Hochschule Düsseldorf
University of Applied Sciences
HSD

Implementierung von CDP

Entwicklung eines Programmiercodes in Python zur Untersuchung und Messung von CDP bei Fahrerassistenzkameras

Bachelor Thesis von Lukas Ebbert

Matrikelnummer: 675102

Im Studiengang Wirtschaftsingenieur Elektrotechnik

Fachbereich Elektro- und Informationstechnik

August 22, 2018

Erstprüfer: Prof. Dr. rer. nat. Alexander Braun
Zweitprüfer: Dr. rer. nat. Marc Geese

Contents

Eidesstattliche Versicherung

Hiermit versichere ich, Lukas Ebbert, an Eides statt, die vorliegende Bachelor Thesis selbständig verfasst und keine weiteren als die angegebenen Hilfsmittel und Quellen benutzt zu haben.

Dies ist die von der Hochschule Düsseldorf zu bewertende Version.

Ort, Datum ____________________ Unterschrift ___________________

Acronyms

ABS	antilock braking system
ACC	adaptive cruise control
CDP	contrast detection probability
HDR	high dynamic range
IEEE	Institute of Electrical and Electronics Engineers
ISP	image signal processor
KPI	key performance indicator
OEC	optical electrical conversion
OEM	original equipment manufacturer
ROI	region of interest
SNR	signal to noise ratio
Tier1	system supplier of OEM
Tier2	component supplier of OEM

1. Introduction

The developing of driver assistance systems is increasing in the last half decade. A lot of people are interested in driver assistance function. So nearly every sold new car includes some driver assistance functions like for example antilock braking system (ABS) which is already legally obligated in some states[2]. But the development is going further and original equipment manufacturer (OEM), also known as car maker for example Daimler, BMW, and others, are trying to introduce adaptive cruise control (ACC) on an autonomous basis. These advanced driver assistance functions are using different types of sensors for example radar, camera, lidar and others. And for all these sensors it is important to describe their quality and functions correctly. But the actual existing image quality standards and their key performance indicator (KPI) are not appropriate enough because their goal is focusing on other use cases. For example the EMVA 1288 is an often used standard to describe image quality but this standard is focused on viewing applications[3]. To solve these occurring requirement problems the Institute of Electrical and Electronics Engineers (IEEE) started a working group to implement a standard for machine vision applications with focus on automotive applications.

In this working group a new KPI was introduced. The KPI is named contrast detection probability (CDP). The goal of this bachelor thesis is to develop a python program to simulate this new defined KPI. This program should deliver a common program for all OEM, system supplier of OEM (Tier1) and all component supplier of OEM (Tier2) to evaluate the CDP of their system or component. At the beginning a basis program was already existing from the P2020 Face to Face Meeting in Brussels which was extended and reworked.

To reach this goal the definition of imaging chain will be explained at the beginning. Afterwards the working group of the IEEE will be described. In the next subsection the KPI called CDP which is evaluated in this thesis will be presented.

In the next chapter the developed program will be explained. This chapter is divided into two section one of these is targeting on the structure of the program the other one is describing the methods of the program. The section functions is divided into three subsection for each application file.

After the program is presented the occurred limits of CDP will be mentioned to understand the use cases of CDP correctly.
In the last chapter different cases of the CDP evaluation are simulated to present the effects of some special selected parameters. This help to understand which parameters do affect the CDP value.

2. Image quality metrics

In this chapter there will be an explanation of an imaging chain, an explanation of the background of this project and an explanation about the new KPI named CDP.

2.1. Imaging chain of a camera system

An imaging chain consists of different steps to process a scene into digital numbers. An example of an imaging chain model is shown in figure 2.1. This imaging chain focuses on automotive applications. It consists of a scene where light beams in all possible directions in the world. This light beams transmit through a windscreen of a car where the light loses intensity and veiling glare is added to the signal. These loses are differ for each specific windscreen because each material behaves different. Afterwards the optic is transmitted. The optic are lenses which are different for each camera. Then the light hits the image sensor often called imager. In the imager the signal is processed by different steps they are also exemplary shown in figure 2.1. In this sensor model at first the optical electrical conversion (OEC) is done. After the signal is converted from an optical signal to an electronic signal dark current is added to the signal. Dark current can be simulated by different models for example by using a random Poisson process. A image sensor has a defined range of values he can handle. All signals above this value will be cut this process is called clipping. In an existing imager the signal will not rise above this value. At the end of the processed steps in the imager the signal will be converted from an analog signal to a digital signal. For example it could be implemented by just cutting the decimal or with a quantization factor often called K-Factor. Then the signal will be processed by an image signal processor (ISP). In an ISP can happen different processing steps for example an interpolation between neighboring pixels or a tone mapping for a high dynamic range (HDR) image. Tone mapping is resizing an image from an image with high resolution to an image with a lower resolution.
In figure 2.1 is only an exemplary imaging chain shown. Every image chain of a specific camera system is different. This example is focusing on automotive applications so the windscreen is part of the imaging chain. A camera system to take images would

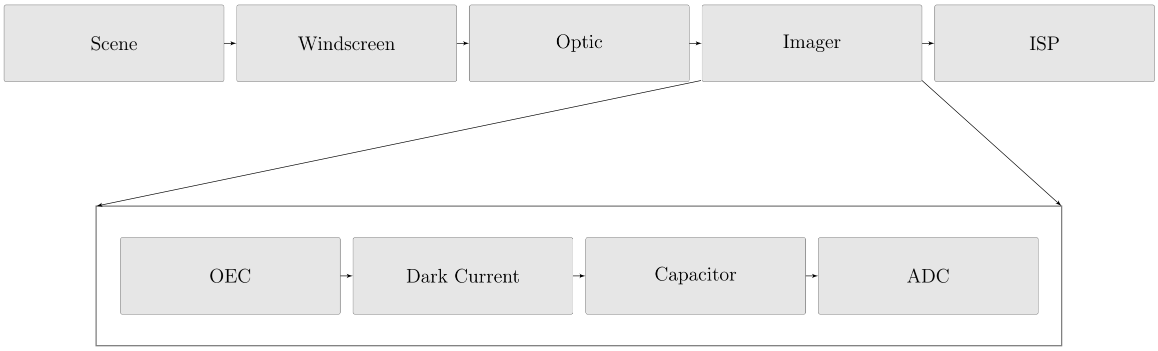


Figure 2.1.: Example of an imaging chain of a digital camera system[4, adapted from]

not include a windscreen. Also the steps which are done by separate parts are different for each model. For example the OEC in every sensor model may differ. Also every supplier makes different approaches of the windscreen to reproduce the scene. Every supplier implements their own sensor model which often diverse[5, p. 188f].

2.2. Project P2020 at the Institute of Electrical and Electronics Engineers

To understand the background of this project the P2020 working group of the IEEE will be explained in this section. The project P2020 is a working group which is developing an image quality standard for automotive applications. This group started working in July 2016. Their goal is to find a common language to describe image quality and camera systems for automotive applications correctly and consistently[7]. The problem occurred with the introduction of cameras in cars to solve the appearing problems by the development of driver assistant systems. There are a lot of standards to describe the quality of cameras, but most of them concentrate on viewing applications like EMVA 1288 and are not created for machine viewing applications.
But at a more detailed look at some of the KPI of EMVA it seems that these KPIs are not convenient to describe image quality for machine viewing applications. As a example the Signal to Noise Ratio (SNR) is suitable which is defined by equation 2.1[3]. Sometimes it is also used with the factor 20.

$$SNR = 10 \cdot \log(Signal/Noise) \tag{2.1}$$

In figure 2.2 is the course of SNR over the imaging chain illustrated. On the y-axis the SNR is plotted and on the x-axis the step of the imaging chain is shown. A patch of same pixels were used for this evaluation. It is visible with the blue line that SNR

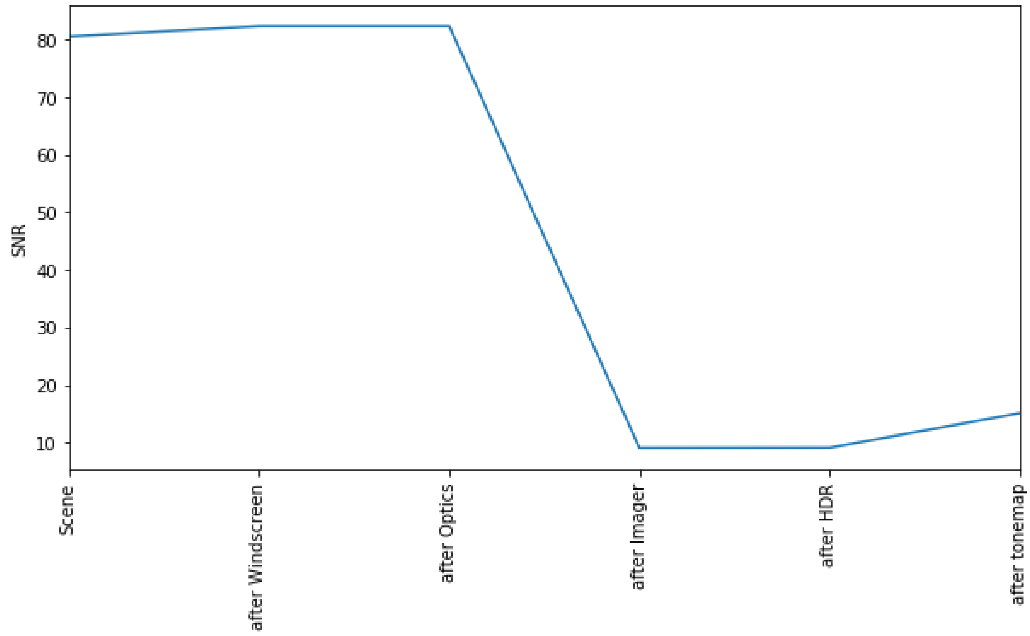


Figure 2.2.: Signal to Noise Ratio course of the Imaging Chain

increases after the windscreen caused through the veiling glare which increases the signal but the noise is not changing. Also the SNR increases through the tone map algorithm because the standard deviation of the signal gets decreased through the quantization effects because the signal is compressed to a lower diversity.

But this does not makes sense because normally after each step the image quality should decrease because with each step of the imaging chain the data quality is decreasing caused through the processing. This is caused by the fact that it is not possible to gain new or better information with a process which is a reproduction of the step before.

Also if the image sensor reaches its full well level the signal to noise ratio (SNR) rises to infinite. Because noise strives to zero. But all information in the image is gone because all pixels have the same value.

Because of this problems a lot of OEM, Tier1 and Tier2 joined forces to develop a language to describe image quality in a more common way. A lot of them are participating in the project P2020 to develop a standard for their specific use case for example Bosch, Valeo, Daimler, ON Semiconductor, Sony Semiconductor and many more[9].

2.3. Contrast detection probability

One of the new KPI's is the so called contrast detection probability. This KPI was developed because camera systems are able to detect objects via contrast. Contrast is the luminance difference of two different light patches. Luminance is the light beamed from a given object. The physical domain of luminance is cd/m^2. The contrast used for CDP is the Weber contrast see equation 2.2[8]. This contrast is defined by the relation of the highest luminance patch to the lowest luminance patch minus 1. This contrast is used because numbers from zero to infinite can occur. So every object has a specific contrast.

$$K_{Weber} = \frac{E_{max}}{E_{min}} - 1 \tag{2.2}$$

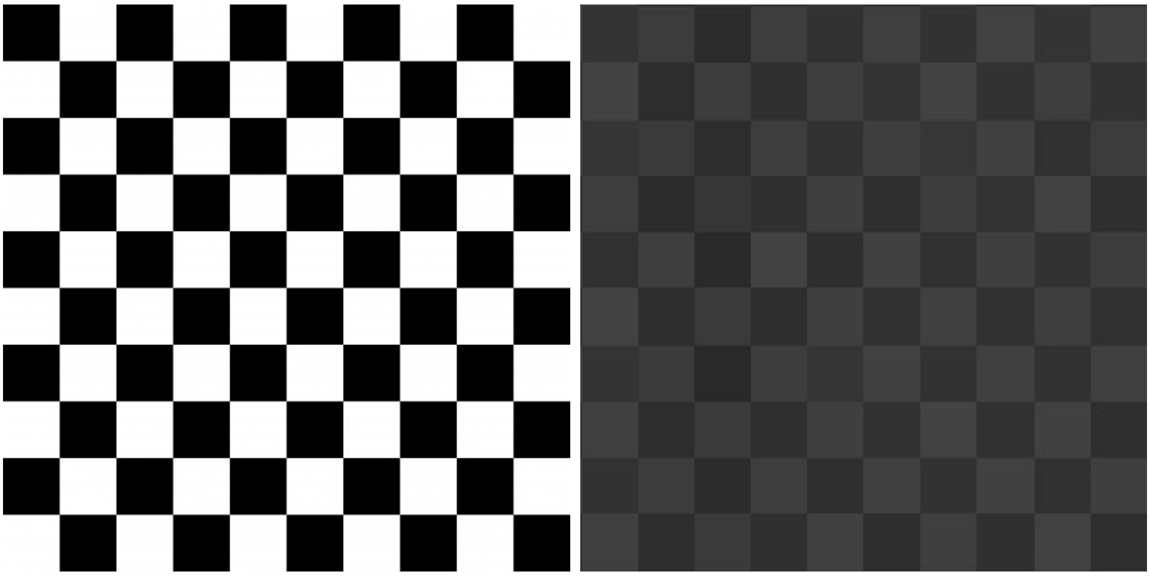
CDP has the goal to be a metric to give information about the probability that a given contrast can be detected by the system or component. Also it was important that it can be used for each part of the imaging chain in system and component level and the possible transfer to other sensor systems of driver assistance functions. The results of CDP are between zero and one because the wish was to have an absolute space where the KPI is suitable. This makes the definition of requirements to the supplier more comfortable.

The definition of CDP is shown in equation 2.3[4].

$$CDP_{K_{in}} = Prob(K_{in}(1-\varepsilon) \leq K_{meas.} \leq K_{in}(1+\varepsilon)) \tag{2.3}$$

This implies for every contrast K_{in} in Weber Contrast, see equation 2.2, the system has a specific probability that a contrast can be recognized under given surroundings for example veiling glare or temperature. The results will be between zero and one. It depends on the defined epsilon. Epsilon will be defined by the use cases and the ability of the used neural networks to detect a specific object. So for example if the neural network behind the camera is able to detect a 100% contrast if contrasts between 50% and 150% are appearing. Then the epsilon has to be 0.5.

To calculate the CDP two different region of interest (ROI) are defined one dark and one bright ROI. In figure 2.3 (a) is a 100 % checkerboard shown. In this example the first ROI would include the black parts and the second ROI would exists of the white parts. With the luminance of the patches measured in cd/m^2. This results in a probability density function gray balks and a summed probability function blue line shown in figure 2.4 the value of the density function is normed to one. On the x-axis



(a) 100% checkerboard as perfect visualization (b) 100% checkerboard after the ISP

Figure 2.3.: Checkerboard with a $K_{Weber} = 100\%$

the values and on the y-axis the probability of these values are illustrated. In this case two signals are appearing with a Weber contrast of 100% which is the K_{in} for this scene. After processed through the imaging chain the data has changed this is visible in figure 2.3 (b). This difference to the original scene is caused by the processing of the imaging chain described in chapter 2.1. So the summed probability and density function has changed, illustrated through figure 2.5. Now the contrast between each pixel from the first ROI to each pixel of the second ROI has to be calculated which results in different appearing contrasts. Figure 2.6 shows all occurring contrasts with the x-axis illustrating the contrast in a familiar illustration like used before. With the summed probability function the CDP can be calculated. The K_{in} as mentioned before is 100%. If the above described epsilon = 0.5 is used. This epsilon produces a confidence interval between 150% and 50% where the contrast can be detected. So to calculate the CDP the value of the summed probability function at the point 150% has to be subtracted by the value of 50%. This equals the CDP in this example it is read from the image as approximately 0.65.

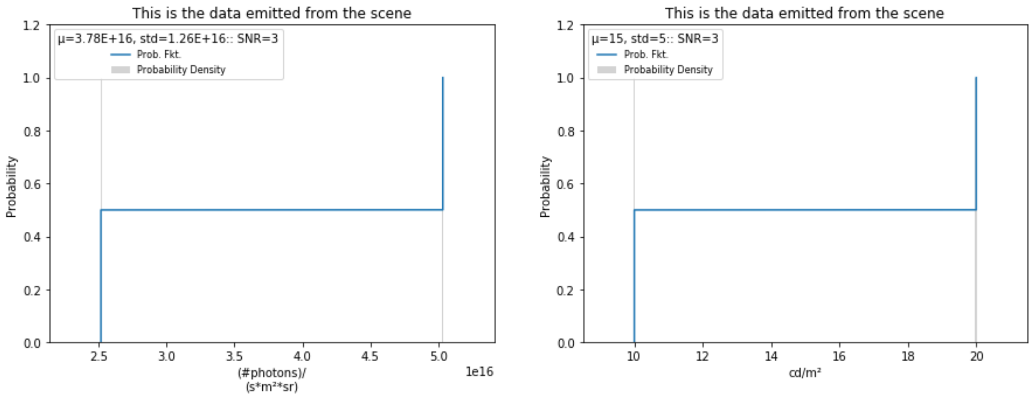


Figure 2.4.: Plot with the values of figure 2.3 (a)

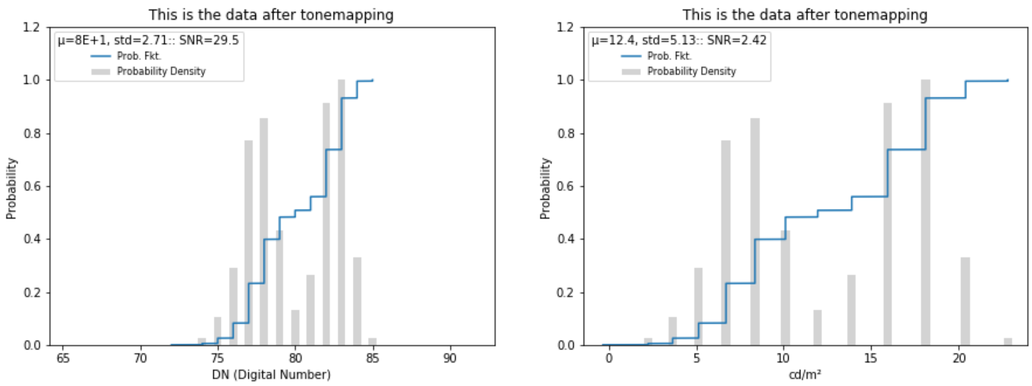


Figure 2.5.: Plot with the values of figure 2.3 (b)

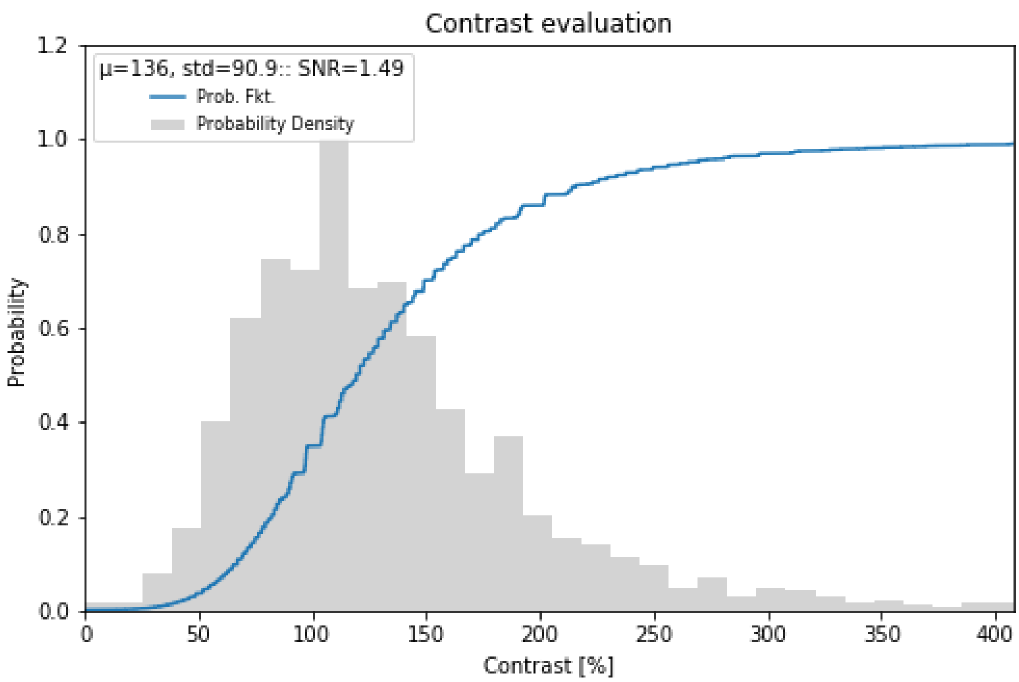


Figure 2.6.: All occurring contrasts of the image

3. Implementing CDP

This chapter describes the structure and the functionality of the program. Therefore the outputs generated by the program are illustrated. The outputs consist of charts and different float numbers. Also the steps which are done by the program are described.

