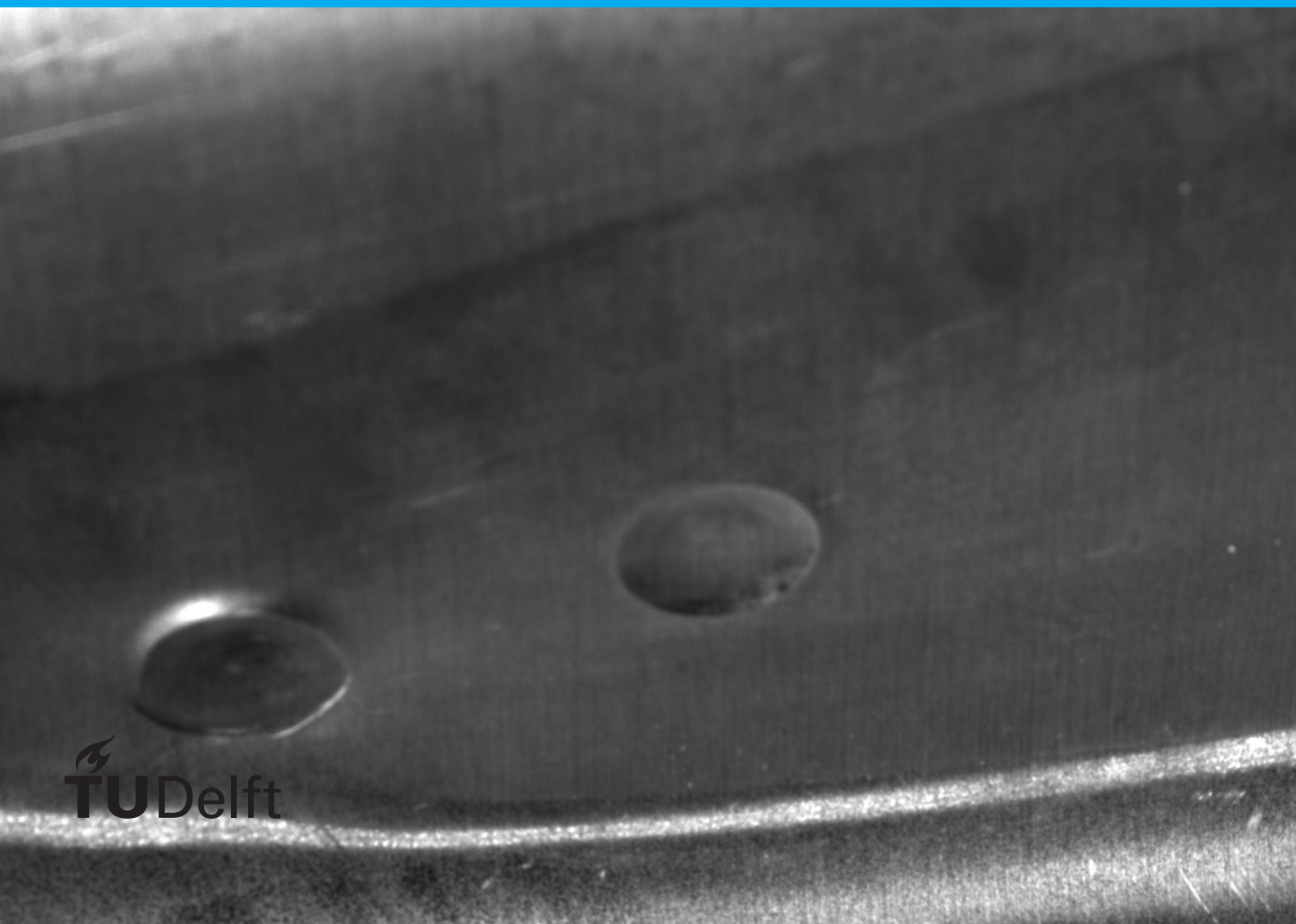
Illumination normalization for Industry 4.0

Specular reflection removal from non-dielectrics

Rick Dijksman

Bachelor Thesis



TU Delft

Illumination normalization for Industry 4.0

Specular reflection removal from non-dielectrics

by

Rick Dijksman

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday February 11, 2022 at 03:00 PM.

Student number:	4912233	
Project duration:	November 10, 2021 – January 19, 2022	
Thesis committee:	Dr. O. Soloviev,	TU Delft, supervisor
	Dr. ir. C. Smith,	TU Delft

TU Delft

Abstract

With the introduction of machine learning algorithms in industry came a new improvement to excising production line. Image recognisance can now be used to detect defects of the product made by these production lines. However these kinds of flaw detection can be fooled by mirror-like specular light effect on these products, which can reduce the reliability of these detection methods. In this thesis two approaches are taken to minimize these spurious specular light features. Here I look at images of welding spots taken from a car-door production line of Fiat Chrysler Automotive Italy. First a median filter is used on multiple images of welding spots, which each have a different illumination. Secondly the same images were fed into a modified algorithm developed by Antonello et al,2018 , originally used to split fore-and background of videos, to remove these spurious features. Then I tried to improve the output of this algorithm so that oversaturated pixels in the output were replaced by a combination of pixel values of the input images that were not oversaturated. Here is found that the median filter does not satisfy the desired goal of this thesis. The algorithm however gives a good filtered output of the input images and therefore are suitable to possibly be used for computer vision applications. The improvements made to the output of the algorithm do not correct the oversaturated pixels the right way, however the method I propose here seems to have some promising results if some improvement to this method are made, mainly concerning the normalisation of these pixels.

Contents

1 Introduction

For a production line to function properly, some form a quality control is needed to ensure the product is good enough to be used or further processed. In this thesis I am considering the welding of car doors by FCA Italy (Fiat). Currently, they do there quality control by letting a human check those welds on possible defects. This limits the speed of the welding line. Preferably the quality control is done on-the-go. This can be achieved by attaching a machine vision head on a robot arm which follows the door. The machine vision head consists of a camera and illumination options. Machine learning algorithms can be used in combination with this machine vision head to detect possible defects. However due to the difficult illumination conditions in combination with the shiny metal, unwanted features like glints can appear in the machine learning classification, which lessens the reliability of the defect detection.
The goal of this thesis is to minimize these unwanted illumination effects so that the reliability of these algorithms is increased. Hereby I try to model the images as a combination of a diffuse-illuminated picture and unwanted specular light features like glints. After that I try to remove the second part of the image to get a illumination-invariant picture of the weld.
I feed multiple images of the same spot, but with different illuminations into a algorithm. This algorithm then tries to remove any glints and maximizes the given information of the images into one picture. This algorithm is a modified algorithm which is used to separate fore-and background in for example webcam videos of streets. I also use a median filter to try to achieve the same goal.

2

Theoretical Background

2.1. Diffuse versus specular reflected Light

When light hits a surface coming from air, two things can happen. Either the light is directly reflected from the surface, which is called specular reflection, or the light penetrates the surface. When the light penetrates the surface, the light can be scattered by all the atoms inside the material it is entering. Eventually the light is either transmitted trough the back of the material or reflected back to the front surface, which is called diffuse reflection. The latter is further consider in this thesis, because we only look at the front of the material.
When the light is specular reflected, the direction of reflection is determined by the law of reflection, however the actual direction can differ because of microscopical differences in the surface normal. With the diffuse reflection, the light is first scattered randomly by the atoms in the material, so when it comes back to the surface, we can assume that the direction has a diffused distribution. The difference of the kinds of reflections are illustrated in figure 2.1.

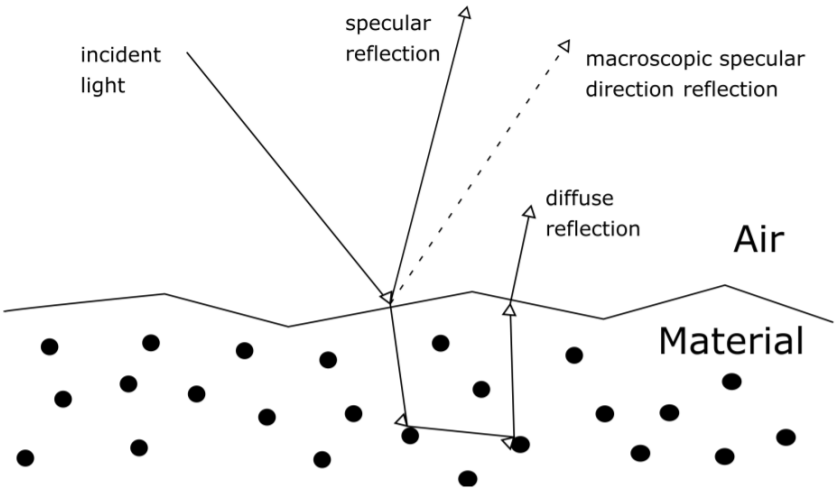


Figure 2.1: Overview of possible reflections of light.It can be reflected specular directly or first be scattered by the particles and then diffusely reflected. [3]

The difference between specular and diffuse reflection in real life can be easily seen if a picture is taken of a metallic object with or without using a flashlight. In figure 2.2 (a) a bright spot in the middle of the rod can be seen, this is the specular reflection caused by the flashlight of the camera. In figure 2.2 (b) no flashlight is used and therefore almost no specular reflection is visible and thus almost all is reflection is diffuse. These picture illustrates rather good why diffuse reflection is beneficial for flow detection, as you can see the scratches a lot better in the second picture.

