabk1=np.std(k1s)
175
176 meank2=np.mean(k2s)
177 stabk2=np.std(k2s)
178
179 meanp1=np.mean(p1s)
180 stabp1=np.std(p1s)
181
182 meanp2=np.mean(p2s)
183 stabp2=np.std(p2s)
184
185 meank3=np.mean(k3s)
186 stabk3=np.std(k3s)
187
188 #-------------------------------------------------------------------------------------
189 # Export mean calibration parameters for evaluation of the calibration
190 #-------------------------------------------------------------------------------------
191
192 g=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_cameramatrix_4_mean.txt',"w+")
```

```
193   g.write('date: '+str(date)+"\n"+'name: '+str(name)+"\n")
194   g.write('Mean fx: \n'+str(meanfx).replace("[","").replace("]","")+"\n"+'Stab mean
      fx: \n'+str(stabfx).replace("[","").replace("]","")+"\n"+'Mean fy:
      \n'+str(meanfy).replace("[","").replace("]","")+"\n"+'Stab mean fy:
      \n'+str(stabfy).replace("[","").replace("]","")+"\n"+"\n"+'Mean cx:
      \n'+str(meancx).replace("[","").replace("]","")+"\n"+'Stab mean cx:
      \n'+str(stabcx).replace("[","").replace("]","")+"\n"+'Mean cy:
      \n'+str(meancy).replace("[","").replace("]","")+"\n"+'Stab mean cy:
      \n'+str(stabcy).replace("[","").replace("]","")+"\n")
195   g.close()
196
197   h=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_distortioncoeff_4_mean.txt',"w+
      ")
198   h.write('date: '+str(date)+"\n"+'name: '+str(name)+"\n")
199   h.write('Mean k1s: \n'+str(meank1).replace("[","").replace("]","")+"\n"+'Stab mean
      k1: \n'+str(stabk1).replace("[","").replace("]","")+"\n"+'Mean k2:
      \n'+str(meank2).replace("[","").replace("]","")+"\n"+'Stab mean k2:
      \n'+str(stabk2).replace("[","").replace("]","")+"\n"+'Mean p1:
      \n'+str(meanp1).replace("[","").replace("]","")+"\n"+'Stab mean p1:
      \n'+str(stabp1).replace("[","").replace("]","")+"\n"+'Mean p2:
      \n'+str(meanp2).replace("[","").replace("]","")+"\n"+'Stab mean p2:
      \n'+str(stabp2).replace("[","").replace("]","")+"\n"+'Mean k3:
      \n'+str(meank3).replace("[","").replace("]","")+"\n"+'Stab mean k3:
      \n'+str(stabk3).replace("[","").replace("]","")+"\n")
200   h.close()
201
202   i=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_numberofframes_4_mean.txt',"w+"
      )
203   i.write('date: '+str(date)+"\n"+'name: '+str(name)+"\n")
204   i.write('Number of frames used for calculation of the cameraparameters: \n'+str(j))
205   i.close()
206
207
208   #-----------------------------------------------------------------------------------
209   # Export mean calibration parameters without comments to use for correction of the
      data
210   #-----------------------------------------------------------------------------------
211
212   cameramatrix=[meanfx, meanfy, meancx, meancy]
213   np.savetxt(path+'/'+date+'/'+name+'/Results/Total/'+name+'_cameramatrix_4_meanT.txt',c
      ameramatrix)
214
215   distortioncoeffs=[meank1, meank2, meanp1, meanp2, meank3]
216   np.savetxt(path+'/'+date+'/'+name+'/Results/Total/'+name+'_distortioncoeff_4_meanT.txt
      ',distortioncoeffs)
217
218   np.savetxt(path+'/'+date+'/'+name+'/Results/Total/'+name+'_error_4_meanT.txt',errors)
219
220
221   cv2.destroyAllWindows()
222
```

A.5 Python code: Generation of corrected data

```
import numpy as np
import cv2
import glob
import pathlib

path=str(pathlib.Path(__file__).parent.absolute()).replace("\\","/")
date='20200509'
name='A2-F1-CB-1'
nameframes='A2-F1-135-P'

#-----------------------------------------------------------------------------------------
# Read and Save Calibration Parameters
#-----------------------------------------------------------------------------------------

# Camera matrix
fileCM=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_cameramatrix.txt',"r")
CM=fileCM.readlines()

# Distortion coefficients
fileDC=open(path+'/'+date+'/'+name+'/Results/Total/'+name+'_distortioncoeff.txt',"r")
DC=fileDC.readlines()

mtx=np.zeros((3,3))
dist=np.zeros((1,5))
mtx[2][2]=1

mtx[0][0]=float(CM[0])  #fx
mtx[1][1]=float(CM[1])  #fy
mtx[0][2]=float(CM[2])  #cx
mtx[1][2]=float(CM[3])  #cy
dist[0][0]=float(DC[0]) #k1
dist[0][1]=float(DC[1]) #k2
dist[0][2]=float(DC[2]) #p1
dist[0][3]=float(DC[3]) #p2
dist[0][4]=float(DC[4]) #k3

#-----------------------------------------------------------------------------------------
# Loop for Undistortion
#-----------------------------------------------------------------------------------------
images = glob.glob(path+'/'+nameframes+'/Frames/'+'*.png')
for fname in images:
    img = cv2.imread(fname)
    h,  w = img.shape[:2]
    newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),0.5,(w,h)) #value
    between 0 (whole picture range) and 1 (cut out of the picture)

    # undistort
    dst = cv2.undistort(img, mtx, dist, None, newcameramtx)

    # crop the image
    x,y,w,h = roi
    dst = dst[y:y+h, x:x+w]

cv2.destroyAllWindows()

```

A.6 Flow chart: Tracer tracking

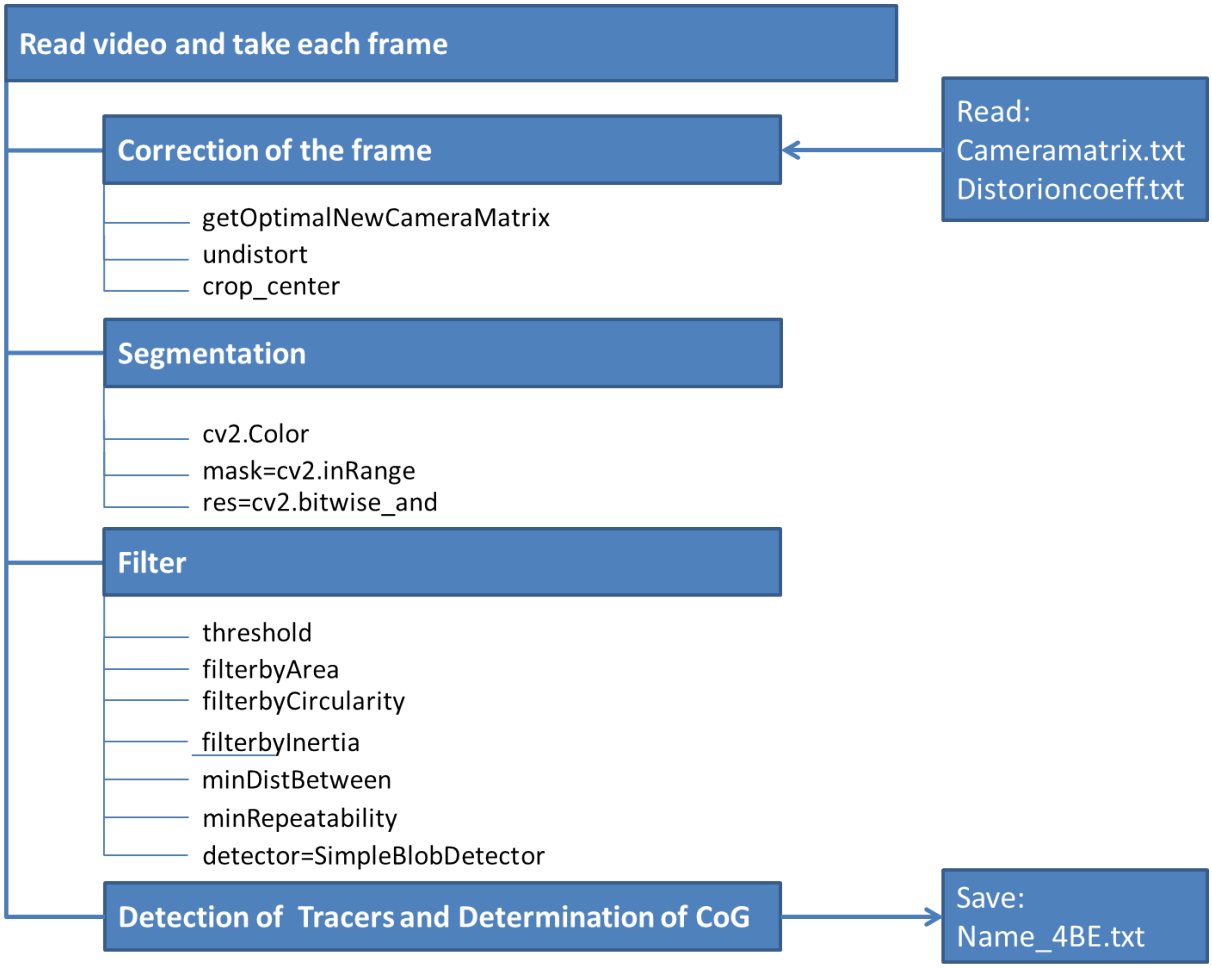


Figure A.3: Schematic diagram of the procedure for detection of the tracers

A.7 Python code: Tracer tracking