3.1. Structure of the program

The program consists of methods and application files. The Application files use the methods or functions to show how to use these methods and gain output. Methods or functions are the processing steps inside an imaging chain explained in chapter 2.1 also methods are the calculation of CDP and SNR.
All files are stored in one main folder. In this main folder are some other folders. These folders include functions, images to evaluate, videos to evaluate and XLS results. The applications files are stored in the main folder. Like the name says the functions files will be found in the functions folder. In the folder images to evaluate the images used for the CDP evaluation have to be saved. Videos to evaluate is to store the videos in this specific folder. If results have to be written to a XLS sheet the created file is stored in XLS results.
It exists ten function files see figure 3.1. These files represent each step of the imaging chain, one python class with a common data structure for the program, one file for each image and video processing and also there is a python file to evaluate the image quality in different ways. The implemented parts of the imaging chain are the parts described in chapter 2.1 additionally a second sensor model is implemented to show the possibility to extend the program. Figure 3.1 shows which application file uses which methods file. On the left side are the three application files shown on the right side the ten method files. The method file C_Test is the second imager model file.

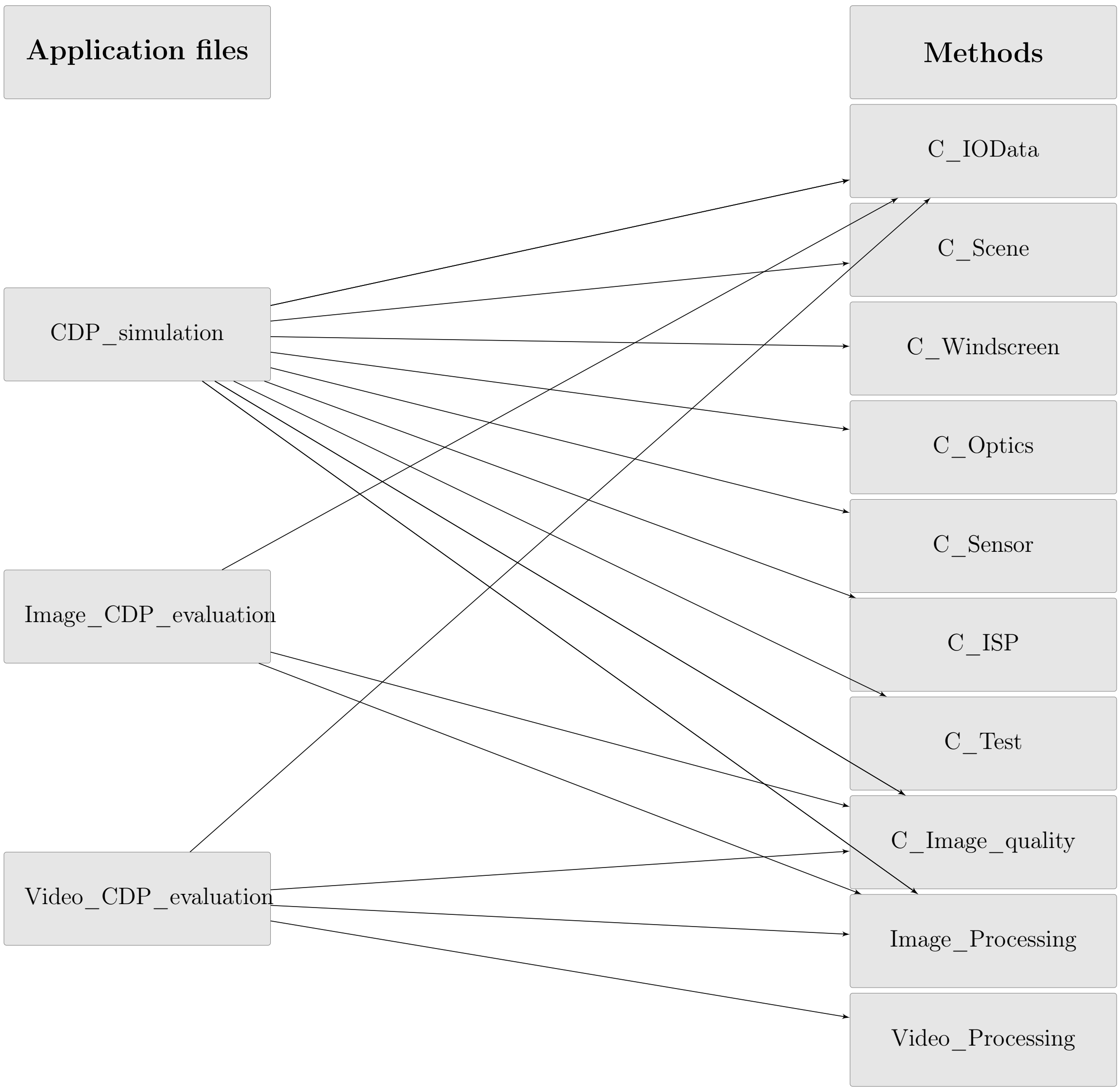


Figure 3.1.: Dependence of the files to each other

3.2. Functions of the program

In this section the functions of the program will be described. Each function is a step in the imaging chain and has a influence on the value of CDP. This influence is illustrated by plotting the relevant data after each step. Also there is an explanation of the generated plots. The functionality is additionally documented in python. The description mentions which variables are necessary, which are optional and which will be returned by the function see the program export in the appendices. If a variable is not known a value is used which could represent a logical value. It is possible to extend the separate parts for a specific use case. There are a lot of different models for each step they could be added but a lot of them are not public so a public one is implemented. Most of the following plots reference to a simulated checkerboard with a contrast of 100% between black and white shown in figure 2.3 (a).

3.2.1. CDP simulation

Class C_IOData

The first Python file to mention is the C_IOData file. This Python file delivers an uniformed data structure for every part of the program. The structure consists of two arrays, one for data the imager is using and one for the data transformed into the physical domain which is cd/m^2 like the evaluation of CDP uses. There is one function for the conversion back to the physical domain. Besides these arrays there are different strings existing. They are responsible for the description of the charts. There are more variables implemented for example pixel pitch and quantum efficiency, they are used to calculate back from a taken image to the physical domain to evaluate the CDP of a taken image.
Also in this class are different functions. There is one main function which calls the sub functions for plotting. The function plots up to two plots for each function call. It plots two charts if both arrays of the data structure are filled otherwise it plots one chart. In the charts the summed probability function over all values appearing in the image or scene and the density function of these values is displayed these values are normed to one. The summed probability function is shown as a blue continuous line and the density function is shown as gray balks. On the y-axis is the probability and on the x-axis is the value from the displayed array plotted. In the upper area of the plots the SNR value, the mean as μ and the standard deviation, std, of the scene is shown, see figure 3.2. Also a legend is shown in the upper part. In this figure the values in the left figure are illustrated in numbers of photons per seconds, square

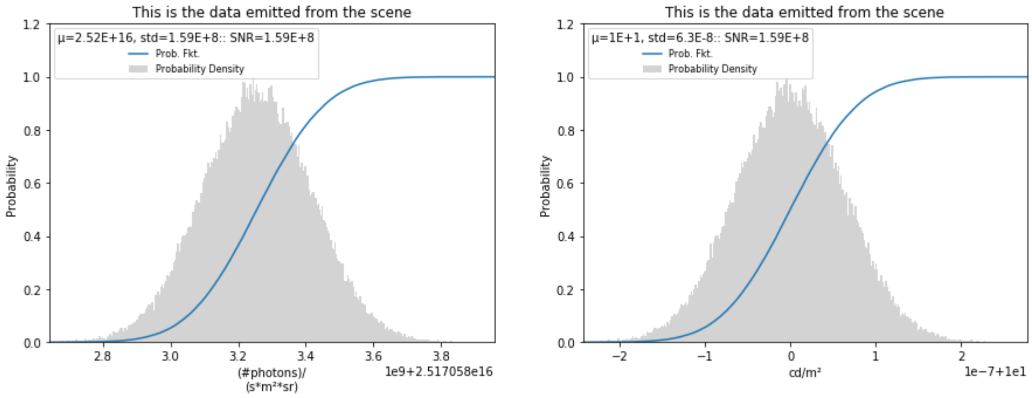


Figure 3.2.: Program output: A generated scene with 10 cd/m^2 intensity and one signal

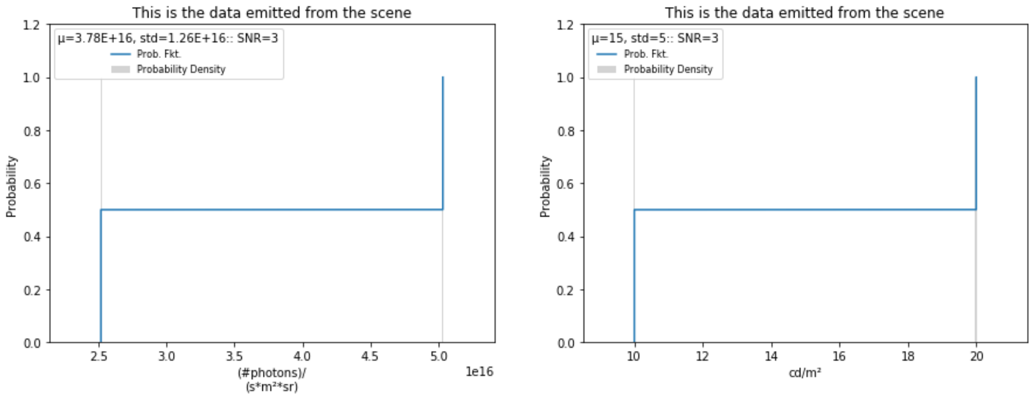


Figure 3.3.: Program output: A generated scene with 20 cd/m^2 intensity

meter and steradian with processing through the imaging chain this domain changes. The values in the right figure are in cd/m^2 for every step in the imaging chain.

Class C_Scene

To go step by step through the imaging chain the class C_Scene is the next to explain. In this class a scene will be generated with a given intensity. With this intensity the light beams which are existing in the real world will be calculated by a mathematical approach with considering the wavelength, speed of light and the Planck's constant.

Also the generation process includes that the light is following a Poisson process because the function for generating a Poisson distribution is limited. It is used an approximation for values larger than 10^3. After the function is done all results will be stored in the data type of class C_IOData. In figure 3.3 the values of a scene with 100% contrast is shown this contrast references to figure 2.3 (a). It is not visible that the light follows a Poisson process. Normally there should be more different peaks but the deviation is very low so the differences are not visible. To show the Poisson process a scene with one signal is plotted in figure 3.2.

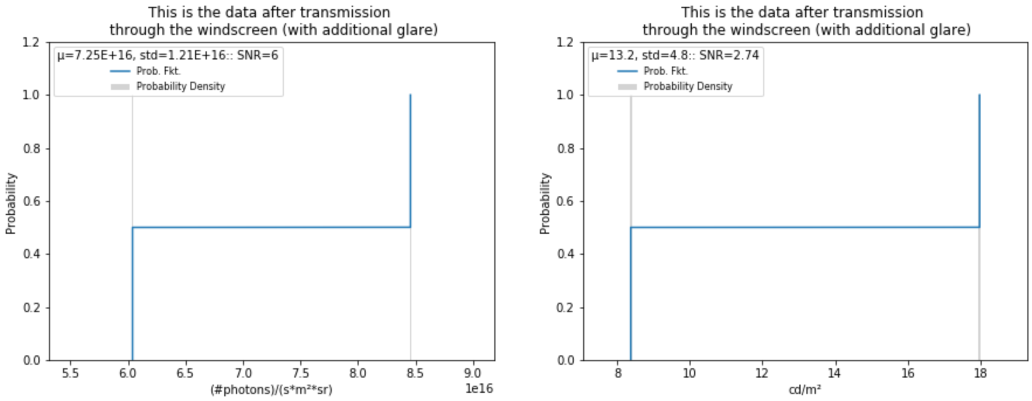


Figure 3.4.: Program output: Scene after processed through the windscreen

Class C_Windscreen

After a scene is generated it transmits through the windscreen. Therefore a class C_Windscreen is part of the program. In this class veiling glare in percent will be added to the data and the data will be processed through the windscreen. This means that the losses caused by transmittance through the windscreen are subtracted from the signal. The transmittance and the veiling glare can be defined by the user. At comparing the right figure of 3.4 with the right figure of 3.3 it is visible that both values have changed. But a bigger changed happened in the left plot of figure 3.4 compared with the left figure of 3.3. This difference comes through the added veiling glare.

Class C_Optics

The next part of the imaging chain is the optic of the camera system. The input data will be transmitted through the optics. For the transmission an easy optic model is used. This optic model is described in equation 3.1 [5, p. 99].

$$E = \frac{t \cdot \pi \cdot \cos^4(\Theta)}{4 \cdot f_{\#}{}^2(1 + m_l)} \cdot L \tag{3.1}$$

By the user the f-number and the transmittance can be defined.

There is a small difference to the output data of the windscreen visible because this optic model allows a nearly perfect back transformation in the physical domain see figure 3.5. Also the domain of the left figure has changed to number of photons per seconds and square meter. The values in the left figure are different because of the processing through the optic and there is no conversion back to the input data done.

Class C_Sensor

After the optic model is processed the image sensor is up next. Before processing

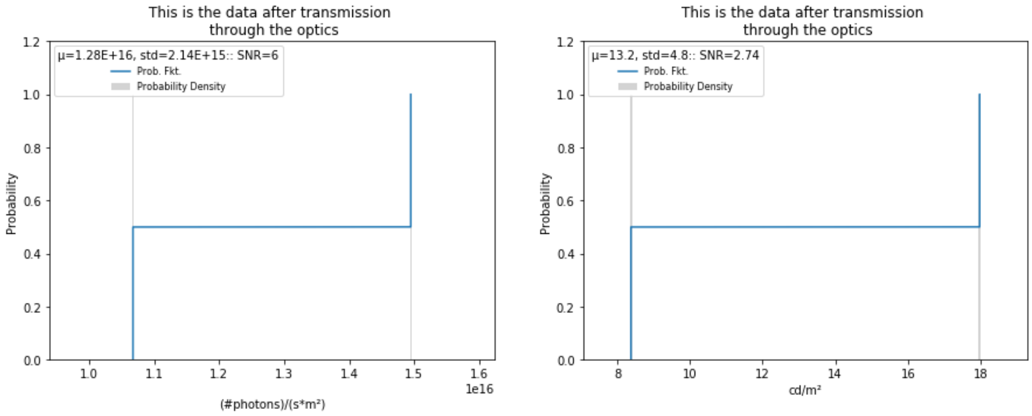


Figure 3.5.: Program output: Data after processed through the optic

the data all variables of the sensor have to be defined. The variables are the pixel pitch for each pixel in meters, the quantum efficiency in percent, the full well capacity in electrons, the system gain called K-Factor, the temperature in °C, the mean dark current for each pixel in electrons, the row wise dark current in electrons, the column wise dark current in electrons, the doubling temperature in °C, the analog digital conversion bits, the image signal processing bits, the size of the sensor with the number of pixel in x- and y-direction and the overall system gain. Of course there are more variables which could be included into a sensor model like a split pixel model. If necessary the variable could be extended for specific use.

Neither these variables processes are implemented in this class. They are done inside the sensor these processes include optical electronic conversion where the photons will be converted in an electrical signal given in electrons see equation 3.2. After the OEC the unity has changed to electrons.

$$\text{eletrons} = \text{input} \cdot (\text{pixel pitch})^2 \cdot \text{exposure time} \cdot \text{quantum efficiency} \tag{3.2}$$

After that a dark current will be added to the signal. The dark current in this model consist of three different random Poisson processes they are one for each pixel, each row and each column. The dark current depends on exposure time and temperature. The temperature and exposure time dependency is shown in equation 3.3[3]. The exposure time will be multiplied to the result.

$$\mu_{dark\_temp} = \mu_{dark} \cdot 2^{\frac{T-T_{ref}}{T_{doubling}}} \cdot \text{exposure time} \tag{3.3}$$

Also a fixed pattern noise is added which is a standard deviation for each pixel, row and column. Then the program checks if the capacitor size is reached everything above

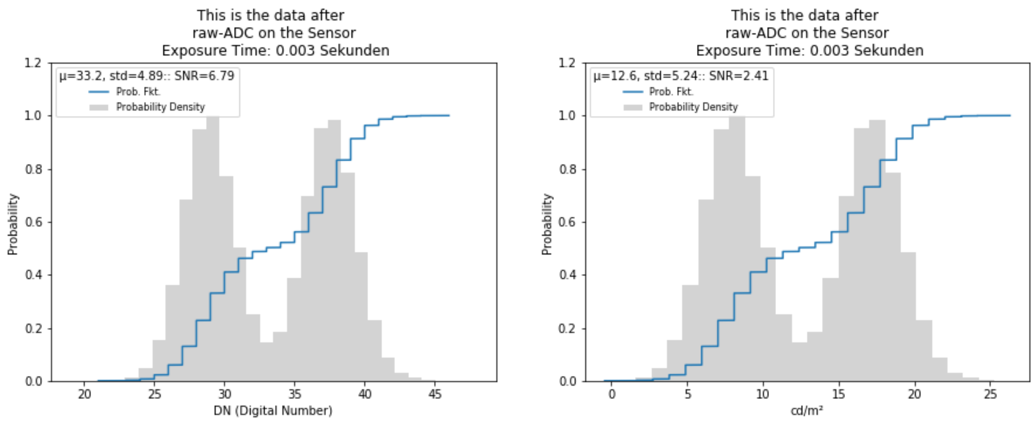


Figure 3.6.: Program output: Imager output with a 3 ms exposure time

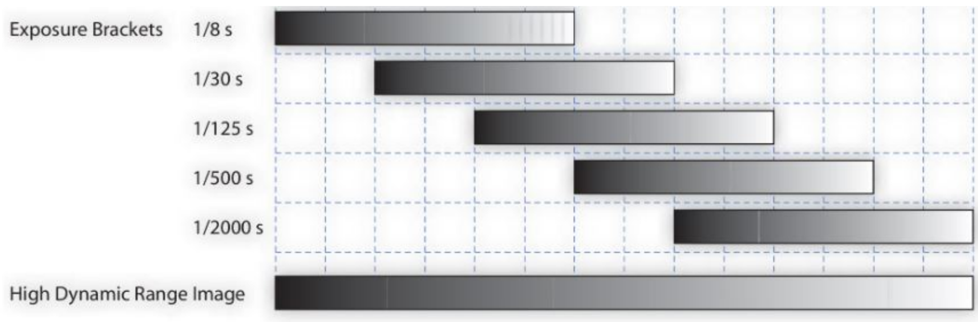


Figure 3.7.: Construction of a HDR image [1, p.108]

this border will be cut this process is called clipping. At the end the electrons will be processed into digital numbers (DN) by multiplying with the system gain called K-factor according to EMVA 1288[3] . Also there is a function which combines the above described functions into one function. The output plot for a sensor is shown in figure 3.6. The unity of the x-Axis has changed to DN because it is the output created by the image sensor. Now the signal is not a single balk anymore. It has changed its shape caused through the dark current and because the sensor is able to convert the emitted photons more exact than they are visible in figure 3.5.