(a) Specular reflection (b) Diffuse reflection

Figure 2.2: Difference between diffuse and specular reflection on a metal rod.

2.2. Construction of pixels in the image

For this thesis a simple model for the construction of the pixels is assumed proposed in [2] . Hereby is assumed that the light the falls on the nth pixel of a senor is a multiplication of the illumination of the scene and the reflectance or transmission of the object into that pixel. Then I assume that the sensor linearly convert the illumination into a voltage $U[n]$. So the output voltage of the sensor can be approximated by formula 2.1,

$$U[n] = b[n]s[n] + v[n] \tag{2.1}$$

, where $U[n]$ is the output voltage, $b[n]$ is proportional to the illumination of the empty scene, $s[n]$ the value of the transmission of reflectance of the object and $v[n]$ is the noise produced by the camera. This voltage value is then converted by a Analog to Digital Converter to a digital value between zero and one, corresponding to the grayscale value.

2.3. Existing Methods

First we will look at some existing methods to remove specular features on images.

2.3.1. High-pass Filter

A high pass-filter can remove any DC or low frequencies component of the picture and keeps only the high frequency features of interest[2]. Since the illumination is mostly consist of low frequency features, we can remove them with a high-pass filter. This makes all edges clearly visible, but because all the low frequency features are removed, the image does not really represent the view you will see in real life. As we are trying to create a realistic image, this method is not suitable for this purpose.

2.3.2. Polarization based filtering

In this method the idea is used that diffuse reflections tend to be less polarized than specular reflection. This can be explained by the nature of specular light. Because this light is reflected directly on the surface, so just one time, compared to the diffuse reflections which are reflected many time. If the light is reflected, part of the light is polarized parallel to the surface. If this happens one time (like with specular light) the resulting ray is partly polarized. For diffuse light this happens many time with different surface orientations, so this light is much less polarised then the specular light. Because of this the specular light intensity entering the camera can be change by putting a polarization filter before the camera and changing its orientation. Therefore a part of the specular light can be filtered away if the orientation of this polarization filter is right. This method is described by Wolff et al [6]. I did not create my own images for this thesis and therefore did not have the opportunity to use any polarization filter when creating the images. Also when using this approach, one only removes a part of the specular light and also a part of the diffuse light.

2.3.3. Removal of Specular Reflections from Image Sequences using feature correspondences

Syed et al [5] proposes a method based on the comparison of sequence of images of the same scene. They compare the different images, where the object is spatially moved, by recognising feature point on those images and then create a overlapping area where they have multiple images of the same area. Because the object was moved the illumination of the overlapping area is different for each image. Because specular point are characterized by high pixel values compared to diffuse pixels, the minimum value of this spot is taken to obtain the value corresponding to the diffuse pixel.This does required at least one image with a diffuse pixel on that spot, otherwise it will still be specular. This approach is based on moving the object of interest relative to the camera, where in the situation of this thesis the object and camera are stationary relative to each other. However the fact that for some of the images a certain spot contains only diffuse light is used in this thesis.

2.4. Julia

The images of the doors in this thesis are processed in a high-level programming language called Julia. Julia is a fairly new language which was launched in 2012. Before I started this research I only had experience in Python, but since syntax of Julia is quite similar to that of Python, it was able to familiarize this new language quite fast. This language was recommended to me by Dr. Soloviev for two main reason. Firstly, the goal of the thesis is to detect the defects in the door on a real time basis. This requires the whole detection sequence to be fast. Julia is a lot faster then Python, because of the way it runs the code. Julia uses a just-in-time compiler which optimizes the way a part of code is executed the first time you run this part of the code. This means that if you run a particular part of the code for the first time, it takes some time to run. But when you then run it the next time, it runs significantly faster. Because in this application only the input images change and not the code, this way of working is very suitable for these kinds of uses. The second reason I used Julia for this thesis is because of a powerful package called StructuredOptimization is written in Julia. This tool plays a really important part in the processing of the images in this thesis and will be further explained in the experimental method.

3

Experimental Method

To achieve to goal of this thesis, I make use of two ways of removing the spurious illumination features. For the first method, a median is taken from a combination of different frames of the same weld. In the second approach, a algorithm designed to split the foreground and background of a video is modified to fit the purpose of the thesis. These methods and how I implemented this are discussed in this chapter.

3.1. Median filter

The median of a set is defined as the value that splits the higher half from the lower half of the set. If the set is ordered for the lowest to the highest value, the median is the value in the middle of this set.

For this thesis I used a median over the different frames of the doors. The median of the values of each pixel from every frame considered was taken to construct a new image with these median values. I used this method because big outlier values do not influence the median as much as for example a mean of those values. This is a good characteristic because for this case big outliers are mostly oversaturated pixels, which do not contain any information and therefore it is desirable that they do not influence the output of the filter that much. For the FCA data set we calculated the median over the 4 pictures of the same spot. For the DCSC case we calculated the median with varying amounts of frames to see how many frames gave the best result. The Julia code I used to accomplish this is shown below.

```
using Images, Statistics

imgpath="./preliminary_data/door941_jul20NOK/spot0"

img = load("$(imgpath)_1.tif") # load first frame
l = 4                          # number of frames
n,m = size(img,1),size(img,2)      # size of frames

Y = zeros(Float64,n,m,l)           #initialize data

for f in 1:l
    img = load("$(imgpath)_$(f).tif")
    Y[:,:,f] .= convert(Array{Float64},Gray.(img)) #load frames in B&W
end

Y_med = median(Y, dims = 3)[:,:,1]  #calculate median over all the frames

Gray.(Y_med)                        #view the resulting grayscale image
```

3.2. Proximal gradient algorithm

For the second part of this thesis, I made use of a algorithm from a paper by Antonello et al [1]. In this paper a new, open-source package implemented in Julia called StructuredOptimization is presented [4]. In this paper an example is given how to use this modeling language to split the fore-and background of a grayscale video. For this model a video can be viewed as a superposition of the background, which stays still, and a foreground with moving object. The frames of the video of interest can been transformed to a matrix consisting of the grayscale values of each pixel. The original video can then be put into a three-dimensional matrix **Y**, whereby the first two dimensions represent the x and y axis and the third one the different frames. When splitting this original video into a moving foreground and steady background, they call the moving foreground the matrix **S** and the steady background **L**, both are also three dimensional matrices. In order to create the latter two matrices, the creators of the algorithm used the following optimization problem given in equation 3.1. Before they use this optimization, the matrices are first transformed so that **Y, L, S** $\in \mathbb{R}^{nm \times l}$, where n and m are the height and width of the frame in pixels. The l-th column of these matrices represent the grayscale pixel values of the vectorized l-th frame.

$$\underset{\mathbf{L},\mathbf{S}}{\text{minimize}} \frac{1}{2}\|\mathbf{L}+\mathbf{S}-\mathbf{Y}\|^2 + \lambda\|vec(\mathbf{S})\|_1 \quad \text{subject to} \quad \text{rank}(\mathbf{L}) \leq 1 \tag{3.1}$$

This optimization problem consist of three parts. The first part makes sure that the **L** and **S** matrices added together closely represent the original image **Y**. The second part can select how sparse the moving foreground image should be. A sparse matrix is a matrix where most of the element are zero. This is an important feature for the foreground, because we want it to only consist of the bits that are moving, which is a small part of image. The sparsity of the matrix is approximated by the L1-norm, which adds the absolute values of all the elements of the matrix. The sparsity of the matrix can be calculated by using the L0-norm, which counts all the non-zero elements of the matrix. However L0 is non-convex, which can lead to a local minimum which possibly is not equal to a global minimum. Usually it is not possible to check if there is another minimum in a non convex problem that further minimizes the function. If we instead use approximate L0 with the L1-norm, which is convex, we can find the global minimum. We can control the amount of sparsity by changing the value of λ. When λ gets bigger, the sparsity of the **S** matrix becomes more important for the minimisation, so **S** becomes sparser. The third part makes sure that the matrix **L** has rank 1 or 0. If the rank is 1, all the columns of the matrix are linear dependent. This means that all the background frames only differ with a scalar, which coincides with just a change in brightness of the image. So the rest of the image stays the same, which is exactly what we want. If the rank of the matrix is zero, the only possibility a zero matrix. This means that there is no background and can occur when a big part of the image is not constant so that a background can not be established.

