Technology
Arts Sciences
TH Köln

Project Documentation

Interactive Lighting Detector

written by

Vera Brockmeyer (Matrikelnr. 11077082)
Laura Anger (Matrikelnr. 11086356)

Image Processing in SS 2017

Supervisor:
Prof. Dr. Dietmar Kunz
Institute for Media- and Phototechnology

Contents

1 Introduction

Vera

Digital image forgery is a actual research topic which analyses various image properties to get a conclusion about the authenticity of a digital image. It is almost split into two sections [9]. The first section is the source digital camera identification that analyses the lens aberration, sensor imperfections like defect pixels or by a colour filter array (CFA) interpolation. Those mentioned properties can give an assumption if an image is captured with a known camera or even camera model.

On the other hand, the second section works on the detection of image forgeries. It analyses acknowledges JPEG patterns and to evaluate how often they are saved and with which application or device. The detail observation of the chromatic aberration camera response function (CRF) can give conclusions about manipulations in form of retouching or composing of new image content.

The latter can also be validated by the estimation of the object belonging light direction because the possibility that strange image content has a different light is very high. But this method is also very difficult because it needs an image sequence of the scene to estimate the 3-dimensional light direction successfully with established algorithms. This is not very useful in reality because in most cases only the suspect image is available without any information about its real environment. Nevertheless, there exists an approach by Johnston [7] that can rather estimate the object's light direction under some pre defined assumptions even from single images with a common least square approach. The assumption are clearly defined as a simplified light model in form of an infinite light source and a constant reflection values of the object's surface.

1.1 Motivation

Vera

In the last decade it became more easily to create an image forgery for everyone. Software applications like *Adobe Photoshop* or *Gimp* and almost every mobile phone or social media platform offers possibilities to sophisticate digital images. Every user can manipulate images or compose a new image of various other images without any significant knowledge. For example, two portraits of prominent people can be combined in a way that the impression of a relationship accrues which does not really exist. If this is made in a imperfect way the forgeries can be recognised very quickly by the human visual system but if they are made in a professional way it is quite challenging to detect them.

In this case, a professional analysis tool is required the analyses the suspect item and gives a reliable rating which can stand in court as evidence. One main problem of the analysis is that many approach asks for a sequence of images from the scene for reliable results but in most cases only the suspect image is available. Thus, an approach which estimates the light vector under some refused assumption has to be

used and eventually confirmed by other image forgery detection methods.

1.2 Project Goal

Vera

According to the previous sections, the project goal is to implement and evaluate the approach of Johnston [7] in C++ with an adequate *Qt* Graphic User Interface (GUI). This GUI offers a segmentation of the object of interest as well as an interactive selection of the most suitable part of its contour. The latter is a requirement of the Johnston approach. Several test images have to be captured with a simulated infinite light source. This light source can be the sun at a cloudless sky, for example. A sun clock needs to be added to the test scene to verify the current light direction and various objects with a simple and even shape. All resulting vectors and contours are printed in the actual image or respectively saved in a text file for the following evaluation.

2 State of the Art

Laura

In the following sections the basic scientific knowledge to understand the *Lighting Detector*, whose functionality is explained in section 4, is presented.
A general introduction to image forensic is given in section 2.1. Furthermore, other approaches using light vectors to detect image manipulation are shortly presented in section 2.2.

2.1 Image Forensic

Laura

In [9] the authors claim image forensic to become more important over the years. Furthermore, they divide the field in two approaches. First of all image forensic can be used to identify the recording device of an image. This can, inter alia, be done by taking sensor imperfections, like for example pixel defects, or the lens aberration of the camera into consideration.
The second field of interest is the detection of image forgery [5]. Besides of using the camera response function there can be other details in the image which can be informative whether an image is real or a forgery. For example the light situation in an image must be consistent. This can be proofed by calculating light vectors in various points in the image. The *Light Detector* is using exactly this method (compare section 4). Related approaches are described briefly in section 2.2.

2.2 Related Approaches

Laura

The *Lighting Detector* was implemented according to the paper by Johnson and Farid [7]. The foundation of their assumptions where set in 2001 by the publication of Nillius and Eklundh on an "*automatic estimation of the projected light source direction*" [10]. Where as the earlier theory is taking three dimensional surface normals to determine the light vectors pointing into the direction if the light source, the newer approach by Johnson and Farid uses only one image, and therefore two dimensional surface normals, to achieve the same goal.
An other related approach, which is also presented by Johnson and Farid estimates the three dimensional light direction from the light's reflections in the eyes of human. Therefore they determine the light vector by using the surface normal and the view direction of the person [8].