Also a function to create a HDR image is implemented. A HDR image is created by using different exposure times for example five different see figure 3.7. HDR images are created for a better display of the high dynamic range appearing in the world. A single image with 8 bit range is able to handle a dynamic range of 48 dB. But in reality most scenes have a range of 80dB. The different exposure times will be combined to one picture.

In figure 3.7 is the creation of the implemented HDR algorithm shown. The brightest pixel will be taken from the shortest exposure time. And the darkest parts will be taken from the longest exposure time. All these values will be optimized for one

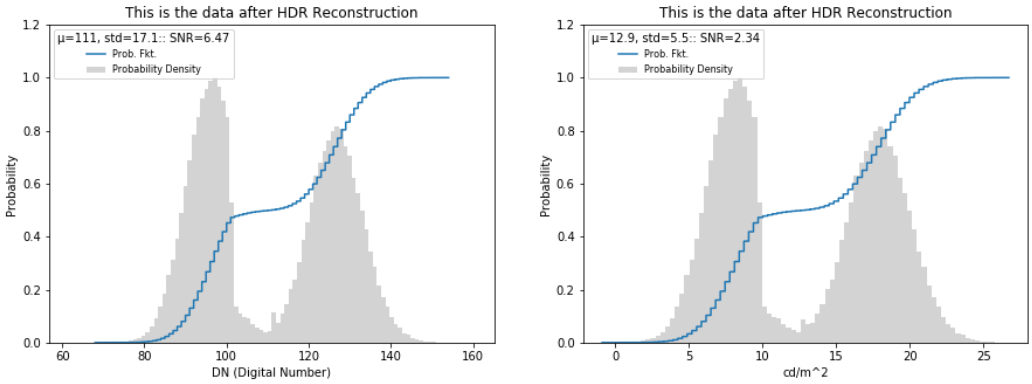


Figure 3.8.: Program output: HDR image of the described image

exposure time by a linear transformation to the longest exposure time. In the figure 3.8 it is visible that the peaks are different this is produced by the HDR algorithm. The values in the right figure are not changing much because it is still the same scene.

Class C_ISP

After the data is processed through the sensor the data will be processed in an ISP. There are lot of options that can be done by an ISP for example different tone map algorithms or an interpolation between the neighboring pixels. In this implementation there is a tone mapping implemented. This tone mapping resizes a HDR image with a big range, range in ISP bits, to an image with a defined tone curve out bits. In the figure 3.9 the output of an image is compressed from an 20 bit image to an eight bit image. The equation 3.4 shows the used algorithm. This causes a discretization of the data which gets visible through the fact that less different values are appearing in figure 3.9 compared to 3.8. After the discretization the highest possible digital number is 255 which corresponds to an eight bit image before the tone mapping the highest possible number was 2^{20}-1.

$$\text{data after ISP} = \log(1 + \text{data}) \cdot 2^{\frac{\text{tone curve out bits}}{\log(2^{\text{ISP bits}})}} \tag{3.4}$$

These eight bit represents the size of separate images used for the HDR image shown in figure 3.8. Where every single image has a low dynamic range and the created image has a high dynamic range.

Class C_Image_Quality

That are all classes related to the imaging chain. But there are also some other classes with functions. One of these is the class C_Image_quality. There are functions implemented to evaluate the CDP and the SNR to compare the results with each other. The CDP evaluation is possible for a Pixel to Pixel evaluation and a ROI to

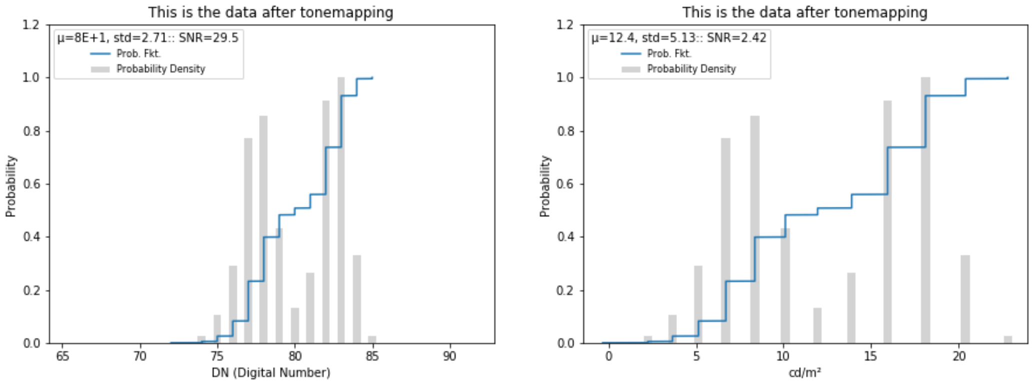


Figure 3.9.: Program output: Tone mapped eight bit image of the described scene

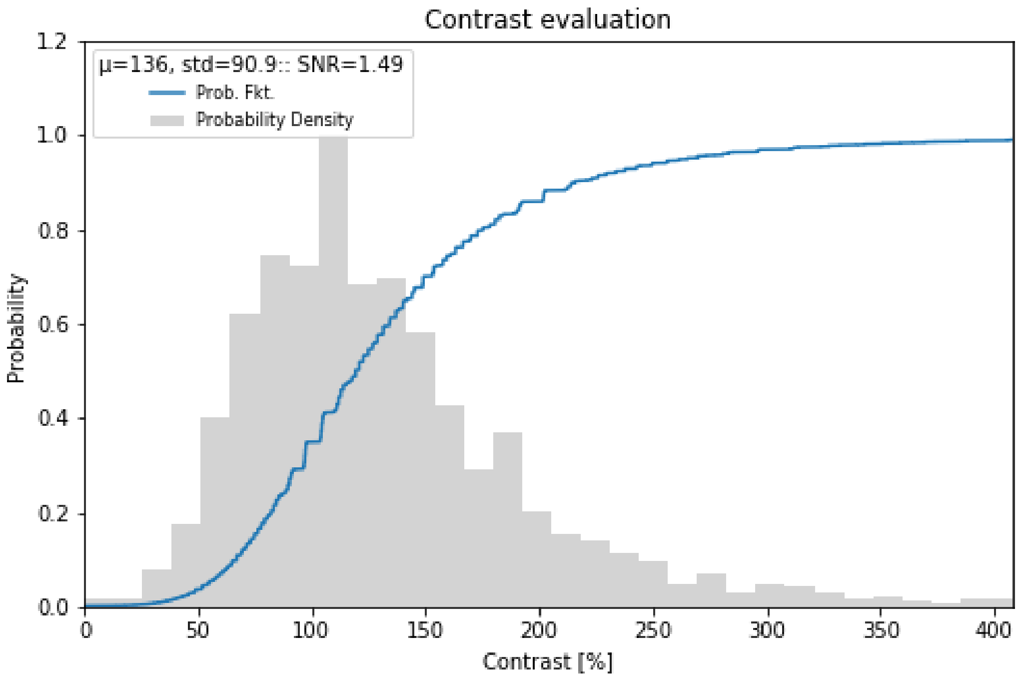


Figure 3.10.: Program output: All occurring contrasts of the image

ROI comparison it will return a float between zero and one and it returns a plot where all calculated contrasts of the image occur already shown in section 2.3. The pixel to pixel evaluation only makes sense when doing a simulation. That is the only use case which makes sense. In figure 3.10 an example of these charts is plotted. This chart follows the illustration like the charts before only the unity of the x-axis has switched to contrast. The float equals the CDP and will be calculated like described in chapter 2.3. For example in this case an epsilon of 0.5 could be used. The K_{in} in figure 3.10 is 100% so the confidence interval is from 50% to 150% with epsilon = 0.5. So the CDP value can be calculated like explained in section 2.3. The program output for this example is 0.66630.

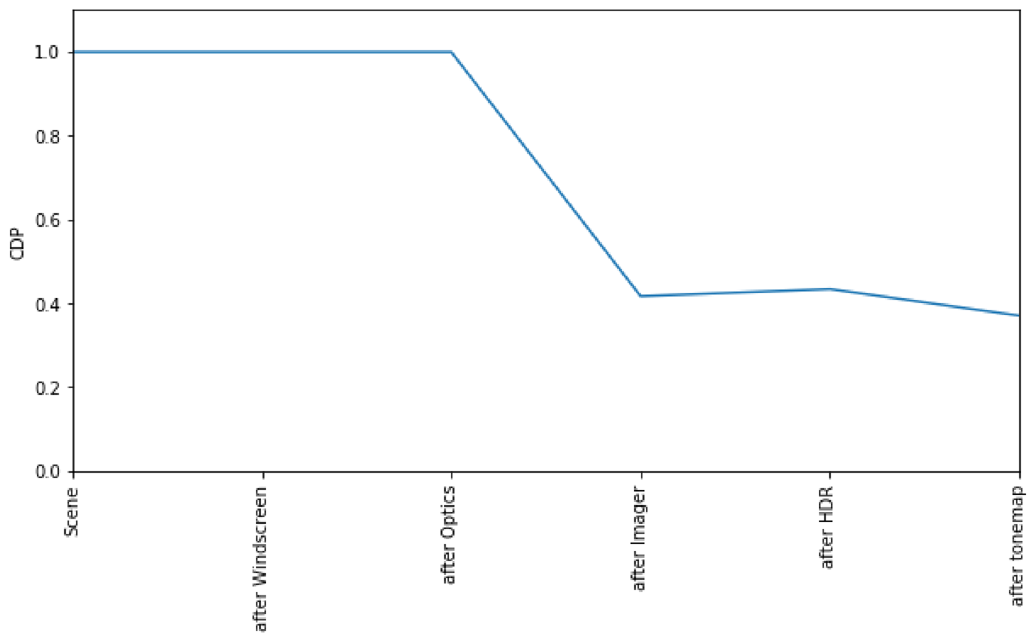


Figure 3.11.: Program output: CDP progress of the imaging chain

Also the possibility exists to make an evaluation over the whole imaging chain to see how the components of the imaging chain affects the KPI's. Examples for these plots are shown in figure 3.11 and 3.12. The display of figure 3.12 is the same as showed in section 2.2. The figure 3.11 is how the CDP behaves over the imaging chain.
One other function is to focus on one specified ROI in the image and evaluate the CDP in this ROI. It is the same function like the CDP function the only difference is that this function concentrates on a specific region of the image. This region can be defined in the program. The output equals the output of the CDP function. This function is useful because CDP is affected by hot and cold pixel and the dark current specific for each pixel. From this follows that the CDP varies for different regions of the sensor and different sensors from the same type of sensor because of production variation. With a bigger ROI this variances could reduce to zero. But still there could be effects like shading, is an effect which appears at the borders of the sensor, or a fixed pattern noise which varies the CDP.

3.2.2. CDP evaluation with Images

Also real world images should be included to evaluate the CDP of these images. Therefore some functions are implemented. The application file for this part is Image_CDP_evaluation. In this part the functions from Image_Processing are used. In the program are implemented the following functionalities:

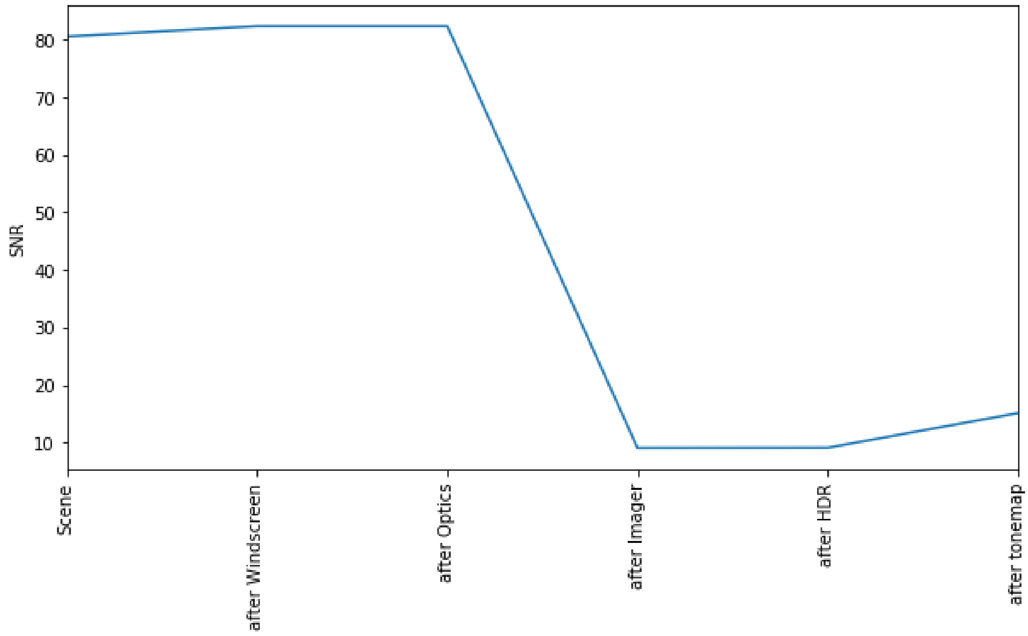


Figure 3.12.: Program output: SNR progress of the imaging chain

- Load an image and convert it to the data type C_IOData

- Recalculation of the image to the physical domain

- Convert an array to an image

- Draw two different ROIs

To load an image it has to be saved to the folder Images to evaluate and the function has to be called. Then it will be converted to a data structure C_IOData which fits the requirements of the program. A recalculation part for the images to transform it to the physical domain. Therefore the specification of the camera has to be defined in the program.
To test these function it was made a experiment. Therefore a camera from type Manta G-505C serial number 503363706, a Kowa LM6NCL lens, a 1000% checkerboard, mounting equipment and a optical table were constructed like shown in figure 3.13. Because luminance is the used domain it is not important to have the distance between the objects.
The luminance of the four defined regions shown in figure 3.13 through the small rectangles were measured with the Luminance Meter Minolta LS-110 serial number 73323007. Afterwards the taken image from the camera was loaded to the function and the luminance of the regions was calculated by the program. The measured results were not the same like the calculated results. This was caused by missing parameters

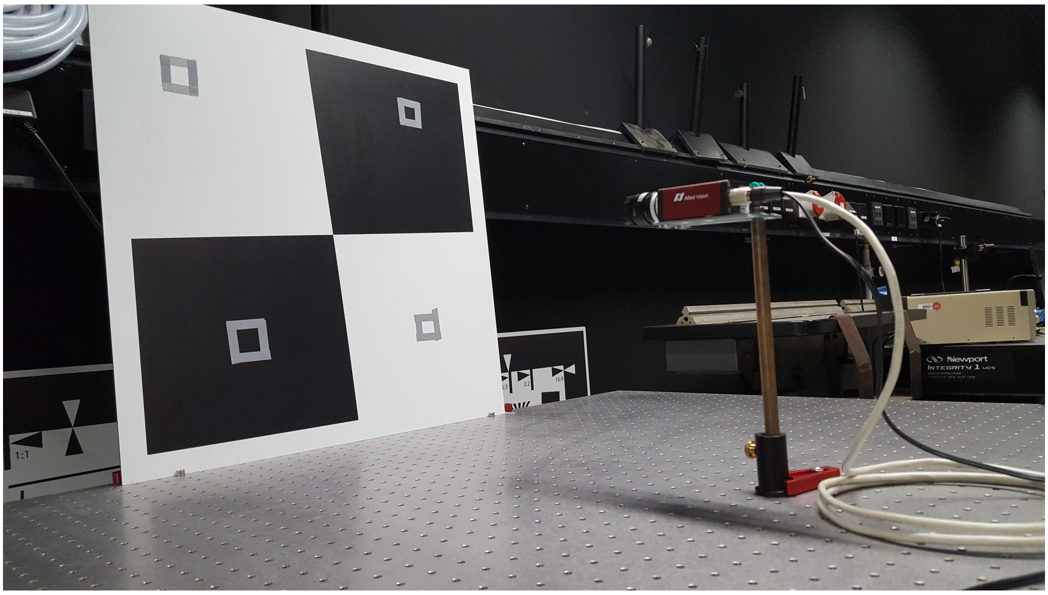


Figure 3.13.: Experimental construction

of the camera specification. The results of the measured luminance and the calculated are shown in table 3.1. Right and left are defined when standing with the face to the checkerboard. It is visible that there is a difference between the results so this can be an approach and no perfect calculation. Most of the parameters of the camera were estimated. To make an good calculation it is necessary to have nearly the whole definition of the used camera system by hand so the calculated luminance will be more exact.

Position of rectangle	Measured in cd/m^2	Calculated in cd/m^2
Upper right	75.6	100.4
Upper left	356	542.72
Lower right	33.6	48.7
Lower left	412	570

Table 3.1.: Measured and calculated luminance of the checkerboard

It is also possible to plot the picture as a three dimensional plot the function is called Plot3D. The generated chart is an html data type to show this data type a suitable program has to be installed for example Internet Explorer, Firefox or others.
With this specified data all functions from the class image quality can be used. In the file Image_processing a function to process simulated data into an image to show the evaluated contrast is implemented. In figure 2.3 are two of these generated pictures shown. In figure 3.14 it is shown how ROIs can be defined through the program.

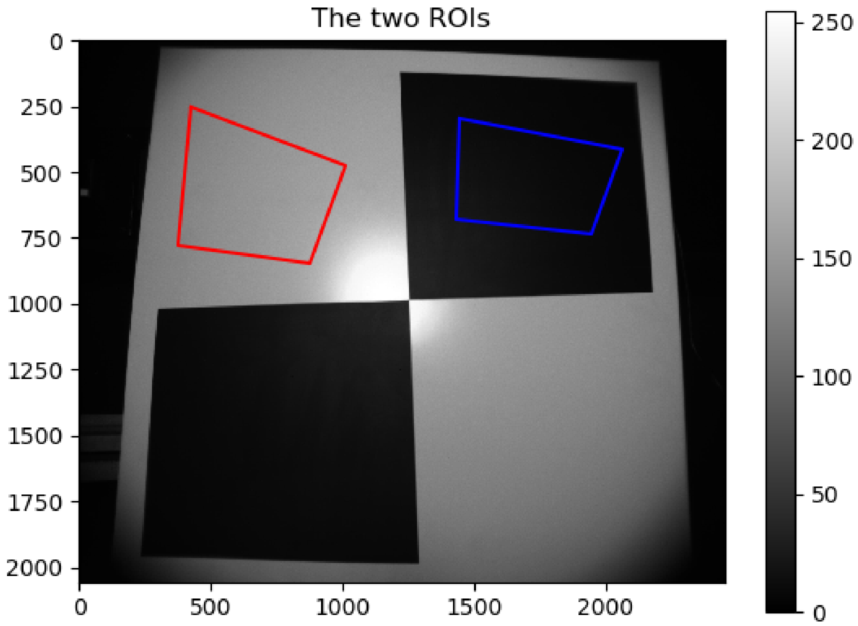


Figure 3.14.: Program output: Define ROI

With the left mouse button the corners of the ROI will be defined and the ROI can be closed by clicking the right mouse button. It is necessary to draw two different ROIs. With theses ROIs the CDP between these ROIs can be evaluated. If the multiplication of the both sizes of lists is smaller than 360 Million. Every pixel will be evaluated to every pixel. Otherwise the smaller ROI will be randomly resized to the size of the bigger one by extending with the data of the array. And the contrast will be evaluated. This resizing is necessary because lists in python are limited through the RAM and the system where python is installed. Also it makes sense because if the ROIs are that big. Noise does not have a big impact on the result of CDP. The CDP evaluation gave a results of 0.712712 for the used parameters of this evaluation.

