

THE HAGUE UNIVERSITY OF APPLIED SCIENCES

IMAGE ACQUISITION AND PROCESSING  
LAB

---

# Final Report

---

*Author/Student:*

Luca van Straaten (18073611)  
Roderik Leijssen (15060292)

*Instructor:*

F. Theinert

November 20, 2022

**THE HAGUE**  
UNIVERSITY OF  
APPLIED SCIENCES

Document version 2.0

# Contents

<b>0</b>	<b>Introduction</b>	<b>1</b>
<b>1</b>	<b>Assignment 1</b>	
	Setup	<b>2</b>
<b>2</b>	<b>Assignment 2</b>	
	Object in front of Dark Background	<b>3</b>
2.a	Object with dark background . . . . .	3
2.b	Optimal exposure . . . . .	3
2.c	Sketch of setup . . . . .	4
2.d	Calculation of ‘angle of view’ . . . . .	4
<b>3</b>	<b>Assignment 3</b>	
	Moving Object	<b>5</b>
3.a	Optimal exposure . . . . .	5
3.b	Sketch of setup . . . . .	5
3.c	Calculation of ‘angle of view’ . . . . .	6
<b>4</b>	<b>Assignment 4</b>	
	Salt and Pepper Noise	<b>7</b>
4.a	Manipulate own image from assignment 2 . . . . .	7
4.b	Write C/C++ code for Brightness correction . . . . .	7
4.c	Write C/C++ code for ‘Salt and Pepper Noise’ correction . . . . .	8
<b>5</b>	<b>Assignment 5</b>	
	Convolution	<b>10</b>
<b>6</b>	<b>Assignment 6</b>	
	Demosaicing Filter	<b>12</b>
6.a	Write C/C++ code for capturing a raw image . . . . .	12
6.b	Convert this grayscale image to a color image by implementing a function to recover the actual colors . . . . .	12
<b>A</b>	<b>Appendix A</b>	<b>16</b>
<b>B</b>	<b>Appendix B</b>	<b>23</b>
<b>C</b>	<b>Appendix C</b>	<b>27</b>
<b>D</b>	<b>Appendix D</b>	<b>31</b>

## 0 Introduction

This lab report documents our results from the lab assignments as part of Image Acquisition and Processing, which is a subject from the minor Embedded Systems 2022.

The original introduction of the study manual[1]states:

The students will work in groups of two and have to attend all lab-sessions in order to pass the course. During the lab-sessions, the students are asked to take images and write software to process them. Students are asked to bring their own laptops with an USB3.0 port. All assignments can be worked out on school-computers with the cameras supplied, but working independently on your own laptop is recommended. Students will receive a virtual machine (Virtual Box) with all software preinstalled on Ubuntu 22.04 LTS.

This report describes the exercises and how we attempted to solve them.

We used the virtual machine and project template provided by our teacher. However the code for assignment 4 to 6 was developed locally on a Debian "Buster" machine with the aid of Makefile's and the packaged Debian version of OpenCV2.

# 1 Assignment 1

## Setup

For this assignment we connected the Camera to the Virtual Machine. And to the project template a case was added to the "switch (key)" statement (see Listing 1). This case was used to take a picture with the camera. The picture was then saved in the folder from which the program was run.

```
1 case 's':  
2     cout << "Saving..." << endl;  
3     // save image using openCV API  
4     imwrite("blahai.png", image);
```

Listing 1: save image to file



Figure 1: Image taken with the camera of a blahai

We pointed the camera at an object and adjusted the aperture and focus to get a good looking picture with a shutter time of 100ms. See image 1

## 2 Assignment 2

### Object in front of Dark Background

For this assignment, we will be capturing a image wich we will also use for assignment 3 and 4. It will be of a model car in front of a dark background. The goal is to make a useful setup to acquire the image and solely adjust the exposurertime to come to a well exposed image [1].