3.2.1. Webcam footage

Firstly, I fed some live webcam footage of different places in the world to this algorithm to see how it reacted to videos it was made to deal with. I used 4 different webcams. They were stationed in Scheveningen (district of The Hague next to the sea) on the boulevard, a street in the city centre of Elburg (small city in the Netherlands), at times square in New York and the George Washington Bridge which connects Manhattan to New Jersey. I cut the recorded webcam footage and cut it into separate frames. Then I fed the frames into the Julia code that was provided as a sources in the paper of Antonello et al. This code used for the Scheveningen footage can be seen below. For each video I used between 7 and 14 frames of the video in the algorithm.

```
using Images, Printf

imgpath="./own_test_data/scheveningen/pictures/"

img = load("$(imgpath)scene00001.png") # load first frame
Frames = 150                           # number of frames in folder street
n,m = size(img,1),size(img,2)          # size of frames
frames = collect(1:10:Frames)          # select a slice of frames
l = length(frames)                     # number of selected frames

Y = zeros(Float64,n,m,l)               #initialize data
```

```
for f in eachindex(frames)
    a = @sprintf("%3.3i",frames[f])
    img = load("$(imgpath)scene0$(a)1.png")
    Y[:,:,f] .= convert(Array{Float64},Gray.(img)) #load frames in B&W
end

Y = reshape(Y,n*m,l); # Y contains the vectorized different frames in each of its coulmns

using StructuredOptimization

L = Variable(n*m,l) #define variables
S = Variable(n*m,l)

lambda = 3e-2

@minimize ls(L+S-Y)+lambda*norm(S,1) st rank(L) <= 1 with PANOC(tol = 1e-4);

L = ~L # extract vectors from variables
S = ~S

S[ S .!= 0 ] .= S[ S .!= 0 ] .+L[ S .!= 0 ] # add background to foreground
                                            # changes in nonzero elements
S[S.== 0] .= 1.0                            # put white in null pixels

Y = reshape(Y,n,m,l)
S = reshape(S,n,m,l)
L = reshape(L,n,m,l)

idx = idx = [1;5;15]
img = Gray.(vcat([ [Y[:,:,i] S[:,:,i] L[:,:,i]] for i in idx]...)) #constructs a combined images of
                                        #the original, moving and background frames
img1 = Gray.(L[:,:,1])   #shows the background image
```

3.2.2. Welding spots

After I fed it with the webcam footage, I wanted to use it to remove the unwanted illumination effect in the data sets of the welding spots. The idea is that if the camera and door are stationary relative to each other, only the light effect changes in the different frames. So the algorithm should put the changing light effect into the 'foreground' **S** matrix and the illumination invariant image of the spot into the 'background' **L** matrix. The algorithm produced a matrix **L** of the same image but with different brightness, so only multiplied with a scalar compared to the other images. So I wrote a piece of code that would normalize the brightness of the resulting 'background' image, which I put under the reshaping of the matrices. This code can be seen below. For the DCSC I also cut parts of the frame of so that only the area of interest with the welding spots is process. I did this to reduce the computing time for the algorithm.

```
L_min , L_max = extrema(L[:,:,1])               #calculate the minimum and maximum
                                                #value of the matrix
L_norm = (L[:,:,1] .- L_min) ./ (L_max-L_min)   #converts the matrix into a matrix
                                                #with a values from 0 to 1
```

3.2.3. multiplication modeling

As discussed in section 2.2 Construction of pixels in the image, a pixel is modeled as a multiplication of the object and its illumination. The algorithm is based on a split of the fore and background which added together return the input images. So I had to make sure that the object and the illumination were not a multiplication but were added together before putting it into the optimization algorithm. This can be done by taking the logarithm of the input matrix, as can be seen in equation 3.2.

$$log(v[n]) = log(b[n]s[n]) = log(b[n]) + log(s[n]) \tag{3.2}$$

When taking the logarithm of the matrix, I had to consider two things. Firstly, the logarithm of zero is not defined. This can be fixed by adding a small constant value to each element of the matrix prior to taking the logarithm. I chose a value of $1 \cdot 10^{-12}$. Secondly, the optimization algorithm only takes positive value. This can solve by first calculating the minimum of the logarithm of the matrix and then subtracting this value from all the elements. After this two operation, I fed the result into the algorithm and then added the minimum value to the output, take the exponent of this and lastly subtract the small value. The code to do this can be seen below.

```
using Images, LinearAlgebra, Statistics

imgpath="./preliminary_data/door941_jul20NOK/spot0"

img = load("$(imgpath)_1.tif") # load first frame
Frames = 4                      # number of frames in folder street
n,m = size(img,1),size(img,2)     # size of frames
frames = collect(Int64,1:4)    # select a slice of frames
l = length(frames)                # number of selected frames

Y = zeros(Float64,n,m,l) ;          #initialize data

for f in 1:l
    img = load("$(imgpath)_$(f).tif")
    Y[:,:,f] .= convert(Array{Float64},Gray.(img)) #loaf frames in B&W
end

Y = reshape(Y,n*m,l); # Y contains the vectorized different frames in each of its coulmns
Y .+= 1.0e-12;        #add small value to make sure all values are nonzero
Y = log.(Y);          #takes elementwise logarithm
ymin = minimum(Y)     #calculate minimum value and subtract this from the matrix
Y .+= -ymin;

using StructuredOptimization

L = Variable(n*m,l) #define variables
S = Variable(n*m,l)

lambda = 2e-1

@minimize ls(L+S-Y)+lambda*norm(S,1) st rank(L) <= 1 with PANOC(tol = 1e-4);


L = ~L # extract vectors from variables
S = ~S



S[ S .!= 0 ] .= S[ S .!= 0 ] .+L[ S .!= 0 ] # add background to foreground
                                            # changes in nonzero elements
S[S.== 0] .= 1.0                            # put white in null pixels

invlog(image) = exp.((image .+ ymin)) .- 1.0e-12     #reverse the operations done to the matrix
                                                       #prior to the optimization algorithm

Y = invlog(Y)
```

```
S = invlog(S)
L = invlog(L)

Y = reshape(Y,n,m,l);
S = reshape(S,n,m,l);
L = reshape(L,n,m,l);


L_min , L_max = extrema(L[:,:,1])
L_norm = (L[:,:,1] .- L_min) ./ (L_max-L_min)
```

3.2.4. Saturated pixels

After I used these methods to filter out the light effect on the pictures of the welding spots, I noticed that if some parts of the welding spots were oversaturated in most of the frames, the algorithm made the filtered picture on that part of the picture also oversaturated. A oversaturated pixel does not give any information about that particular spot. Therefore I wanted to replace these pixels with some combination of the values of that pixel which were not oversaturated. As the optimization algorithm is a non-linear filter I could not consider the whole output image as some linear combination of the given input frames. However Dr. Soloviev suggested that if a small group of pixels is taken, a linear approximation is justified. I therefore split up the output image of the algorithm into small patches of 3x3 pixels. For each small patch I approximated the pixel value output on spot [i,j] of the optimization algorithm as stated in equation 3.3.

$$L[i,j] = \sum_l w_l Y_l[i,j] \tag{3.3}$$

Hereby is $L[i,j]$ the output value , w_l the weight factor of the l-th input frame and $Y_l[i,j]$ the value of that pixel on the l-th frame. For each patch of 3x3 pixel I first calculated the corresponding weight factors by means of the least squares solution given in equation 3.4.

$$\hat{w} = Y^\top L \tag{3.4}$$

Then I look for each pixel in the patch if some of corresponding Y values are 1, which means they are oversaturated. If this is the case I calculate a new value for this pixel by only using the values of Y that are not oversaturated with there corresponding weight factor. So if for instance only the first second and fourth frame of that pixel are not oversaturated, a new value for the output is calculated as follows:

$$L[i,j] = N \cdot (w_1 Y_1[i,j] + w_2 Y_2[i,j] + w_4 Y_4[i,j]),$$

where N is some normalisation constant. If all Y values of this pixel are 1, I replace the output value with 1, because the only information we then have of that pixel is that it is oversaturated. The normalisation constant is needed because I excluded some weight values that as a complete set defined the output image. To calculate this normalisation constant I took two approaches. The first one is based on the sum of the weight factors. Hereby the normalisation factor is calculated by the sum of all the weight factors dividing by the sum of all the weight factors that are not excluded for that particular pixel. The second approach is based on the mean value of the patch. The normalisation value is then calculated by the mean value of the output of the algorithm in that patch divide by the mean value of the Y values on the patches that are not excluded.

The code for this saturated pixel replacement is displayed below, Hereby the first normalisation constant is used.

```
filter_size = 3                 #patch size
n_pieces = Int(n/filter_size)   #the amount of patches fitted on the x and y axis
m_pieces = Int(m/filter_size)

threshold = 1


Y = reshape(Y, (filter_size,n_pieces,filter_size,m_pieces, l));   #both Y and L are reshaped so that al
Y = permutedims(Y, [1,3,2,4,5]);
```

```
Y = reshape(Y,(:,l));

L_norm = reshape(L_norm, (filter_size,n_pieces,filter_size,m_pieces));
L_norm = permutedims(L_norm, [1,3,2,4]);
L_norm = reshape(L_norm,(:));

Y_shaped_3 = deepcopy(Y)
L_norm_shaped_3 = deepcopy(L_norm)
```

```
for i in range(1, n*m, step= filter_size^2) #loops trough each patch
    W_local = Y[i:i+(filter_size^2-1),:] \ L_norm[i:i+(filter_size^2-1)]   #t local weight
                                                                          #factors are calculated
    normal = sum(W_local)
    for j in range(i, i+(filter_size^2-1), step=1) #loops trough each pixel of the patch
        W_local_ov = deepcopy(W_local)
        W_trace = ones(l) #traces the frames that are or are not included
        for k in 1:l
            if Y[j,k] >= 1
                W_local_ov[k] = 0   #excludes the value of oversaturated pixels
                W_trace[k] = 0
            end
        end
        Y_local_nonzero = Y[j,:] .* W_trace
        # check if there are oversaturated pixel and if there is, change the value of the output
        if W_trace != ones(l)
            L_norm[j] =  dot(Y[j,:] , W_local_ov) * normal/sum(W_local_ov)
        elseif W_trace == zeros(l)
            L_norm[j] = 1
        end

    end
end
```

```
Y = reshape(Y,(filter_size,filter_size ,n_pieces,m_pieces, l));   #reshapes both Y and L back
                                                                  #to the form that they make a image
Y = permutedims(Y, [1,3,2,4, 5]);
Y = reshape(Y, (n,m,l));

L_norm = reshape(L_norm,(filter_size,filter_size ,n_pieces,m_pieces));
L_norm = permutedims(L_norm, [1,3,2,4]);
L_norm = reshape(L_norm, (n,m));
```