Hast du noch andere Ansätze gefunden die enen ähnlichen Ansatz verfolgen??? Alle paper die ich sonst gefunden habe, fahren einen anderen Ansatz. HAHA, warum nur?

3 Materials

Laura

The following sections describe the resources and tools required for the completion of the project. Furthermore, the test images are presented in chapter 3.3.

3.1 Hardware

Laura

During the implementation phase, the application was run on two computers, which are described in the following two sections. Both computers needed to be able to deal with the software components described in section 3.2. An extract from your data sheet is shown in table 1 respectively table 2.

3.2 Software

Laura

In order to develop the *Interactive Lighting Detector Qt* was used (compare section 3.2.1). To take advantage of already existing functionalities the *OpenCV*-library, which is described in section 3.2.2, was taken advantage of.

3.2.1 QT

Laura

The *QT Creator* was invented by *The Qt Company*. It is an integrated software development environment in the programming language C++. More functionality can be added by using the Qt project's library, which is called *Qt*. As a cross-platform tool, the *QT Creator* can be used on all common operating systems [12].
Besides extensive database functions and XML-support the software can build graphic user interfaces (GUI).
For this project the algorithm was transcribed in source code using the *Qt Creator* and the GUI was designed in the *Qt Designer* [11].

NAME?	Description
Processor	??
RAM	??
Graphic Card	??
Operating System	??

Table 1: Extract from the Data Sheet of the NAME?

.

Acer Aspire 5820TG	Description
Processor	Intel Core i3 CPU @ 2.40 GHz
RAM	4 GB
Graphic Card 1	AMD Mobilty Radeon HD 5000 Series
Graphic Card 2	Intel(R) HD Graphics
Operating System	Windows 10 Education 64 bit

Table 2: Extract from the Data Sheet of the Acer Aspire 5820TG Notebook.

3.2.2 OpenCV

Laura

The *Open Source Computer Vision* (OpenCV) is an open source library for image- and video processing, which is among others available in the programming language *C++*. It has been introduced ten years ago and is developed by various programmers since then. This library offers the most common algorithms, as well as current developments in image processing [4].
On the case of the implementation of the *Light Detector* the library was mainly used for the detecting of the contours (compare section 4.1) and solving the minimization problem (compare section 4.4) introduced by Johnson and Farid [7].

3.3 Testimages

Laura

Due to the assumption that the objects shown on the test images described in section 3.3.1 have a too complicated shape, a second batch of images was made (compare section 3.3.2). Images of both batches were used to test the functionality of the the algorithms used for the lighting detection. All images have in common that besides the actual object they show a sundial to simplify the determination of the light direction for the user.

3.3.1 First Batch

Laura

Four examples of the first batch of test images are shown on figure 1. Next to the mandatory sundial there are different objects depicted, like a helmet, a handbag, a bucket or a hot-water bottle. Those objects differ in their surface texture, as well as their size. They are shot from different angles to produce different light directions. it is necessary that the objects are not trimmed at the boarders of the image, because the algorithm requires a full contour of the selected object.

Figure 1: Examples of the Test Images of the first Batch.

3.3.2 Second Batch

Laura

In contrast to the first batch, the test images described in this section show easier objects with a round surface. As depicted in figure 2 all images show the madatory sundial and one or two table tennis balls in yellow and pink, which have a matt texture. For the actual algorithm of the *Light Detector* only one of this balls is taken into consideration (compare section 4). Properties like size of the object, the camera angle and the lighting direction differs in each image.