3.2.3. Video CDP evaluation

Also a python application exists to split a video in frames and evaluate the CDP for every frame. The video has to be stored in the folder Videos to evaluate. The frames will be also stored in this folder in a new created sub folder named like the video. The results of the CDP evaluation of these frames will be stored in a xls sheet in the folder XLS results. In the frame evaluation the pixel to pixel evaluation is used because the implementation of the ROI to ROI evaluation is difficult. This is caused by the movement of the objects in the video. Which leads to the consequence that the evaluated ROIs have to be consequently new defined in each frame. There are already algorithms to find the regions in a video but most of them are not public and it was not possible to develop a algorithm. Also a basis to evaluate the CDP to is needed which makes the CDP evaluation in a video more complicated because a reference measurement with a luminance camera is needed. This makes a experimental construction more complex.

4. Application and limits of CDP

It was planned to use CDP as a metric to evaluate the quality of a new defined simulation model for lenses. But there occurred a lot of challenges which have to be solved.

One challenge is to define the correct region where the evaluation has to be done because in a single image or frame it has to be a ROI to ROI evaluation. A pixel to pixel evaluation in an image would not measure CDP but something else which has nothing in common with CDP. This was caused by a misunderstanding of CDP because it was assumed that CDP is a Pixel to Pixel evaluation but after talking to the author of the CDP paper it was clear that CDP is a ROI to ROI evaluation.

To do this the idea was to use a superpixel or contour detection but then the CDP depends on this model. Then every supplier would implement their own superpixel or contour detection model. This will cause problems because it is not possible to compare the CDP from camera system A with the CDP of camera system B because they could have used different models. So the results are not comparable because one supplier could have a better contour detection than camera B in view of CDP but a worse camera specification in view of CDP. The other one could have it the reversed way but the CDP result of whole system could be the same. Also a superpixel or a contour detection model is like introducing a threshold to make it easier to measure. And once a threshold is introduced other suppliers start to introduce their own threshold. So after a small time of practicing CDP there are a lot of different threshold and the idea of CDP has gone.

It makes sense to use CDP in an ideal environment where every variable is known and manageable to calculate a more accurate CDP value and a value which is comparable to other components. The ROI can be defined accurate and also other aspects like for example veiling glare are under control. This is important because CDP is influenced by a lot of variables. Also CDP is only valid for the one investigated component due to production variances. But it is expectable that the CDP does not varies in a big range for the production series. With a big number of different samples the evaluated production series could be defined in case of CDP like a normal distribution with a mean of 0.75 and deviation of 0.01 under the given surroundings. This is useful to

define the requirements which have to be fulfilled by the supplier.
The next challenge is to have a basis to evaluate the CDP to. In the simulation a contrast is defined and it can be simulated nearly perfect and the defined scene shows only this contrast. But in real world there are lots of variables to take into account to get the original contrast of objects because the contrast is influenced by the color of objects and the pollution of the object. Also the contrast of a lot of objects is not the same contrast like in a clean environment. For example a dirty traffic sign doesn't have the expected value like a clean one also other variables influence the contrast of the traffic sign for example the age, date of production and others. So it would be useful to take an image with an luminance camera to get the correct contrast to evaluate to. If no image with a luminance camera is taken the evaluation will look at this defined and expected contrast and because of the defined confidence interval the CDP could have the correct value for the scene itself. But the CDP evaluation of the processed image of the scene and the processed simulation will not be logical if both are compared to each other. Because it is expectable that both values should be nearly the same. But the confidence interval has to be shifted in case of the taken image because the evaluation is looking at the wrong basis of the contrast.
After the implementation of an image evaluation the same was done for a video. But there are occurring the same challenges like they are occurring in a single image. Also it is more complicated to define the evaluated ROI because of the movement of the car the ROI is changing every frame. This challenge is already solved in other use cases but they are not implemented in this program. Also there the CDP value depends of the quality of these recognition algorithms. One other concern is to have basis to evaluate to. Which makes the measurement more complicated because normally it would require a measurement with a luminance camera to have a basis.

5. CDP simulation with different parameters

In this chapter CDP will be simulated with different parameters to show their influences on CDP. Therefore the program has been simulated with two different contrasts and the variables around these contrast have been changed differently to show the effects on CDP.

Parameter	Values	Values
Contrast	30 %	100 %
Intensity	1 cd/m^2	10 cd/m^2
Temperature	50 °C	100 °C
Exposure time	10 ms	15 ms
Pixel Pitch	2 μm	3 μm

Table 5.1.: The different Parameters of CDP evaluations

In table 5.1 are shown the different evaluated parameters and in table 5.2 are the parameters which did not change. Of course it would make sense to show the CDP in dependency on every variable but that would be a high amount of different settings. To show the generated effects on CDP for the above parameter different settings have been simulated. Every result is not 100% reproducible because in the program are different random poisson processes which generate different data for each function call so the results will vary. It have been done different evaluation to show the variance between the measurements. It were made fifteen samples the mean was 0.681248141 with a standard deviation of $3.51974 \cdot 10^{-5}$ for the third setting in table 5.3.

Parameter	Value
Windscreen transmittance	0.96
Veiling glare percentage	100 %
F-number	2
Optic transmission	0.9
Quantum efficiency	0.7
Full well capacity	15000 electrons
K-Factor	0.25
Row wise dark current	5 electrons
Column wise dark current	10 electrons
Pixel wise dark current	35 electrons
Doubling Temperature	10 °C
ADC bits	12
ISP bits	20
epsilon	0.5

Table 5.2.: The general used setting

The settings in table 5.2 have been used for all following evaluations only the mentioned parameters in table 5.1 will differ. The used parameters are already explained in section 3.2.1. The K-Factor and the doubling temperature have been used according to the EMVA 1288. All these simulation have been done from scene to imager like shown in figure 2.1. This includes scene, windscreen, optics and imager.

With the result in table 5.3 and comparing the fourth with the fifth it is visible that smaller contrasts are more difficult to detect because most of the times CDP is lower than for higher contrasts. This is caused by the smaller difference between the ROIs. The main reason for this behavior is that the dark current is influencing the contrast more than it would do with higher contrasts. Also it is visible that the dark current effects the CDP a lot for example the CDP decreases if a higher dark current is added to the signal. In this example the higher value of the dark current is simulated through a higher temperature which causes a more dark electrons because the dark current depends on the temperature. This effect is described in this program by equation 5.1[3]. For example compare the first to the second result.

$$\mu_{dark\_temp} = \mu_{dark} \cdot 2^{\frac{T - T_{ref}}{T_{doubling}}} \tag{5.1}$$

Parameter	Value	Value	Value	Value	Value	Value	Value	Value
Contrast	100 %	100 %	100 %	100 %	30 %	30%	30%	30%
Intensity	1 cd/m^2	1 cd/m^2	10 cd/m^2	1 cd/m^2	1 cd/m^2	1 cd/m^2	1 cd/m^2	10 cd/m^2
Temperature	50 °C	100 °C	100 °C	100 °C	100 °C	100 °C	50 °C	50 °C
Exposure time	10 ms	10 ms	10 ms	10 ms	10 ms	15 ms	15 ms	15 ms
Pixel Pitch	2 μm	2 μm	2 μm	3 μm	3 μm	3 μm	3 μm	3 μm
CDP	0.8287198	0.1523090	0.6816225	0.2325489	0.15827843	0.1578064	0.8420468	0.99834205

Table 5.3.: Different CDP evaluations

So the CDP would not increase just by decreasing the dark current the temperature has to stay at a comparable level. It is possible to decrease the temperature to optimize the CDP. One other effect caused by the dark current is that just increasing the exposure time does not automatically increase the CDP, see results five and six, because the dark current also depends on the exposure time so the signal is still distorted.

But if the scene has a higher intensity the CDP increases a lot because then the dark current and other disruptive factors does not have a big influence on the CDP see results of the measurements two compared to three. But if the sensor reaches its full well capacity the CDP value falls to zero because all contrasts haven been gone.

The CDP can be optimized by increasing the pixel pitch compare the second to fourth evaluation. But normally with a bigger pixel pitch the dark current specific for each pixel increases. This is caused because the materials have a bigger surface area which causes new dark electrons.

6. Summary

At the beginning of this thesis an exemplary imaging chain was explained. In this thesis the imaging chain exists of a windscreen, an optic, an imager, and an ISP. In every part of the imaging chain are done different processing steps for example in the imager an optical electronic conversion or clipping can be done. The functionality of these steps have also been described.

The thesis focuses on CDP so it was explained what the definition of CDP means. The definition is shown in equation 6.1.

$$CDP_{K_{in}} = Prob(K_{in}(1-\varepsilon) \leq K_{meas.} \leq K_{in}(1+\varepsilon)) \tag{6.1}$$

It is also important that all the used contrasts are given as Weber contrast. The result of CDP will be between zero and one.

The main part of this thesis is about the implementation of a program to evaluate the CDP on taken images or simulate the CDP with given camera specifications. It exists of different classes these classes represent all mentioned parts of the imaging chain, a class for image quality and classes for video and image processing. Also there are three application files whose show how to use the written functions and how the display of the output looks like.

Also it was mentioned that CDP is a ROI to ROI evaluation. This is important to know because some specific use cases are not possible to implement in an useful way. So an evaluation with a video is difficult to implement because the interesting ROIs are changing in a driving scene every frame. Of course it is already possible to do this. But this was not part of this thesis. The most problematic thing about this is that it has to be done a measurement to have a basis contrast to evaluate to. This would led to a very complicated experimental set up.

After this was done a CDP evaluation for different parameters was done to show the effects of these parameters on CDP. One conclusion was that higher contrast are easier to detect because the effects of the dark current and other disruptive factors do not have a big impact on the picture because the contrast difference between the ROIs is big enough to not led the measured contrast fall out of the confidence interval. One other conclusion was that the dark current does affect the CDP a lot and is one main reason for very low CDP because CDP is a noise influenced KPI. Also it was visible that longer exposure times does not led to an automatically increased CDP. If the values of the signal and dark current are similar the CDP will stay at the same value. There have already been some talks about CDP which show the positive things of CDP for example there have been some at the Autosens [6][10]. The developing will go further and it will be introduced with the Project P2020 of the IEEE as an useful KPI.

A. Code export from Python

Appendix 1: CDP Simulation

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 22 09:34:58 2018

@author: lueb5102
to run this programm PIL(image, install with "install pillow"), cv2,
matplotlib, numpy and xlwt have to be installed on the system
"""

from PIL import Image
import numpy as np
import functions.C_IOData as Data
import functions.C_Scene as Scene
import functions.C_Windscreen as Windscreen
import functions.C_Optics as Optics
import functions.C_Sensor as Sensor
import functions.C_ISP as ISP
import functions.C_Image_quality as IQ
import functions.C_Test as Test
import functions.Video_processing as Video_processing
import functions.Image_processing as Image_processing
import matplotlib.pyplot as plt


###############################################################################
"""This part of the program is a simulation over the whole imaging chain
there is no real world input"""

"""Evaluation for different illumination scenarios.
To show the dependancy of the Scene intensity"""

```

```
"""scene_intensities_patch = np.logspace(-3,9,200)
CDP = np.zeros_like(scene_intensities_patch)
CDP_scene = np.zeros_like(scene_intensities_patch)
SNR = np.zeros_like(scene_intensities_patch)
for scene_intensity, idx in zip(scene_intensities_patch, range(len(
scene_intensities_patch))):
    Samples = int(126420)
    Test_Scene = Scene.C_Scene(scene_intensity, Samples)
    Output_Scene = Test_Scene.get_Output()
    for i in range (0, Samples):
        if i % 2==0:
            Output_Scene.data [i] = 1*Output_Scene.data[i]
            Output_Scene.cd_m2 [i] = 1*Output_Scene.cd_m2[i]
        else:
            Output_Scene.data [i] = 2*Output_Scene.data[i]
            Output_Scene.cd_m2 [i] = 2*Output_Scene.cd_m2[i]

    Test_Windscreen = Windscreen.C_Windscreen( 20,
                                              0.96)
    Windscreen_Output = Test_Windscreen.get_Output(Output_Scene)
    Test_Optic = Optics.C_Optics(2, 0.9)
    Optic_Output = Test_Optic.get_Output(Windscreen_Output)

    exposureTime_s1=3e-3
    exposureTime_s2=5e-3
    exposureTime_s3=7e-3
    pixel_pitch_m=2e-6
    quantum_efficiency=0.7
    full_well=15000
    K=0.25
    temp=100.
    mu_dark=35.
    row_dark_current = 5
    column_dark_current = 10
    doubling_temp=10
    ADC_bits=12
    ISP_bits = 20
    number_of_pixel_x = 301
    number_of_pixel_y = 420

```

```
    Test_Sensor = Sensor.C_Sensor(
                pixel_pitch_m,
                quantum_efficiency,
                full_well,
                K,
                temp,
                mu_dark,
                row_dark_current,
                column_dark_current,
                doubling_temp,
                ADC_bits,
                ISP_bits,
                number_of_pixel_x,
                number_of_pixel_y)
    Imager_Output = Test_Sensor.get_Output_with_exposureTime(Optic_Output,
                                                             exposureTime_s2)

    CDP[idx], Contrast = IQ.C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                                        Imager_Output.cd_m2, 100, Test_Sensor)
    CDP_scene[idx], Contrast = IQ.C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                                        Output_Scene.cd_m2, 100, Test_Sensor)
    SNR[idx] = IQ.C_Image_quality.evaluate_SNR_Array(Imager_Output.cd_m2)


plt.figure()
plt.semilogx(scene_intensities_patch, CDP,CDP_scene)
plt.title('CDP')
plt.figure()
plt.semilogx(scene_intensities_patch, SNR)
plt.title('SNR')
"""
###############################################################################
"DefinitionsScene"
Test_Intensity= 20#5e15
Samples = int(126420) #(because the defined Sensorsize is 301*420= 126420)
Test_Scene = Scene.C_Scene(Test_Intensity, Samples)
Output_Scene = Test_Scene.get_Output()

#100% Contrast Checkerbox contrast can also be changed to other contrasts
for i in range (0, Samples):
```

```
    if i % 2==0:
        Output_Scene.data [i] = 1*Output_Scene.data[i]
        Output_Scene.cd_m2 [i] = 1*Output_Scene.cd_m2[i]
    else:
        Output_Scene.data [i] = 0.5*Output_Scene.data[i]
        Output_Scene.cd_m2 [i] = 0.5*Output_Scene.cd_m2[i]

Output_Scene.doPrint()


################################################################################
"Defnition windscreen"
Glare_Percentage = 100
#subtraction of the glare photons (incl their uncertainty) decreases the SNR
#of the target quantity (e.g. the SNR in cd/m^2)
Windscreen_Transmittance = 0.96
Test_Windscreen = Windscreen.C_Windscreen( Glare_Percentage,
                                         Windscreen_Transmittance)
Windscreen_Output = Test_Windscreen.get_Output(Output_Scene)
Windscreen_Output.doPrint()


################################################################################
"Definition optics"
F_number = 2
Optic_Transmission = 0.9
Test_Optic = Optics.C_Optics(F_number, Optic_Transmission)
Optic_Output = Test_Optic.get_Output(Windscreen_Output)
Optic_Output.doPrint()

################################################################################
"Definition sensor"
exposureTime_s1=3e-3
exposureTime_s2=15e-3
exposureTime_s3=7e-3
pixel_pitch_m=2e-6
quantum_efficiency=0.7
full_well=15000
K=0.25
temp=100.
```

```
mu_dark=35.
row_dark_current = 5
column_dark_current = 10
doubling_temp=10
ADC_bits=12
ISP_bits = 20
number_of_pixel_x = 301
number_of_pixel_y = 420

Test_Sensor = Sensor.C_Sensor(
                pixel_pitch_m,
                quantum_efficiency,
                full_well,
                K,
                temp,
                mu_dark,
                row_dark_current,
                column_dark_current,
                doubling_temp,
                ADC_bits,
                ISP_bits,
                number_of_pixel_x,
                number_of_pixel_y)

Test_class_Sensor = Sensor.C_Sensor(
                pixel_pitch_m,
                quantum_efficiency,
                full_well,
                K,
                temp,
                mu_dark,
                row_dark_current,
                column_dark_current,
                doubling_temp,
                ADC_bits,
                ISP_bits,
                number_of_pixel_x,
                number_of_pixel_y)
'Above one image sensor and now a new one which is defined in the classe C_Test'
Test_own_class = Test.C_Test()
```

```
Output = Test_own_class.get_Output_with_exposureTime(Optic_Output,
                                                     exposureTime_s2)
Output.doPrint()

###############################################################################
'Images with single exposure time'
Imager_Output = Test_Sensor.get_Output_with_exposureTime(Optic_Output,
                                                         exposureTime_s2)
Imager_Output.doPrint()
Imager_Output3 = Test_Sensor.get_Output_with_exposureTime(Optic_Output,
                                                          exposureTime_s1)
Imager_Output3.doPrint()
image2 = Image_processing.Array_to_image(Imager_Output.data,
                                         Test_Sensor.number_of_pixel_x,
                                          Test_Sensor.number_of_pixel_y)

'Resizing of the image to see the checkerboard'
x, y = image2.size
newsize = x * 4, y * 4
image3 = image2.resize(newsize, resample =Image.NEAREST)
image3.show()

###############################################################################
"Building a HDR image with n-exposure times wih the sensor Test_own_class"
Mode = "Linear"
Output_HDR_linear = Test_own_class.get_HDR_Image(Optic_Output,
                        [exposureTime_s2, exposureTime_s1, exposureTime_s3],
                        Input_Mode = Mode)
Output_HDR_linear.doPrint()

###############################################################################
'Building a HDR image with Test_Sensor'
Mode = "Normal"
HDR_Output_normal = Test_Sensor.get_HDR_Image(Optic_Output,
                        [exposureTime_s1, exposureTime_s2, exposureTime_s3],
                        Input_Mode = Mode)
HDR_Output_normal.doPrint()

###############################################################################
"Definition ISP"
```

```
Defined_ISP = ISP.C_ISP(Test_Sensor)

################################################################################
"Do tonemapping"
Tonemap_Output = Defined_ISP.tonemap(HDR_Output_normal)

'Produce an image with the tonemapped numbers'
image2 = Image_processing.Array_to_image (Tonemap_Output.data,
                                          Test_Sensor.number_of_pixel_x,
                                          Test_Sensor.number_of_pixel_y )

'Resizing the image to get a better display of the image'
x, y = image2.size
newsize = x * 4, y * 4 #to make the checkerboard visible by increasing the pixel
image3 = image2.resize(newsize, resample =Image.NEAREST)
image3.show()
Tonemap_Output.doPrint()

################################################################################
"Evaluate contrast of the defined scene after the imager"
Contrast_to_evaluate = 100
CDP, Contrast = IQ.C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                                                    Imager_Output.cd_m2,
                                                   Contrast_to_evaluate,
                                                   Test_Sensor)
print ("CDP = ", CDP)
Contrast.doPrint()

################################################################################
"Contrast over the whole imaging chain at 100%"
IQ.C_Image_quality.evaluation_CDP_imaging_chain(
        Contrast_to_evaluate, Input_Sensor = Test_Sensor,
        Input_Scene = Output_Scene.cd_m2,
        Input_Windscreen = Windscreen_Output.cd_m2,
        Input_Optics = Optic_Output.cd_m2,Input_Imager =  Imager_Output.cd_m2,
        Input_HDR = HDR_Output_normal.cd_m2,
        Input_Tonemap = Tonemap_Output.cd_m2)

################################################################################
"SNR evaluation of the whole imaging chain"
```