```
import cv2
import numpy as np
import sys
import pathlib
import glob
import matplotlib.pyplot as plt
from PIL import Image
import os
import math


# cropbox
def crop_center(img,a1,a2,b1,b2):
    return img[a1:a2,b1:b2]


#-----------------------------------------------------------------------------------------
# Name of the experiment
#-----------------------------------------------------------------------------------------
date='200617'
framename=date
number='_2'
fileextension='.mp4'
path=str(pathlib.Path(__file__).parent.absolute()).replace("\\","/")

#-----------------------------------------------------------------------------------------
# Calibration
#-----------------------------------------------------------------------------------------

pathCali='G:/Liebl/4-analysis/tracer tracking/calibration/'
dateCali='20200509'
nameCali='A2-F1-CB-1'

#-----------------------------------------------------------------------------------------
# Set data
#-----------------------------------------------------------------------------------------

v_x0=1
v_y0=1
delta_t=1
radius=1


#cropbox 135mm: 50mm x 50mm
# a width, b lenght
a1=710
a2=1300
b1=383
b2=974
cropbox=(a1,b1,a2,b2)

#-----------------------------------------------------------------------------------------
# Read and Save Calibration Parameters
#-----------------------------------------------------------------------------------------

# Camera matrix
fileCM=open(pathCali+'/'+dateCali+'/'+nameCali+'/Results/Total/'+nameCali+'_cameramatr
ix_4_meanT.txt',"r")
CM=fileCM.readlines()

# Distortion coefficients
fileDC=open(pathCali+'/'+dateCali+'/'+nameCali+'/Results/Total/'+nameCali+'_distortion
coeff_4_meanT.txt',"r")
DC=fileDC.readlines()

mtx=np.zeros((3,3))
dist=np.zeros((1,5))
mtx[2][2]=1

mtx[0][0]=float(CM[0])  #fx
mtx[1][1]=float(CM[1])  #fy
mtx[0][2]=float(CM[2])  #cx
```

```
mtx[1][2]=float(CM[3])  #cy
dist[0][0]=float(DC[0]) #k1
dist[0][1]=float(DC[1]) #k2
dist[0][2]=float(DC[2]) #p1
dist[0][3]=float(DC[3]) #p2
dist[0][4]=float(DC[4]) #k3

#-------------------------------------------------------------------------------------------
# Save to folder
#-------------------------------------------------------------------------------------------

f=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_dots.txt',"w+")
f.write('Framenumber NumberofDetectedDots MaxDiameter MinDiameter MeanDiameter
MedianDiameter StandardDeviation'+"\n" )

m=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_keydata.txt',"w+")
m.write('Name MeanNumberofDetectedDots StandardDeviation'+"\n" )
m.write(date+number)

j=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_dots_x.txt',"w+")
k=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_dots_y.txt',"w+")


#-------------------------------------------------------------------------------------------
# Get frames from video, undistort them and detect the tracers
#-------------------------------------------------------------------------------------------

# Take video
cap=cv2.VideoCapture(path+'/'+date+"/"+framename+number+"/"+framename+number+fileexten
sion)

frames=list()
correctedframes=list()
framenumber=list()
allblobs=list()
zahl=0
br=0

#-------------------------------------------------------------------------------------------
# Loop for reading the video, segmentation, filter and detection of the tracer
#-------------------------------------------------------------------------------------------
# Read until video is completed
while(cap.isOpened()):

  # Capture frame-by-frame
  ret, frame = cap.read()
  if ret == True:

        zahl+=1
##        print('Framenumber: '+str(zahl))
        f.write(str(zahl))
        f.write(' ')
        framenumber.append(zahl)


        imgO = frame
        h,  w = imgO.shape[:2]
        newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),0,(w,h))
        #value between 0 (whole picture range) and 1 (cut out of the picture)

        # undistort the original image
        dst = cv2.undistort(imgO, mtx, dist, None, newcameramtx)

        # crop the orinigal image
        x,y,w,h = roi
        dst = dst[y:y+h, x:x+w]
        correctedframes.append(dst)
        frames.append(frame)
        framecut=crop_center(dst,b1,b2,a1,a2)
        img=framecut
        gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)

```

```
140           #-------------------------------------------------------------------------------
141           # Segmentation
142
              #-------------------------------------------------------------------------------
143           # Convert BGR to HSV
144           hsv=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
145
146           # Get value sensitivity
147           sens=50
148
149           # Define range of white color in HSV with adjustable sensitivity
150           lower_white = np.array([ 0 , 0 , 100-sens ])
151           upper_white = np.array([ 0 , 0 , 100     ])
152
153           # Threshold the HSV image to get only white colors, (in the range of the
              mask: 1 (white))
154           mask = cv2.inRange(hsv, lower_white, upper_white)
155
156           # Bitwise-AND mask and original image, in the frame ther is the frame and
              mask ist true; Bitwise: where are 1s in the frame: show the color)
157           res = cv2.bitwise_and(img,img, mask= mask)
158
159           img=res
160
161   ##        # MedianBlur
162   ##        blursize=1 #size = blursizexblursize
163   ##        resblur = cv2.medianBlur(res,blursize)
164   ##        img=resblur
165
166
167           #-------------------------------------------------------------------------------
168           # Create Filter and Detection of tracer
169
              #-------------------------------------------------------------------------------
170
171           im = img
172
173           # Setup SimpleBlobDetector parameters
174           params = cv2.SimpleBlobDetector_Params()
175
176           # Change thresholds
177           params.thresholdStep = 5
178           params.minThreshold = 200
179           params.maxThreshold = 255
180
181           # Filter by Color (one specific color needed: not recommended)
182           params.filterByColor = False
183           params.blobColor = 255
184
185           # Filter by Area. see: Excel: picture-pillow
186           params.filterByArea = True
187           params.minArea = 83  # 135mm
188           params.maxArea = 180  # 135mm
189
190           # Filter by Circularity
191           params.filterByCircularity = True
192           params.minCircularity = 0.7 # circularity of a square
193
194           # Filter by Convexity (Area of the Blob / Area of it's convex hull)
195           params.filterByConvexity = False
196           params.minConvexity = 0.87
197
198           # Filter by Inertia (ratio of widest to thinnest point, measures how
              elongated a shape is, circle:1, ellipse:0<x<1, line:0)
199           params.filterByInertia = True
200           params.minInertiaRatio = 0.01
201   ##        params.maxInertiaRatio = 1 #
202
203           params.minDistBetweenBlobs = 1.0  # blobs closer will be merged
204           params.minRepeatability = 2
```

```
205
206
207             # Create a detector with the parameters
208             ver = (cv2.__version__).split('.')
209             if int(ver[0]) < 3 :
210                     detector = cv2.SimpleBlobDetector(params)
211             else :
212                     detector = cv2.SimpleBlobDetector_create(params)
213
214
215             # Detect blobs
216             keypoints = detector.detect(im)
217             blobnumber=len(keypoints)
218             print('Number of detected circles: '+ str(blobnumber))
219             f.write(str(blobnumber))
220             f.write(' ')
221             allblobs.append(blobnumber)
222
223             # Draw detected blobs as red circles
224             # cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures the size of the circle
                corresponds to the size of blob
225             im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]),
                (0,255,0), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
226
227             center=list()
228             y1=list()
229             x1=list()
230             diameter=list()
231
232
                d=open(path+"/"+date+"/"+framename+number+"/"+"/doPTV/"+framename+number+'_det
                ectedDOTS_'+str(zahl)+'.txt',"w+")
233
                e=open(path+"/"+date+"/"+framename+number+"/"+"/doPTV/"+framename+number+'_4BE
                _'+str(zahl)+'.txt',"w+")
234             d.write('Dot X: ')
235             d.write('Dot Y: '+ '\n')
236             for keypoint in keypoints:
237                 x = keypoint.pt[0]
238                 y = keypoint.pt[1]
239                 s = keypoint.size/2
240                 KeyPoints=[x,y,s]
241                 center.append(KeyPoints)
242                 point=[x,y]
243                 y1.append(y)
244                 x1.append(x)
245                 diameter.append(s*2)
246                 d.write(str(x)+ ' ')
247                 d.write(str(y)+ '\n')
248                 e.write(str(x)+ ','+str(y)+ '\n')
249                 j.write(str(x)+ ',')
250                 k.write(str(y)+ ',')
251             j.write('\n')
252             k.write('\n')
253
254             centerint = np.uint16(np.around(center)) #circle only can handle int
255
256             # Check the detected circles
257             for i in centerint:
258                 # draw the outer circle
259                 cv2.circle(im,(i[0],i[1]),i[2],(0,255,0),2)
260                 # draw the center of the circle4.
261                 cv2.circle(im,(i[0],i[1]),2,(0,0,255),1)
262             print('diameter: '+str(diameter))
263
264             if len(diameter)>0:
265                 # Calculation of characteristics of detected dots
266                 maxi=np.amax(diameter)
267                 mini=np.amin(diameter)
268                 delta=maxi-mini
269                 medi=np.mean(diameter)
270                 medi3=np.median(diameter)
```