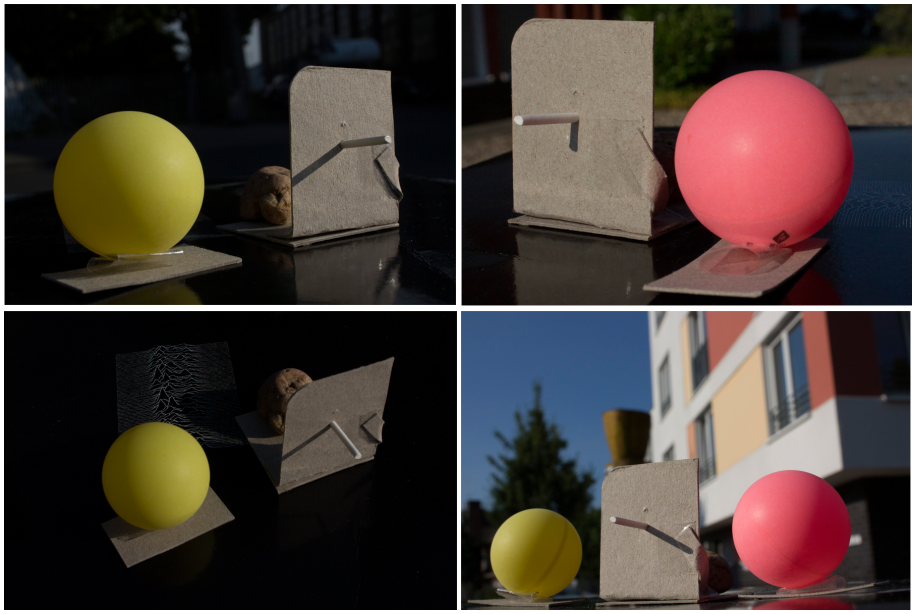


Figure 2: Examples of the Test Images of the second Batch.

4 System

Vera

This chapter describes the contour searching method and different approaches to detect the light direction of an object in detail. In the beginning, the contours of an object of interest are generated in the application to get all pixel coordinates along an adequate contour patch. Followed by the actual computations to ascertain the light direction of an infinitive light source that shines on the object belonging to the contour patch.

4.1 Contours

Vera

To estimate the light direction of an object in a digital image, it is firstly needed to locate the object of interest and get any shape informations. All informations are described by the contour which is forced to be perfectly even to guarantee a sucessful light direction estimation. The most important required informations are all pixel coordinates along the outer bound of the object in an ordered way.

Beside a correctly calculation, an application with a good usability is preferred as well. An pre implemented interactive segmentation [**?**] was integrated into the application to generate the affordable contours with manual constraints that are defined by the user. This method is called *Live-Wire* [1] and based on a maximum flow problem. However, several test showed that this extension calculates noisy and fault contours as well as lacks of a robust use control. The main problem was that the generated contour seems to follow the valley between two edges of an RGB image and snap to the most significant edge in a wide neighbourhood. Even more constraints set by the user did not result in adequate and useful shapes.

According to this bad results, the team decided to extract the contours from perfectly even mask images of the objects of interest like in in Figure 3. These images are made in manually in *Adobe Photoshop CS 6* with the pen tool that offers perfectly round and even Bezier Curves. All created masks are shrink about almost two pixels to ensure that only pixel informations of the actual object are used for further proceedings and none from the background.

If the file name of a mask image of interest is the same as the belonging RGB image with the extension `_mask.jpg`, it will be loaded automatically into the application at the same time when the belonging RGB image is selected. This mask data will be used for all further contour calculations.

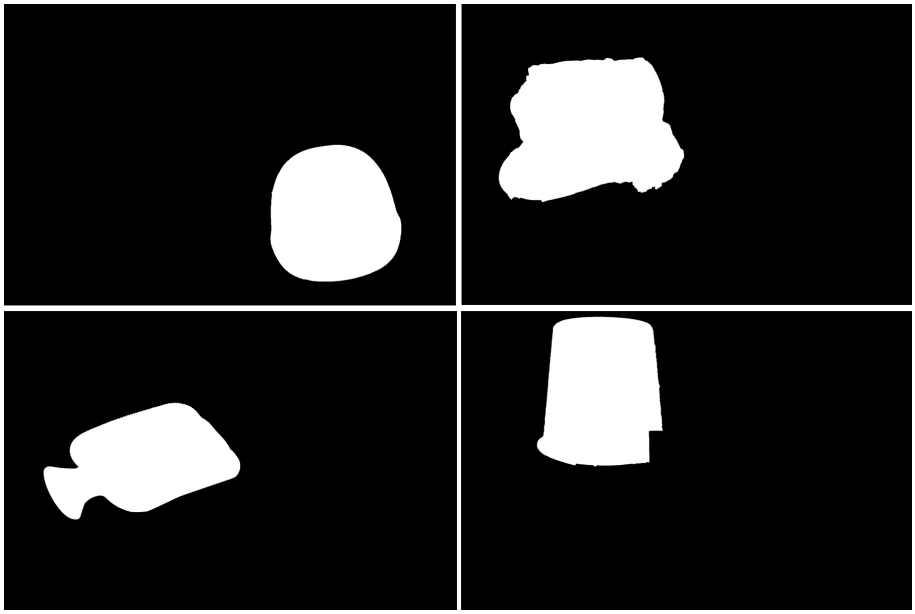


Figure 3: Manual created mask images of the first batch.