```
IQ.C_Image_quality.evaluation_SNR_imaging_chain(
        Output_Scene.data, Input_Windscreen = Windscreen_Output.data,
        Input_Optics = Optic_Output.data, Input_Imager =  Imager_Output.data,
        Input_HDR = HDR_Output_normal.data,
        Input_Tonemap = Tonemap_Output.data )

plt.show()
```

Appendix 2: Image CDP evaluation

```
# -*- coding: utf-8 -*-
"""
Created on Wed May  9 10:15:35 2018

@author: lueb5102
to run this programm PIL(image), cv2, matplotlib, numpy and xlwt have
to be installed on the system
In Spyder this part of the program does not run. There is some Problem with
QT Application. It is running correctly by using the cmd of windows.
"""
from PIL import Image
import numpy as np
import functions.C_IOData as Data
import functions.C_Scene as Scene
import functions.C_Windscreen as Windscreen
import functions.C_Optics as Optics
import functions.C_Sensor as Sensor
import functions.C_ISP as ISP
import functions.C_Image_quality as IQ
import functions.C_Test as Test
import functions.Video_processing as Video_processing
import functions.Image_processing as Image_processing
import plotly
import matplotlib.pyplot as plt


############################
'Example real world image'
"""The function opens and writes the data of the Image into an array and shows
the image in 2D"""
image_focus = Data.C_IOData ()
```

```
image_focus.data , number_of_pixel_x , number_of_pixel_y = Image_processing.Image_to_Array(
                                                            'img_focus.png')

"do not run this part if you computer hasn't at least 16GB Ram."
"The image is also resized to 1000 Pixel in y-Direction. This part takes a while."
Image_processing.plot3D (image_focus.data , number_of_pixel_x = number_of_pixel_x ,
                        number_of_pixel_y = 1000)

##############################################################################
' Definition of the spezification  camera'
Camera_Spezifikation = Data.C_IOData()
Camera_Spezifikation.ADCBits = 12
Camera_Spezifikation.Exposure_Times = [5e-3]
Camera_Spezifikation.quatum_efficieny = 0.4
Camera_Spezifikation.sensorsize = 3.45e-6
Camera_Spezifikation.F_number = 2.6
Camera_Spezifikation.K = 1
Camera_Spezifikation.data = image_focus.data
Camera_Spezifikation = Image_processing.DN_to_cd_m2conversion(Camera_Spezifikation)
Camera_Spezifikation.doPrint()

##############################################################################
"Define a ROI"
Point1 = [2100,2000]
Point2 = [100,100]
Contrast_to_evaluate = 1000
Contrast_ROI = Data.C_IOData()

"""There will be a rectangle between the both points , which will also be shown
as a single image"""
CDP_in_ROI , Contrast_in_ROI = IQ.C_Image_quality.evaluate_CDP_ROI (
        Point1 , Point2 , Camera_Spezifikation , Contrast_to_evaluate ,
        number_of_pixel_x = number_of_pixel_x ,
        number_of_pixel_y = number_of_pixel_y)
#Xsize/Ysize of the picture itself the size is defined above
print("CDP_ROI = ", CDP_in_ROI)

##############################################################################
'CDP evaluation for the complete image'
CDP , Contrast = IQ.C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
```

```
        Camera_Spezifikation.cd_m2, Contrast_to_evaluate,
        number_of_pixel_x = number_of_pixel_x,
        number_of_pixel_y = number_of_pixel_y)

print ("CDP = ", CDP)
Contrast.doPrint()

plt.show()

##############################################################################
#this function is not running in Spyder, because there are some problems with the interface
#but it does run with the cmd Editor in windows
"This function gives an interface to define two different ROIs by clicking into"
"the image. Afterwards the CDP between these ROIs is evaluated."
Name_Image = 'img_focus.png'
ROI1, ROI2 = Image_processing.draw_ROI(Name_Image)
ROI1cd_m2 = Camera_Spezifikation.cd_m2_function(ROI1)
ROI2cd_m2 = Camera_Spezifikation.cd_m2_function(ROI2)
CDP_ROI_to_ROI, Contrast_ROI = IQ.C_Image_quality.evaluate_CDP_ROI_to_ROI(
                                                    ROI1cd_m2, ROI2cd_m2, 1000)
print (CDP_ROI_to_ROI)
SNR = IQ.C_Image_quality.evaluate_SNR_ROI(ROI1cd_m2)
print("SNR = ", SNR)

##############################################################################
"Open the image and convert it to an array"
Name_Image = 'img_focus.png'
image_focusdata, number_of_pixel_x, number_of_pixel_y = Image_processing.Image_to_Array(
                                                                    Name_Image)


"""The spezifikation of the camera is defined above. We are now changing the
input data."""
Camera_Spezifikation.data = image_focusdata
Camera_Spezifikation = Image_processing.DN_to_cd_m2conversion(Camera_Spezifikation)
Camera_Spezifikation.doPrint()

CDP, Contrast= IQ.C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
        Camera_Spezifikation.cd_m2, Contrast_to_evaluate, number_of_pixel_x =
        number_of_pixel_x,  number_of_pixel_y = number_of_pixel_y )
```

```
print ("CDP = ", CDP)
#Contrast.doPrint()
plt.show()


##############################################################################
"""Contrast evaluation for a range of contrast with a stepsize which has to
defined"""
CDP_Row, Contrasts = IQ.C_Image_quality.evaluation_from_x_to_y (
        Camera_Spezifikation.cd_m2, Start = 10, Stop = 100, Stepsize = 25,
        number_of_pixel_x = number_of_pixel_x,
        number_of_pixel_y = number_of_pixel_y)

CDP_Row2, Contrasts = IQ.C_Image_quality.evaluation_from_x_to_y (
        Camera_Spezifikation.cd_m2, Start = 10, Stop = 100, Stepsize = 25,
        number_of_pixel_x = number_of_pixel_x,
        number_of_pixel_y = number_of_pixel_y)
print (CDP_Row, Contrasts)
print (CDP_Row2, Contrasts)

##############################################################################
"Write the results above this function in an excel sheet"
CDPs = np.array([CDP_Row, CDP_Row2]) # input of CDP with different Pictures
#with the same evaluated Contrasts
#Contrasts = the evaluated Contrasts of the CDPs above
IQ.C_Image_quality.write_to_xls(Contrasts, CDPs, name_of_sheet = "results")
#returns the results above in a xls datei
plt.show()
```

Appendix 3: Video CDP evaluation

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jun 22 09:36:07 2018

@author: lueb5102
to run this programm PIL(image), cv2, matplotlib, numpy and xlwt have
to be installed on the system
"""
from PIL import Image
import numpy as np
```

```
import functions.C_IOData as Data
import functions.C_Scene as Scene
import functions.C_Windscreen as Windscreen
import functions.C_Optics as Optics
import functions.C_Sensor as Sensor
import functions.C_ISP as ISP
import functions.C_Image_quality as IQ
import functions.Video_processing as Video_processing
import functions.Image_processing as Image_processing

################################################################################
' Definition of the camera spezifikation'
Camera_Spezifikation = Data.C_IOData()
Camera_Spezifikation.ADCBits = 12
Camera_Spezifikation.Exposure_Times = [5e-3]
Camera_Spezifikation.quatum_efficieny = 0.4
Camera_Spezifikation.sensorsize = 3.45e-6
Camera_Spezifikation.F_number = 2.6
Camera_Spezifikation.K = 1

"This is the evaluation of a real world video "
Name_of_Scene = "Snapchat-374723380.mp4"
Sheet_Name = "Video"

'''The results of this function will be written into an xls-sheet with the name
of Sheet_name
'''
Video_processing.Video_CDP_evaluation_from_x_to_y (Name_of_Scene,
                                                    Camera_Spezifikation, Start = 10,
                                                    Stop = 30, Stepsize = 10,
                                                    Name_of_sheet = Sheet_Name)
```

Appendix 4: Class C_IOData

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  3 12:53:55 2018

@author: lueb5102
"""

```

```


import numpy as np
import matplotlib.pyplot as plt
from decimal import *

#Data Class: input and output of each block in the imaging chaine
class C_IOData:
    # Data Contrainer and Printing abilities.

    # Idea: *The data numbers change when the signal passes though the imaging chain.
    #       *The cd/m^2 represents always the best reconstruction of the original
    #           scenes cd/m^2 value
    #           -  it changes as well (e.g. if quantization of the data happens)
    def __init__(self):
        self.info_str = "emtpy info-string"

        # number representation of the current scene:
        self.data = np.array(0)
        self.datatype_str = "invalid"

        # sometimes it is better to recalculate later, therefore use
        self.stochaticProcess = "invalid"
        #self.data_expectValue = 0


        #Backtransformation into the scene intensity given in  cd per square metre in
        self.cd_m2_function = 0
        self.cd_m2 = np.array(self.data)
        #these data is necessary to recalculate from Image Data to cd/m^2
        self.Exposure_Times = np.array(0)
        self.Sensorsize = 0
        self.quatum_efficieny = 0
        self.mu_dark = 0
        self.temp = 25
        self.K = 1
        self.ADCBits = 0
        self.Tonemap_algorithmus = 0
        self.ISP_bits = 0
        self.full_well = 0
```

```
        self.transmission_Optics = 1
        self.transmission_Windscreen = 1
        self.glare_photons = 0
        self.F_number = 0

    def doPrint(self):
        '''
        Prints the plots of the data
        '''
        print(">----- Start ------<")
        print(self.info_str)
        print(">------ End -------<")
        self.get_plot()

    def get_ProbablityFunction(self, Data, SampleMax = 10000):
        '''
        Calculates the Probability Function
        Data: Array-Like. With the input data for the plot.
        SampleMax: Integer. With the number of samples in data.

        returns:
            X_Data: Array-Like. The data for the plot of the x-Axis.
            Prob_Fkt: Array-Like. The data for the plot of the y-Axis.

        '''

        InputData = Data.flatten()
        InputDataCount = len(InputData)
        SampleCnt = np.minimum(SampleMax, InputDataCount)

        print("UsedSamples =", SampleCnt)

        x_Data = np.sort(Data)

        Prob_Fkt = Data

        Prob_Fkt = Prob_Fkt.astype(float)
        size = np.size(Data)

        for i in range (0, size):
```

```
            Prob_Fkt [i] = 1/size

        Prob_Fkt = np.cumsum(Prob_Fkt)

        print("get_ProbablityFunction -Done")

        return x_Data, Prob_Fkt

    def get_DensityFunktion(self, x_Axis, DataIN):
        '''
        Calculates the density function for the input data. Sometimes the
        presentation of the plots is not good enough. Therfore some other solutions
        have been tried but these solutions are not that good and have to be
        fixed. They have been commented off the program.
        x_Axis: Array-Like. Generated from the probability function.
        DataIN: Array-Like. Y-Axis of the probability function.

        returns:
            OutX_Axis: Array-Like. The X-axis data of the density function.
            DensityFkt: Array-Like. The Y-axis data of the density function.
            BarWidth: Array-Like. The width of each bar.

                '''

        '''if np.size(np.unique(x_Axis)) > 50000000:
            hist, bins = np.histogram(x_Axis, bins=200)
            width = (bins[1] - bins[0])
            center = (bins[:-1] + bins[1:]) / 2
            hist = hist/np.max(hist)
            return center, hist, width

        else:'''
        print ("test",np.size(np.unique(x_Axis)))
        bins = np.minimum(700, np.size(np.unique(x_Axis)))
        if self.cd_m2_function == (0):
            print("contrast")
            hist, bins = np.histogram(x_Axis, bins=500,range = (x_Axis.min(),
                                                                x_Axis.max()*0.95))
        if np.size(np.unique(x_Axis)) < 20:
            hist = np.zeros_like(np.unique(x_Axis))
```

```
            unique = np.unique(x_Axis)
            center = unique
            for i in range (0, np.size(unique)):
                for j in range (0,np.size(x_Axis)):
                    if unique[i] == x_Axis[j]:
                        hist[i] = hist[i]+1
        else:
            hist, bins = np.histogram(x_Axis, bins = bins)
            width = np.diff(bins)
            center = (bins[:-1] + bins[1:]) / 2


        if np.size(np.unique(x_Axis)) < 20:
            width = np.ones_like(unique)*0.5

        if np.size(np.unique(x_Axis)) > 5000:
            width = width * 3

        else:
            width = width

        #print(bins)
        hist = hist/np.max(hist)
        return center, hist, width

        """hilfsvariable = np.size(np.unique (x_Axis))
        if hilfsvariable < 30:
            hilfsvariable = 50
        if hilfsvariable > 100:
            hilfsvariable = 100

        x_AxisLin = np.linspace(x_Axis[0], x_Axis[-1], hilfsvariable)

        Data = np.interp(x_AxisLin, x_Axis, DataIN)

        OutputProbability = np.linspace(0, 1, 1000)
        OutX_Axis = np.interp(OutputProbability, Data, x_AxisLin)

        # get probability density function
        DensityFkt = np.diff(OutputProbability)
```

```
        BarWidth = np.diff(OutX_Axis)

        BarMask = BarWidth > 0#BarWidthLimit
        BarMask = np.append(BarMask,[False])


        OutputProbability = OutputProbability[BarMask]
        OutX_Axis = OutX_Axis[BarMask]
        ##print("BarMask -Done")

        DensityFkt = np.diff(OutputProbability)
        BarWidth = np.diff(OutX_Axis)
        OutX_Axis = np.delete(OutX_Axis,0)
        for i in range (0, OutX_Axis.size):
            if OutX_Axis [i] == ('nan'):
                OutX_Axis [i] = 0


        DensityFkt = DensityFkt / BarWidth
        DensityFkt = DensityFkt / np.max(DensityFkt) *0.75

        return OutX_Axis, DensityFkt, BarWidth"""
        """else:

            x_Unique = np.unique(x_Axis)
            #print (x_Unique)
            #BarWidth = np.ones_like(x_Unique) * np.diff(x_Unique) * 0.5
            y_Achse = np.zeros_like(x_Unique)
            for j in range (0, np.size(x_Unique)):
                for i in range (0, np.size(x_Axis)):
                    if x_Unique [j] == x_Axis [i]:
                        y_Achse [j] = y_Achse[j] + 1
            print ("Density Done")
            if np.size(x_Unique)>20:
                BarWidth = np.ones_like (x_Unique)
            else:
                BarWidth = np.ones_like (x_Unique)*0.5

                y_Achse = y_Achse/np.max(y_Achse)
                return x_Unique, y_Achse, BarWidth"""
```

```python


    def get_plot(self):
        '''
        One function for plotting the correct data
        It is seperated in two parts so that the plots of the data, contrasts,
        SNR over the imaging chain and CDP over the imaging chain charts look
        like each other.
        '''


        if self.cd_m2_function == (0):
            figure, axes = plt.subplots(1,1, figsize=(8,5))
        else :
            figure, axes = plt.subplots(1,2, figsize=(15,5))
        # get probability function
        x_AxisProb, ProbFkt = self.get_ProbablityFunction(self.data)

        _mean = np.mean(self.data)
        _meanSTR = str(Decimal('{:.2e}'.format(_mean)).normalize())

        _std = np.std(self.data)
        _stdSTR = str(Decimal('{:.2e}'.format(_std)).normalize())

        if np.std(self.data) != 0:
            _SNR = _mean/_std
        else:
            _SNR = 0
        _SNRString = str(Decimal('{:.2e}'.format(_SNR)).normalize())
        Legendtitle = "μ=" + _meanSTR + ", std=" + _stdSTR  + ":: SNR=" +_SNRString

        #print("get_ProbablityFunction -Done")
        x_AxisDens, DensityFkt, BarWidth = self.get_DensityFunktion(x_AxisProb,
                                                                    ProbFkt)
        #print("x_Axis = ",x_Axis)

        if self.cd_m2_function == (0) :

            axes.plot(x_AxisProb, ProbFkt, label="Prob. Fkt.")

```

```
            axes.bar(x_AxisDens, DensityFkt, width=BarWidth, color="lightgray",
                     label="Probability Density")
            axes.set_ylim(0,1.2)
            #the xlim must be changed to get a display area which shows all interesting
            #information
            axes.set_xlim(x_AxisProb[0],x_AxisProb[int (x_AxisProb.size*0.99)])
            axes.set_title(self.info_str)
            axes.set_xlabel(self.datatype_str)
            axes.set_ylabel("Probability")
            axes.legend(loc='upper left',title=Legendtitle,prop={'size':8})
            return
        else:
            axes[0].plot(x_AxisProb, ProbFkt, label="Prob. Fkt.")

            axes[0].bar(x_AxisDens, DensityFkt, width=BarWidth,
                    color="lightgray",label="Probability Density")
            axes[0].set_ylim(0,1.2)
            #the display area is greater because otherwise the probability function
            #is not correctly visible
            axes[0].set_xlim(x_AxisProb[0]- ((x_AxisProb[0]+x_AxisProb[-1])/20)
            ,x_AxisProb[-1] +  ((x_AxisProb[0]+x_AxisProb[-1])/20))

            axes[0].set_title(self.info_str)
            axes[0].set_xlabel(self.datatype_str)
            axes[0].set_ylabel("Probability")
            axes[0].legend(loc='upper left',title=Legendtitle,prop={'size':8})

            x_AxisProb, ProbFkt = self.get_ProbablityFunction(
                    self.cd_m2_function(self.data))
            #x_AxisProb, ProbFkt = self.get_ProbablityFunction(self.cd_m2)
            _mean = np.mean(self.cd_m2_function(self.data))
            _meanSTR = str(Decimal('{:.2e}'.format(_mean)).normalize())

            _std = np.std(self.cd_m2_function(self.data))
            _stdSTR =  str(Decimal('{:.2e}'.format(_std)).normalize())
            Legendtitle = "μ=" + _meanSTR + ", std=" + _stdSTR

            if np.std(self.data) != 0:
                _SNR = _mean/_std
            else:
```

```
                _SNR = 0

            _SNRString = str(Decimal('{:.2e}'.format(_SNR)).normalize())
            Legendtitle = "μ=" + _meanSTR + ", std=" + _stdSTR  + ":: SNR="  +_SNRString

        #print("get_ProbablityFunction -Done")
            x_AxisDens, DensityFkt, BarWidth = self.get_DensityFunktion(
                    x_AxisProb,ProbFkt)

            axes[1].plot(x_AxisProb,ProbFkt,label="Prob. Fkt.")
            axes[1].bar(x_AxisDens, DensityFkt, width=BarWidth,
                    color="lightgray",label="Probability Density")
            #axes[1].plot(x_AxisDens,DensityFkt)
            axes[1].set_ylim(0,1.2)
            axes[1].set_xlim(x_AxisProb[0] - ((x_AxisProb[0]+x_AxisProb[-1])/20
                ),x_AxisProb[-1] + ((x_AxisProb[0]+x_AxisProb[-1])/20))
            axes[1].set_title(self.info_str)
            axes[1].set_xlabel("cd/m^2")
            axes[1].set_ylabel("Probability")
            axes[1].legend(loc='upper left',title=Legendtitle,prop={'size':8})
```

Appendix 5: Class C_Windscreen

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  3 12:59:48 2018

@author: lueb5102
"""

import functions.C_IOData as Data
import numpy as np

class C_Windscreen:
    """ Scene: Input scene in cd/m^2
        Glare photons: Choose from 0 to full sun ;)
        Tranmission of the windscreen"""
    def __init__(self, glare_photons_percentage=20.,
                 transmission=0.92):
        self.glare_photons_percentage = glare_photons_percentage
        self.transmission = transmission
```

```

        return

    def get_Output(self, DataIn):
        '''
        DataIn: Data from type C_IOData. Is the data before the windscreen

        returns:
            Windscreen_Output: Data from type C_IOData. Is the data through the
                                windscreen with veiling glare
        '''

        glare_mean = np.mean(DataIn.data)

        glare_mean = glare_mean * self.glare_photons_percentage/100
        if(glare_mean > 10e3):
            #print("Geese [Bosch]: There is a problem here with Gaussian
            #Numbers and the display of the probability density")

            # Geese: There was Problem with np.random.normal [doesn't work for
            #data with very large numebers e.g. 1e15... ]
            glare_data = np.random.random(DataIn.data.shape)
            for i in range(0,25):
                glare_data = glare_data + np.random.random(glare_data.shape)

            glare_data = glare_data - np.mean(glare_data)
            glare_data = glare_data / np.std(glare_data)

            glare_data = glare_data * np.sqrt(glare_mean)
            glare_data = glare_data + glare_mean

        else:

            glare_data = np.ones_like(DataIn.data)*glare_mean
            glare_data = np.random.poisson(glare_data)

        intensity_after_windscreen = self.transmission * (DataIn.data
                                                          + glare_data)
        intensity_after_windscreen = np.maximum(0, intensity_after_windscreen)
        Windscreen_Output = Data.C_IOData()
```

```
        Windscreen_Output.data = intensity_after_windscreen
        Windscreen_Output.stochaticProcess = "Poisson"

        Windscreen_Output.cd_m2_function = lambda x: DataIn.cd_m2_function (
                x-glare_mean/self.transmission)

        Windscreen_Output.cd_m2 = Windscreen_Output.cd_m2_function(
                Windscreen_Output.data)

        Windscreen_Output.datatype_str = "(#photons)/(s*m^2*sr)"
        Windscreen_Output.info_str = """This is the data after transmission
        through the windscreen (with additional glare)"""

        return Windscreen_Output
```

Appendix 6: Class C_Optics

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  3 13:05:22 2018

@author: lueb5102
"""

import functions.C_IOData as Data
import numpy as np

class C_Optics:
    """ Jaehne, Sec Ed, page 96,
    Input: Scene input as a light field
    Output: incident illuminance [lm/m^2] on sensor (but without angular
    dependency)"""
    #illuminance incident on sensor with wavelength (spectral illuminance)

    def __init__(self, f_number=2., transmission=0.9):

        self.f_number = f_number
        self.transmission = transmission
        return

    def get_Output(self, DataIn):
```

```
        '''
        DataIn: Data from type C_IOData. The input data which is going through
                the optics
        returns:
            Optic_Output: Data from type C_IOData. The data after transmission
                            through the optic.