```
271                 stab=np.std(diameter)
272
273                 f.write(str(maxi)+ ' ' + str(mini)+ " "+str(medi)+ " "+str(medi3)+ "
                    "+str(stab)+"\n")
274             d.close()
275             e.close()
276
277             # Show blobs
278     ##        cv2.namedWindow("Keypoints", cv2.WINDOW_NORMAL) #in case that the window
        is to big
279     ##        cv2.namedWindow("detected circles", cv2.WINDOW_NORMAL)
280     ##        cv2.imshow("Keypoints", im_with_keypoints)
281
                cv2.imwrite(path+"/"+date+"/"+framename+number+'/Filter_after.png',im_with_key
                points)
282     ##        cv2.imshow('detected circles',im)
283             cv2.imwrite(path+"/"+date+"/"+framename+number+'/DetectedTracer.png',im)
284
285             # Press Q on keyboard to  exit
286             if cv2.waitKey(27) & 0xFF == ord('q'):
287                     break
288
289       # Break the loop
290       else:
291         print('frame breake: '+str(zahl))
292         break
293
294     cap.release()
295
296     # Number of frames
297     n=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_numberofframes.txt',"
        w+")
298     n.write(str(zahl))
299     n.close()
300
301     meandots=np.mean(allblobs)
302     stabdots=np.std(allblobs)
303     m.write(' '+str(meandots)+' '+str(stabdots))
304     m.close()
305     f.close()
306     j.close()
307     k.close()
308
309     #-----------------------------------------------------------------------------------------
310     # Tracking using the 4BestEstimate algorithm
311     #-----------------------------------------------------------------------------------------
312     # Number of frames in video
313     folder=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_numberofframes.t
        xt')
314     i=int(folder.read())
315     print('Number of frames: '+str(i))
316
317     # Total number od detected dots
318     dots_all=list()
319     dots_total=0
320     numberdots=0
321
322
323     # Maximum number of detected dots in one frame
324     frameN=1
325     numberofdots=list()
326     while frameN <= i:
327
            file=path+"/"+date+"/"+framename+number+"/doPTV/"+framename+number+'_4BE_'+str(fra
            meN)+'.txt'
328         dots=np.loadtxt(file, str, delimiter=',').astype(np.float) # dots[x][y]
329         numberofdots.append(len(dots))
330         dots_total+=len(dots)
331         frameN+=1
332     maxnumberofdots=max(numberofdots)
333     print('Max number of dots in one frame: '+str(maxnumberofdots))
334     print('Total number of dots in all frame: '+str(dots_total))
```

```
335
336
337   # Matrices to save all the tracks
338   trackingx=np.zeros((dots_total, i))
339   trackingy=np.zeros((dots_total, i))
340
341   #-------------------------------------------------------------------------------------------
342   # Parameters for tracking (to be set)
343   #-------------------------------------------------------------------------------------------
344
345   # Radius of search volumes
346   # For F1
347   radiusF1=30
348   # For F2
349   radiusF2=30
350   # For F3
351   radiusF3=30
352
353   # Time step
354   deltaT=1
355
356   #-------------------------------------------------------------------------------------------
357   # Loop for all frames
358   #-------------------------------------------------------------------------------------------
359   framenumber=1
360   while framenumber< (i-2):
361   ##while framenumber< 15:
362
363       print()
364       print()
365       print('Step: '+str(framenumber))
366       print()
367
368       #-------------------------------------------------------------------------------------
369       # Step 1: Read detected tracer
370       #-------------------------------------------------------------------------------------
371
372       #-------------------------------------------------------------------------------------
373       # Step 1.1: Read F_n: F_0
374       #-------------------------------------------------------------------------------------
375       j0=framenumber
376       k=0
377       x_0=list()
378       y_0=list()
379       center0=list()
380
381
          file=path+"/"+date+"/"+framename+number+"/doPTV/"+framename+number+'_4BE_'+str(j0)
          +'.txt'
382       dots=np.loadtxt(file, str, delimiter=',').astype(np.float) # dots[x][y]
383       n0=len(dots)
384       dots_all.append(n0)
385       numberdots+=n0
386       while k < n0:
387           try:
388               x_0.append(dots[k][0])
389               y_0.append(dots[k][1])
390               center0.append([dots[k][0],dots[k][1]])
391           except IndexError:
392               x_0.append([dots[0]])
393               y_0.append([dots[1]])
394               center0.append([dots[0],dots[1] ])
395               k=n1
396           k+=1
397
398       if len(center0)==0:
399           x_0.append([0])
400           y_0.append([0])
401           center0.append([0,0])
402
403   ##    print('center0:'+str(center0))
404   ##    print('len(center0):'+str(len(center0)))
```

```
405   ##    print()
406
407       #-------------------------------------------------------------------------------
408       # Step 1.2: Read F_n+1: F_1
409       #-------------------------------------------------------------------------------
410       j1=j0+1
411       k=0
412       x_1=list()
413       y_1=list()
414       center1=list()
415
416
          file=path+"/"+date+"/"+framename+number+"/doPTV/"+framename+number+'_4BE_'+str(j1)
          +'.txt'
417       dots=np.loadtxt(file, str, delimiter=',').astype(np.float) # dots[x][y]
418       n1=len(dots)
419       while k < n1:
420           try:
421               x_1.append(dots[k][0])
422               y_1.append(dots[k][1])
423               center1.append([dots[k][0],dots[k][1]])
424           except IndexError:
425               x_1.append([dots[0]])
426               y_1.append([dots[1]])
427               center1.append([dots[0],dots[1] ])
428               k=n1
429           k+=1
430
431       if len(center1)==0:
432           x_1.append([0])
433           y_1.append([0])
434           center1.append([0,0])
435
436   ##    print('center1:'+str(center1))
437   ##    print('len(center1):'+str(len(center1)))
438   ##    print()
439
440       #-------------------------------------------------------------------------------
441       # Step 1.3: Read F_n+2: F_2
442       #-------------------------------------------------------------------------------
443       j2=j1+1
444       k=0
445       x_2=list()
446       y_2=list()
447       center2=list()
448
449
          file=path+"/"+date+"/"+framename+number+"/doPTV/"+framename+number+'_4BE_'+str(j2)
          +'.txt'
450       dots=np.loadtxt(file, str, delimiter=',').astype(np.float) # dots[x][y]
451       n2=len(dots)
452       while k < n2:
453           try:
454               x_2.append(dots[k][0])
455               y_2.append(dots[k][1])
456               center2.append([dots[k][0],dots[k][1]])
457           except IndexError:
458               print('error')
459               x_2.append([dots[0]])
460               y_2.append([dots[1]])
461               center2.append([dots[0],dots[1] ])
462               k=n2
463           k+=1
464
465       if len(center2)==0:
466           x_2.append([0])
467           y_2.append([0])
468           center2.append([0,0])
469
470   ##    print('center2: '+str(center2))
471   ##    print('len(center2): '+str(len(center2)))
472   ##    print()
```

```
473
474         #-------------------------------------------------------------------------------
475         # Step 1.4: Read F_n+3: F_3
476         #-------------------------------------------------------------------------------
477         j3=j2+1
478         k=0
479         x_3=list()
480         y_3=list()
481         center3=list()
482
483
            file=path+"/"+date+"/"+framename+number+"/doPTV/"+framename+number+'_4BE_'+str(j3)
            +'.txt'
484         dots=np.loadtxt(file, str, delimiter=',').astype(np.float) # dots[x][y]
485         n3=len(dots)
486         while k < n3:
487             try:
488                 x_3.append(dots[k][0])
489                 y_3.append(dots[k][1])
490                 center3.append([dots[k][0],dots[k][1]])
491             except IndexError:
492                 x_3.append([dots[0]])
493                 y_3.append([dots[1]])
494                 center3.append([dots[0],dots[1] ])
495                 k=n3
496             k+=1
497
498         if len(center1)==0:
499             x_3.append(0)
500             y_3.append(0)
501             center3.append([0,0])
502
503     ##    print('center3: '+str(center3))
504     ##    print('len(center3): '+str(len(center3)))
505     ##    print()
506
507         #-------------------------------------------------------------------------------
508         # If framenumber=1: save center 0 to the storage matrices trackingx and trackinygy
509         #-------------------------------------------------------------------------------
510
511         if framenumber==1:
512             row=0
513             while row<len(center0):
514                     trackingx[row][j0-1]=str(center0[row][0])
515                     trackingy[row][j0-1]=str(center0[row][1])
516                     row+=1
517
518         #-------------------------------------------------------------------------------
519         # Step 2: Find couplesF0F1
520         #-------------------------------------------------------------------------------
521         # Radius of search volume for F1
522         SVF1=radiusF1**2
523         couplesF0F1=list()
524
525         a0=0
526         while a0<len(center0):
527             center_x0=center0[a0][0]
528             center_y0=center0[a0][1]
529             a1=0
530             distances01=list()
531             distance01p=list()
532             distances01p=list()
533             count=0
534             while a1<len(center1):
535                 # Circle
                    ((x,y)2_g)https://stackoverflow.com/questions/12262017/python-checking-if-
                    coordinates-are-within-circle
536                 test=(center1[a1][0]-center_x0)**2 + (center1[a1][1]-center_y0)**2
537                 if test <=SVF1 :
538                     d_F0_f=[center0[a0][0],center0[a0][1]]
539                     d_F1_f=[center1[a1][0],center1[a1][1]]
540                     distances01.append(test)
```

```
                    distance01p.append([d_F0_f, d_F1_f])
                    distances01p.append([d_F0_f, d_F1_f,test])
                a1+=1

            if len(distances01)==0:
                coupleF0F1=[[center0[a0][0],center0[a0][1]],[0,0]]
                couplesF0F1.append(coupleF0F1)

            if len(distances01)>0:
                if len(distances01)==1:
                    coupleF0F1=[distance01p[0][0],distance01p[0][1]]
                    couplesF0F1.append(coupleF0F1)
                else:
                    distance01=min(distances01)
                    coupleF0F1=distance01p[distances01.index(distance01)]
                    couplesF0F1.append(coupleF0F1)
            a0+=1