4.1.1 Calculate Contours

Vera

Several *OpenCV* functions are used to compute all required pixel coordinates along the outer object bound. In the beginning a common erosion filter with a filter size of three is applied to the mask image to eliminate small outliers and to shrink the mask to ensure that it is not located directly or outside the object bound. In the next step, a Canny filter with a filter size of five is used to extract the edges. Then the *OpenCV* function `findContours()` calculates the vector representations of the binary canny image with the border following algorithm of Suzuki et Al. [13]. All results are not optimized by any approximations because otherwise important coordinate data will be lost. The resulting contours will be printed into the preview of the GUI to allow the user a selection of the wanted sub-contour to run the light direction estimation.

4.2 Generate Subcontours

Vera

As mentioned, the user is forced to select an adequate part of the contour to contain the further calculations. He can do so in the GUI with a simple rectangle selection in the preview. This selection can be renewed, if affordable, and saved. After the storage all points of the contour which are within the rectangle are stored and sorted in a so called sub-contour. At this point, the sort is needed because all contour points are stored like a ring buffer that starts at the highest point and then follows counter clock wise the boundary. Thus, it has to be reordered to ensure an open and gapless sub-contour, if the selection includes the top of the complete contour.

After this process, we have an sub-contour that includes all important edge points along the boundary but it still lacks of all pixel coordinates along this part of the boundary. This circumstance can be solved by making a binary image and print the required sub-contour into it with a line thickness of one pixel. Then a *OpenCV* `LineIterator` is applied to this image. It is treated as versatile implementation of the Bresenham algorithm [3] and allows the notice and storage of every required coordinate. Now, these coordinates can be used for the following approaches of the light direction estimation in a single digital image.

4.3 Illumination Model

Vera

To understand the following approaches, it is useful to give a short introduction to the definition of an illumination model. One of the most acknowledged illumination model is the Blinn model [2](see Equation 1) which defines a basic illumination in a 3D space. This illumination is a sum of ambient a, diffuse d and specular sp light parts where $\vec{n}$ is the surface normal, $\vec{l}$ is the light direction vector and $\vec{h}$ is the direction of the maximum highlights. The p are variables that are related to the proportions of each reflection type which are part of the sum and s is the amount of the specular reflection.

$$I(x, y, z) = (\vec{n} \cdot \vec{l}) * p_d + (\vec{n} \cdot \vec{h})^s * p_s + p_a = d + sp + a \tag{1}$$

Due to the number of unknown variables and vectors, the Blinn model is too complex to calculate the 3-vector $\vec{l}$ with standard approaches like the least square method. Hence, some assumptions to simplify the mode had to be made [7]. The first assumption is to state the the current surface reflects light isotropically. This surface has a constant reflectance value r and is illuminated by a faraway infinitive light source. Finally, the angle between $\vec{n}$ and $\vec{l}$ is in the range of 0° and 90°. All previous assumptions lead to the simplified light model of equation 2.

$$I(x, y, z) = r(\vec{n} \cdot \vec{l}) + a \tag{2}$$

4.4 Different Approaches

Laura

Because no reliable solution could be found, three different approaches were tried out. All three approaches are based on the paper of Johnson and Farid [7]. Whereas two of this approaches remained unchanged (compare section 4.4.1 and 4.4.2), the last approach takes the assumptions of Johnson and Farid and expand them (compare section 4.4.3).
To standardise the explanations in the following sections a list of necessary variables is introduced:

- $\vec{L}$: vector, which points in the direction of the light source (3D)
- $\vec{N}(x, y)$: surface normal at the point (x,y) (3D)
- $I(x, y)$: intensity at the point (x,y)
- A: constant ambient light term
- R: constant reflectance value

All equations in this section, as well as its subsections, are taken from [7]. The basic idea behind the approach of Johnson and Farid is summed up in equation 3.

$$I(x, y) = R(\vec{N}(x, y) \cdot \vec{L}) + A \tag{3}$$

In the following sections we do not have a three dimensional room, as all evaluations are made based on images. All of them have in common, that their assumptions are based on a infinite light source in a two dimensional image space. Therefore $\vec{L}$ and $\vec{N}(x, y)$ needs to be redefined as follows:

- $\vec{L}$: vector, which points in the direction of the light source (2D)
- $\vec{N}(x, y)$: surface normal at the point (x,y) (2D)

For the second and the third approach, the light vectors are summed up in patches of size four.

4.4.1 1. Approach: One Lightvector

Laura

The first approach tries to simplify the two dimensional case by solving equation 4 and getting one finale light vector per contour of an object plus the ambient light term. The Matrix M can be found in Johnsons and Farids paper [7] in equation (6) and p denotes the number of points on a contour with the same assumed reflectance. All other variables are explained in section 4.4. Thereby the reflectance along the entire contour is thought of as constant.

$$E(\vec{L}, A) = \left|\left| M \begin{pmatrix} L_x \\ L_y \\ A \end{pmatrix} - \begin{pmatrix} I(x_1, y_1) \\ I(x_2, y_2) \\ \vdots \\ I(x_p, y_p) \end{pmatrix} \right|\right|^2 = \left|\left| M\vec{v} - \vec{b} \right|\right|^2 \tag{4}$$

This options can be activated in our script *mainwindow.cpp* by setting *usePatches* to *false*. Equation 4 is than solved using *Singular Value Decomposition*. Then the lightvector $\vec{L}$ is drawn onto the test image to visualize the result.