        '''
        Optic_Output = Data.C_IOData()

        Conv_Factor_Optics = self.transmission * np.pi * 1.0/(
                4.0 * self.f_number**2 );
        #Factor for from formel Paper Marc Geese  p. 4 calculation transformation
        #factor for optics

        Optic_Output.data = Conv_Factor_Optics * DataIn.data #in lm/m^2

        Optic_Output.cd_m2_function = lambda x: DataIn.cd_m2_function(
                x / Conv_Factor_Optics) #Funktion in cd/m^2 Skala
        Optic_Output.cd_m2 = Optic_Output.cd_m2_function(Optic_Output.data)
        Optic_Output.datatype_str = "\n(#photons)/(s*m^2)"
        Optic_Output.info_str = "This is the data after transmission \n through the optics"

        return Optic_Output
```

Appendix 7: Class C_Sensor

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  3 13:05:56 2018

@author: lueb5102
"""

import functions.C_IOData as Data
import numpy as np

class C_Sensor():
    """ Image sensor class """
    def __init__(self,
                 pixel_pitch_m=2e-6,
```

```
                 quantum_efficiency=0.7,
                 full_well=15000,
                 K=0.23,
                 temp=50.,
                 mu_dark=35.,
                 row_dark_current = 5.,
                 column_dark_current = 10.,
                 doubling_temp=10.,
                 ADC_bits=12,
                 ISP_bits = 20,
                 number_of_pixel_x = 300,
                 number_of_pixel_y = 420,
                 overall_system_gain = 1,
                 FPN_Pixel_std = 5,
                 FPN_Column_std = 10, FPN_Row_std = 5):
        """
        Pixel pitch: Float. Pixel pitch, actually pixel side length in microns.
                    The pixel is a square.
        Quantum efficiency: Float number. Quatum efficiency of sensor as ratio.
                            Number between 0 and 1
        Full well capacity:Integer. Number of photoelectrons
        K: Float. System overall conversion gain --> DN per photoelectron (see
            EMVA 1288)
        Temperature:Float. In deg celsius
        Mu dark: Float. Expected dark current electrons per sec at 25 deg
                celsius (EMVA 1288)
        Doubling temp: Float. Doubling temperature (EMVA 1288)
        ADC: Integer. ADC discretization in bits
        number of pixel x: Integer. The number of pixel in x-axes
        number of pixel y: Integer. The number of pixel in y-axes"""
        self.pixel_pitch_m = pixel_pitch_m
        self.quantum_efficiency = quantum_efficiency
        self.full_well = full_well
        self.K = K
        self.temp = temp
        self.mu_dark = mu_dark
        self.row_dark_current = row_dark_current
        self.column_dark_current = column_dark_current
        self.reference_temp = 25
        self.doubling_temp = doubling_temp
```

```
        self.ADC_bits = ADC_bits
        self.ISP_bits = ISP_bits
        self.overall_system_gain = overall_system_gain
        self.number_of_pixel_x = number_of_pixel_x
        self.number_of_pixel_y = number_of_pixel_y
        self.FPN_Pixel_std = FPN_Pixel_std
        self.FPN_Column_std = FPN_Column_std
        self.FPN_Row_std = FPN_Row_std


    ###########################################################################
    def OEC (self, IO_Data_In, exposure_Time):
        '''
        This function simulates the Optical electrical Conversion.

        IO_Data_In: Data type from class C_IOData. Is the data after the optics.
        exposure_Time: Float. The exposure time of the image.

        returns:
            Data_OEC: Array-Like. Is the data after the optical elctrical
                        conversion.
            Conv_Factor_Sensor: Float. Is the factor which is generated from
                                the sensor.
        '''

        Conv_Factor_Sensor = (self.pixel_pitch_m**2) * exposure_Time * self.quantum_efficiency

        Data_OEC = IO_Data_In.data * Conv_Factor_Sensor

        return Data_OEC, Conv_Factor_Sensor

    ###########################################################################
    def DarkCurrent (self, Data_OEC, exposure_Time):
        '''
        This function adds dark current to the data after OEC.

        Data_OEC: Array-Like. The data after the OEC-conversion.
        exposure_Time: Float. The exposure time of the image.

        returns:
```

```
            Data_DarkCurrent: Array-Like. The data with added dark current.
            mu_temp_electrons: Float. The mean of the generated dark electrons.

        '''

        dark_current_pixel = np.random.poisson(self.mu_dark, Data_OEC.size)
        dark_current_pixel = np.random.normal(dark_current_pixel,
                                              self.FPN_Pixel_std)
        dark_current = np.ones_like(dark_current_pixel)
        dark_current_row = np.random.poisson(self.row_dark_current,
                                             self.number_of_pixel_y)
        dark_current_row = np.random.normal(dark_current_row, self.FPN_Row_std)
        dark_current_column = np.random.poisson(self.column_dark_current,
                                                self.number_of_pixel_x)
        dark_current_column = np.random.normal(dark_current_column,
                                               self.FPN_Column_std)
        for i in range (0, self.number_of_pixel_y):
            for j in range (0, self.number_of_pixel_x):
                dark_current[i*self.number_of_pixel_x + j] = dark_current_pixel[
                        i*self.number_of_pixel_x + j] + dark_current_row [i
                                                    ] + dark_current_column[j]

        mu_dark_sensor = dark_current * 2**((
                self.temp - self.reference_temp)/self.doubling_temp)
        mu_temp_electrons = np.mean(mu_dark_sensor) * exposure_Time
        # Adapt the dark electrons on the exposure time
        temp_electrons = mu_dark_sensor * exposure_Time

        Data_DarkCurrent = Data_OEC + temp_electrons

        return Data_DarkCurrent, mu_temp_electrons

    ###########################################################################
    def Capacitor (self, Data_DarkCurrent):
        '''
        This function looks up if the capacity is exceeded. And cuts off all
        data above the capacity.(Clipping)
        Data_DarkCurrent: Array-Like. The data with the generated dark current.

        returns:
```

```
            Data_Capacitor: Array-Like. All electrons over the capacity will be
                                cutted.

        '''

        Data_Capacitor = np.minimum (Data_DarkCurrent, self.full_well )

        return Data_Capacitor

    ###########################################################################
    def ADC (self, Data_Capacitor):
        '''
        This functions converts the electrons into digital numbers.
        Data_Capacitor: Array-Like. This is the Data after the capacitor

        returns:
            Data_ADC: Data type from class C_IOData. Is the data after analog
                      digital conversion. The data is in integer numbers.

        '''
         #... and convert to DN
        Data_ADC = Data_Capacitor * self.K
        Data_ADC = Data_ADC.astype(int)

         #ADC
        Data_ADC = np.minimum(Data_ADC, 2**self.ADC_bits)
        Data_ADC = np.maximum(Data_ADC, 0)

        return Data_ADC

    ###########################################################################
    def get_Output_with_exposureTime (self, IO_Data_In, exposure_Time):
        '''
        This function generates output with the given exposure time.

        IO_Data_In: Data type from class C_IOData. Is the data after the optic.
        exposure_Time: Float. The exposure time for the image.

        returns:
            Output: Data type from class C_IOData. Is the data with the given
```

```
                    given exposue time.
        '''
        IO_Data_In.exposure_times = exposure_Time
        Data_OEC, ConvFactor_Sensor = self.OEC (IO_Data_In, exposure_Time)
        Data_DarkCurrent, mu_temp_electrons = self.DarkCurrent (
                Data_OEC, exposure_Time)
        Data_Capacitor = self.Capacitor(Data_DarkCurrent)
        Data_ADC = self.ADC (Data_Capacitor)

        Output = Data.C_IOData()
        Output.data = Data_ADC
        Output.cd_m2_function = lambda x: IO_Data_In.cd_m2_function(
                ((x / self.K) - mu_temp_electrons)/ ConvFactor_Sensor)
        Output.cd_m2 = Output.cd_m2_function(Output.data)
        Output.datatype_str = "DN (Digital Number)"
        Output.info_str = """This is the data after \n raw-ADC on the Sensor \n
        Exposure Time: %s Sekunden""" %exposure_Time

        return Output


    ###########################################################################
    def get_HDR_Image(self, IO_Data_In, Exposure_Times, Input_Mode = "Normal"):
        '''
        This function generates a HDR image with n-exposure times.

        IO_Data_In: Array (n*m). This is am array with data for different
                    exposure times.
        Exposure_Times: Array. There n-exposures possible. But the IO_Data_In
                        has the same n defined.
        Input_Mode: String. At the moment only "Normal" is existing. But it is
                    possible to implement other HDR algorithms. For example in
                    this function or in a new class.

        returns:
            Imager_Ouput:Data type from class C_IOData.
                        Is the Input_Data combined to a HDR image.

        '''

```

```
        #Exposure_Times has to be an array
        if Input_Mode == "Normal":

            Imager_Output = Data.C_IOData()
            Exposure_Times = np.sort(Exposure_Times)
            exposure_ratio = np.array(np.ones_like(Exposure_Times))
            Imager_Outputdata = [[np.ones_like(
                IO_Data_In.data)]for _ in range (len (Exposure_Times))]
            for i in range (0, len(Exposure_Times)):

                Data_OEC, ConvFactor_Sensor = self.OEC (IO_Data_In, Exposure_Times[i])
                Data_DarkCurrent, mu_temp_electrons = self.DarkCurrent (
                Data_OEC, Exposure_Times[i])
                Data_Capacitor = self.Capacitor(Data_DarkCurrent)
                Data_ADC = Data_Capacitor * self.K

                 #ADC
                Data_ADC = np.minimum(Data_ADC, 2**self.ADC_bits)
                Data_ADC = np.maximum(Data_ADC, 0)
                Imager_Outputdata[i] = Data_ADC
                Imager_Output = Data.C_IOData()
                Imager_Output.data = Data_ADC
                Imager_Output.cd_m2_function = lambda x: IO_Data_In.cd_m2_function(
                ((x / self.K) - mu_temp_electrons)/ ConvFactor_Sensor)
                Imager_Output.cd_m2 = Imager_Output.cd_m2_function(Imager_Output.data)
                Imager_Output.datatype_str = "DN (Digital Number)"
                Imager_Output.info_str = """This is the data after \n raw-ADC on the
                Sensor \n Exposure Time: %s Sekunden""" %Exposure_Times[i]
                #Imager_Output.doPrint()
                if (i < len(Exposure_Times)):
                    Imager_Outputdata [i] = Imager_Outputdata [i] * Exposure_Times[-1]/Exposure_Time
                    exposure_ratio [i] = Exposure_Times[-1] / Exposure_Times[i]



            HDR_Reconstruction = Imager_Outputdata[-1]
            for j in range (0, (len(Exposure_Times))):
                for i in range (0, len (HDR_Reconstruction)):
                    if HDR_Reconstruction[i] > 2**(self.ISP_bits/len(
                            Exposure_Times)*(j)) :
```

```
                        HDR_Reconstruction [i] = Imager_Outputdata [len(
                                    Exposure_Times)-1-j][i]

                        if HDR_Reconstruction [i] < 2**(self.ISP_bits/len(
                                Exposure_Times)*(j+1)):

                            HDR_Reconstruction [i] = Imager_Outputdata [len(
                                    Exposure_Times)-1-j][i]



            Imager_Output.data = HDR_Reconstruction * self.overall_system_gain

            Imager_Output.data = np.minimum(HDR_Reconstruction.astype(int),
                                            2**self.ISP_bits)

            Imager_Output.cd_m2 = Imager_Output.cd_m2_function(Imager_Output.data)
            Imager_Output.datatype_str = "DN (Digital Number)"
            Imager_Output.info_str = "This is the data after HDR Reconstruction"

        return Imager_Output
```

Appendix 8: Class C_Test

```
# -*- coding: utf-8 -*-
"""
Created on Mon May 28 10:45:37 2018

@author: lueb5102
"""
import functions.C_IOData as Data
import functions.C_Sensor as Sensor
import numpy as np

class C_Test():

    def __init__(self, K = 0.25, row_dark_current = 5.,
                 column_dark_current = 10., temp = 100, overall_system_gain = 1,
                 number_of_pixel_x = 301, number_of_pixel_y = 420,
                 FPN_Pixel_std = 5,
                 FPN_Column_std = 10, FPN_Row_std = 5):
```

```


        self.pixel_pitch_m = 3e-6
        self.ADC_bits = 12
        self.full_well = 15000
        self.quantum_efficiency = 0.8
        self.doubling_temp = 10
        self.reference_temp = 25
        self.K = K
        self.mu_dark = 35
        self.row_dark_current = row_dark_current
        self.column_dark_current = column_dark_current
        self.temp = temp
        self.overall_system_gain = overall_system_gain
        self.number_of_pixel_x = number_of_pixel_x
        self.number_of_pixel_y = number_of_pixel_y
        self.FPN_Pixel_std = FPN_Pixel_std
        self.FPN_Column_std = FPN_Column_std
        self.FPN_Row_std = FPN_Row_std

    def OEC (self, IO_Data_In, exposure_Time):
        '''
        This function uses the function from the sensor class because the
        calculation is the same.
        '''

        Data_OEC, Conv_Factor_Lightsensor = Sensor.C_Sensor.OEC (self,
                                                  IO_Data_In, exposure_Time)

        return Data_OEC, Conv_Factor_Lightsensor



    def DarkCurrent (self, Data_OEC, exposure_Time):
        '''
        This function uses the function from the sensor class because the
        calculation is the same.
        '''

        Data_DarkCurrent, mu_temp_electrons = Sensor.C_Sensor.DarkCurrent (self
```

```
                                                      , Data_OEC, exposure_Time)

        return Data_DarkCurrent, mu_temp_electrons


    def Capacitor (self, Data_DarkCurrent):
        '''
        This function uses the function from the sensor class because the
        calculation is the same.
        '''

        Data_Capacitor = Sensor.C_Sensor.Capacitor (self, Data_DarkCurrent)

        return Data_Capacitor


    def ADC (self, Data_Capacitor):
        '''
        This function uses the function from the sensor class because the
        calculation is the same.
        '''

        Data_ADC = Sensor.C_Sensor.ADC (self, Data_Capacitor)

        return Data_ADC


    def get_Output_with_exposureTime (self ,IO_Data_In, Exposure_Time):
        '''
        This function uses the function from the sensor class because the
        calculation is the same.
        '''

        Output = Sensor.C_Sensor.get_Output_with_exposureTime (self,
                                                      IO_Data_In, Exposure_Time)

        return Output

    def get_HDR_Image(self, IO_Data_In, Exposure_Times, Input_Mode = 0):
        '''
```

```
        In case the Input_Mode is zero or "Normal" this function uses the
        function from class sensor.

        This function generates a HDR image with n-exposure times. If the
        Input_Mode is Linear.

        IO_Data_In: Array (n*m). There is the data for each exposure time
                    inside
        Exposure_Times: Array. There are n-exposures Possible. But the
                        IO_Data_In has to have the same size in n.
        Input_Mode: String. In this class the mode "Linear" is possible. In
                    case "Normal" or "0" the algorithm from class sensor will
                    be used.

        returns:
            Imager_Ouput:Data type from class C_IOData.
                        Is the "Input_Data" combined to a HDR image.

        '''

        if Input_Mode == "Normal":

            Sensor.C_Sensor.get_HDR_Image (self, IO_Data_In, Exposure_Times,
                                           Input_Mode)

        if Input_Mode == 0:

            Sensor.C_Sensor.get_HDR_Image (self, IO_Data_In, Exposure_Times,
                                           Input_Mode)



        if Input_Mode == "Linear":

            Picture = np.ones(len(IO_Data_In.data))
            Linear = np.array(0)
            Exposure_Times = np.sort(Exposure_Times)
            for i in range (0, len(Exposure_Times)):
                Output = self.get_Output_with_exposureTime(IO_Data_In,
                                                           Exposure_Times[i])
                Picture = Output.data
```

```
                Picture = Picture / Exposure_Times[i]
                Linear = Linear + Picture

                data = Linear / len (Exposure_Times)
                data = data * np.max(Exposure_Times)
                Output.data = data.astype(int)


            Output.data = Output.data * self.overall_system_gain
            Output.info_str = "This is the data after HDR Reconstruction"

        return Output
```

Appendix 9: Class C_Image_quality

```
# -*- coding: utf-8 -*-
"""
Created on Thu May  3 13:08:32 2018

@author: lueb5102
"""

import functions.C_IOData as Data
from PIL import Image
import functions.C_Sensor as Sensor
import functions.Image_processing as Image_processing
import numpy as np
import matplotlib.pyplot as plt
import xlwt
import mpl_toolkits.mplot3d.axes3d as p3
import os
import plotly.plotly as py
import plotly.graph_objs as go
import plotly
import random
from random import randint


class  C_Image_quality ():
    ###########################################################################
    def evaluate_CDP_Pixel_to_Pixel(CDP_Input, Scene_Contrast, Input_Sensor = 0,
```

```
                    number_of_pixel_x = 0, number_of_pixel_y = 0):
        '''
        This function evaluates the CDP from Pixel to Pixel.
        CDP_Input: Array-Like has to be in cd/m^2.
        Scene_Contrast: Integer has to be between zero and infinite.
        Input_Sensor: Data from Class Sensor where number_of_pixel_x and
                        number_of_pixel_y be filled with integer if not filled
                        the variable number_of_pixel_x and number_pixel_y have
                        to have integers inside.
        number_of_pixel_x: Integer is necessary otherwise Input_Sensor must be
                            existing.
        number_of_pixel_y: Integer is necessary otherwise Input_Sensor must be
                            existing.

        returns:
            cdp: Float number in the area 0 to 1. It is the calclated CDP
            Output: Datatype from class C_IOData, with data and all strings to
                    print the plot.
        '''
        if Input_Sensor == 0:
            if number_of_pixel_x == 0:
                raise NameError ("At least number_of_pixel_x or a Input_Sensor is necessary")

            if number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")

        else:
            if Input_Sensor.number_of_pixel_x == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")
            else:
                number_of_pixel_x = Input_Sensor.number_of_pixel_x
            if Input_Sensor.number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_x or a Input_Sensor is necessary")
            else:
                number_of_pixel_y = Input_Sensor.number_of_pixel_y

        'The confidence_delta is defined in the paper of Marc Geese p.10'
        confidence_delta = 0.5
        'To be sure that all CDP_Input is positive'
        CDP_Input = np.maximum(0, CDP_Input)
```

```

        Contrast =  np.ones(2 * CDP_Input.size - number_of_pixel_x - number_of_pixel_y)
        helping_value = 0

        for i in range (0, CDP_Input.size-1):


            if i % number_of_pixel_x == 0 and i != 0:
                helping_value = helping_value + 1

            else:
                if CDP_Input [i] > CDP_Input [i+1]:
                    if CDP_Input[i+1] != 0:
                        #the result is already in %
                        Contrast[i - helping_value] = (CDP_Input[i] /
                                CDP_Input[i+1] * 100 - 100)
                    else:
                        Contrast[i - helping_value] = 10000

                else:
                    if CDP_Input[i] != 0:
                        #the result is already in %
                        Contrast[i - helping_value] = (CDP_Input[i+1] /
                                CDP_Input[i] * 100 - 100)
                    else:
                        Contrast[i - helping_value] = 10000



            if (i + number_of_pixel_x) < CDP_Input.size:
                if CDP_Input [i] > CDP_Input [i+ number_of_pixel_x]:
                    if CDP_Input [i + number_of_pixel_x] != 0:
                            #the result is already in %
                        Contrast[i + CDP_Input.size -
                                 number_of_pixel_y - helping_value] = (
                                 CDP_Input[i] / CDP_Input[i
                                    + number_of_pixel_x] * 100 - 100)
                    else:
                        Contrast[i + CDP_Input.size - number_of_pixel_y
                                 - helping_value] = 10000