    ##    print('couplesF0F1: '+str(couplesF0F1))
    ##    print('len(couplesF0F1): '+str(len(couplesF0F1)))
    ##    print()

        #-------------------------------------------------------------------------------
        # If framenumber=1: save center 1 to the storage matrices trackingx and trackinygy
        #-------------------------------------------------------------------------------
        if framenumber==1:
            lenfoundcouplesF1F2=len(couplesF0F1)
            row=0
            while row<len(center0):
                row2=0
                count=0
                while row2<len(couplesF0F1):
                    if couplesF0F1[row2][0][0]!=0:
                        if trackingx[row][j1-2]==couplesF0F1[row2][0][0]:
                            trackingx[row][j1-1]=couplesF0F1[row2][1][0]
                            trackingy[row][j1-1]=couplesF0F1[row2][1][1]
                    row2+=1
                row+=1

            # Which center1 are not in mincouples01:  append to center1_not
            center1_not=list()
            testC1=0
            while testC1<len(couplesF0F1):
                testMC01=0
                count=0
                while testMC01<len(center1):
                    if couplesF0F1[testC1][1][0]==center1[testMC01][0]:
                        print('Center1 already appended')
                    else:
                        count+=1
                    if count==len(center1):
    ##                    print('Center1 not append before')
                        center1_not.append([center1[testC1][0],center1[testC1][1]])
                    testMC01+=1
                testC1+=1

    ##        print('Center1_not: '+str(center1_not))
    ##        print()

            rowsxNF1=lenfoundcouplesF1F2
            test=0
            while test<len(center1_not):
                trackingx[rowsxNF1][j1-1]=center1_not[test][0]
                trackingy[rowsxNF1][j1-1]=center1_not[test][1]
                test+=1
                rowsxNF1+=1

            lenfoundcouplesF1F2+=len(center1_not) # counting the rows in the storage
            matrices, which are already used for a track

        #-------------------------------------------------------------------------------
        # Step 3: Calculate the velocity and acceleration of F_n:F_1 and velocity of
```

```
        F_n-1:F0
612     #-------------------------------------------------------------------------------
613     # Initial position
614     x_m1=[0]*n0
615     y_m1=[0]*n0
616
617     # v0
618     if framenumber==1:
619         vx_0=list()
620         vx_0=[(i-j)/deltaT for i,j in zip(x_0,x_m1)]
621         vy_0=list()
622         vy_0=[(i-j)/deltaT for i,j in zip(y_0,y_m1)]
623         vx_1=vx_0
624         vy_1=vy_0
625
626     if len(x_1)>len(x_0):
627             aa=0
628             x_0_new=[0]*len(center1)
629             y_0_new=[0]*len(center1)
630             while aa<len(couplesF0F1):
631                 x_0_new[aa]=couplesF0F1[aa][0][0]
632                 y_0_new[aa]=couplesF0F1[aa][0][1]
633                 aa+=1
634             x_0=x_0_new
635             y_0=y_0_new
636
637     if len(x_1)<len(x_0):
638             aa=0
639             x_1_new=[0]*len(center0)
640             y_1_new=[0]*len(center0)
641             while aa<len(couplesF0F1):
642                 x_1_new[aa]=couplesF0F1[aa][1][0]
643                 y_1_new[aa]=couplesF0F1[aa][1][1]
644                 aa+=1
645             x_1=x_1_new
646             y_1=y_1_new
647
648     if len(center0)==1 and len(center1)==1:
649         x_0=x_0[0]
650         y_0=y_0[0]
651         x_1=x_1[0]
652         y_1=y_1[0]
653         # v1
654         vx_1=list()
655         vx_1=[(x_1[0]-x_0[0])/deltaT ]
656         vy_1=list()
657         vy_1=[(y_1[0]-y_0[0])/deltaT]
658
659     if len(center0)==1 and len(center1)==0:
660         x_0=x_0[0]
661         y_0=y_0[0]
662         x_1=[0]
663         y_1=[0]
664         # v1
665         vx_1=list()
666         vx_1=(x_1[0]-x_0[0])/deltaT
667         vy_1=list()
668         vy_1=(y_1[0]-y_0[0])/deltaT
669
670     else:
671         # v1
672         vx_1=list()
673         vx_1=[(i-j)/deltaT for i,j in zip(x_1,x_0)]
674         vy_1=list()
675         vy_1=[(i-j)/deltaT for i,j in zip(y_1,y_0)]
676
677     #-------------------------------------------------------------------------------
678     # Step 4: Calcluation of extimated point F2_g
679     #-------------------------------------------------------------------------------
680     if len(center0)==0 and len(center1)==0:
681         # Estimated point of F2: F2_g
682         x_2_g=[0]
```

```
683             y_2_g=[0]
684             center2_g=[[0,0]]
685             couplesF0F1=[[[0,0],[0,0]]]
686 
687         else:
688             # Estimated point of F2: F2_g
689             x_2_g=[(i+j*deltaT) for i,j in zip(x_1,vx_1)]
690             y_2_g=[(i+j*deltaT) for i,j in zip(y_1,vy_1)]
691             center2_g=[[(i+j*deltaT),(k+l*deltaT)]  for i,j,k,l in zip(x_1,vx_1,y_1,vy_1)]
692 
693             #-------------------------------------------------------------------------------
694             # If no F0 was found, the estimated F2 is set to F1
695             #-------------------------------------------------------------------------------
696             testi=0
697             while testi<len(center2_g):
698                 if vx_1[testi]==x_1[testi]:
699                     center2_g[testi][0]=x_1[testi]
700                     center2_g[testi][1]=y_1[testi]
701                 testi+=1
702 
703 ##    print('center2_g: '+str(center2_g))
704 ##    print('len(center2_g): '+str(len(center2_g)))
705 ##    print()
706 
707         #-------------------------------------------------------------------------------
708         # Step 5: Find couplesF0F1F2F2c
709         #-------------------------------------------------------------------------------
710 
711         #-------------------------------------------------------------------------------
712         # Step 5.1: Find couplesF1F2F2c
713         #-------------------------------------------------------------------------------
714         # Radius of search volume for F2
715         #radiusF2=100
716         SVF2=radiusF2**2
717         couplesF1F2F2c=list()
718 
719         if len(center2)>len(center2_g): #more dots in picture (center2) as in the one
        before (center1)
720             a2_g=0
721             while a2_g<len(center2_g):
722                 center_x2_g=center2_g[a2_g][0]
723                 center_y2_g=center2_g[a2_g][1]
724                 cand_2=list()
725 
726                 a2=0
727                 count=0
728                 while a2<len(center2):
729                 # Circle
                    ((x,y)2_g)https://stackoverflow.com/questions/12262017/python-checking-if-
                    coordinates-are-within-circle
730                     test2=(center2[a2][0]-center_x2_g)**2 +
                        (center2[a2][1]-center_y2_g)**2
731                     if ((center2[a2][0]-center_x2_g)**2 +
                        (center2[a2][1]-center_y2_g)**2) <=SVF2 :
732                             cand_2.append([center2[a2][0],center2[a2][1]])
733 
                                couple F1F2F2c=[[x_1[a2_g],center2_g[a2_g][0],center2[a2][0]],[
                                y_1[a2_g],center2_g[a2_g][1],center2[a2][1]]]
734                             couplesF1F2F2c.append(coupleF1F2F2c)
735                     a2+=1
736                 if len(cand_2)==0:
737 
                        coupleF1F2F2c=[[x_1[a2_g],center2_g[a2_g][0],0],[y_1[a2_g],center2_g[a
                        2_g][1],0]] # no candidate for estimated F2* in F2 (found dots)
738                     couplesF1F2F2c.append(coupleF1F2F2c)
739                 a2_g+=1
740 
741         if len(center2)==len(center2_g): #same number of dots in picture (center2) as in
        the one before (center1)
742             a2_g=0
743             while a2_g<len(center2_g):
744                 center_x2_g=center2_g[a2_g][0]
```

```
745             center_y2_g=center2_g[a2_g][1]
746             as012=list()
747             a012p=list()
748             as012p=list()
749             cand_2=list()
750
751             a2=0
752             count=0
753             while a2<len(center2):
754             # Circle ((x,y)2_g)https://stackoverflow.com/questions/12262017/python-checking-if-coordinates-are-within-circle
755                 test2=(center2[a2][0]-center_x2_g)**2 + (center2[a2][1]-center_y2_g)**2
756                 if ((center2[a2][0]-center_x2_g)**2 + (center2[a2][1]-center_y2_g)**2) <=SVF2 :
757                         cand_2.append([center2[a2][0],center2[a2][1]])
758
                            coupleF1F2F2c=[[x_1[a2_g],center2_g[a2_g][0],center2[a2][0]],[y_1[a2_g],center2_g[a2_g][1],center2[a2][1]]]
759                         couplesF1F2F2c.append(coupleF1F2F2c)
760                 a2+=1
761             if len(cand_2)==0:
762
                    coupleF1F2F2c=[[x_1[a2_g],center2_g[a2_g][0],0],[y_1[a2_g],center2_g[a2_g][1],0]] # no candidate for estimated F2* in F2 (found dots)
763                 couplesF1F2F2c.append(coupleF1F2F2c)
764             a2_g+=1
765
766
767     if len(center2)<len(center2_g): #less dots in picture (center2) as in the one before (center1)
768         a22_g=0
769         while a22_g<len(center2_g):
770             center_x2_g=center2_g[a22_g][0]
771             center_y2_g=center2_g[a22_g][1]
772             as012=list()
773             a012p=list()
774             as012p=list()
775             cand_2=list()
776
777             a22=0
778             count=0
779             while a22<len(center2):
780             # Circle ((x,y)2_g)https://stackoverflow.com/questions/12262017/python-checking-if-coordinates-are-within-circle
781                 test2=(center2[a22][0]-center_x2_g)**2 + (center2[a22][1]-center_y2_g)**2
782                 if ((center2[a22][0]-center_x2_g)**2 + (center2[a22][1]-center_y2_g)**2) <=SVF2 :
783                         cand_2.append([center2[a22][0],center2[a22][1]])
784
                            coupleF1F2F2c=[[x_1[a22_g],center2_g[a22_g][0],center2[a22][0]],[y_1[a22_g],center2_g[a22_g][1],center2[a22][1]]]
785                         couplesF1F2F2c.append(coupleF1F2F2c)
786                 a22+=1
787
788             if len(cand_2)==0:
789
                    coupleF1F2F2c=[[x_1[a22_g],center2_g[a22_g][0],0],[y_1[a22_g],center2_g[a22_g][1],0]] # no candidate for estimated F2* in F2 (found dots)
790                 couplesF1F2F2c.append(coupleF1F2F2c)
791             a22_g+=1
792
793 ##    print('couplesF1F2F2c: '+str(couplesF1F2F2c))
794 ##    print('len(couplesF1F2F2c): '+str(len(couplesF1F2F2c)))
795 ##    print()
796
797     #-------------------------------------------------------------------------------
798     # Step 5.2: Append F0 to found couplesF1F2F2c and create couplesF0F1F2
799     #-------------------------------------------------------------------------------
```