For the second way of calculating the normalisation constant the for loop above is changed to the following.

```
for i in range(1, n*m, step= filter_size^2) #loops trough each patch
    W_local = Y[i:i+(filter_size^2-1),:] \ L_norm[i:i+(filter_size^2-1)]   #t local weight
                                                                          #factors are calculated
    L_mean = mean(L_norm[i:i+(filter_size^2-1)])
    for j in range(i, i+(filter_size^2-1), step=1) #loops trough each pixel of the patch
        W_local_ov = deepcopy(W_local)
        W_trace = ones(l) #traces the frames that are or are not included
```

```
        for k in 1:l
            if Y[j,k] >= 1
                W_local_ov[k] = 0   #excludes the value of oversaturated pixels
                W_trace[k] = 0
            end
        end
        Y_mean =  W_trace .* mean(Y[i:i+(filter_size^2-1),:], dims=1)
        # check if there are oversaturated pixel and if there is, change the value of the output
        if W_trace != ones(l)
            L_norm[j] =  dot(Y[j,:] , W_local_ov) * L_mean / mean( Y_mean[Y_mean .!= 0] )
        elseif W_trace == zeros(l)
            L_norm[j] = 1
        end

    end
end
```

4

Results and Discussion

This chapter discusses the results of used filters. Firstly, I present the two data sets on which the filters are used. Then the results of the median filter are presented and discussed. After that we take a look at the result of the proximal gradient algorithm on the webcam footage. Then we look how the optimization problem works on the two data sets of the welding spots. Lastly I will discuss the result of the modified code to take care of the oversaturated pixels.

4.1. Data sets

In this thesis I use two different data sets of pictures of welts taken by both FCA Italy and researchers of the Delft Center of System and Control (DCSC). Both are images of welts in car-doors produce at a welding line of FCA Italy.

4.1.1. FCA Italy

FCA made a sets of grayscale pictures of welding spots from two different doors. From each spot four pictures are taken whereby each picture has a different illumination. The set up consisted of a camera and two sets of LED-strips. The LED-strips were attached on both sides of the camera. The different illumination was then achieved by leaving the LED's both off, turning the left or right one on or lighting them both at the same time. The set-up for the pictures can be seen in figure 4.1 and the resulting pictures in figure 4.2

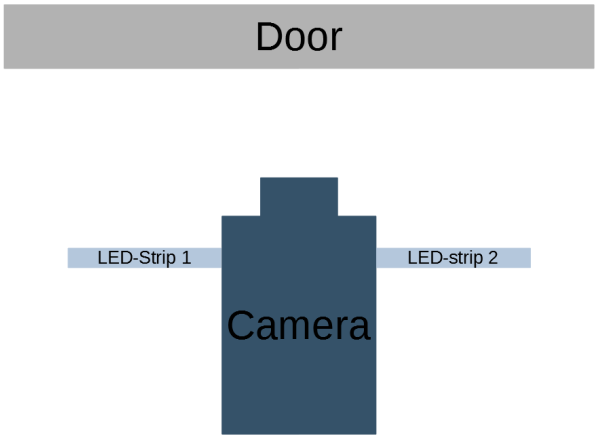
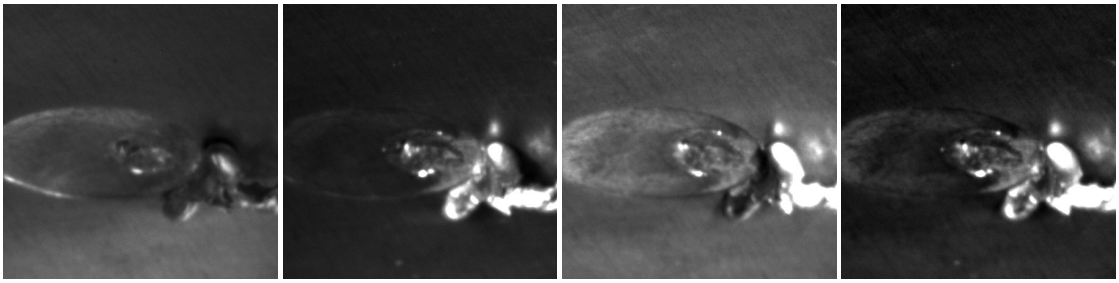


Figure 4.1: Set-up for the FCA data set



(a) LED-strips are both off (b) Only the left LED-strip is on (c) Only right LED-strip is on (d) Both LED-strip are on

Figure 4.2: An example of the different pictures of the same welding spot taken by FCA Italy

4.1.2. DCSC

The second data set is made by researchers of DCSC who visited the production line themselves in November 2021 to make there own images. The set consist of two videos cut in frames of a door with 3 welds on it. During the video the camera and door remain stationary and a light source is moved by hand behind the camera. The sets contain 124 and 126 frames of the same door with varying illumination. Some frames of the second set of images can be seen in figure 4.3.

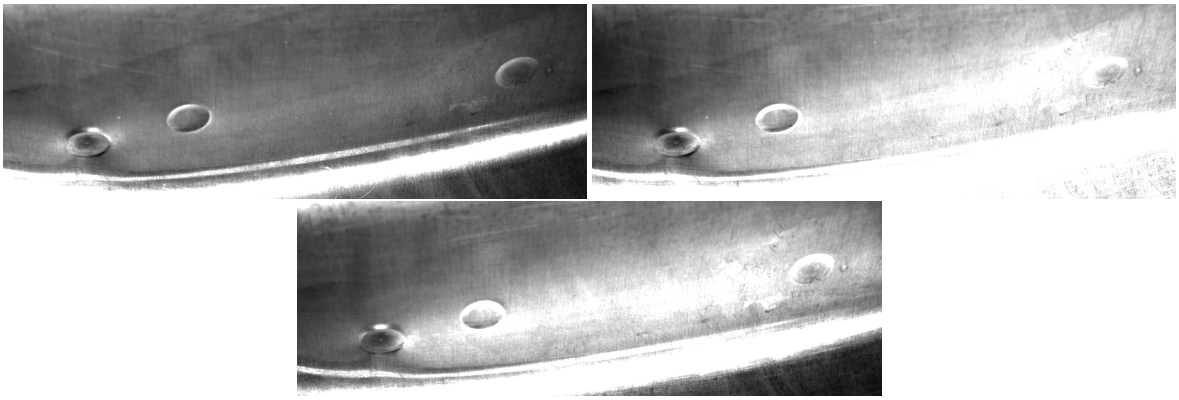


Figure 4.3: An example of 3 frames fro the data set made by the researchers of DCSC

4.2. Median Filter

I first tried to use this filter on the FCA Italy data. The result can be seen in figure 4.4. The upper 4 images are the original input images and the bottom one is the output. It can be seen that the output image does not get clearer. The whole image just gets vaguer than the originals and especially the border of the circular welding spot is less clear to see.