The code in Listing 2 was used to take the image, or adjust the exposure time. The image was then saved in the folder from wich the program was run. for the full code see appendix A.

```
1  case ' ':
2      if (imgSave(image, "output.png")) {
3          cout << "Image saved succesfully!" << endl;
4      } else {
5          cout << "Error saving file." << endl;
6      }
7      break;
8  case ',':
9      cam0.setExpoMs(--cfg.exposureMS);
10     cout << "Exposure adjusted to " << cfg.exposureMS << endl;
11     break;
12 case '.':
13     cam0.setExpoMs(++cfg.exposureMS);
14     cout << "Exposure adjusted to " << cfg.exposureMS << endl;
15     break;
16 case '[':
17     cam0.setExpoMs(cfg.exposureMS -= 10);
18     cout << "Exposure adjusted to " << cfg.exposureMS << endl;
19     break;
20 case ']':
21     cam0.setExpoMs(cfg.exposureMS += 10);
22     cout << "Exposure adjusted to " << cfg.exposureMS << endl;
23     break;
```

Listing 2: save image to file

#### 2.a Object with dark background



Figure 2: Object in front of Dark Background

The image above is the image we took of the model car, it has a matelic gold paint with black stripes across. It was challenging to get the car well exposed, because the metallic paint is reflective. So we needed even light positioned in a way that the reflections would not go into the camera. Enough light was needed for the black parts of the car to not be ender exposed. We placed the object away from the background so we could make shine the light only on the car and not the background. The layout of the setup is shown in section 2.c.

#### 2.b Optimal exposure

We got the best result using a 420 ms exposure time. This is the time the camera takes to gather light on the sensor. The image is shown in figure 2. The image is well exposed and the background is dark. The car is well

visible and the details are clear. The image is not overexposed and the background is black but not saturated.

## 2.c Sketch of setup

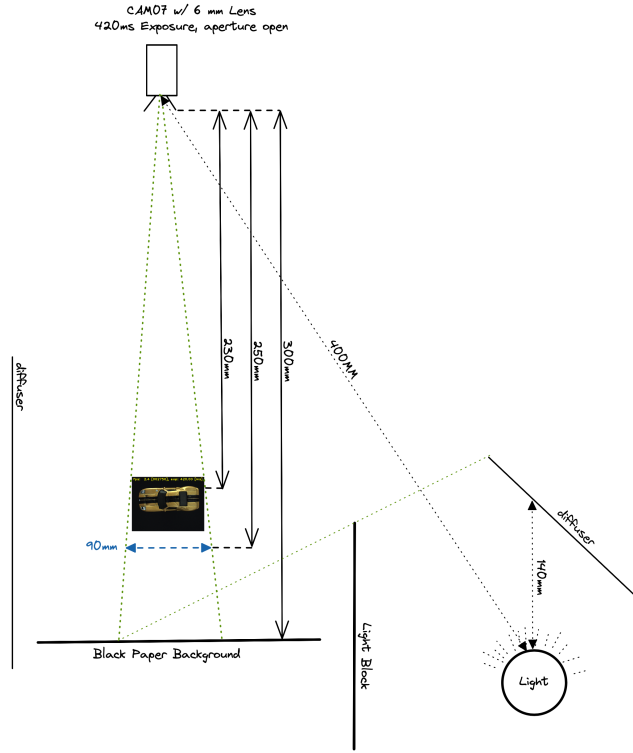


Figure 3: Sketch of setup

## 2.d Calculation of ‘angle of view’

To calculate the angle of view  $\alpha$  we need the following information:

- Sensor size [mm]  $d = 8.89mm$
- focal length  $f = 6mm$

And we used the following formula:

$$\alpha = \frac{180}{\pi} \cdot 2 \arctan \frac{d}{2 \cdot f} = \frac{180}{\pi} \cdot 2 \arctan \frac{8.89}{2 \cdot 6} = 73.1 \text{ degrees}$$

### 3 Assignment 3

#### Moving Object

The description of this assignment was: Take an image of a considerably fast moving object (rotating disk) without any motion-blur and without reflection from any light-sources. You will not be able to synchronize the camera, so find a solution which will not need any synchronization. Make a sketch of the required setup first, discuss multiple solutions in the group. [1]

```
1 // capture 50 frames
2 if (capt50 == true) {
3     capt50 = false;
4     for(int i = 0; i < 50; i++) {
5         cam0.captureFrame(&image);
6         // save image with frame number
7         imwrite("../capt/img" + to_string(i) + ".png", image);
8     }
9 }
```

Listing 3: save image to file

#### 3.a Optimal exposure

We use a strobe light to expose this image. The stroke frequency is not important but should be sufficiently slow to make it impossible for two exposures to occur in a single frame, and it should also be slow so that the capacitors inside the strobe enough time to charge to give the lights its maximum brightness. We took 50 images, and saved them to the file system. The first image which looks like 4 was handpicked and the other images were ignored. The image is slightly underexposed which could have been resolved with a lower RPM setting of the strobe, or by placing the strobe closer to the disc. Adding a difuser would eliminate reflections.