```

```
            else:
                if CDP_Input[i] != 0:
                     #the result is already in %
                    Contrast[i + CDP_Input.size - number_of_pixel_y -
                            helping_value] = CDP_Input[i +
                               number_of_pixel_x] / CDP_Input[i] * 100 - 100
                else:
                    Contrast[i + CDP_Input.size - number_of_pixel_y
                            - helping_value] = 100000
                        #Contrast_after_HDR[i] = Contrast_after_HDR_h[i]




    CDP_Input = CDP_Input.flatten().astype(np.float)
    CDP_Input = CDP_Input.flatten().astype(np.float)

    Lower_Bound = Scene_Contrast - confidence_delta * Scene_Contrast
    Upper_Bound = Scene_Contrast + confidence_delta * Scene_Contrast
    Lower_idx = (Contrast < Lower_Bound).astype(np.int)
    Lower_idx = Lower_idx.sum()
    Upper_idx = (Contrast < Upper_Bound).astype(np.int)
    Upper_idx = Upper_idx.sum()

    Output = Data.C_IOData()
    Output.datatype_str= "Contrast [%]"
    Output.info_str = "Contrast evaluation"

    Output.data = Contrast

    cdp = (Upper_idx - Lower_idx) / np.size (Contrast)
    return cdp, Output

###########################################################################
def evaluate_CDP_ROI_to_ROI (ROI_1, ROI_2, Scene_Contrast):
    '''
    This function evaluates the CDP between two defined ROIs.
    ROI_1: Array-Like has to be in cd/m^2.
    ROI_2: Array-Like has to be in cd/m^2.
    Scene_Contrast: Integer has to be between zero and infinite.
```

```

    returns:
        cdp: Float number in the area 0 to 1. It is the calclated CDP
        Output: Datatype from class C_IOData, with data and all strings to
                print the plot.
    '''

    confidence_delta = 0.5
    size_contrast = ROI_1.size * ROI_2.size

    if size_contrast < 360000000:

        Contrast = np.ones(size_contrast)

        for i in range (0, ROI_1.size):
            for j in range (0, ROI_2.size):
                if np.mean(ROI_1) > np.mean(ROI_2):
                    Contrast[i*j+j] = ROI_1[i]/ROI_2[j] * 100 - 100
                else:
                    Contrast[i*j+j] = ROI_2[j]/ROI_1[i] * 100 - 100

    else:
        if ROI_1.shape != ROI_2.shape:
            if ROI_1.size > ROI_2.size:

                for i in range (ROI_2.size, ROI_1.size):

                    ROI_2 = np.append(ROI_2, ROI_2[randint(0, ROI_2.size-1)])


            if ROI_2.size > ROI_1.size:

                for i in range (ROI_1.size, ROI_2.size):

                    ROI_1 = np.append(ROI_1, ROI_1[randint(0, ROI_1.size-1)])

        if np.mean(ROI_1) > np.mean (ROI_2):
            Contrast = ROI_1 / ROI_2 * 100 - 100 #Weber Contrast in %
        else:
            Contrast = ROI_2 / ROI_1 * 100 - 100 #Weber Contrast in %
```

```python

        Lower_Bound = Scene_Contrast - confidence_delta * Scene_Contrast
        Upper_Bound = Scene_Contrast + confidence_delta * Scene_Contrast
        Lower_idx = (Contrast < Lower_Bound).astype(np.int)
        Lower_idx = Lower_idx.sum()
        Upper_idx = (Contrast < Upper_Bound).astype(np.int)
        Upper_idx = Upper_idx.sum()

        Output = Data.C_IOData()
        Output.datatype_str= "Contrast [%]"
        Output.info_str = "Contrast evaluation"

        Output.data = Contrast

        cdp = (Upper_idx - Lower_idx) / np.size (Contrast)
        return cdp, Output



    ###########################################################################
    def evaluate_SNR_Array(SNR_Input_1,  Zero_Signal_Value=0):
        '''
        SNR_Input_1: Array-Like. The data where the SNR has to be calculated.
        Zero_Signal_Value: Integer. This number is not necessary.

        returns:
            SNR: Float
        '''

        for i in range (0, SNR_Input_1.size):
            if i % 2 == 0:
                SNR_Input_1[i]= 0


        Hilfsarray = SNR_Input_1 > 0
        SNR_Input_1 = SNR_Input_1[Hilfsarray]
        if np.std(SNR_Input_1) < 1e-9:
           SNR = np.inf
        else:
            if SNR_Input_1.all () == 0:
```

```
                SNR =  10 * np.log10(0.001)
            else:
                SNR = 10 * np.log10((np.mean(SNR_Input_1) - Zero_Signal_Value
                                    ) / np.std(SNR_Input_1))
        return SNR


    ###########################################################################
    def evaluate_SNR_ROI(SNR_Input_1,  Zero_Signal_Value=0):
        '''
        SNR_Input_1: Array-Like. The data where the SNR has to be calculated.
        Zero_Signal_Value: Integer. This number is not necessary.

        returns:
            SNR: Float
        '''


        if np.std(SNR_Input_1) < 1e-9:
           SNR = np.inf
        else:
            if SNR_Input_1.all () == 0:
                SNR =  10 * np.log10(0.001)
            else:
                SNR = 10 * np.log10((np.mean(SNR_Input_1) - Zero_Signal_Value
                                    ) / np.std(SNR_Input_1))
        return SNR

    ###########################################################################
    def evaluate_CDP_ROI (Point_1, Point_2, ROI_Input, Contrast,
                          Input_Sensor = 0 , number_of_pixel_x = 0,
                          number_of_pixel_y = 0):
        '''
        This function evaluates the CDP in a defined ROI.
        Point1: Tuple-object with x and y Point
        Point2: Tuple-object with x and y Point
        ROI_Input: Data from class C_IOData
        Contrast: Integer or Float. The contrast where the CDP has to be
                  evaluated
        Input_Sensor: Data from class Sensor where number_of_pixel_x and
```

```
                            number_of_pixel_y be filled with integer if not filled
                            the variable number_of_pixel_x and number_pixel_y have
                            to have integers inside
        number_of_pixel_x: Integer is necessary otherwise Input_Sensor must be
                                existing
        number_of_pixel_y: Integer is necessary otherwise Input_Sensor must be
                                existing
        returns:
            CDP: Float whith the calculated CDP for the evaluated contrast
            Contrast_ROI: Data from type C_IOData to plot the contrast progress
        '''
        if Input_Sensor == 0:
            if number_of_pixel_x == 0:
                raise NameError ("At least number_of_pixel_x or a Input_Sensor is necessary")

            if number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")

        else:
            if Input_Sensor.number_of_pixel_x == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")
            if Input_Sensor.number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_x or a Input_Sensor is necessary")
        d = -1

        if Point_2[0] < Point_1[0]:
            Point_help = Point_2[0]
            Point_2[0] = Point_1[0]
            Point_1[0] = Point_help

        if Point_2[1] < Point_1[1]:
            Point_help = Point_2[1]
            Point_2[1] = Point_1[1]
            Point_1[1] = Point_help
        if Point_1[0]> number_of_pixel_x:
            Point_1[0] = number_of_pixel_x
        if Point_1[1]> number_of_pixel_y:
            Point_1[1] = number_of_pixel_y
        if Point_2[0]> number_of_pixel_x:
            Point_2[0] = number_of_pixel_x
```

```python
        if Point_2[1]> number_of_pixel_y:
            Point_2[1] = number_of_pixel_y

        if Input_Sensor == 0:
            ROI = Data.C_IOData()

            ROI.data = np.ones((Point_2[0] - Point_1[0]) * (
                    Point_2[1] - Point_1[1]))

            ROI.cd_m2 = np.ones((Point_2[0] - Point_1[0]) * (
                    Point_2[1] - Point_1[1]))

            for i in range (Point_1[1], Point_2[1]):

                for j in range (Point_1[0],Point_2[0]):

                    d = d + 1 # just a variable to increase the array
                    ROI.data[d] = ROI_Input.data[i*number_of_pixel_x+j-1]
                    ROI.cd_m2 [d] = ROI_Input.cd_m2[i*number_of_pixel_x+j-1]


            ROI.data = ROI.data.astype(int)
            x = Point_2[1]-Point_1[1]
            image2 = Image_processing.Array_to_image (ROI.data, (
                    Point_2[0] - Point_1[0]),x)


            image2.convert('L')

            image2.show()
        if Input_Sensor == 0:
            Input_Sensor = Sensor.C_Sensor(
                    10, number_of_pixel_x = number_of_pixel_x,
                    number_of_pixel_y = number_of_pixel_y)
        if Input_Sensor == 0:
            CDPs, Contrast_ROI = C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                                                        ROI.cd_m2,
                                                        Contrast,
                                                        Input_Sensor = Input_Sensor)
        else:
```

```
            CDPs , Contrast_ROI = C_Image_quality . evaluate_CDP_Pixel_to_Pixel (
                                                                ROI . cd_m2 ,
                                                                Contrast ,
                                                                number_of_pixel_x =
                                                                number_of_pixel_x ,
                                                                number_of_pixel_y =
                                                                number_of_pixel_y )

        return CDPs , Contrast_ROI

    ###########################################################################
    def evaluation_CDP_imaging_chain ( Contrast_to_Evaluate , Input_Scene ,
                                      Input_Sensor = 0 , number_of_pixel_x = 0 ,
                                      number_of_pixel_y = 0 , Input_Windscreen = 0 ,
                                      Input_Optics = 0 , Input_Imager =  0 ,
                                      Input_HDR = 0 , Input_Tonemap = 0) :
        '''
        Contrast_to_evaluate : Float or Integer . The contrast where the CDP
                                has to be evaluated .
        Input_Sensor : Data from class sensor where number_of_pixel_x and
                       number_of_pixel_y be filled with integer
        Input_Scene : Array - Like . The data has to be in Cd / m ^2. Is necessary for
                     the calculation .
        Input_Windscreen : Array - Like . The data has to be in Cd / m ^2. Is not
                           necessary for the calculation .
        Input_optics : Array - Like . The data has to be in Cd / m ^2. Is not
                       necessary for the calculation .
        Input_Imager : Array - Like . The data has to be in Cd / m ^2. Is not
                       necessary for the calculation .
        Input_HDR : Array - Like . The data has to be in Cd / m ^2. Is not
                    necessary for the calculation .
        Input_Tonemap : Array - Like . The data has to be in Cd / m ^2. Is not
                        necessary for the calculation .

        At the end there will be a plot with the CDP over the defined Imaging
        Chain
        '''
        if Input_Sensor == 0:
            if number_of_pixel_x == 0:
                raise NameError ( " At least number_of_pixel_x or a Input_Sensor is necessary " )
```

```
            else:
                Input_Sensor.number_of_pixel_x =   number_of_pixel_x
            if number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")
            else:
                Input_Sensor.number_of_pixel_y =   number_of_pixel_y
        else:
            if Input_Sensor.number_of_pixel_x == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")
            if Input_Sensor.number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_x or a Input_Sensor is necessary")



        i = 0
        Contrast_image = Data.C_IOData()
        if Input_Scene.any != 0:
            CDP_Scene, Contrast_image.data  = C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                    Input_Scene, Contrast_to_Evaluate, Input_Sensor)
            y = np.array(CDP_Scene)
            x = np.array('Scene')
            z = np.array(i)

        if Input_Windscreen.any != 0:
            CDP_Windscreen, Contrast_image.data  = C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                    Input_Windscreen, Contrast_to_Evaluate, Input_Sensor)
            y = np.append (y, CDP_Windscreen)
            x = np.append (x, 'after Windscreen')
            i = i+1
            z = np.append (z, i)

        if Input_Optics.any != 0:
            CDP_Optics, Contrast_image.data  = C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                    Input_Optics, Contrast_to_Evaluate, Input_Sensor)
            y = np.append (y, CDP_Optics)
            x = np.append (x, 'after Optics')
            i = i+1
            z = np.append (z, i)

        if Input_Imager.any != 0:
```

```
        CDP_Imager, Contrast_image.data  = C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                Input_Imager, Contrast_to_Evaluate, Input_Sensor)
        y = np.append (y, CDP_Imager)
        x = np.append (x, 'after Imager')
        i = i+1
        z = np.append (z, i)

    if Input_HDR.any != 0:
        CDP_HDR, Contrast_image.data  = C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                Input_HDR, Contrast_to_Evaluate, Input_Sensor)
        y = np.append (y, CDP_HDR)
        x = np.append (x, 'after HDR')
        i = i+1
        z = np.append (z, i)

    if Input_Tonemap.any != 0:
        CDP_tonemap, Contrast_image.data  = C_Image_quality.evaluate_CDP_Pixel_to_Pixel(
                Input_Tonemap, Contrast_to_Evaluate, Input_Sensor)
        y = np.append (y, CDP_tonemap)
        x = np.append (x, 'after tonemap')
        i = i+1
        z = np.append (z, i)



    Figure, Axes = plt.subplots(1,1, figsize=(10,5))
    Axes.plot(z,y)
    Axes.set_xticklabels (x, rotation ='vertical')
    Axes.set_ylabel ('CDP')
    Axes.set_ylim (0,1.1)
    Axes.set_xlim (0, np.max(z))

    return

#############################################################################
def evaluation_SNR_imaging_chain (Input_Scene, Input_Windscreen = 0,
                                 Input_Optics = 0, Input_Imager =  0,
                                 Input_HDR = 0,
                                 Input_Tonemap = 0, Zero_Signal_Value = 0):

```

```
    '''
    Input_Scene: Array-Like. The Data has to be in Cd/m^2. This Data is
                necessary.
    Input_Windscreen: Array-Like. The Data has to be in Cd/m^2. Is not
                        necessary for the calculation.
    Input_optics: Array-Like. The Data has to be in Cd/m^2. Is not
                        necessary for the calculation.
    Input_Imager: Array-Like. The Data has to be in Cd/m^2. Is not
                        necessary for the calculation.
    Input_HDR: Array-Like. The Data has to be in Cd/m^2. Is not
                        necessary for the calculation.
    Input_Tonemap: Array-Like. The Data has to be in Cd/m^2. Is not
                        necessary for the calculation.

    At the end there will be a plot with the SNR over the defined Imaging
    Chain
    '''
    i = 0
    if Input_Scene.any != 0:
        SNR_Scene  = C_Image_quality.evaluate_SNR_Array (Input_Scene,
                                          Zero_Signal_Value)
        y = np.array(SNR_Scene)
        x = np.array('Scene')
        z = np.array(i)

    if Input_Windscreen.any != 0:
        SNR_Windscreen= C_Image_quality.evaluate_SNR_Array (Input_Windscreen,
                                          Zero_Signal_Value)
        y = np.append (y, SNR_Windscreen)
        x = np.append (x, 'after Windscreen')
        i = i+1
        z = np.append (z, i)

    if Input_Optics.any != 0:
        SNR_Optics = C_Image_quality.evaluate_SNR_Array(Input_Optics,
                                          Zero_Signal_Value)
        y = np.append (y, SNR_Optics)
        x = np.append (x, 'after Optics')
        i = i+1
        z = np.append (z, i)
```

```

    if Input_Imager.any != 0:
        SNR_Imager  = C_Image_quality.evaluate_SNR_Array (Input_Imager,
                                                           Zero_Signal_Value)
        y = np.append (y, SNR_Imager)
        x = np.append (x, 'after Imager')
        i = i+1
        z = np.append (z, i)

    if Input_HDR.any != 0:
        SNR_HDR  = C_Image_quality.evaluate_SNR_Array (Input_HDR,
                                                       Zero_Signal_Value)
        y = np.append (y, SNR_HDR)
        x = np.append (x, 'after HDR')
        i = i+1
        z = np.append (z, i)

    if Input_Tonemap.any != 0:
        SNR_Tonemap  = C_Image_quality.evaluate_SNR_Array (Input_Tonemap,
                                                            Zero_Signal_Value)
        y = np.append (y, SNR_Tonemap)
        x = np.append (x, 'after tonemap')
        i = i+1
        z = np.append (z, i)



    Figure, Axes = plt.subplots(1,1, figsize=(10,5))
    Axes.plot(z,y)
    Axes.set_xticklabels (x, rotation ='vertical')
    Axes.set_ylabel ('SNR')
    Axes.set_xlim (0, np.max(z))

    return

############################################################################
def evaluation_from_x_to_y (Input, Start = 10, Stop = 100, Stepsize = 10,
                            Input_Sensor = 0,  number_of_pixel_x = 0,
                            number_of_pixel_y = 0):
    '''
```

```
        Input: Array-Like. The data has to be in cd/m^2 Domain.
        Start: Integer. Start contrast for the evaluation.
        Stop: Integer. Stop contrast for the evaluation.
        Stepsize: Integer. Difference between two contrasts.
        Input_Sensor: Data from class Sensor where number_of_pixel_x and
                        number_of_pixel_y be filled with integer if not filled
                        the variable number_of_pixel_x and number_pixel_y have
                        to have integers inside
        number_of_pixel_x: Integer is necessary otherwise Input_Sensor must be
                            existing
        number_of_pixel_y: Integer is necessary otherwise Input_Sensor must be
                            existing

        returns:
            CDP_out: Array-Like with the different CDPs from Start to Stop with
                    the defined stepsize
            Contrasts: Array-Like with the evaluated Contrasts in the defined
                        Domain
        '''
        if Input_Sensor == 0:
            if number_of_pixel_x == 0:
                raise NameError ("At least number_of_pixel_x or a Input_Sensor is necessary")
            if number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")
        else:
            if Input_Sensor.number_of_pixel_x == 0:
                raise NameError ("At least number_of_pixel_y or a Input_Sensor is necessary")
            if Input_Sensor.number_of_pixel_y == 0:
                raise NameError ("At least number_of_pixel_x or a Input_Sensor is necessary")

        #calculation of number of steps which are necessary
        Helping_Value = int((Stop - Start) / Stepsize+1)
        for j in range (0, Helping_Value):

            if Input_Sensor == 0:

                Contrast = Data.C_IOData()
                CDP, Contrast.data = C_Image_quality.evaluate_CDP_Pixel_to_Pixel (
                        Input, Start + Stepsize * j,
                        number_of_pixel_x =  number_of_pixel_x,
```

```
                    number_of_pixel_y =  number_of_pixel_y)

            if j == 0:
                CDP_out = np.array(CDP)
                Contrasts = np.array(Start)
            else:
               CDP_out = np.append(CDP_out, CDP)
               Contrasts = np.append(Contrasts, Start + Stepsize * j)
        else:
            Contrast = Data.C_IOData()
            CDP, Contrast.data = C_Image_quality.evaluate_CDP_Pixel_to_Pixel (
                    Input, Start + Stepsize*j, Input_Sensor = Input_Sensor)
            if j == 0:
                CDP_out = np.array(CDP)
                Contrasts = np.array(Start)
            else:
               CDP_out = np.append(CDP_out, CDP)
               Contrasts = np.append(Contrasts, Start + Stepsize * j)


    return CDP_out, Contrasts

###########################################################################
def write_to_xls (Contrasts, CDPs, name_of_sheet = "test"):
    '''
    Contrasts: Array-Like. The evaluated contrasts which have to be written
                in the xls sheet.
    CDPs: Array-Like. The associated CDPs to the evaluated contrasts.

    Name_of_sheet: String. Name of sheet where the results have to be
                    written.

    returns:
        After this function there will be an xls with the name of sheet in
        the folder

    '''
    os.chdir("XLS results")
    Book = xlwt.Workbook(encoding = "utf-8")
    Sheet1 = Book.add_sheet("Sheet 1")
```

```
        Sheet1.write(0, 0, "Contrasts in %")

        Number_of_Images = len (CDPs)
        Number_of_Contrasts = len(Contrasts)

        for i in range (0, Number_of_Contrasts):

            Helping_Value = float(Contrasts[i])
            Sheet1.write(i+1, 0, Helping_Value)

        for j in range (0, Number_of_Images):
            Sheet1.write (0, (j+1), "Picture number %s" %(j+1))

            for i in range (0, Number_of_Contrasts):
                Helping_Value = float(CDPs[j][i])
                Sheet1.write (i+1, j+1, Helping_Value)

        Book.save ("%s.xls" %name_of_sheet)
        os.chdir("...")
        return
```