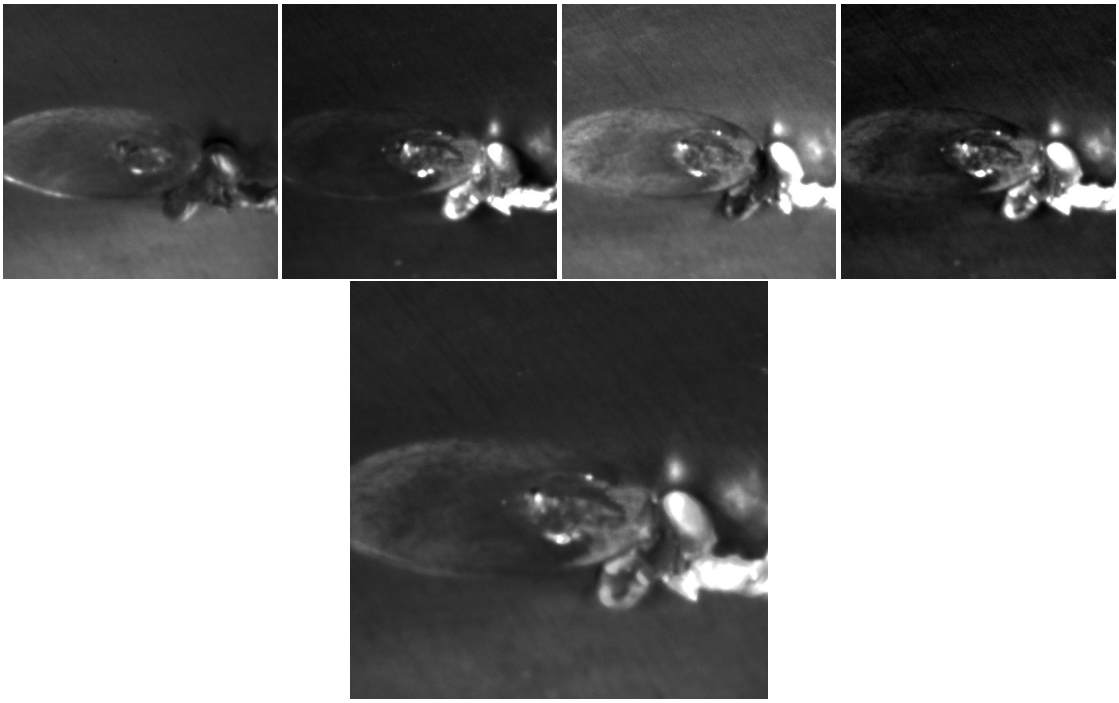
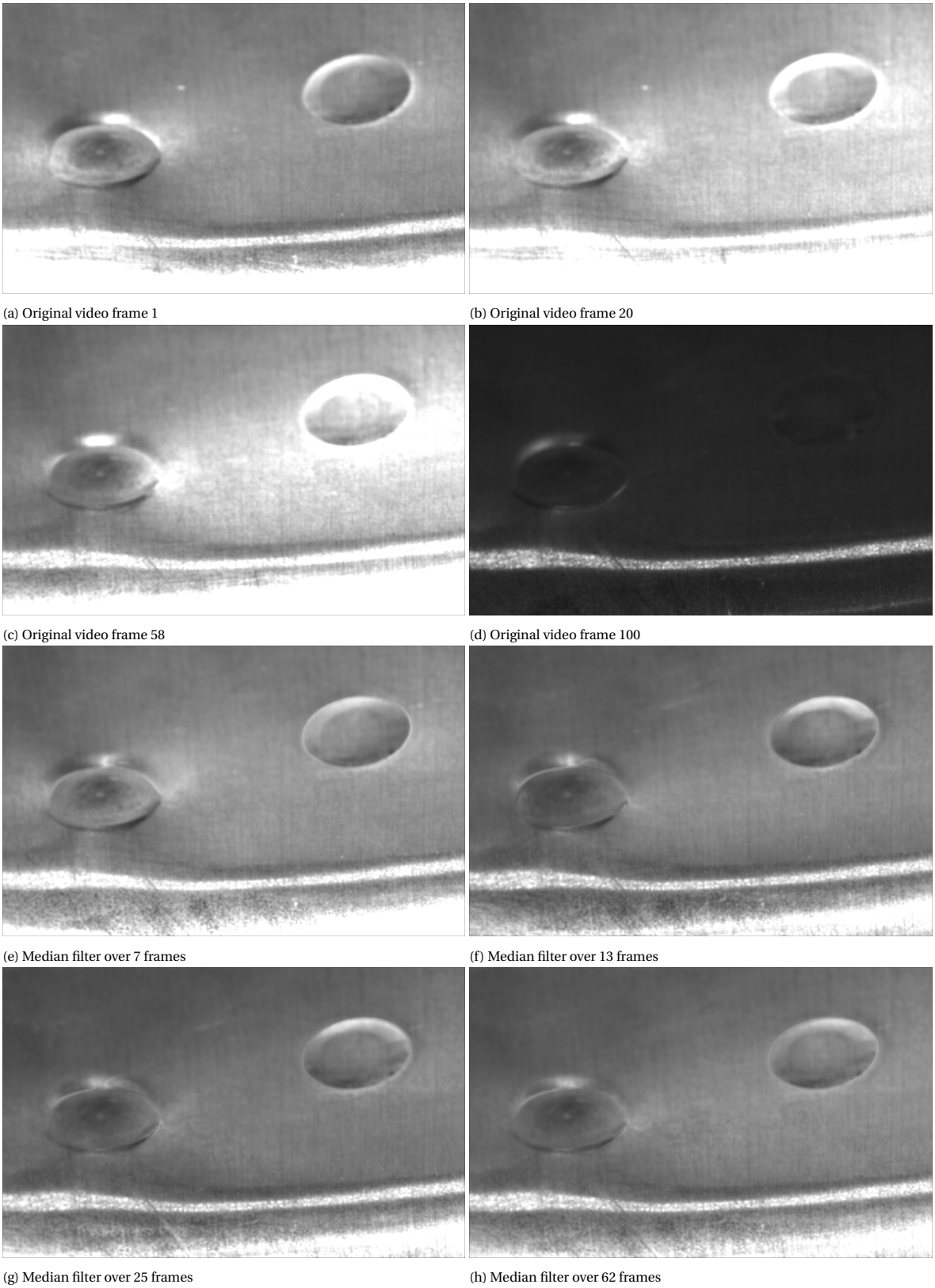


Figure 4.4: The output image of the median filter on the FCA Italy filter with its input images above it.

Then I used this filter on the data collected by the DCSC team. Because I had more than 100 frames of the same weld, I experimented by feeding different amount of frames into this filter. In figure 4.5 first some of the input frames are showed followed by some resulting images with different amounts of frames.

In these images the circumference of the circle are enhanced, but like in the FCA Italy case, the pictures also get vaguer. This is undesirable because the detection of things like scratches is much worse. If the number of frames the filter takes in is increased, the enhancement of the circles gets more present, but it also becomes more like a sort of artificial circle instead of a representation of reality. The filter does however deal quite well with the oversaturated pixels that can be seen at the bottom of the images. I therefore think this filter might be useful to use for the detection of the welding spots, but not for flaw detection.



(a) Original video frame 1

(b) Original video frame 20

(c) Original video frame 58

(d) Original video frame 100

(e) Median filter over 7 frames

(f) Median filter over 13 frames

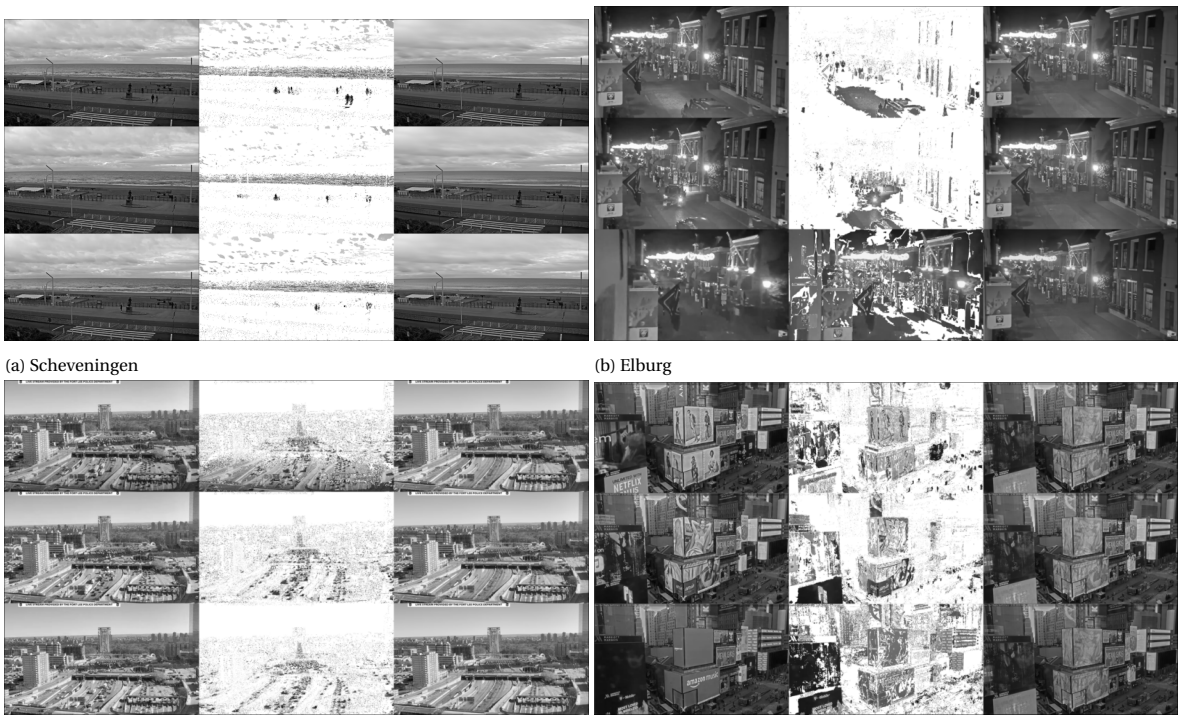
(g) Median filter over 25 frames

(h) Median filter over 62 frames

Figure 4.5: The output images for a median filter over the DCSC data taken over different amounts of frames and some of its input frames.

4.3. Proximal gradient algorithm for webcam footage

As said in the experimental method, I used 4 different webcam video to find out the characteristics of this algorithm. In figure 4.6 you can see how the algorithm reacted to those videos. In each image you can see for 3 different frames: the original frame, the part of the image that was moving at that moment and the constant background from left to right.



(a) Scheveningen

(b) Elburg

(c) George Washington Bridge

(d) Times Square

Figure 4.6: The splitting of webcam footage done by the algorithm. From left to right: original frame (Y), moving part (S), steady background (L). Three different frames of the video are depicted underneath each other.

4.3.1. Scheveningen

The algorithm preform really well on the footage of the boulevard of Scheveningen. In figure 4.7 can be seen that the people that were in the shot were removed perfectly. Also because the waves and the clouds are moving, the background image contains what seems to be a calmer sea and sky. In this particular footage not a lot of the image is moving so the algorithm seems to have no problem with removing the moving parts.