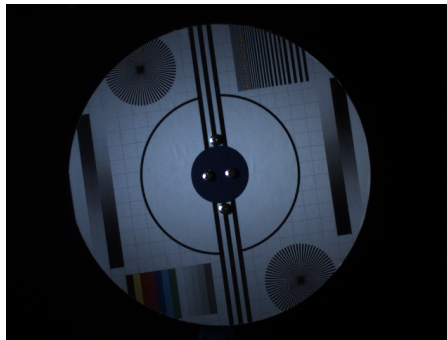


Figure 4: Image of moving object

#### 3.b Sketch of setup

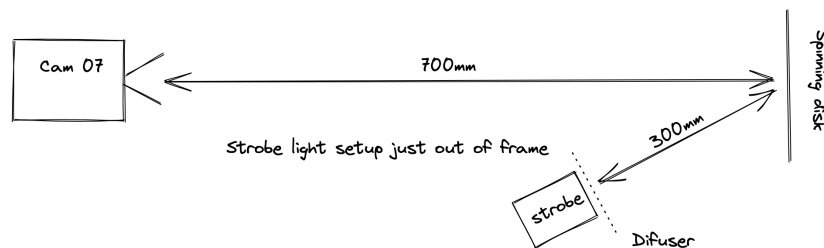


Figure 5: Sketch of setup

### 3.c Calculation of ‘angle of view’

We use the same camera and lens as assignment two, So the angle of view will be identical as calculated in section [2.d](#).



## 4 Assignment 4

### Salt and Pepper Noise

#### 4.a Manipulate own image from assignment 2

In this assignment, a program was used to darken the image and add RGB salt and pepper noise to our original image from assignment 2. The noise must be filtered out and the brightness must be corrected.

We implemented a median filter and a gamma correction function to filter our image. The result is shown in figure 6.



Figure 6: Image with salt and pepper noise

#### 4.b Write C/C++ code for Brightness correction

A Brightness correction filter was written and can be found in appendix B. The filter function is shown in listing 4.

```
1 void gammaCorrection(Mat* input, Mat* output, float gamma) {
2     Size s = input->size();
3     long h, w;
4     float sum;
5
6     std::cout << "Correcting gamma" << std::endl;
7     uint8_t out[s.height][s.width];
8     uint8_t lut[256];
9
10
11     for (int i = 0; i < 256; i++) { //create a lookup table, with the gamma correction curve.
12         lut[i] = saturate_cast<uint8_t>(pow((float)(i / 255.0), gamma) * 255.0f); //saturate
13         //std::cout << unsigned(lut[i]) << " "; //print the function for testing. cout prints
14         //uint8_t as chars so we cast it.
15     }
16
17     for (w=0; w<s.width; w++){ //loop over the image,
18         for(h=0; h<s.height; h++){
19             sum = lut[(input->at<uint8_t>(h,w))]; //the original output value will be scaled
20             //to the value in the LUT.
21             out[h][w]=(uint8_t)sum;
22         }
23     }
24     std::memcpy(output->data, out, s.height*s.width*sizeof(uint8_t)); //copy our standard 2D
25     //array to a new buffer that OpenCV understands
26 }
```

We had several options for correcting brightness. At first we considered simply increasing the value of each pixel by addition. However, this would significantly increase the brightness of the black background turning it dark grey. For this reason we settled on using a gamma correction curve instead. This would spread the histogram "bins" rather than

```
moving them up. Determining the correct gamma correction constant was trial and error,
exponents below 0,5 would make the object too bright.
```