```
800         couplesF0F1F2F2c=list()
801         couplesF0F1F2c=list()
802 
803         test1=0
804         while test1<len(couplesF1F2F2c):
805             test2=0
806             count=0
807             while test2<len(couplesF0F1):
808                 if couplesF1F2F2c[test1][0][0]==couplesF0F1[test2][1][0] and
                    couplesF0F1[test2][1][0]!=0:
809 
                            coupleF0F1F2F2c=[[couplesF0F1[test2][0][0],couplesF1F2F2c[test1][0
                            ][0],couplesF1F2F2c[test1][0][1],couplesF1F2F2c[test1][0][2]],[cou
                            plesF0F1[test2][0][1],couplesF1F2F2c[test1][1][0],couplesF1F2F2c[t
                            est1][1][1],couplesF1F2F2c[test1][1][2]]]
810                         couplesF0F1F2F2c.append(coupleF0F1F2F2c)
811 
                            coupleF0F1F2c=[[couplesF0F1[test2][0][0],couplesF1F2F2c[test1][0][
                            0],couplesF1F2F2c[test1][0][2]],[couplesF0F1[test2][0][1],couplesF
                            1F2F2c[test1][1][0],couplesF1F2F2c[test1][1][2]]]
812                         couplesF0F1F2c.append(coupleF0F1F2c)
813                 else:
814                     count+=1
815                 if count==len(couplesF0F1):
816 
                            coupleF0F1F2F2c=[[0,couplesF1F2F2c[test1][0][0],couplesF1F2F2c[tes
                            t1][0][1],couplesF1F2F2c[test1][0][2]],[0,couplesF1F2F2c[test1][1]
                            [0],couplesF1F2F2c[test1][1][1],couplesF1F2F2c[test1][1][2]]]
817                         couplesF0F1F2F2c.append(coupleF0F1F2F2c)
818 
                            coupleF0F1F2c=[[0,couplesF1F2F2c[test1][0][0],couplesF1F2F2c[test1
                            ][0][2]],[0,couplesF1F2F2c[test1][1][0],couplesF1F2F2c[test1][1][2
                            ]]]
819                         couplesF0F1F2c.append(coupleF0F1F2c)
820                 test2+=1
821             test1+=1
822 
823 ##    print('couplesF0F1F2F2c: '+str(couplesF0F1F2F2c))
824 ##    print('len(couplesF0F1F2F2c): '+str(len(couplesF0F1F2F2c)))
825 ##    print('couplesF0F1F2c: '+str(couplesF0F1F2c))
826 ##    print('len(couplesF0F1F2c): '+str(len(couplesF0F1F2c)))
827 ##    print()
828 
829         #-------------------------------------------------------------------------------
830         # Step 6: Determine foundF1F2c
831         #-------------------------------------------------------------------------------
832         # Radius of search volume for F3
833         #radiusF3=100
834         SVF3=radiusF3**2
835 
836         couplesF1F2F2cF3c=list()
837         foundcouplesF1F2c=list()
838 
839         a2c=0
840         while a2c<len(couplesF0F1F2F2c):
841             center_x2_cand=couplesF1F2F2c[a2c][0][2]
842             center_y2_cand=couplesF1F2F2c[a2c][1][2]
843             mindists3=list()
844             noF2=list()
845             cand_3=list()
846 
847             # v1
848             vx_1=(couplesF0F1F2F2c[a2c][0][1]-couplesF0F1F2F2c[a2c][0][0])/deltaT
849             vy_1=(couplesF0F1F2F2c[a2c][1][1]-couplesF0F1F2F2c[a2c][1][0])/deltaT
850 
851             # v2
852             vx_2=(couplesF0F1F2F2c[a2c][0][3]-couplesF0F1F2F2c[a2c][0][1])/deltaT
853             vy_2=(couplesF0F1F2F2c[a2c][1][3]-couplesF0F1F2F2c[a2c][1][1])/deltaT
854 
855             # a1
856             ax_1=(vx_2-vx_1)/deltaT
857             ay_1=(vy_2-vy_1)/deltaT
```

```
858
859             # Estimated point of F3: F3_g
860             x1=couplesF0F1F2F2c[a2c][0][1]
861             y1=couplesF0F1F2F2c[a2c][1][1]
862             x_3_g=x1+vx_1*2*deltaT+ax_1*(2*deltaT)**2
863             y_3_g=y1+vy_1*2*deltaT+ay_1*(2*deltaT)**2
864             center3_g=[[x_3_g,y_3_g]]
865
866             #-------------------------------------------------------------------------
867             # If no F0 was found, velocity v1 is set to zero
868             #-------------------------------------------------------------------------
869             if abs(vx_1)==abs(x1):
870                     x_3_g=x1
871                     y_3_g=y1
872                     center3_g=[[x1,y1]]
873
874     ##        print('center3_g: '+str(center3_g))
875     ##        print('len(center3_g): '+str(len(center3_g)))
876
877             as0123=list()
878             a0123p=list()
879             as0123p=list()
880             F3dists=list()
881
882             a3=0
883             count=0
884             while a3<len(center3):
885                 # Circle
                    ((x,y)2_g)https://stackoverflow.com/questions/12262017/python-checking-if-
                    coordinates-are-within-circle
886                 if ((center3[a3][0]-x_3_g)**2 + (center3[a3][1]-y_3_g)**2) <=SVF3 :
887                     cand_3.append([center3[a3][0],center3[a3][1]])
888
                        coupleF1F2F2cF3c=[couplesF1F2F2c[a2c][0][0],couplesF1F2F2c[a2c][0][1],
                        couplesF1F2F2c[a2c][0][2],center3[a3][0]],[couplesF1F2F2c[a2c][1][0],c
                        ouplesF1F2F2c[a2c][1][1],couplesF1F2F2c[a2c][1][2],center3[a3][1]]
889                     couplesF1F2F2cF3c.append(coupleF1F2F2cF3c)
890                     F3dist=(center3[a3][0]-x_3_g)**2 + (center3[a3][1]-y_3_g)**2
891                     F3dists.append(F3dist)
892
                        a0123p.append([[couplesF0F1F2F2c[a2c][0][0],couplesF0F1F2F2c[a2c][0][1
                        ],couplesF0F1F2F2c[a2c][0][2],couplesF0F1F2F2c[a2c][0][3],center3[a3][
                        0]],[couplesF0F1F2F2c[a2c][1][0],couplesF0F1F2F2c[a2c][1][1],couplesF0
                        F1F2F2c[a2c][1][2],couplesF0F1F2F2c[a2c][1][3],center3[a3][1]]])
893
                        as0123p.append([[[couplesF0F1F2F2c[a2c][0][0],couplesF0F1F2F2c[a2c][0]
                        [1],couplesF0F1F2F2c[a2c][0][2],couplesF0F1F2F2c[a2c][0][3],center3[a3
                        ][0]],[couplesF0F1F2F2c[a2c][1][0],couplesF0F1F2F2c[a2c][1][1],couples
                        F0F1F2F2c[a2c][1][2],couplesF0F1F2F2c[a2c][1][3],center3[a3][1]],F3dis
                        t]])
894                 a3+=1
895
896     ##        print('len(cand_3): '+str(len(cand_3)))
897     ##        print()
898
899             if len(F3dists)==0:
900     ##              print('len(cand_3)==0: No F2 appended to track')
901
                    coupleF1F2F2cF3c=[[couplesF1F2F2c[a2c][0][0],couplesF1F2F2c[a2c][0][1]
                    ,couplesF1F2F2c[a2c][0][2],3333],[couplesF1F2F2c[a2c][1][0],couplesF1F
                    2F2c[a2c][1][1],couplesF1F2F2c[a2c][1][2],3333]]
902                 couplesF1F2F2cF3c.append(coupleF1F2F2cF3c)
903
904
905             if len(F3dists)>0:
906                     if len(F3dists)==1:
907                         F3dist=F3dists[0]
908
                        foundcouplesF1F2c.append([couplesF1F2F2cF3c[a2c][0][0],couplesF1F2
                        F2cF3c[a2c][0][2],couplesF1F2F2cF3c[a2c][1][0],couplesF1F2F2cF3c[a
                        2c][1][2],F3dist])
909                     else:
```