(a) Original image

(b) Background image

Figure 4.7: The result of filtering the background of the Scheveningen webcam footage by the algorithm.

4.3.2. Elburg

For the Elburg webcam footage the algorithm also preform rather well. It also removes the cars and people and because this footage was taken in the evening, some effect of lighting can also be seen. The traffic sign in the middle of the image is a good example that is relevant for this thesis. In the original picture, depicted in figure 4.8, you can see that this sign lights up because of the reflection of the headlights of the car. In the filtered background image generated by the algorithm, this reflection is removed, which is quit a nice feature considering the end goal of this thesis.

(a) Original image (b) Background image

Figure 4.8: The result of filtering the background of the Elburg webcam footage by the algorithm.

4.3.3. George Washington Bridge

In this footage the limits of the algorithm become visible. As there are a lot of cars moving in this video and the background, which in this case is a empty road, is hardly visible, the algorithm has problems with filtering out all these cars. Therefore we can still see some feature of the cars that were driving there .

(a) Original image (b) Background image

Figure 4.9: The result of filtering the background of the George Washington Bridge webcam footage by the algorithm.

4.3.4. Times Square

If parts of an image have no constant background we can begin to see some strange effects. This can be seen in the footage of times square, displayed in figure 4.10. Because the billboard in the image do not really have a background, the algorithm replaces this part by some weird combination of picture that were displayed in the time this was recorded. However most of the people were removed and some of the cars that were not standing still were also removed.

(a) Original image (b) Background image

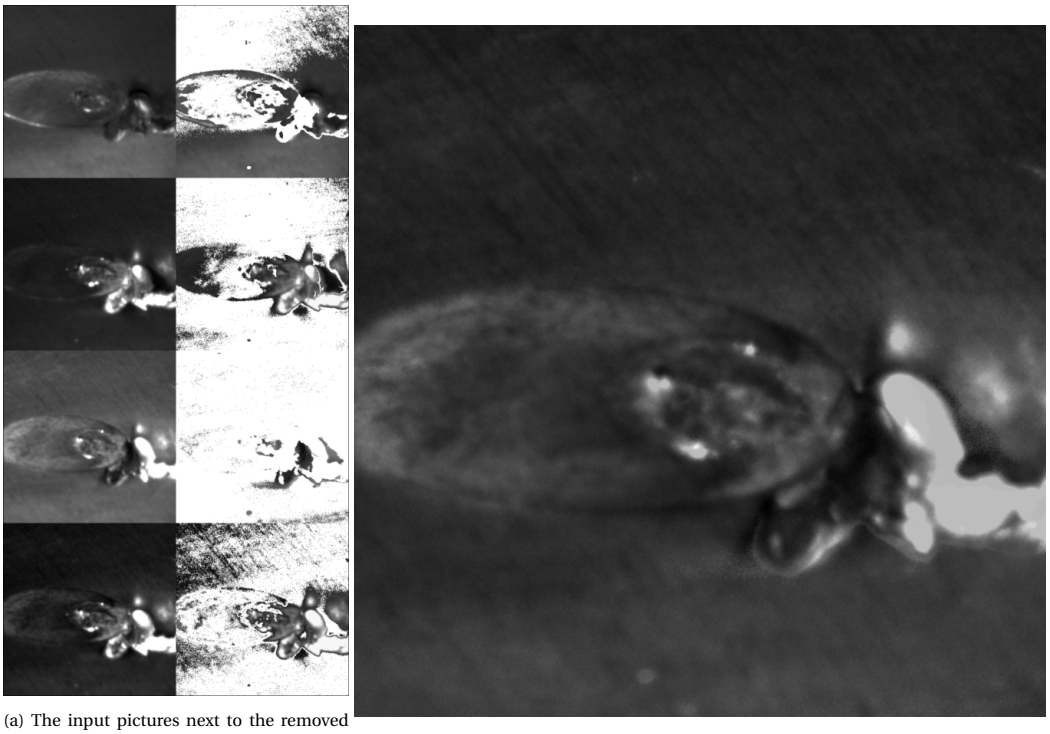
Figure 4.10: The result of filtering the background of the Times Square webcam footage by the algorithm.

4.4. Proximal gradient algorithm for removal of spurious illumination effect

After I used the algorithm on webcam footage I modified the code to use it on the welding spot images. I first run the algorithm without taking the logarithm before feeding it into the algorithm.

4.4.1. Without logarithm

I first tested it on the FCA Italy footage. The result can be seen in figure 4.11. on the left we see the four input images and next to that the parts that were filtered out by the algorithm. Normally the algorithm return four different output pictures who only differ in brightness. The output picture we see on the right is a normalized version of these where the grayscale values lie between 0 and 1.



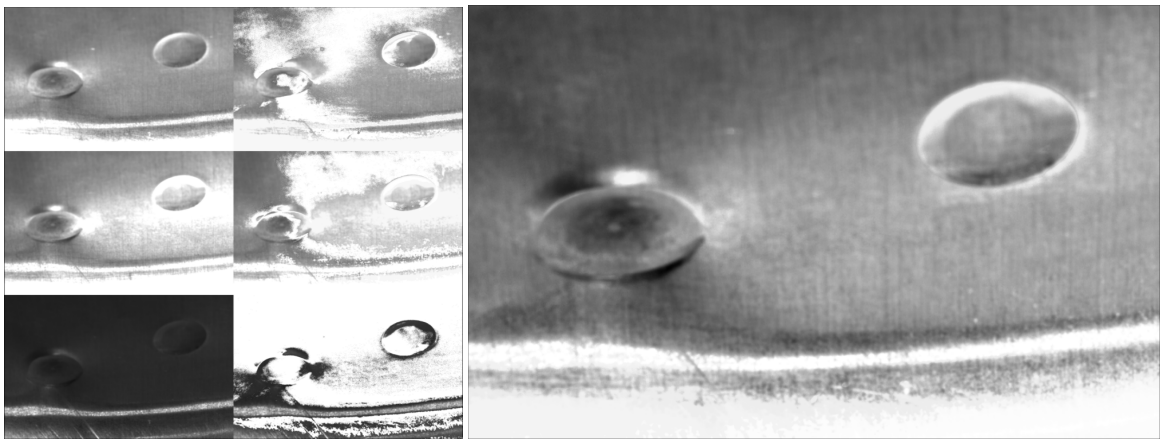
(a) The input pictures next to the removed parts. (b) The output picture normalized

Figure 4.11: On the left the four input pictures are shown. To the right of them are parts that were removed by the algorithm per input image. On the right the output image of the algorithm is shown.

In this output the circular welding spot can been see rather good and if we compare it with the input images, it is clear that it removes a lot of unwanted reflections. For instance in the first input image we see a reflection on the bottom left side of the welding spot, which is perfectly filtered away in the output image.

However we can see that for areas where the majority of input images are over saturated, like on the bulge on the right side of the weld, the resulting output picture is also oversaturated. That is undesirable because we do have information about this area from for example the first input image. In section 4.4.3 I tried to solve this problem.

Then I ran the footage taken by the DSCS team. For this data set I decided to feed the algorithm with 7 frames, because I had much more frames of the same spots available. The result can been seen in figure 4.12 with the same layout as in figure 4.11.



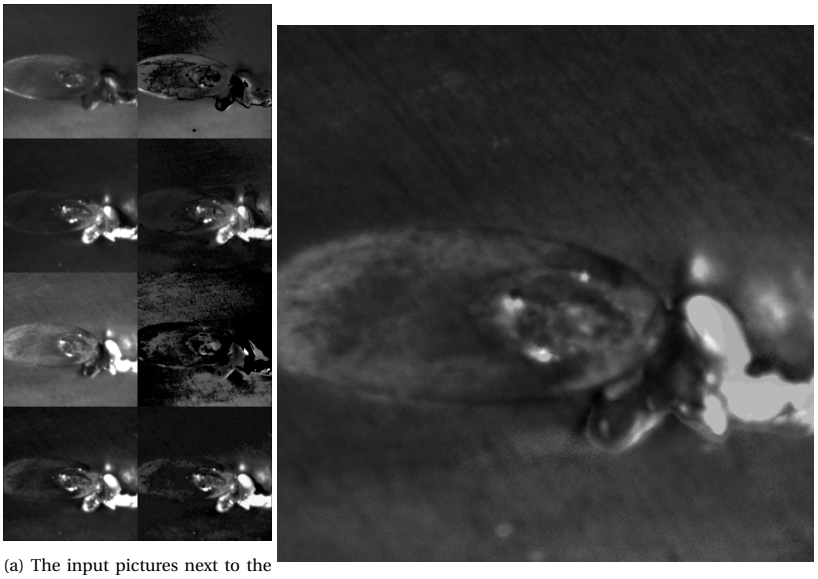
(a) The input pictures next to the removed parts. (b) The output picture normalized

Figure 4.12: On the left three of the seven input pictures are shown. To the right of them are parts that were removed by the algorithm per input image. On the right the output image of the algorithm is shown.