Listing 4: Brightness correction

## 4.c Write C/C++ code for ‘Salt and Pepper Noise’ correction

A ‘Salt and Pepper Noise’ correction filter was written and can be found in appendix B. The filter function is shown in listing 5.

```
1 void filter(Mat* input, Mat* result) {
2     Size s = input->size();
3     long h, w;
4     long sum;
5
6     //std::cout << "Input type was : " << input->type() << std::endl;
7     uint8_t out[s.height][s.width];
8
9     std::cout << s.height << " " << s.width << std::endl;
10
11     for (w=0; w<s.width; w++){ //loop over the image,
12         for(h=0; h<s.height; h++){
13             sum = 0;
14             std::vector<uint8_t> median;
15
16             for (int _x = -1; _x < 2; ++_x) //for every pixel, loop over every pixel in a 3x3
kernel
17                 {
18                     for (int _y = -1; _y < 2; ++_y)
19                     {
20                         int idx_y = h + _y; //sum the incrementor with the kernel's, so we can
identify borders
21                         int idx_x = w + _x;
22
23                         if (idx_x < 0 || idx_x > s.width) //the kernel goes outside of the image,
therefore break and ignore that pixel.
24                             break;
25
26                         if (idx_y < 0 || idx_y > s.height)
27                             break;
28
29                         median.push_back(input->at<uint8_t>(idx_y,idx_x)); // Add all the pixels
from the kernel into a vector
30                     }
31                 }
32                 std::sort(std::begin(median), std::end(median)); //sort all pixel values from high
to low
33
34                 for (auto it = median.begin(); it != median.end(); ++it) {
35                     sum = median.at(median.size()/2); //The pixel at the y,x coordinate is now the
median from our 3x3 sliding window
36                 }
37                 out[h][w]=(uint8_t)sum;
38             }
39         }
40         // this did not work, since the out array was never copied to memory, causing image data
pointing to nowhere!
41         //result = Mat(s.height, s.width, CV_8U, out);
42         // Instead, the data from out needs to be copied directly to Mat result with the correct
size
43         std::memcpy(result->data, out, s.height*s.width*sizeof(uint8_t));
44     }
45 }
```

46 The reason why we chose to implement a median filter is to preserve sharp edges to minimize loss of detail on the object, whereas a Gaussian or mean filter would blur sharp edges. In our code, a 3x3 sliding window is ran over the image, and the center pixel is replaced with the median of the window. The pixel values from the kernel are pushed back into a std::vector, this allows us to use STL library operations like std::sort to order the values. 47 After applying the median filter, the resulting image looks slightly blurred, but sharp transitions are mostly preserved. Note that the 1px wide text at the top of the image is partially removed. This is due to the nature of a median filter and is very challenging to

```
preserve.
```

Listing 5: Noise correction filter

The result after applying the noise removal filter is shown below [7](#).

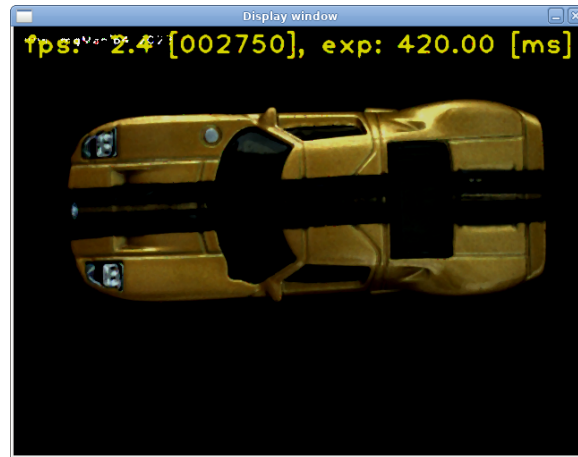


Figure 7: Filtered image

The final result is shown in the figure below [8](#). We chose a gamma correction factor of 0.5 but other values can also be used.

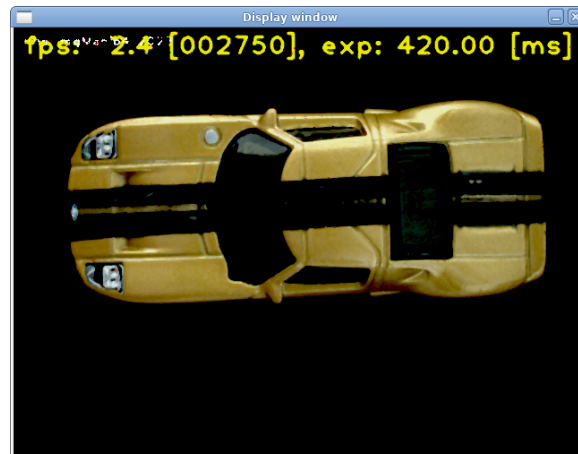


Figure 8: Gamma correction on the filtered image

## 5 Assignment 5

### Convolution

We wrote a convolutie 5x5 kernel with all 1's. The kernel put over the image from lab 2 is shown in figure 9. Note how the output of the filter will slightly blur the image.



Figure 9: 5x5 kernel over image from lab 2

The code we used to make the kernel is shown in listing 6. and the full code can be found in appendix C.