```
910                         F3dist=min(F3dists)
911                         coupleF1F2F2cF3c=a0123p[as0123.index(a0123)]
912                         couplesF1F2F2cF3c.append(coupleF1F2F2cF3c)
913
                            foundcouplesF1F2c.append([couplesF1F2F2cF3c[a2c][0][0],couplesF1F2
                            F2cF3c[a2c][0][2],couplesF1F2F2cF3c[a2c][1][0],couplesF1F2F2cF3c[a
                            2c][1][2],F3dist])
914             a2c+=1
915
916     ##    print('couplesF1F2F2cF3c: '+str(couplesF1F2F2cF3c))
917     ##    print('len(couplesF1F2F2cF3c): '+str(len(couplesF1F2F2cF3c)))
918     ##    print()
919     ##    print('foundcouplesF1F2c: '+str(foundcouplesF1F2c))
920     ##    print('len(foundcouplesF1F2c): '+str(len(foundcouplesF1F2c)))
921     ##    print()
922
923         #-------------------------------------------------------------------------------
924         # Step 7: Find foundF1F2
925         #-------------------------------------------------------------------------------
926         foundF1F2=list()
927
928         aa=0
929         while aa<len(foundcouplesF1F2c):
930             F1=foundcouplesF1F2c[aa][0]
931             bb=0
932             mindist=list()
933             mindistp=list()
934             minip=list()
935             count=0
936             while bb<len(foundcouplesF1F2c):
937                 if F1==foundcouplesF1F2c[bb][0]:
938                     dist=foundcouplesF1F2c[bb][4]
939                     mindist.append(dist)
940
                        minip.append([[foundcouplesF1F2c[bb][0],foundcouplesF1F2c[bb][1]],[fou
                        ndcouplesF1F2c[bb][2],foundcouplesF1F2c[bb][3]]])
941
                        mindistp.append([[[foundcouplesF1F2c[bb][0],foundcouplesF1F2c[bb][1]],
                        [foundcouplesF1F2c[bb][2],foundcouplesF1F2c[bb][3]]],dist])
942                     count+=1
943                 bb+=1
944             if count>1:
945                 minidist=min(mindist)
946                 foundF1F2i=mindistp[minip.index(minidist)]
947             else:
948                 foundF1F2i=minip[0]
949             foundF1F2.append(foundF1F2i)
950             aa+=1
951
952         #-------------------------------------------------------------------------------
953         # Step 8: Find F2, with no matching F1
954         #-------------------------------------------------------------------------------
955         #which center2 are not in foundF1F2: append to center2_not
956         center2_not=list()
957
958         testC2=0
959         if len(foundF1F2)==0:
960             while testC2<len(center2):
961                 center2_not.append([center2[testC2][0],center2[testC2][1]])
962                 testC2+=1
963         else:
964             while testC2<len(center2):
965                 testMC12=0
966                 count=0
967                 while testMC12<len(foundF1F2):
968                     if center2[testC2][0]==foundF1F2[testMC12][0][1]:
969                         print('center2[testC2][0]==foundF1F2[testMC12][1][0]')
970                     else:
971                         count+=1
972                     if count==len(foundF1F2):
973                         center2_not.append([center2[testC2][0],center2[testC2][1]])
974                     testMC12+=1
```

```
975                 testC2+=1
976
977     ##    print('center2_not: '+str(center2_not))
978     ##    print('len(center2_not): '+str(len(center2_not)))
979     ##    print()
980
981         #-------------------------------------------------------------------------------
982         # Step 9: Store found F2s in the matrices trackingx and trackingy (in the row of
            the matching F1)
983         #-------------------------------------------------------------------------------
984         if framenumber==1:
985             lenfoundcouplesF1F2=len(center0)+len(center1)-len(couplesF0F1)
986             row=0
987             while row<lenfoundcouplesF1F2: #
                lenfoundcouplesF1F2=len(center0)+len(center1)-len(mincouples01)
988                 test1=trackingx[row][j2-2] # F1_x in trackingx
989                 rows4=0
990                 while rows4<len(foundF1F2):
991                     if trackingx[row][j2-2]==foundF1F2[rows4][0][0]:
992                         trackingx[row][j2-1]=foundF1F2[rows4][0][1]
993                         trackingy[row][j2-1]=foundF1F2[rows4][1][1]
994                     rows4+=1
995                 row+=1
996             lenfoundcouplesF1F2+=len(center2_not)
997
998         if framenumber>1:
999             row=0
1000            while row<lenfoundcouplesF1F2:
1001                test1=trackingx[row][j2-2] # F1_x in trackingx
1002                rows4=0
1003                while rows4<len(foundF1F2):
1004                    if trackingx[row][j2-2]==foundF1F2[rows4][0][0]:
1005                        trackingx[row][j2-1]=foundF1F2[rows4][0][1]
1006                        trackingy[row][j2-1]=foundF1F2[rows4][1][1]
1007                    rows4+=1
1008                row+=1
1009
1010            rowsxNF2=lenfoundcouplesF1F2
1011            test=0
1012            while test<len(center2_not):
1013                trackingx[rowsxNF2][j2-1]=center2_not[test][0]
1014                trackingy[rowsxNF2][j2-1]=center2_not[test][1]
1015                test+=1
1016                rowsxNF2+=1
1017            lenfoundcouplesF1F2+=len(center2_not)
1018
1019    ##        print('lenfoundcouplesF1F2 after: '+str(lenfoundcouplesF1F2))
1020    ##        print()
1021
1022        # Export the tracking matrices for each frame
1023
            np.savetxt(path+"/"+date+"/"+framename+number+"/Tracking/Steps/"+framename+number+
            'trackingx_'+str(framenumber)+'.txt', trackingx, delimiter=',')
1024
            np.savetxt(path+"/"+date+"/"+framename+number+"/Tracking/Steps/"+framename+number+
            'trackingy_'+str(framenumber)+'.txt', trackingy, delimiter=',')
1025
1026        framenumber+=1
1027
1028    #-------------------------------------------------------------------------------
1029    # Step 10: Export the found tracks and the final tracking matrices trackingx and
        trackingy
1030    #-------------------------------------------------------------------------------
1031    trackingx_final=list()
1032    trackingy_final=list()
1033
1034    stepi=0
1035    while stepi< lenfoundcouplesF1F2+1:
1036        trackingx_final.append(trackingx[stepi])
1037        trackingy_final.append(trackingy[stepi])
1038
            np.savetxt(path+"/"+date+"/"+framename+number+"/Tracking/Tracks/"+framename+number
```

```
            +'trackingx_'+str(stepi)+'.txt', trackingx[stepi], delimiter=',')
1039
            np.savetxt(path+"/"+date+"/"+framename+number+"/Tracking/Tracks/"+framename+number
            +'trackingy_'+str(stepi)+'.txt', trackingy[stepi], delimiter=',')
1040        stepi+=1
1041
1042    np.savetxt(path+"/"+date+"/"+framename+number+"/Tracking/"+framename+number+'trackingx
        _final.txt', trackingx_final, delimiter=',')
1043    np.savetxt(path+"/"+date+"/"+framename+number+"/Tracking/"+framename+number+'trackingy
        _final.txt', trackingy_final, delimiter=',')
1044
1045    cv2.waitKey(0)
1046    cv2.destroyAllWindows()
1047
1048
1049
1050
1051
1052
1053
1054
1055
```

A.8 Python code: Film thickness

A.8.1 Calibration for the calculation of the film thickness

```
import cv2
import numpy as np
import sys
import glob
import pathlib
import matplotlib.pyplot as plt
from PIL import Image
from matplotlib.colors import hsv_to_rgb

# Name of the experiment
date='200603'
framename=date
number='_1'
fileextension='.mp4'
path=str(pathlib.Path(__file__).parent.absolute()).replace("\\","/")
print(path)

# cropbox
def crop_center(img,a1,a2,b1,b2):
    return img[a1:a2,b1:b2]


#-------------------------------------------------------------------------------------
# Calibration Camera
#-------------------------------------------------------------------------------------

pathCali='C:/Users/lieb_la/Desktop/Calibration/'
dateCali='20200509'
nameCali='B3-F2-CB-1'

#-------------------------------------------------------------------------------------
# Get frames from video
#-------------------------------------------------------------------------------------

#cropbox 135mm: 50x50mm
# a width, b lenght
a1=910
a2=1400
b1=250
b2=750
cropbox=(a1,b1,a2,b2)


#-------------------------------------------------------------------------------------
# Read and Save Calibration Parameters
#-------------------------------------------------------------------------------------

# Camera matrix
fileCM=open(pathCali+'/'+dateCali+'/'+nameCali+'/Results/Total/'+nameCali+'_cameramatrix_4_meanT.txt',"r")
CM=fileCM.readlines()

# Distortion coefficients
fileDC=open(pathCali+'/'+dateCali+'/'+nameCali+'/Results/Total/'+nameCali+'_distortioncoeff_4_meanT.txt',"r")
DC=fileDC.readlines()

mtx=np.zeros((3,3))
dist=np.zeros((1,5))
mtx[2][2]=1

mtx[0][0]=float(CM[0])  #fx
mtx[1][1]=float(CM[1])  #fy
mtx[0][2]=float(CM[2])  #cx
mtx[1][2]=float(CM[3])  #cy
dist[0][0]=float(DC[0]) #k1
dist[0][1]=float(DC[1]) #k2
dist[0][2]=float(DC[2]) #p1
dist[0][3]=float(DC[3]) #p2
dist[0][4]=float(DC[4]) #k3


```