Appendix 10: Image_processing

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jun 19 12:55:38 2018

@author: lueb5102
"""
import numpy as np
import functions.C_IOData as Data
from PIL import Image
import os
import pylab as pl
import sys
import matplotlib.pyplot as plt
import matplotlib.path as mplPath
import plotly.plotly as py
import plotly.graph_objs as go
import plotly
import random
```

```


def DN_to_cd_m2conversion (Image_Information):
    '''
    Is the Backtransformation from digital numbers back into the cd/m^2 domain.
    There are some brackets which are not necessary. Their only sense is to get all
    relevant code onto one A4 Page.
    Image_Information: Data from type C_IOData. There is all informationen about
                       the used camera inside. With these parameters the
                       backtransformation will be done. Not all Paremeters are
                       necessary but this will led to inaccurate results.
    returns:
        Image_Information: It is the same variable from the Input. But now als
                           the cd_m^2function and the cd_m2 part of the variable
                           has got some results. These Results are calculated
                           inside this function

    '''

    c = 299792458 # in m/s
    h = 6.62607004e-34
    wavelength_m = 500e-9
    luminos_efficiency = 1000
    Conv_Fac = h * c * luminos_efficiency / wavelength_m
    reference_temp = 25.
    doubling_temp = 10

    #go back to Sensor
    Image_Information.cd_m2 = np.ones_like(Image_Information.data)
    Conv_Light = Image_Information.sensorsize ** 2 * (
                Image_Information.Exposure_Times[0] *
                Image_Information.quatum_efficieny / Image_Information.K)
    Image_Information.cd_m2_function = lambda x: x* Conv_Light / Conv_Fac
    Image_Information.datatype_str = "DN (Digital Number)"
    Image_Information.info_str = "This is the data of the Image"

    for i in range (0, np.size(Image_Information.cd_m2)):

        Image_Information.cd_m2 [i] = (Image_Information.data[i] * Conv_Light -
        (Image_Information.mu_dark * 2 **((
```

```
                Image_Information.temp - reference_temp)/doubling_temp) *
        Image_Information.Exposure_Times[0]))

    for j in range (0, np.size(Image_Information.cd_m2)):

        Image_Information.cd_m2[j]= Image_Information.cd_m2[j] / Conv_Fac

    #go back to optics
    if (Image_Information.transmission_Optics != 0):

        Image_Information.cd_m2 = (Image_Information.cd_m2 /
                Image_Information.transmission_Optics)
        Image_Information.cd_m2_function = (lambda x: x * Conv_Light /
                Conv_Fac / Image_Information.transmission_Optics)

    if (Image_Information.F_number != 0):
        #same formula like class optics
        Conv_Optic = np.pi * 1 / ( 4 * Image_Information.F_number ** 2)
        Image_Information.cd_m2 = Image_Information.cd_m2 * Conv_Optic
        Image_Information.cd_m2_function = (lambda x: x * Conv_Light /
                Conv_Fac / Image_Information.transmission_Optics * Conv_Optic)

    #go back to windscreen
    if (Image_Information.transmission_Windscreen != 0):

        Image_Information.cd_m2 = (Image_Information.cd_m2 /
                                   Image_Information.transmission_Windscreen)
        Image_Information.cd_m2_function = (lambda x: x* Conv_Light / Conv_Fac
                                    / Image_Information.transmission_Optics *
                                    Conv_Optic /
                                    Image_Information.transmission_Windscreen*(
                                            1-Image_Information.glare_photons))

    return Image_Information

##############################################################################
def Image_to_Array (Name_of_Image):
    '''
    This function converts an input image to an array.

```

```
    Name_of_Image: String. Just the name of the image the rest will be done
                    inside this function. Image has to be stored in the CDP
                    folder.

    returns:
        Array: Array. With the Data from the image.
        Number_of_pixel_x: Integer. Number of Pixel in the x-Axis.
        Number_of_pixel_y: Integer. Number of Pixel in the y-Axis.

    '''

    os.chdir("Images to evaluate")
    Input_Image = Image.open("%s" %Name_of_Image)
    Input_Image.show()

    if Input_Image.mode != "L":
        Input_Image = Input_Image.convert("L")

    number_of_pixel_x, number_of_pixel_y = Input_Image.size

    Array = np.ones ((number_of_pixel_x)*(number_of_pixel_y))

    for i  in range (0, number_of_pixel_y):
        for j in range (0, number_of_pixel_x):
            Array[number_of_pixel_x * i + j] = Input_Image.getpixel((j,i))
    os.chdir("..")
    return Array, number_of_pixel_x, number_of_pixel_y



##############################################################################
def Array_to_image (Array, number_of_pixel_x, number_of_pixel_y):
    '''
    This function converts an Input array to an Image.

    Array: Array. With the Data for the image.
        Number_of_pixel_x: Integer. Number of pixel in the x-axis.
        Number_of_pixel_y: Integer. Number of pixel in the y-axis.

    returns:
```

```
        image: Data type from class PIL Image.

    '''

    Array.astype(int)
    image = Image.new ('RGB', (number_of_pixel_x , number_of_pixel_y), (210))

    #image after the ISP
    for i in range (0, number_of_pixel_y):
        for j in range (0, number_of_pixel_x):

            image.putpixel((j,i), (Array[i*number_of_pixel_x+j],Array[
                    i*number_of_pixel_x+j]
            ,Array[i*number_of_pixel_x+j]))
    image = image.convert("L")
    return image

###########################################################################
def plot3D (Image_Data, Input_Sensor = 0, number_of_pixel_x = 0,
                number_of_pixel_y = 0):
        '''
        Plots the image as a 3D plot
        Image_Data: Array-Like. With the data of each pixel of the image
        Input_Sensor: Data from Class Sensor where number_of_pixel_x and
                        number_of_pixel_y be filled with integer if not filled
                        the variable number_of_pixel_x and number_pixel_y have
                        to have integers inside
        number_of_pixel_x: Integer is necessary otherwise Input_Sensor must be
                            existing
        number_of_pixel_y: Integer is necessary otherwise Input_Sensor must be
                            existing

        returns:
            A 3d-plot with the data (z-Axis), pixel in x(X-Axis) and pixel in
            y  (y-Axis) will be generated.

        '''


        y_Achse = np.ones(number_of_pixel_y * number_of_pixel_x)
```

```
        d_Achse = np.ones((number_of_pixel_y, number_of_pixel_x ))

        if Input_Sensor == 0:
            for i in range (0, number_of_pixel_y):
                for j in range (0, number_of_pixel_x):
                    y_Achse[(j + i * number_of_pixel_y)] = j
                    d_Achse[i][j] = Image_Data[i*number_of_pixel_x+j]

        plotly.offline.plot({"data": [go.Surface(z=d_Achse)]})
        return

##############################################################################
def draw_ROI (name_of_image):
    """This function opens the given Image and converts it to an grey scale
    image afterwards it gives an interface to draw two different ROIs into the
    image.
    input:
        name_of_image: The name of the image where the ROIs should be drawn.

    returns:
        ROI1: First defined ROI. Array_Like with the image data.
        ROI1: First defined ROI. Array_Like with the image data.
    """

    # create image
    os.chdir("Images to evaluate")
    convert = Image.open('%s' %name_of_image).convert('L')
    convert.save ('%s_grey.JPG' %name_of_image)
    img = pl.imread('%s_grey.JPG'%name_of_image)

    # show the image
    pl.imshow(img, interpolation='nearest', cmap="gist_gray")
    pl.colorbar()
```

```
    pl.title("left click: line segment         right click: close region")

    # let user draw first ROI
    ROI1 = roipoly(roicolor='r')

    # show the image with the first ROI
    pl.imshow(img, interpolation='nearest', cmap="gist_gray")
    pl.colorbar()
    ROI1.displayROI()
    pl.title('draw second ROI')

    # let user draw second ROI
    ROI2 = roipoly(roicolor='b')

    # show the image with both ROIs and their mean values
    pl.imshow(img, interpolation='nearest', cmap="gist_gray")
    pl.colorbar()
    [x.displayROI() for x in [ROI1, ROI2]]
    pl.title('The two ROIs')
    pl.show()

    ROI1 = np.extract(ROI1.getMask(img),img)
    ROI2 = np.extract(ROI2.getMask(img),img)
    os.chdir("..")
    return ROI1, ROI2

"""
Copied from https://github.com/jdoepfert/roipoly.py.
"""

class roipoly:

    def __init__(self, fig=[], ax=[], roicolor='b'):
        if fig == []:
            fig = plt.gcf()

        if ax == []:
            ax = plt.gca()

        self.previous_point = []
```

```
        self.allxpoints = []
        self.allypoints = []
        self.start_point = []
        self.end_point = []
        self.line = None
        self.roicolor = roicolor
        self.fig = fig
        self.ax = ax
        #self.fig.canvas.draw()

        self.__ID1 = self.fig.canvas.mpl_connect(
            'motion_notify_event', self.__motion_notify_callback)
        self.__ID2 = self.fig.canvas.mpl_connect(
            'button_press_event', self.__button_press_callback)

        if sys.flags.interactive:
            plt.show(block=False)
        else:
            plt.show()

    def getMask(self, currentImage):
        ny, nx = np.shape(currentImage)
        poly_verts = [(self.allxpoints[0], self.allypoints[0])]
        for i in range(len(self.allxpoints)-1, -1, -1):
            poly_verts.append((self.allxpoints[i], self.allypoints[i]))

        # Create vertex coordinates for each grid cell...
        # (<0,0> is at the top left of the grid in this system)
        x, y = np.meshgrid(np.arange(nx), np.arange(ny))
        x, y = x.flatten(), y.flatten()
        points = np.vstack((x,y)).T

        ROIpath = mplPath.Path(poly_verts)
        grid = ROIpath.contains_points(points).reshape((ny,nx))
        return grid

    def displayROI(self,**linekwargs):
        l = plt.Line2D(self.allxpoints +
                       [self.allxpoints[0]],
                       self.allypoints +
```

```
                     [self.allypoints[0]],
                     color=self.roicolor, **linekwargs)
        ax = plt.gca()
        ax.add_line(l)
        plt.draw()

    def displayMean(self,currentImage, **textkwargs):
        mask = self.getMask(currentImage)
        meanval = np.mean(np.extract(mask, currentImage))
        stdval = np.std(np.extract(mask, currentImage))

        string = "%.3f +- %.3f" % (meanval, stdval)
        plt.text(self.allxpoints[0], self.allypoints[0],
                 string, color=self.roicolor,
                 bbox=dict(facecolor='w', alpha=0.6), **textkwargs)

    def __motion_notify_callback(self, event):

        if event.inaxes:
            ax = event.inaxes
            x, y = event.xdata, event.ydata
            # Move line around
            if (event.button == None or event.button == 1) and self.line != None:
                self.line.set_data([self.previous_point[0], x],
                                   [self.previous_point[1], y])
                self.fig.canvas.draw()



    def __button_press_callback(self, event):

        if event.inaxes:

            x, y = event.xdata, event.ydata
            ax = event.inaxes
            # If you press the left button, single click
            if event.button == 1 and event.dblclick == False:
                if self.line == None: # if there is no line, create a line
                    self.line = plt.Line2D([x, x],
                                           [y, y],
                                           marker='o',
```

```
                                       color=self.roicolor)
                self.start_point = [x,y]
                self.previous_point =  self.start_point
                self.allxpoints=[x]
                self.allypoints=[y]

                ax.add_line(self.line)
                self.fig.canvas.draw()
                # add a segment
            else: # if there is a line, create a segment
                self.line = plt.Line2D([self.previous_point[0], x],
                                       [self.previous_point[1], y],
                                       marker = 'o',color=self.roicolor)
                self.previous_point = [x,y]
                self.allxpoints.append(x)
                self.allypoints.append(y)

                event.inaxes.add_line(self.line)
                self.fig.canvas.draw()
        # close the loop and disconnect
        elif ((event.button == 1 and event.dblclick==True) or
              (event.button == 3 and event.dblclick==False)) and self.line != None:
            self.fig.canvas.mpl_disconnect(self.__ID1) #joerg
            self.fig.canvas.mpl_disconnect(self.__ID2) #joerg

            self.line.set_data([self.previous_point[0],
                                self.start_point[0]],
                               [self.previous_point[1],
                                self.start_point[1]])
            ax.add_line(self.line)
            self.fig.canvas.draw()
            self.line = None

            if sys.flags.interactive:
                pass
            else:
                #figure has to be closed so that code can continue
                plt.close(self.fig)
```

Appendix 11: Video_processing

```python
# -*- coding: utf-8 -*-
"""
Created on Wed Jun 13 13:25:38 2018

@author: lueb5102
"""


import cv2
from PIL import Image
import numpy as np
import functions.C_Image_quality as Image_Quality
import functions.Image_processing as Image_processing
import os

def Video_CDP_evaluation_from_x_to_y (
        Name_Scene, Image_Information, Start = 10, Stop = 30, Stepsize = 10,
        Input_Sensor = 0,  Name_of_sheet = "results"):
    '''
    Name_Scene: String with the name of the video
    Image_Information: Data from type C_IOData
    Start: Integer
    Stop: Integer
    Stepsize: Integer
    Input_Sensor: Data from class sensor where number_of_pixel_x and
                  number_of_pixel_y be filled with integer if not filled
                  the variable number_of_pixel_x and number_pixel_y have
                  to have integers inside
    number_of_pixel_x: Integer is necessary otherwise Input_Sensor must be
                       existing
    number_of_pixel_y: Integer is necessary otherwise Input_Sensor must be
                       existing
    Name_of_sheet: String. Under this name the results of the CDP-evaluation
                will be stored in a xls-sheet.

    '''
    os.chdir("Videos to evaluate")



```

```
    Vidcap = cv2.VideoCapture("%s" %Name_Scene)
    Success, Frame = Vidcap.read()
    Count = 0
    Success = True
    os.makedirs ("..\\Videos to evaluate\\%s frames" %Name_Scene)
    os.chdir("%s frames" %Name_Scene)
    'Fragment the Video into single frames'
    while Success:

        Success, Frame = Vidcap.read()

        if Success == True:

            X_Size, Y_Size = Frame.shape[:2]
            img = Image.new("RGB",(X_Size, Y_Size), 210)

            for i in range (0, X_Size):
                for j in range (0, Y_Size):
                    img.putpixel((i,j),(Frame[i][j][2], Frame[i][j][1],
                                        Frame[i][j][0]))

            img = img.convert("L")

            img.save("frame_%d.png" %Count) #save as JPEG
            print('Read a new frame: %d ' %Count, Success)
            Count = Count + 1


    'Create an array to write each CDP in the array'
    CDP_frames = np.zeros ((Count-1, (int((Stop-Start)/Stepsize+1))))

    'In this for Loop each Image will be opened and the CDP will be calculated'
    for i in range (0, Count-1):

        'Open the produced single Frame'
        img = Image.open("frame_%d.png" %i)
        number_of_pixel_x, number_of_pixel_y = img.size
        Image_Information.data = np.zeros(number_of_pixel_x * number_of_pixel_y)

        'Write the Information of the Image to an Array'
```

```
        for j  in range (0, number_of_pixel_x):
            for k in range (0, number_of_pixel_y):
                Image_Information.data[number_of_pixel_y * j + k] = img.getpixel((j,k))

        'Convert the Single frame back to real world illumniance'
        Image_Information = Image_processing.DN_to_cd_m2conversion (
                Image_Information)

        ' Evaluate the Contrast from start to stop of the frame'
        CDP_frame, Evaluated_Contrasts = Image_Quality.C_Image_quality.evaluation_from_x_to_y (
                Image_Information.cd_m2, Start, Stop, Stepsize,
                                number_of_pixel_x = number_of_pixel_x,
                                number_of_pixel_y = number_of_pixel_y)

        'Write the results from above into the array'
        for j in range (0, len (Evaluated_Contrasts)):
            CDP_frames [i][j] = CDP_frame[j]
    os.chdir("..")
    os.chdir("..")
    'Write the results from above into an xls-Sheet'
    Image_Quality.C_Image_quality.write_to_xls(Evaluated_Contrasts, CDP_frames,
                                             name_of_sheet = Name_of_sheet)

    return
```

Bibliography

[1] Christian Bloch. *Das HDRI-Handbuch: High Dynamic Range Imaging für Fotografen und Computergrafiker*. 1. Auflage. Heidelberg: dpunkt.-Verl., 2008. ISBN: 9783898644303. URL: http://deposit.d-nb.de/cgi-bin/dokserv?id=2964553&prov=M&dok_var=1&dok_ext=htm.

[2] *Der Weg des ABS vom Flugzeug ins Auto*. 19.03.2010. URL: https://www.auto-motor-und-sport.de/test/abs-die-geschichte-des-anti-blockier-systems/?block=1&private=1 (visited on 08/07/2018).

[3] EMVA. *Standard for Characterization of Image Sensors and Cameras*. 30.12.2016. (Visited on 08/07/2018).

[4] Marc Geese, Ulrich Seeger, and Alfredo Paolillo. *Detection Probabilities: Performance Prediction for Sensors of Autonomous Vehicles.*

[5] Bernd Jähne. *Digitale Bildverarbeitung: Und Bildgewinnung*. 7., neu bearbeitete Aufl. 2012. 2012. ISBN: 9783642049521. URL: http://dx.doi.org/10.1007/978-3-642-04952-1.

[6] Marc Geese. *Image quality and safety in automotive video applications - YouTube*. URL: https://www.youtube.com/watch?v=luiewsmZcrg (visited on 08/12/2018).

[7] Members of the IEEE P2020 Working Group. *IEEE P2020 Automotive Imaging White Paper*. IEEE, 2018.

[8] Hans-Christian Pape et al., eds. *Physiologie*. 7., vollst. überarb. und erw. Aufl. s.l.: Georg Thieme Verlag KG, 2014. ISBN: 9783137960072. DOI: 10.1055/b-002-98019. URL: http://dx.doi.org/10.1055/b-002-98019.

[9] R. Stead et al. *IEEE - SA P2020: Face-to-Face Meeting*. 2017. URL: https://auto-sens.com/wp-content/uploads/2016/06/20170209_IEEE-SA_P2020_BoschMeeting_FINAL.pdf (visited on 07/31/2018).

[10] Ulrich Seeger. *Challenges with video camera image quality in functional safety for autonomous driving - YouTube*. URL: https://www.youtube.com/watch?v=BbK4kywyquE (visited on 08/12/2018).