We slightly deviated from the original assignment function prototype by replacing the `int*` argument to a 2D array pointer. We think that is not possible to easily call the kernel by reference without changing this argument, and makes the function more sensible to use. The size is fixed at compile time however.

The code does not make use of any OpenCV iteration templates, which would have drastically improved the performance. However, the assignment stated that it was not allowed to rely on OpenCV functionality despite having to use the `cv2::Mat` class.

The code below will loop over the height and width of the image, and on each pixel loop through the kernel passed as an argument. The result is the use of nested for loops, which quadratically increases processing time. It is highly inefficient and took roughly 13 seconds to execute on my machine (5587.52 BogoMips)

```
1 //int* from original assignment replaced with pointer to the kernel instead, because that
  //makes more sense
2 void convolve5 (Mat* inputImg, Mat* outImg, int (*kernel5)[5][5]) {
3     //I assume the mat is CV_8UC1 since I want to process BGR channels individually
4     Size s = inputImg->size(); // get size of image
5     const uint8_t kernel_size = 5; // todo: replace with sizeof
6     uint8_t out[s.height][s.width]; // create output array
7     int x, y, h, w, i, j, sum; // declare variables
8
9     std::cout << "Input type was : " << inputImg->type() << std::endl; //debug
10
11     for (h=0; h<s.height; h++) { // for each row
12         for (w=0; w<s.width; w++){ // for each column
13             sum = 0; // reset sum
14             for (i=0; i<kernel_size; i++) { // for each kernel row
15                 for (j=0; j<kernel_size; j++){ // for each kernel column
16                     y=h-i+1; x=w-j+1; // calculate the position of the pixel in the image
17                     // if the pixel is outside the image, set it to the border
18                     if (y<0) y=0;
19                     if (y>s.height-2) y=s.height-2;
20                     if (x<0) x=0;
21                     if (x>s.width-2) x=s.width-2;
22                     sum += ((*kernel5)[i][j]) * inputImg->at<uint8_t>(y,x); // add the
  product of the kernel and the pixel to the sum
23                 }
24             }
25             sum /= kernel_size*kernel_size; //divide the result of the pixel by 5^2
26             if (sum<0) sum=0; // if the result is negative, set it to 0
27             if (sum>255) sum=255; // if the result is greater than 255, set it to 255
28             std::memcpy(outImg->data, out, s.height*s.width*sizeof(uint8_t)); // copy the
  result to the output array
29             out[h][w]=(uint8_t)sum; // set the result to the output array
```

```
30     }  
31 }  
32 }
```

Listing 6: 5x5 kernel

## 6 Assignment 6

### Demosaicing Filter

#### 6.a Write C/C++ code for capturing a raw image

```
1 // capture raw image
2 imshow(camName, image);
3 if (captrAW == true) {
4     captrAW = false;
5     // save raw image
6     cfg.camMode = CAM_MODE_RAW;
7     cam0.captureFrame(&image);
8     imwrite("../capt/RAW.png", image);
9
10    // save color image
11    cfg.camMode = CAM_MODE_COL;
12    cam0.captureFrame(&image);
13    imwrite("../capt/COL.png", image);
14 }
```

Listing 7: save image to file

the output of the code is shown in figure 10.

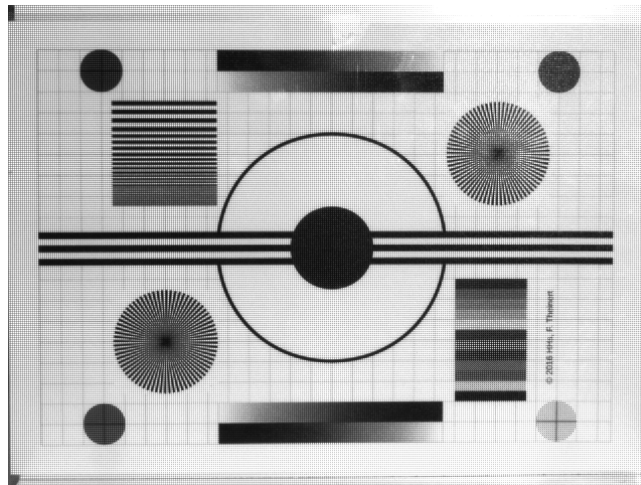


Figure 10: test image

#### 6.b Convert this grayscale image to a color image by implementing a function to recover the actual colors

The function prototype should look as follows:

```
1 void deBayer(Mat *rawImg, Mat *outImg);
```