```
71
72  frames=list()
73  correctedframes=list()
74  FT_all=list()
75  FT_sort=list()
76  framenumber=list()
77  zahl=0
78  framenumber_withFT=list()
79
80  # Print the detected lines
81  p=open(path+"/"+framename+number+'_Lines.txt',"w+")
82  p.write('Framenumber, Xs, Ys, Xs, Ys, Slope'+"\n")
83
84  g=open(path+"/"+framename+number+'_LinesXMean.txt',"w+")
85  g.write('Framenumber, XMeans'+"\n")
86
87  l=open(path+"/"+framename+number+'_LinesXMeanSorted.txt',"w+")
88  l.write('Framenumber, XMeans'+"\n")
89
90
91  lenxmeans_new=list()
92  xmeanALL=list()
93  minlines=list()
94  frame_withFT=0
95
96  # Take video and read it until video is completed: if Calibration is done with
    frames from a video
97  ##cap=cv2.VideoCapture(path+'/'+date+"/"+framename+number+"/"+framename+number+fileext
    ension)
98  ##while(cap.isOpened()):
99  ##  # Capture frame-by-frame
100 ##  ret, frame = cap.read()
101 ##  if ret == True:
102
103 for filename in glob.glob('*.png'): # if Calibration is done with frames
104         frame = cv2.imread(filename)
105         zahl+=1
106         print('Framenumber: '+str(zahl))
107         framenumber.append(zahl)
108
109         #-----------------------------------------------------------------------------
110         # Correction of the data using the calibration
111         #-----------------------------------------------------------------------------
112
113         imgO = frame
114         h,  w = imgO.shape[:2]
115         newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),0,(w,h))
          #value between 0 (whole picture range) and 1 (cut out of the picture)
116
117         # Undistort the original image
118         dst = cv2.undistort(imgO, mtx, dist, None, newcameramtx)
119
120         # Crop the orinigal image
121         x,y,w,h = roi
122         dst = dst[y:y+h, x:x+w]
123         correctedframes.append(dst)
124         frames.append(frame)
125         framecut=crop_center(dst,b1,b2,a1,a2)
126
          cv2.imwrite(path+'/Frames/'+framename+number+'_correctedframecut'+str(zahl)+'.
          png',framecut)
127         img=framecut
128
129         #-----------------------------------------------------------------------------
130         #Segmentation
131         #-----------------------------------------------------------------------------
132
133         # Convert BGR to HSV
134         hsv=cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
135
136         # Get value sensitivity
137         sens=20
```

```
138
139         # Define range of white color in HSV with adjustable sensitivity
140         lower_white = np.array([ 0 , 0 , 100-sens])
141         upper_white = np.array([ 0 , 0 , 100     ])
142
143         # Threshold the HSV image to get only white colors, (in the range of the
            mask: 1 (white))
144         mask = cv2.inRange(hsv, lower_white, upper_white)
145
146         # Bitwise-AND mask and original image, in the frame ther is the frame and
            mask ist true; Bitwise: where are 1s in the frame: show the color)
147         res = cv2.bitwise_and(img,img, mask= mask)
148         cv2.imwrite(path+"/Frames/"+framename+number+'_res'+str(zahl)+'.png',res)
149
150         # Find the edges
151         edges = cv2.Canny(res,100,250) #,apertureSize=3)
152         cv2.imwrite(path+"/Frames/"+framename+number+'_edges'+str(zahl)+'.png',edges)
153
154         #-------------------------------------------------------------------------
155         # Read detected lines from calibration
156         #-------------------------------------------------------------------------
157         ymeans=list()
158         xmeans=list()
159         y1s=list()
160         x1s=list()
161         y2s=list()
162         x2s=list()
163         slopes=list()
164         angles=list()
165
166         # Criteria for the line detection
167         minLineLength = 5 #min lenght of lines, line segments shorter than that are
            rejected
168         maxLineGap = 4 #maximum gap between line segments to treat them as a single
            line
169         minangle=85 #eliminate all lines with angle >x° (horizontal lines)
170
171         lines = cv2.HoughLinesP(edges,1,np.pi/180,10,minLineLength,maxLineGap)
172         g.write(str(zahl)+',')
173         l.write(str(zahl)+',')
174         try:
175             for line in lines:
176                 x1,y1,x2,y2 = line[0]
177                 angle = np.arctan2(y2 - y1, x2 - x1) * 180. / np.pi
178                 if abs(angle) >= minangle:
179                     cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
180                     angles.append(angle)
181                     cv2.line(img,(x1,y1),(x2,y2),(0,255,0),2)
182                     x1s.append(x1)
183                     x2s.append(x2)
184                     y1s.append(y1)
185                     y2s.append(y2)
186                     slope=(y2-y1)/(x2-x1)
187                     p.write(str(zahl)+','+str(x1)+','+
                        str(y1)+','+str(slope)+','+str(x2)+','+ str(y2)+"\n")
188                     slopes.append(slope)
189                     xmean=(x1+x2)/2
190                     g.write(str(xmean)+',')
191                     xmeans.append(xmean)
192                     ymean=(y1+y2)/2
193                     ymeans.append(ymean)
194         except TypeError:
195             print('No lines detected')
196
197         p.write('\n')
198         g.write('\n')
199
200         xmeans.sort()
201         xmeans_new=list()
202         means=list()
203         mindist=5
204         count=0
```

```
205             if len(xmeans)>1:
206                 while count < len(xmeans):
207                     count2=0
208                     counti=0
209                     while count2 < len(xmeans):
210                         if count != count2:
211 
212                             if abs(xmeans[count]-xmeans[count2])< mindist:
213                                 mean=(xmeans[count]+xmeans[count2])/2
214                                 means.append(mean)
215                             else:
216                                 counti+=1
217                             if counti==len(xmeans)-1:
218                                 xmeans_new.append(xmeans[count])
219                         count2+=1
220                     count+=1
221             else:
222                 xmeans_new=xmeans
223 
224             means2=means
225             x=0
226             while x <len(means):
227                 x2=0
228                 while x2<len(means2):
229                     if x!=x2:
230                         if means[x]==means2[x2]:
231                             del means2[x2]
232                     x2+=1
233                 x+=1
234 
235             x3=0
236             while x3<len(means2):
237                 xmeans_new.append(means2[x3])
238                 x3+=1
239 
240             xmeans_new.sort()
241             lenxmeans_new.append(len(xmeans_new))
242             xmeanALL.append(xmeans_new)
243             l.write(str(xmeans_new).replace("[","  ").replace("]","  "))
244             l.write('\n')
245 
246             cv2.imwrite(path+'/Frames/'+framename+number+'_detectedLines'+str(zahl)+'.png'
                ,img)
247     ##
248     ##  # Break the loop if a video is used for calibration
249     ##  else:
250     ##    break
251 
252     ##cap.release()
253     p.close()
254     g.close()
255     l.close()
256 
257     # Save to folder
258     xmeans_MEAN=list()
259     maxlenxmeans_new=max(lenxmeans_new)
260     x=0
261     while x< maxlenxmeans_new:
262         b=0
263         y=0
264         while y<len(xmeanALL):
265             b+=xmeanALL[y][x]
266             y+=1
267         xmeans_MEAN.append(b/y)
268         x+=1
269     print('xmeans_MEAN: '+str(xmeans_MEAN))
270 
271     #-------------------------------------------------------------------------------------
272     # Calculation of calibration parameters for the determination of the film thickness
273     #-------------------------------------------------------------------------------------
274     # Scale on the pin
    ScalePIN=list(range(1, len(xmeans_MEAN)+1))
```

```
275
276   # Linear fit
277   m2,b2 = np.polyfit(xmeans_MEAN,ScalePIN,  1)
278
279   # Save
280   z=open(path+"/"+framename+number+'_CalibrationFT.txt',"w+")
281   z.write(str(m2)) # slope
282   z.write('\n')
283   z.write(str(b2)) # intercept
284   z.close()
285
286   cv2.waitKey(0)
287   cv2.destroyAllWindows()
288
289
290
```

A.8.2 Determination of the film thickness

```
import cv2
import numpy as np
import sys
import pathlib
import matplotlib.pyplot as plt
from PIL import Image
from matplotlib.colors import hsv_to_rgb

# name of the experiment
date='200617'
framename=date
number='_1'
fileextension='.mp4'
path=str(pathlib.Path(__file__).parent.absolute()).replace("\\","/")

# save to folder
p=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_filmthickness.txt',"w+")
p.write('Framenumber xmaxdots Filmthickness'+"\n")

k=open(path+"/"+date+"/"+framename+number+"/"+framename+number+'_filmthickness_mean.txt',"w+")

#-----------------------------------------------------------------------------------------
# Calibration Camera
#-----------------------------------------------------------------------------------------

pathCali='G:/Liebl/4-analysis/tracer tracking/calibration/'
dateCali='20200509'
nameCali='B3-F2-CB-1'

#-----------------------------------------------------------------------------------------
# Calibration FT
#-----------------------------------------------------------------------------------------

pathCaliFT=path+"/calibration/"
dateCaliFT='200603'
nameCaliFT='_1'

fileCaliFT=open(pathCaliFT+'/'+dateCaliFT+'/'+dateCaliFT+nameCaliFT+'/'+dateCaliFT+nameCaliFT+'_CalibrationFT.txt',"r")
CalibrationFT=fileCaliFT.readlines()
slope=float(CalibrationFT[0])
intercept=float(CalibrationFT[1])

#-----------------------------------------------------------------------------------------
# Cropbox (Same size than in calibration!)
#-----------------------------------------------------------------------------------------

# cropbox
def crop_center(img,a1,a2,b1,b2):
    return img[a1:a2,b1:b2]

#cropbox 135mm: 50x50mm
# a width, b lenght
a1=910
a2=1400
b1=250
b2=750
cropbox=(a1,b1,a2,b2)

maxLimit=300 # Has to be adjusted!!