4.4.2 2. Approach: Averaging Lightvectors

Laura

For the second approach it is not only assumed, that the light source is infinite, but also that the reflection created by it is constant within each surface patch. The basic idea can be found in section 2.2.1 in the paper of Johnson and Farid [7].
It is assumed, that a minimization problem according to equation 5 needs to be solved. The Matrix M can be found in [7] in equation (8).

$$E_1(\vec{L}^1, ..., \vec{L}^n, A) = \left|\left| M \begin{pmatrix} L_x^1 \\ L_y^1 \\ \vdots \\ L_x^n \\ L_y^n \\ A \end{pmatrix} - \begin{pmatrix} I(x_1^1, y_1^1) \\ \vdots \\ I(x_p^1, y_p^1) \\ \vdots \\ I(x_1^n, y_1^n) \\ \vdots \\ I(x_p^n, y_p^n) \end{pmatrix} \right|\right|^2 = \left|\left| M\vec{v} - \vec{b} \right|\right|^2 \tag{5}$$

This options can be activated in our script *mainwindow.cpp* by setting *usePatches* to *true* and *useHighestIntensity* to *false*. As stated in section 4.4.1 equation 5 is solved using *Singular Value Decomposition* as well. This leads to calculating one light vector per patch. To get the final vector $\vec{L}$ all of this light vectors are averaged.

4.4.3 3. Approach: Lightvector with highest Intensity

Laura

This last approach uses the same basic idea as the second one described in section 5. It is also based on dividing the surface normals $\vec{N}$ into patches and solving equation 5 using *Singular Value Decomposition*. Afterwards there is no averaging done, but the vector $\vec{L}$ belonging to the patch with the highest intensities is taken as the final light vector.
This idea is not described in the paper of Johnson and Farid [7]. It was a try make the results of the *Lighting Detector* more trustworthy.
This options can be activated in our script *mainwindow.cpp* by setting *usePatches* to *true* and *useHighestIntensity* to *true*.

5 Evaluation

Vera und Laura: Stichpunkte

Vera: Ausformulierung

Stichpunkte Laura:
Allgemein:

- es ist auffällig, dass die autoren bei den Komplexeren Aufnahmen (mit den Promis) keine einzelne Lichtvekoren mehr einzeichnen.
- Es wurde versucht möglichst homogene Objekte zum testen zu nehmen (vor allem der zweite Satz der Testbilder)

Beobachtungen unserer Versuche:

- lediglich eine visuelle Auswertung
- Wir bekomen immer wieder Vektoren, die in die richtige Richtung zeigen (bzw. um 180 gedrehte Vektoren) aber der Algorihtmus funtkioniert nicht zuverlässig
- Da Ergebnisse eher zufällig wirken, macht es keinen Sinn die drei Ansätze direkt miteinander zu vergleichen, der zweite Ansatz scheint jedoch am häufigste richtig zu liegen.

6 Project Management

Laura

In this section the planning and the management of the *Lighting Lab* is described. As there were only two persons involved this project the following part is going to be a bit more compressed than the reader might expected it.
To give the customer an all-encompassing idea about the *Interaction Lab* a project proposal was handed in, which is written in German. It is divided in two sections. First of all a motivation is given and afterwards the exercise, which should be implemented is determined. It is translated into English and can be found in section sec:ProjectProposal.
Before starting the actual project it is wise to make some mindful project plans in order to organize the project. First of all a catalogue of requirements and specifications (compare section 6.2) was written to formulate all important basic information of the project.
The project structure plan, which is presented in section 6.3, serves to arrange the work packages of the project in a sorted way. The relating milestone description can found in section 6.4.
For staying in time while the implementation phase, a time exposure is required (compare section 6.5).
A list with all early detectable risks can be found in the table in section 6.6.
In section 6.7 a final resume about the changes during the actual realisation of the project is given.