Listing 8: function prototype

To convert the RAW image, we first split the image into 3 channels, one for each color. Then we interpolate the missing values in the channels. The result is a color image. The function for splitting is shown in listing 9.

```
1
2 for(row=0; row<rawImg->rows; row++){ //todo: make this evaluation smaller to increase speed
3     for(col=0; col<rawImg->cols; col++){
4         if (row % 2 == 0 && col % 2 == 0) //odd row, odd column
5             outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>(row, col);
6         if (row % 2 == 0 && col % 2 == 1) //odd row, even column
7             outImgMID->at<Vec3b>(row, col).val[BGR_RED] = rawImg->at<uint8_t>(row, col);
8         if (row % 2 == 1 && col % 2 == 0) //even row, odd column
9             outImgMID->at<Vec3b>(row, col).val[BGR_BLUE] = rawImg->at<uint8_t>(row, col);
10        if (row % 2 == 1 && col % 2 == 1) //even row, even column
11            outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>(row, col);
```

```

12 }
13 }

```

Listing 9: splitt image

And the function for interpolating is shown in listing 10, where as an example the green channel is interpolated. This function interpolates the missing values in the channels, by using the values of the neighboring pixels. The result is a color image.

```

1 // Interpolate green channel by taking the value of the nearest neighbour that is green
2 std::cout << "Interpolating green channel" << std::endl;
3
4 for(row=0; row<outImgMID->rows; row++){
5     for(col=0; col<outImgMID->cols; col++){
6         if (outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] == 0){
7             if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_GREEN] != 0)
8                 outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>(row, col-1).
9                 val[BGR_GREEN];
10            else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row, col+1).val[BGR_GREEN]
11                != 0)
12                outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>(row, col+1).
13                val[BGR_GREEN];
14            else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_GREEN] != 0)
15                outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>(row-1, col).
16                val[BGR_GREEN];
17            else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1, col).val[BGR_GREEN]
18                != 0)
19                outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>(row+1, col).
20                val[BGR_GREEN];
21        }
22    }
23 }

```

Listing 10: interpolate missing values

This gives the output image shown in figure 11. Which is different from the color image shown in figure 12. The difference between the original and the output, can be explained by an incorrect splitting of the image, or wrong interpolation.

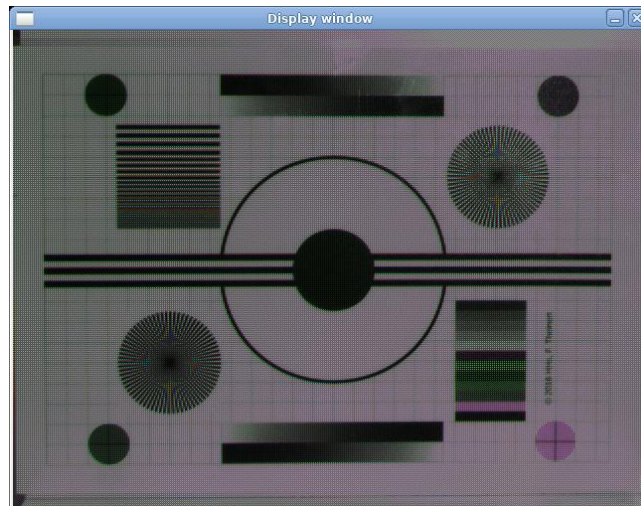


Figure 11: Output image

For the full code see appendix D.



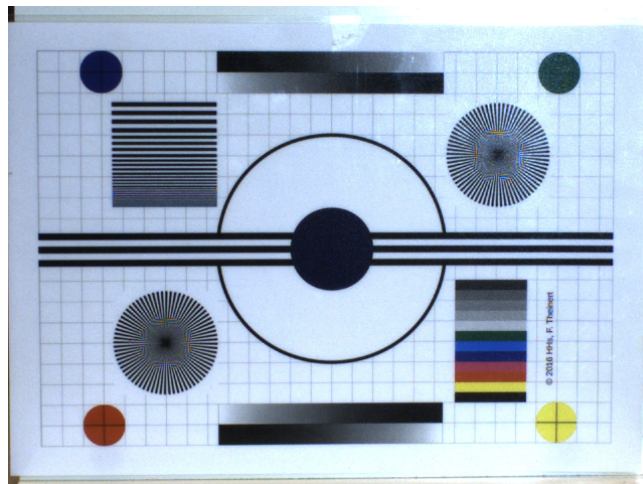


Figure 12: Original image



## References

Theinert, F. (n.d.). *Reader image acquisition and processing*. The Hague University of Applied Sciences.