Here we can see that the algorithm also did quit good in filtering out the unwanted illumination effects. However there is still some glints to be seen around both the welding spots, but they are significantly reduce compared to the input pictures. Also the metal around the spots is clearly visible compared to for example the second input picture depicted in figure 4.12. Concerning the bottom of the picture, this algorithm gives a poorer representation than the median filter. The rest of the image however is displayed much better here.

4.4.2. With logarithm

After this I first took the logarithm of the images before feeding it into the algorithm to see if this provided any significant improvements. I first tried this with the FCA Italy data, which can be seen in figure 4.13.

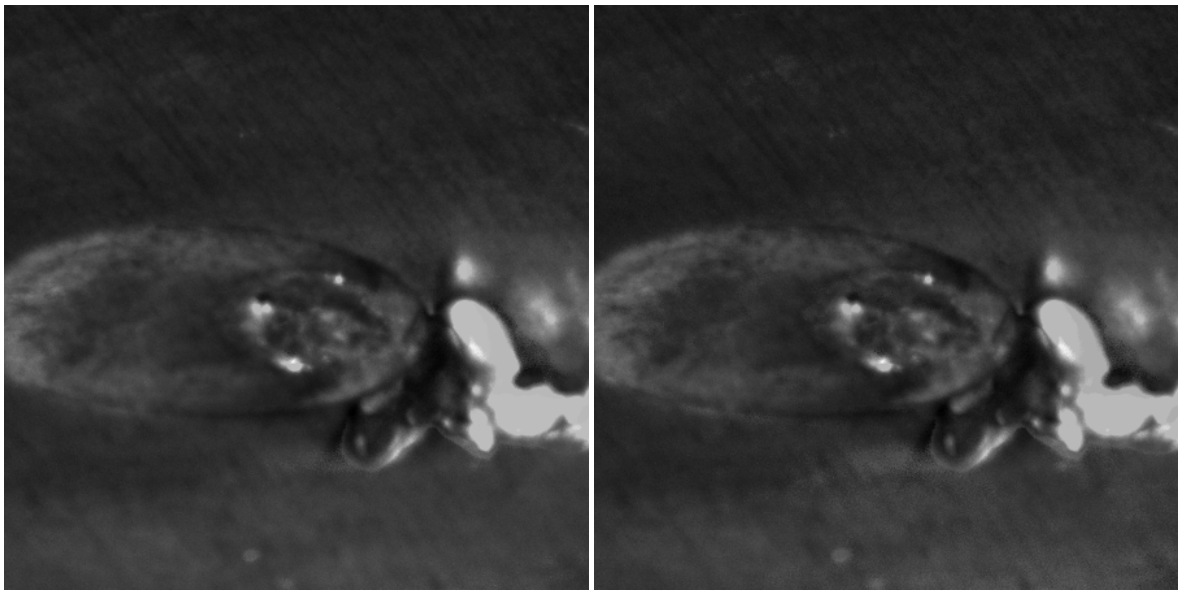


(a) The input pictures next to the removed parts.

(b) The output picture normalized

Figure 4.13: On the left the four input pictures are shown. To the right of them are parts that were removed by the algorithm with the use of taken the logarithm per input image. On the right the output image of the algorithm is shown.

If we compare this to the non logarithmic case, we can see that more is removed here over the whole picture instead of only some areas. It does however not remove a lot in the areas that were not affected in the non logarithmic case. Therefore if we compare the two outputs in figure 4.14, I do not see any significant different. Since this is the case, I would recommend to use the algorithm without first taking the logarithm of the input. This calculation takes some extra time and up top of that the algorithm take significantly more time to run for the data where the logarithm is taken of.

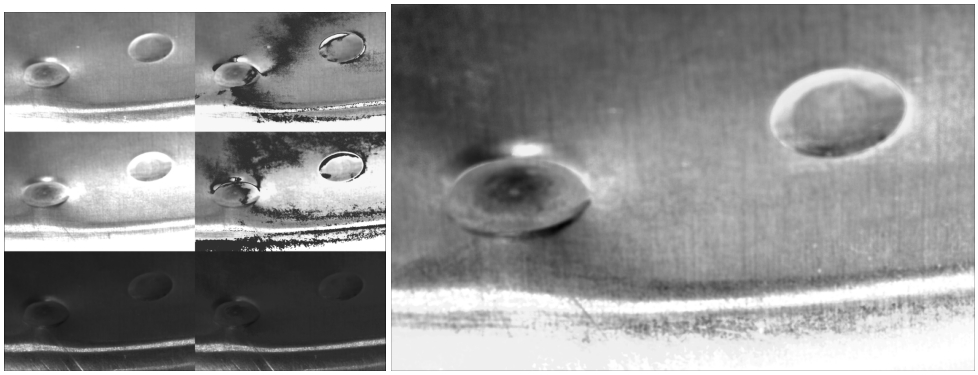


(a) Output non-logarithmic

(b) Output logarithmic

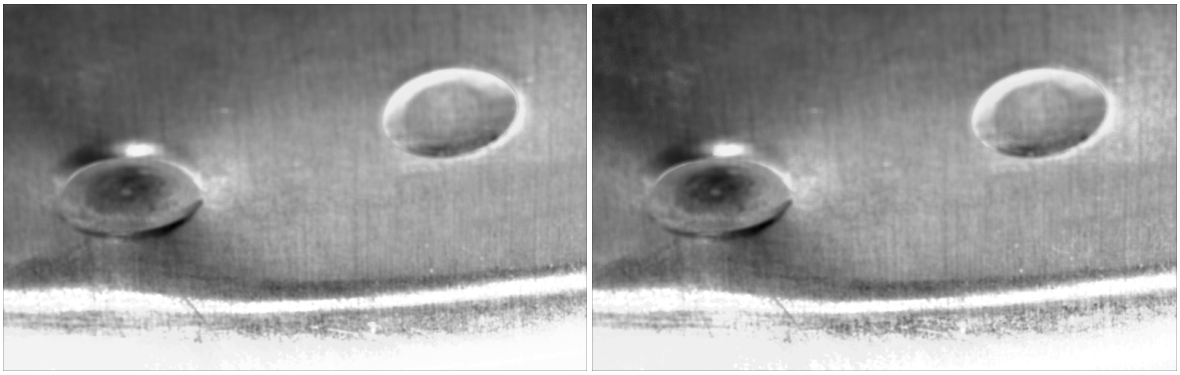
Figure 4.14: A comparison between the output with and without taking the logarithm during the algorithm for the FCA Italy data.

Then I also use the logarithm for the DCSC case. As can be seen in figure 4.15 and 4.16 this implementation of the logarithm does not make any significant improvements to the output. So with the same reasoning as for the FCA Italy case, I conclude that the non logarithmic use of the algorithm is recommended.



(a) The input pictures next to the removed parts. (b) The output picture normalized

Figure 4.15: On the left the three of the seven input pictures are shown. To the right of them are parts that were removed by the algorithm with the use of taken the logarithm per input image. On the right the output image of the algorithm is shown.

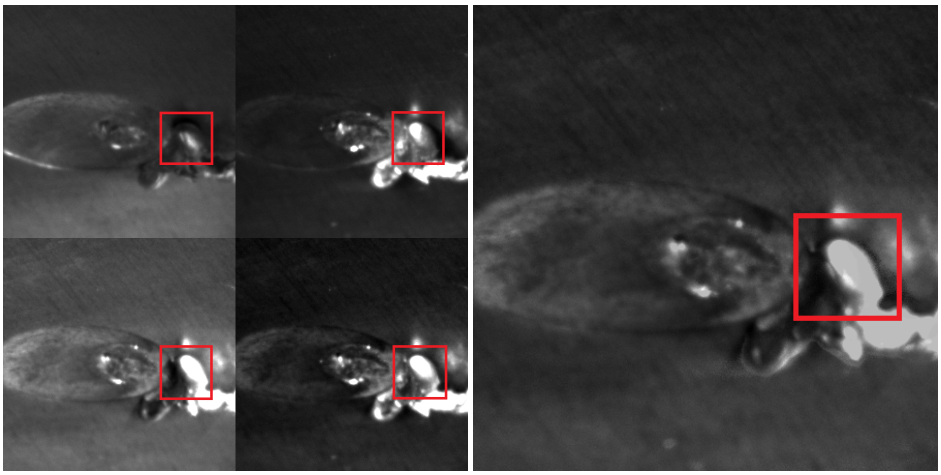


(a) Output non-logarithmic (b) Output logarithmic

Figure 4.16: A comparison between the output with and without taking the logarithm during the algorithm for the DCSC data.