6.1 Project Proposal

Laura

Motivation

Due to progressive changes in image processing programs and high-resolution images, the manipulation of digital images is became easier. A common form of manipulation is the so-called "*image splicing*". Therefore image regions of at least two images where combined to a new one. The transitions between the individual image parts can get invisible for the user by using accurate and user-friendly image processing tools. According to [6] there is is still no algorithm, which makes images completely forgery-proof by adding watermarks to it. There is put much effort in the integrity of images and the recognition of manipulated image parts, despite the lack of prior knowledge of the image content. This should be improved by the *Interaction Lab*. An algorithm should be established, which proofs the consistence in light directions on various surfaces presented on an image. The light vectors will be conveniently estimated to give an assertion on whether the image is manipulated or not.

Work Steps

Until now there is no implementation given by *OpenCV*, which estimates the light

vector of an infinite light source on one surface and compares it with the according vector of another surface in the same image. An approach in the programming language *C++* should be implemented using the assumptions of Johnson and Farid [7] to compute, analyse and visualize the light vectors.
The following work stages are necessary:

- Creation of test images with an infinite light source, e.g. in nature on a sunny day
- Implementation of a GUI for visualize the approach
- Implementation of the algorithm of Johnson and Farid [7]
- Optional: The approach might be amended by also taking a local (not infinite) light source into consideration [7]

6.2 Catalogue of Requirements and Specification

Laura

Project Manager: Laura Anger
Team Member: Vera Brockmeyer
Supervisor: Prof. Dr. Dietmar Kunz

Project Goals

This project aims to implement an interactive image forensic tool to detect lighting detections of various objects in an image. The tool is limited to images with infinitive light sources. All resulting direction vectors are compared with each other to give a statement if the image is partly digital modified.

The development of the tool requires the following parts:

- a set of test images with a simulated infinitive light source
- an interactive partial contour detection tool
- a calculation and validation of lighting detection vectors
- a graphic user interface which includes Live-Wire and visualisation tools

The project will be realised in the programming languages C++ with the OpenCV, the Dlib C++ Library and the Qt Library. The release of the project is planned on 4th August 2017.

List of Requirements

Set of Test Images

A set of ten test images is required to validate the image forensic tool. These images have to be captured with an simulated infinitive light source. Thus, all image are taken at a sunny day with no cloudy heaven. Each image shows either different types of objects with surfaces and contours like reflecting metal and glass, diffuse natural tissues as well as different persons. Another interesting option is the validation of concave and convex objects. Furthermore, a sun clock is placed in each image to display the current lighting direction precisely. Finally, all collected test images are partly composed together with images of other lighting.

Calculation and Validation of lighting detection vectors

The Method of the image forensic tool to calculate and validate lighting detection vectors of objects in an image is given by the publication of Johnson [7]. Johnson offers an algorithm that calculates the lighting direction of infinite light sources by minimizing several samples of direction vectors along a contour segment of an object. Each sample is computed by extracting and minimizing features from the inner and nearby neighborhood of a contour patch. Every required minimization is computed by the least square method of the Dlib C++ library. After a successful computation of the vectors the method computes angles of all calculated vectors and compare them with each other. If the difference between them is off a predefined threshold, the object has probably been retouched into the image.

Interactive contour detection tool

The interactive Live-Wire tool [1] is going to be realized to compute the required surface contour segments of the objects. This tool serves a segmentation algorithm which is initialized and controlled by seed points. Every seed point is set by the user with the mouse cursor and a cost minimisation of local features computes the optimal path between the last two seed points. These required features are extracted with OpenCV functions and minimised with the Dlib C++ library. Furthermore, all specified seed-points and resulting contour segments between them are previewed in the GUI.

Graphic User Interface (GUI)

The GUI includes two main parts. First, the interface of the Live-Wire tool, which provide the interactive generation of the contour segment in the current image. This image is opened and displayed in a preview panel. The second part serves visualisations of the computed final direction vectors, all vectors of each contour patch as well as an indication of the digitally modified image sections. For the implementation is the QT Library used.

Functionally test of the prototypes

All prototypes are tested with the predefined set of images with a simulated infinitive light source. For the second prototype, the accuracy of the Live-Wire is going to be