## A Appendix A

```

1  /*
2   hhs_cam.cpp
3
4   Get frames from DaHeng USB3.0 camera and
5   display them in window
6   Created on: 2022 / 07
7   Author: Fidelis Theinert
8   Reading DaHeng cameras with OpenCV 4.5
9   Version 1.0
10  */
11
12  #include <iostream>
13  #include <string>
14  #include <stdio.h>
15
16  #include <opencv.hpp>
17  #include <highgui.hpp>
18
19  #include "dh0.h"
20
21  // Namespace for using cout.
22  using namespace std;
23
24  // Namespace for OpenCV
25  using namespace cv;
26
27
28  /**
29
30   DEFINITIONS AND MACROS
31
32   ****
33
34   // blue green red is order used in openCV
35   #define COL_BLUE          0
36   #define COL_GREEN         1
37   #define COL_RED           2
38
39   #define COLMODE_COL       0
40   #define COLMODE_GREY      1
41
42   ****
43
44   PROTOTYPES OF NOT EXPORTED FUNCTIONS
45
46   ****
47
48   //int imgSave(int cnt, Mat img, string fname);
49   int Config(int argc, char **argv, struct ImgConf *camCfg);
50   int PrintHelp(void);
51   void InitWindows(string);
52   void ConvertGrey(Mat*);
53
54   ****

```

```

55 **
56 PROTOTYPES OF EXPORTED FUNCTIONS
57
58 ****
59 */
60 /****
61 **
62 DEFINITIONS OF GLOBALS
63
64 ****
65 */
66 int ShowFPS = true;
67 int DisplayMode = COLMODE_COL;
68
69 struct ImgConf {
70     int resolution;
71     int camMode;
72     double exposureMS;
73 };
74
75 //
76 ****
77 int main(int argc, char *argv[]) {
78     //
79     ****
80     double t;
81     int key;
82     int cntframe = 0;
83
84     string camName;
85     char countxt[90];
86
87     struct ImgConf cfg;
88
89     // set default values
90     cfg.resolution = CAM_RES_640_480;
91     cfg.camMode = CAM_MODE_COL;
92     cfg.exposureMS = 12.34; // setting default exposure time in milliseconds
93
94     // set the configuration according to commandline-parameters
95     Config(argc, argv, &cfg);
96
97     // call the constructor and open default camera
98     // if this does NOT succeed the program will abort here (see: constructor)
99     dh cam0(0);
100
101     // declare the matrix where our image is stored
102     Mat image;
103
104     // set camera-mode and exposure-time
105     cam0.setMode(cfg.resolution, cfg.camMode);
106     cam0.setExpoMs(cfg.exposureMS);
107
108     // get camera name
109     cam0.getName(&camName);

```

```

108 cout << "using device '" << camName << "' " << endl;
109
110 // initialize our OpenCV display window
111 InitWindows(camName);
112
113 // discard first image to let camera settle
114 cam0.captureFrame(&image);
115
116 // get systemtime to calculate frame-rate later on
117 t = (double) getTickCount();
118
119 // get actual exposuretime
120 cam0.getExpoMs (&cfg.exposureMS);
121 cout << "Using resolution: " << image.cols << " by " << image.rows
122      << ", exposuretime: " << cfg.exposureMS << " ms" << endl;
123
124 // here the main-loop starts, read one frame
125 while (cam0.captureFrame(&image) == CAM_OK) {
126     // increment frame counter
127     cntframe++;
128
129     // check if retrieving image was successful
130     if (!image.empty()) {
131
132         // check is we have to convert the image to grey-scale
133         switch (DisplayMode) {
134             case COLMODE_COL: // normal color
135                 break;
136
137             case COLMODE_GREY: // grey-scale
138                 ConvertGrey(&image);
139                 break;
140         }
141
142         // check if we have to display frame-rate
143         if (ShowFPS == true) {
144             // define location where to display the frame-rate
145             Point org;
146             org.x = 10;
147             org.y = 30;
148
149             // calculate the expired time since last acquisition of frame
150             t = ((double) getTickCount() - t) / getTickFrequency();
151
152             sprintf(countxt, "fps: %4.1f [%06d], exp: %6.2f [ms]",
153                    (1.0 / t), cntframe, cfg.exposureMS);
154             // sprintf(countxt, "fps: %4.1f [%06d], exp: %6.2f [ms]",
155             //          (1.0 / t), cntframe, cam0.getExpoMs());
156
157             // get new time
158             t = (double) getTickCount();
159
160             // print string to image-buffer
161             putText(image, countxt, org, 1, 2, Scalar(0, 255, 255), 2, 16,
162                    false);
163         }
164
165         // display frame in standard window