#-----------------------------------------------------------------------------------------
# Read and Save Calibration Parameters
#-----------------------------------------------------------------------------------------

# Camera matrix
fileCM=open(pathCali+'/'+dateCali+'/'+nameCali+'/Results/Total/'+nameCali+'_cameramatrix_4_meanT.txt',"r")
CM=fileCM.readlines()

```

```
69  # Distortion coefficients
70  fileDC=open(pathCali+'/'+dateCali+'/'+nameCali+'/Results/Total/'+nameCali+'_distortion
    coeff_4_meanT.txt',"r")
71  DC=fileDC.readlines()
72
73  mtx=np.zeros((3,3))
74  dist=np.zeros((1,5))
75  mtx[2][2]=1
76
77  mtx[0][0]=float(CM[0])  #fx
78  mtx[1][1]=float(CM[1])  #fy
79  mtx[0][2]=float(CM[2])  #cx
80  mtx[1][2]=float(CM[3])  #cy
81  dist[0][0]=float(DC[0]) #k1
82  dist[0][1]=float(DC[1]) #k2
83  dist[0][2]=float(DC[2]) #p1
84  dist[0][3]=float(DC[3]) #p2
85  dist[0][4]=float(DC[4]) #k3
86
87
88  #-------------------------------------------------------------------------------
89  # Get frames from video, undistort them and detect particles
90  #-------------------------------------------------------------------------------
91  frames=list()
92  correctedframes=list()
93  framenumber=list()
94  framenumber_withFT=list()
95  zahl=0
96  FT_all=list()
97  FT2_all=list()
98
99  #take video
100 cap=cv2.VideoCapture(path+'/'+date+"/"+framename+number+"/"+framename+number+fileexten
    sion)
101 # Read until video is completed
102 while(cap.isOpened()):
103   # Capture frame-by-frame
104   ret, frame = cap.read()
105   if ret == True:
106
107         zahl+=1
108         print('Framenumber: '+str(zahl))
109         p.write(str(zahl)+',')
110 ##
    cv2.imwrite(path+"/"+date+"/"+framename+number+'/Frames/'+framename+number+'_frame'+st
    r(zahl)+'.png',frame)
111         framenumber.append(zahl)
112
113         #-------------------------------------------------------------------------
114         # Correction of the data using the calibration
115         #-------------------------------------------------------------------------
116
117         imgO = frame
118         h,  w = imgO.shape[:2]
119         newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),0,(w,h))
            #value between 0 (whole picture range) and 1 (cut out of the picture)
120
121         # Undistort the original image
122         dst = cv2.undistort(imgO, mtx, dist, None, newcameramtx)
123
124         # Crop the orinigal image
125         x,y,w,h = roi
126         dst = dst[y:y+h, x:x+w]
127         correctedframes.append(dst)
128         frames.append(frame)
129         framecut=crop_center(dst,b1,b2,a1,a2)
130 ##
    cv2.imwrite(path+"/"+date+"/"+framename+number+'/Frames/'+framename+number+'_framecut'
    +str(zahl)+'.png',framecut)
131         im=framecut
132
133         #-------------------------------------------------------------------------
```

```
134         # Detection of particles
135         #-------------------------------------------------------------------------------
136
137         # Setup SimpleBlobDetector parameters.
138         params = cv2.SimpleBlobDetector_Params()
139
140         # Change thresholds
141         params.thresholdStep = 5
142         params.minThreshold = 60
143         params.maxThreshold = 255 # limit, so that the pin is not detected: has to
            be checked
144
145         # Filter by Color (one specific color needed: not recommended)
146         params.filterByColor = False
147         params.blobColor = 255
148
149         # Filter by Area depending on the distance between the camera and the
            particle tracer mixture. see: Excel: picture-pillow
150         params.filterByArea = True
151         params.minArea = 40  # 60mm:127, 80mm:92, 100mm:60, 135mm:40, 150mm:26,
            200mm:17
152         params.maxArea = 175  # 60mm:500, 80mm:366, 100mm:238, 135mm:158, 150mm:102,
            200mm:66
153
154         # Filter by Circularity
155         params.filterByCircularity = True
156         params.minCircularity = 0.7 # circularity of a square
157
158         # Filter by Convexity (Area of the Blob / Area of it's convex hull)
159         params.filterByConvexity = False
160         params.minConvexity = 0.87
161
162         # Filter by Inertia (ratio of widest to thinnest point, measures how
            elongated a shape is, circle:1, ellipse0-1, line:0)
163         params.filterByInertia = True
164         params.minInertiaRatio = 0.01
165
166         params.minDistBetweenBlobs = 1.0  # keypoints closer will be merged
167         params.minRepeatability = 2
168
169
170         # Create a detector with the parameters
171         ver = (cv2.__version__).split('.')
172         if int(ver[0]) < 3 :
173                 detector = cv2.SimpleBlobDetector(params)
174         else :
175                 detector = cv2.SimpleBlobDetector_create(params)
176
177
178         # Detect blobs
179         keypoints = detector.detect(im)
180         blobnumber=len(keypoints)
181
182
183         # Draw detected blobs as red circles
184         # cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS ensures that the size of the
            circle corresponds to the size of blob
185         im_with_keypoints = cv2.drawKeypoints(im, keypoints, np.array([]),
            (0,0,255), cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
186
187         center=list()
188         ycenterdots=list()
189         xcenterdots=list()
190         diameter=list()
191         diametermm=list()
192
193         for keypoint in keypoints:
194             x = keypoint.pt[0]
195             y = keypoint.pt[1]
196             s = keypoint.size/2
197             KeyPoints=[x,y,s]
198             center.append(KeyPoints)
```

```
199             ycenterdots.append(y)
200             xcenterdots.append(x)
201             diameter.append(s*2)
202 
203         centerint = np.uint16(np.around(center)) #circle only can handle int
204 
205         # To check the detected circles
206         for i in centerint:
207         # Draw the outer circle
208             cv2.circle(im,(i[0],i[1]),i[2],(0,255,0),2)
209         # Draw the center of the circles.
210             cv2.circle(im,(i[0],i[1]),2,(0,0,255),1)
211 
212         #-------------------------------------------------------------------------------
213         # Find the highest detected particle
214         #-------------------------------------------------------------------------------
215         # Values of the highest dot
216         ymin_centerdots=min(ycenterdots)
217         xmax_centerdots=max(xcenterdots)
218         print('xmax_centerdots: '+str(xmax_centerdots))
219         radiusofcentermax=diameter[xcenterdots.index(xmax_centerdots)]/2
220         xmax_centerdotswithRadius=xmax_centerdots+radiusofcentermax
221         print('xmax_centerdotswithRadius: '+str(xmax_centerdotswithRadius))
222         p.write(str(xmax_centerdots)+',')
223 
224         # Draw a line to the maximum detected dot
225         cv2.line(im,(int(xmax_centerdotswithRadius),
            0),(int(xmax_centerdotswithRadius),1080), (255,0,0),1)
226         cv2.line(im,(int(xmax_centerdots), 0),(int(xmax_centerdots),1080),
            (0,0,255),1)
227 
            cv2.imwrite(path+"/"+date+"/"+framename+number+'/Frames/'+framename+number+'_d
            etectedCircles'+str(zahl)+'.png',im)
228 
229         #-------------------------------------------------------------------------------
230         # Film thickness calculation
231         #-------------------------------------------------------------------------------
232         if xmax_centerdots < maxLimit:
233             FT=xmax_centerdots*slope+intercept
234             FT2=xmax_centerdotswithRadius*slope+intercept
235             FT_all.append(FT)
236             FT2_all.append(FT2)
237             framenumber_withFT.append(zahl)
238             p.write(str(FT)+',')
239         p.write("\n")
240 
241     # Break the loop
242     else:
243       break
244 
245 cap.release()
246 p.close()
247 
248 
249 # Calculation of mean film thickness
250 FT_mean= np.mean(FT_all)
251 FT_stab=np.std(FT_all)
252 print('FT_Mean (Center of the highest dot): '+str(FT_mean))
253 print('FT_stab: '+str(FT_stab))
254 print('Number of FT measurements with xmax_dots<Limit:'+str(len(FT_all)))
255 k.write('FT_Mean (Center of the highest dot): '+str(FT_mean)+'\n')
256 k.write('FT_stab: '+str(FT_stab)+'\n')
257 k.write('Number of FT measurements with xmax_dots<Limit:'+str(len(FT_all)))
258 k.close()
259 
260 FT2_mean= np.mean(FT2_all)
261 FT2_stab=np.std(FT2_all)
262 print('FT2_Mean (Center+radius of the highest dot): '+str(FT2_mean))
263 print('FT2_stab: '+str(FT2_stab))
264 
265 # Plot of the Filmthickness
266 plt.plot(framenumber_withFT,FT_all,'bo',markersize=1)
```

```
267   plt.title('Filmthickness')
268   plt.xlabel('Framenumber')
269   plt.ylabel('Filmthickness in mm')
270   #plt.show()
271   plt.savefig(path+"/"+date+"/"+framename+number+'/'+framename+number+'_FT.png')
272   plt.close()
273
274   cv2.waitKey(0)
275   cv2.destroyAllWindows()
276
277
278
```