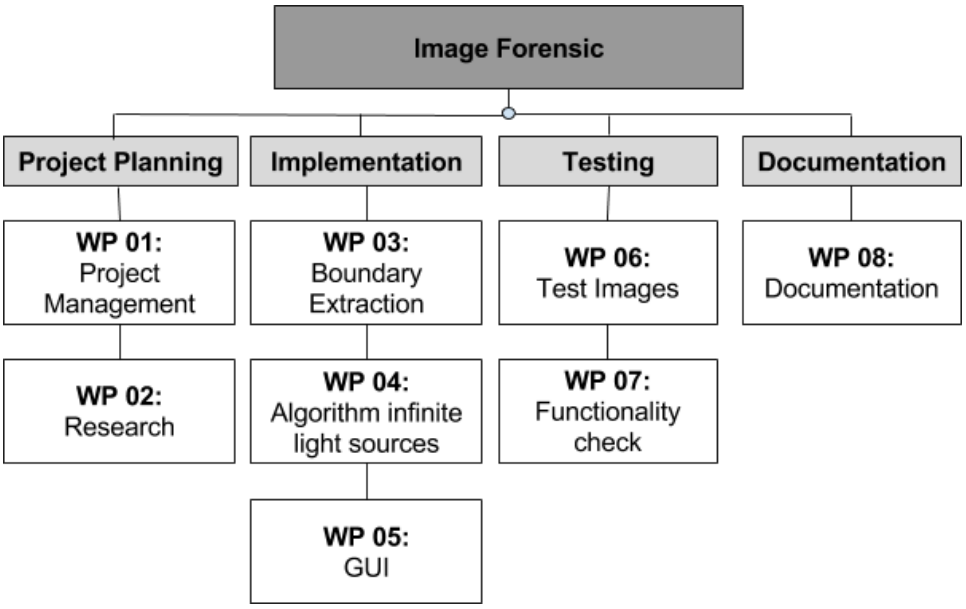
tested visually by the developers. On the other hand, the angles of each computed direction vector are going to be compared with the angle of the sun clock shadows to validate the functionality of third prototype.

Dates

There will be a first prototype in form of a Paper Mock Up on the 20th of April 2017 and a second prototype on the 9th June 2017. The final version of the Interactive Lightning Detector will be realised until the 17th of July 2017. The deadline for this project is the 4th of August 2017.

6.3 Project Structure Plan

Laura



6.4 Milestones

Laura

Number	Milestone	Description	How to get there	Planned Date
1	Start Implementation		Tasks 1.1, 1.2 and 1.3 needs to be done before, as well as WP 2.	17.04.2017
2	First Prototype (GUI)	Paper Mockup of the GUI should show the possibilties for the future user.	WP 2 should be concluded.	20.04.2017
3	Second Prototype	GUI is working, as well as the edgedetection	Mostly MS 1 and MS 2 must finished.	09.06.2017
4	Third Prototype	Algorithm by Johnson is implemented and working.	Mostly WP 2, WP 3 must finished.	10.07.2017
5	Final Version	Implementation is ready.	All bugs found while testing, must be fixed.	17.07.2017
6	Submission	This includes source code and the documentation	The final version of the Image Forensic Tool is ready, as well as the documentation.	04.08.2017

6.5 Time Exposure

Laura

Number	Workpackage	Subpackage	Planned Time Period	Working time in days			Description
				minimum	average	maximum	
1	WP 01: Project Management			3	4	5	
1.1		Create Projectdescription	27.03.2017 - 01.04.2017	1	1	1	Brief description of the project. Can be found in our Ilias folder.
1.2		Define Workpackages	01.04.2017 - 10.04.2017	1	2	3	Includes the definition of the Work packages (project-structure- plan. pdf in our Ilias Folder), as well as this time exposure.
1.3		Calculate Risks	01.04.2017 - 10.04.2017	1	1	1	This plan describes the risks, which may arise. Will be found in our Ilias Folder.
2	WP 02: Research			8	12	16	
2.1		General Topic Search	13.03.2017 - 27.03.2017	3	4	5	
2.2		Boundary Extraction	03.04.2017 - 17.04.2017	2	3	4	Search a semi-automatic algorithm and become familiar with its possible implementation
2.3		Algorithm of Johnson	03.04.2017 - 17.04.2017	2	3	4	Understand the algorithm of Johnson and find its mathematical basic functions in a library
2.4		GUI Implementation	03.04.2017 - 17.04.2017	1	2	3	Find out about the possible ways to build a GUI in c++
3	WP 03: Boundary Extraction			8	12	14	
		Implementation of Live-Wire	24.04.2017 - 09.06.2017	8	12	14	Implementation of the interactive Live-Wire tool; interagtion of the tool in the GUI
4	WP 04: Algorithm of Johnson			6	11	18	
4.1		Compute Lighting Vectors as Johnson	10.06.2017 - 10.07.2017	4	7	12	Exctract Features for Lighting Vector Computation, compute Minimizations with Math Lib
4.2		Validate Lighting Vectors	10.06.2017 - 10.07.2017	2	4	6	Measure Angles of each surface and validate them
5	WP 05: GUI			3	5	7	
5.1		Basic Surface	24.04.2017 - 09.06.2017	1	1	1	The GUI frame can be seen on the screen and basic functions, like loading a new image are included
5.2		Boundary Extraction	24.04.2017 - 09.06.2017	1	2	3	The GUI gives the user the possibilty to mark areas in the image to allow the algorithm described under 3 to run the semi-automatic boundary extraction
5.3		Visualisation	24.04.2017 - 09.06.2017	1	2	3	Feautures like object boundaries and the light vectors can be drawn into the images.
6	WP 06: Test Images			2	2	2	
6.1		make images	**15.05.2017-21.05.2017**	1	1	1	Pictures with an infinite light source are needed. The presented objects should differ in number and form as well as viewing angle.
6.2		select images		1	1	1	An adequate number of useful images must be selected
7	WP 07: Functionality check			3	5	7	
7.1		Testing	10.07.2017-14.07.2017	2	2	2	External Parties should be invited to test the application. This includes NO usability study.
7.2		Correction	14.07.2017-17.07.2017	1	3	5	The most pressing issues must be fixed.
8	WP 08: Documentation			12	18	24	
8.1		write documentation	17.07.2017 - 03.08.2017	10	15	20	Description of the actual system as well as its functionality.
8.2		add comments to source code	17.07.2017 - 03.08.2017	1	2	3	Clean up the soucre code and add descriptions.
8.3		print documentation	04.08.2017	1	1	1	Print documentation and submission.
			Estimated Duration	**45**	**69**	**93**	