4.4.3. Saturated Pixels

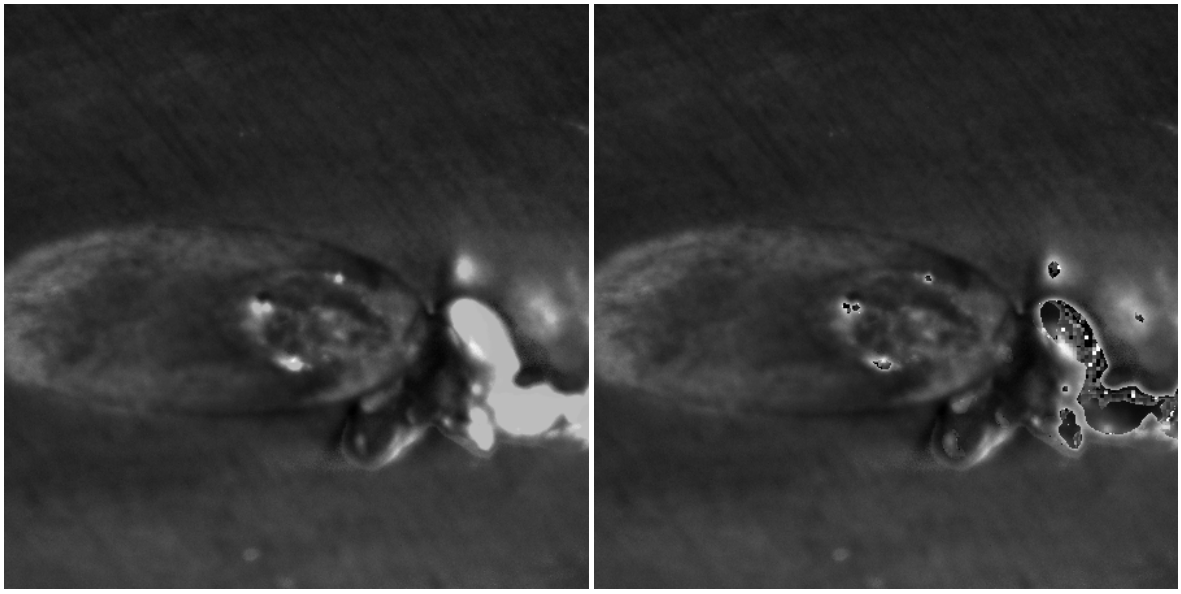
As discussed in section 4.4.1, if in most of the input images some area is oversaturated, the algorithm also fills this area with oversaturated pixels. A example of such area is highlighted in figure 4.17.



(a) Input images (b) Output image

Figure 4.17: Highlighted area were the algorithm output takes a oversaturated value.

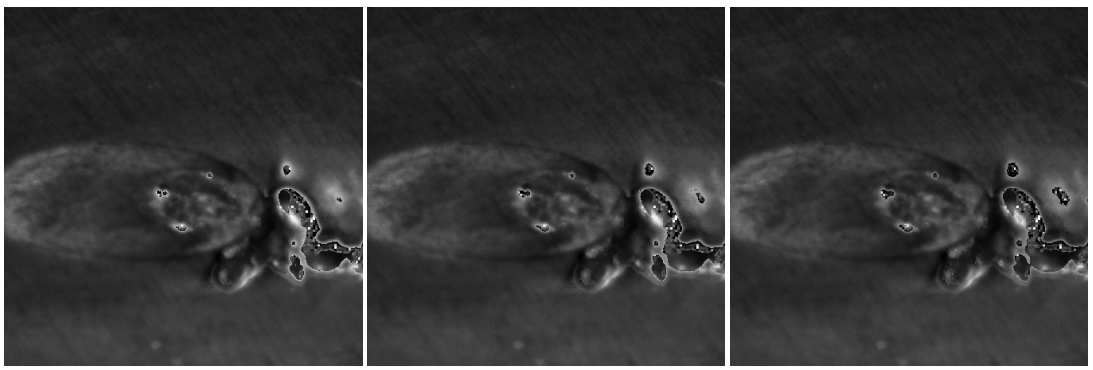
I would like to further improve this area so that it only uses the information of the pixels that are not oversaturated. Therefor I apply the extra piece of code presented in the experimental method. This piece of code is applied with two different normalisation constants. First I tried the normalisation based on the sum of the weight factors. The result of this can been seen in figure 4.18.



(a) Output without excluding saturated pixels (b) Output image with excluding saturated pixels

Figure 4.18: A comparison between the output of the algorithm with or without the code that replaces the oversaturated pixels with the normalisation constant based on the sum of the weight factors.

Here we see that the extra piece of code rightly selects the areas where the pixels of some input images were oversaturated. However the value that replace them does not seem to blend in with the rest of the image although more information about this area is given. Also a clear line is seen around the areas that were effected by the extra code. This can be explained because the areas directly around these oversaturated spots are almost oversaturated and have a grayscale-value close to 1. If I set the threshold for which I consider the pixel oversaturated to a value lower then 1, we can decrease this bright border around it.This can been seen in figure 4.19.

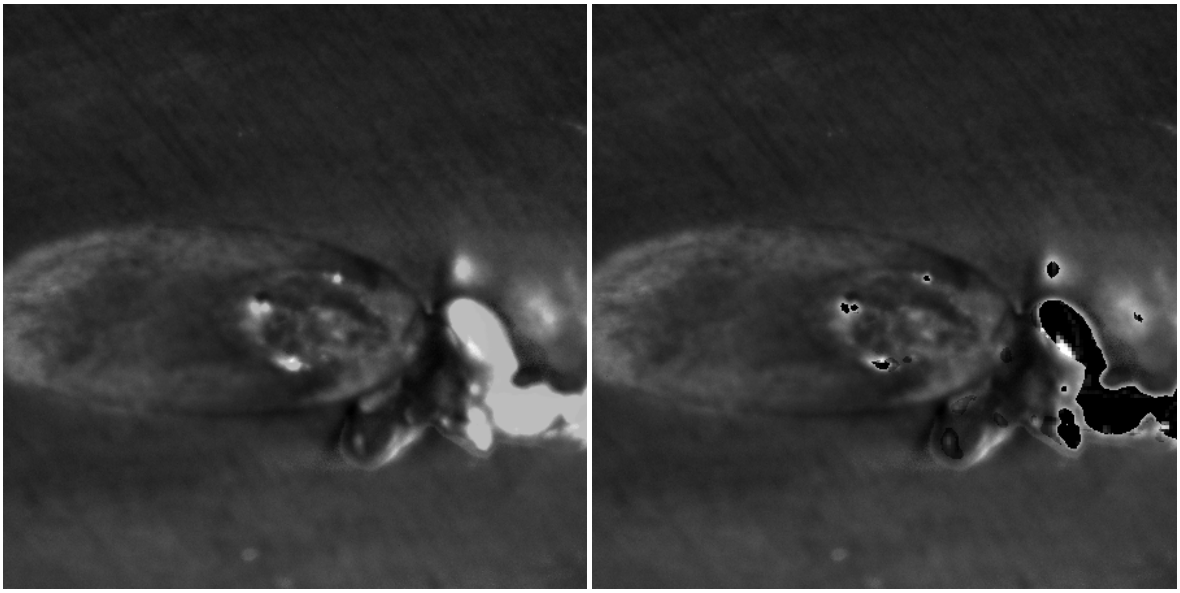


(a) Threshold = 1 (b) Threshold = 0.9 (c) Threshold = 0.8

Figure 4.19: The output of the algorithm with the saturated pixels replaced with the normalisation constant based on the sum of the weight factors with different threshold values.

Then I tried the same code but with the normalisation constant based mean of the values in that particular patch. The result is displayed in figure 4.20.

In this image the right areas are also rightly picked. However the values that replace these areas are totally not correct. I do think this approach on the normalisation is a promising one though. Unfortunately due to limited time, I did not have the time to further investigate what is going wrong here. Therefore I suggest that



(a) Output without excluding saturated pixels

(b) Output image with excluding saturated pixels

Figure 4.20: A comparison between the output of the algorithm with or without the code that replaces the oversaturated pixels with the normalisation constant based on the mean value of the particular patch.

for further research, one can try to improve this way of normalising the replaced values. Also the first way of normalising seems to work quite descent and might do the trick if some improvements are made.

5

Conclusion

The goal of this thesis was to minimize unwanted illumination effect on images of metallic doors. Different approaches were made to achieve this goal. Firstly a median filter was applied to the data. This made the resulting image vaguer then the original and sometimes even distorted image. Then a algorithm developed by Antonello et al was used to filter the input images. This algorithm gave a much more promising result. A lot of unwanted specular features that were present on the input images were removed without sacrificing clarity of the image. I then tried to further improve the output of this algorithm by taking the logarithm of the input before feeding it into the algorithm but this only significantly increased the processing time without improving the resulting output. At last I tried to remove the saturated pixels in the output of this algorithm and replacing them by non oversaturated pixels. This method seems promising to further improve the process, but due to time shortage the right replacement value could not be found, most probably due to the wrong normalisation. Therefore I suggest to further investigate this replacement of oversaturated pixels to perfect this method of illumination normalisation.

Bibliography

[1] Niccoló Antonello, Lorenzo Stella, Panagiotis Patrinos, and Toon van Waterschoot. Proximal gradient algorithms: Applications in signal processing. *arXiv preprint arXiv:1803.01621*, 2018.

[2] Bruce G. Batchelor. *Machine Vision Handbook*. Springer, London, 2 edition, 2012.

[3] Jorge Bonekamp. Multi-image optimization based specular reflection removal from non-dielectric surfaces. 2021.

[4] N. Antonello L. Stella. Structuredoptimization.jl. `https://github.com/JuliaFirstOrder/StructuredOptimization.jl`, 2017.

[5] MZ Abbas Shah, Stephen Marshall, Paul Murray, et al. Removal of specular reflections from image sequences using feature correspondences. *Machine Vision and Applications*, 28(3):409–420, 2017.

[6] Lawrence B Wolff and Terrance E Boult. Constraining object features using a polarization reflectance model. *Phys. Based Vis. Princ. Pract. Radiom*, 1:167, 1993.