6.6 Risks

Laura

Description and Source of the Risk	Possible Area of Appereance	Effects	Preventive Measures
1. Technical Problems			
1.1 Implementation Problems	mostly Implementation	Some parts of the implementation might take longer than planned. Maybe other solutions need to be taken into consideration	Try to get to know how complicated the implementation is during the research period
1.2 Algorithm	mostly Implementation	Sometimes the described algorithm is limited. For instace the described results can only be achieved using a specific type of input data.	One should read the according paper carefully. Sometimes more or less hidden hints are given on how good the algorithm works.
1.3 GUI latency	Preview of image	preview of image has a latency which is reconsiable for the users	regard an low costly implementation and prevent from unnecessary memory and code implementations
1.4 wrongness of Lighting Vectors	concave and convex objects	lighting direction is misinterpret because for concave objects the lighting seems to come from the opposite direction	observe problem and figure out characteristics of the problem
2. Time Problems			
2.1 time-collision with other projects	all Workpackages	On Task or a whole Workpackage can't be finished in time.	Try to plan all upcoming projects as detailed as possible to avoid time overlap
2.2 false planning	all Workpackages	It is not possible to stick to the Sheduling	Try to make a thoughtful project planning and do not forget buffer time
2.3 illness	all Workpackages	Restriction of workability and thus problems with finishing a task or a even a whole workpackage in time.	

6.7 Final Conclusion Projectmanagement

7 Conclusion

Vera und Laura

References

[1] William A. Barrett and Eric N. Mortensen. Interactive live-wire boundary extraction. *Medical Image Analysis*, 1(4):331 – 341, 1997.

[2] James F. Blinn. Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198, July 1977.

[3] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965.

[4] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek. A brief introduction to opencv. pages 1725–1730, May 2012.

[5] H. Farid. Image forgery detection. *IEEE Signal Processing Magazine*, 26(2):16–25, March 2009.

[6] Yu-Feng Hsu and Shih-Fu Chang. Detecting image splicing using geometry invariants and camera characteristics consistency. In *ICME*, 2006.

[7] Micah K. Johnson and Hany Farid. Exposing digital forgeries by detecting inconsistencies in lighting. In *Proceedings of the 7th Workshop on Multimedia and Security*, pages 1–10, New York, NY, USA, 2005. ACM.

[8] Micah K. Johnson and Hany Farid. Exposing digital forgeries through specular highlights on the eye. In Teddy Furon, François Cayre, Gwenaël J. Doërr, and Patrick Bas, editors, *Information Hiding*, volume 4567 of *Lecture Notes in Computer Science*, pages 311–325, 2008.

[9] T. Van Lanh, K. S. Chong, S. Emmanuel, and M. S. Kankanhalli. A survey on digital camera image forensic methods. In *2007 IEEE International Conference on Multimedia and Expo*, pages 16–19, July 2007.

[10] P. Nillius and J. O. Eklundh. Automatic estimation of the projected light source direction. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–1076–I–1083 vol.1, 2001.

[11] Qt. Qt Designer Manual. http://doc.qt.io/qt-5/qtdesigner-manual.html. Aufgerufen: 13. Juli 2017.

[12] Ray Rischpater. *Application Development with Qt Creator - Second Edition*, volume 2. Packt Publishing, 2014.

[13] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985.