```

```

166     imshow(camName, image);
167 }
168
169 // make frame visible
170 key = waitKey(1);
171
172 // if (key != -1)
173 //     cout << "key: '" << key << "' " << endl;
174
175 // check for 'Esc' (or 'backspace' or 'enter') to stop
176 if ((key == 0x1b) || (key == 0x08) || (key == 0x0d)) {
177     cout << "Stopping Cam!" << endl;
178     cam0.close();
179     break;
180 } else {
181     // check for keyboard commands
182     switch (key) {
183
184     case '?':
185         cout << "ROI width = " << image.cols << ", height = "
186             << image.rows << endl;
187         break;
188
189     case 'e':
190         cout << "Exposure time set to: " << cfg.exposureMS << " ms"
191             << endl;
192         break;
193
194     case ' ':
195         // save image using openCV API
196         break;
197     }
198 }
199 }
200
201 return 0;
202 }
203
204 //
205 *****
206 void ConvertGrey(Mat *image) {
207     //
208     *****
209     // go through all cols and rows and convert each pixel to gray value
210     // grey = 0.299 * red + 0.587 * green + 0.114 * blue
211     for (int r = 0; r < image->rows; r++) {
212         for (int c = 0; c < image->cols; c++) {
213             Vec3b &rgb = image->at<Vec3b>(r, c);
214
215             rgb[COL_RED] = (unsigned char) (0.299 * (float) rgb[COL_RED]
216                 + 0.587 * (float) rgb[COL_GREEN]
217                 + 0.114 * (float) rgb[COL_BLUE]);
218             rgb[COL_GREEN] = rgb[COL_RED];
219             rgb[COL_BLUE] = rgb[COL_RED];
220         }
221     }
222 }
223 //

```

```

223 *****
224 int Config(int argc, char **argv, struct ImgConf *camCfg) {
225 //
226 *****
227 // read commandline-parameters one by one
228 if (argc > 1) {
229     for (int i = 1; i < argc; i++) {
230         if (argv[i][0] == '-') {
231             // check for help
232             if (argv[i][1] == '?') {
233                 PrintHelp();
234             }
235             // check for frames per second display
236             if (argv[i][1] == 'F') {
237                 cout << "show FPS!" << endl;
238                 ShowFPS = true;
239             }
240             // check for grey-scale display
241             if (argv[i][1] == 'G') {
242                 cout << "show grey-scale image" << endl;
243                 DisplayMode = COLMODE_GREY;
244             }
245         }
246     }
247 } else {
248     PrintHelp();
249 }
250 return 0;
251 }
252 //
253 *****
254 int PrintHelp(void) {
255 //
256 *****
257 cout << "DaHeng USB3 Camera-Framework, V1.0" << endl;
258 cout << "(c) F. Theinert 2022" << endl;
259 cout << "Commandline options: -F -G -?" << endl;
260 cout << "  -F show frames per second" << endl;
261 cout << "  -G grey-scale image" << endl;
262 cout << "  -? this help-screen" << endl;
263 return 0;
264 }
265 //
266 *****
267 void InitWindows(string camName) {
268 //
269 *****
270 // make HighGui OpenCV window for display
271 namedWindow(camName, WINDOW_AUTOSIZE | WINDOW_GUI_NORMAL);
272 }

```

```
275 |  
276 | /// * EOF hhs_cam.cpp */  
277 |
```



**B   Appendix B**

```

1 #include <iostream>
2
3 #include <opencv2/core.hpp>
4 #include <opencv2/imgcodecs.hpp>
5 #include <opencv2/highgui.hpp>
6 #include <opencv2/opencv.hpp>
7
8 using namespace cv;
9
10 void filter(Mat* input, Mat* result) {
11     Size s = input->size();
12     long h, w;
13     long sum;
14
15     //std::cout << "Input type was : " << input->type() << std::endl;
16     uint8_t out[s.height][s.width];
17
18     std::cout << s.height << " " << s.width << std::endl;
19
20     for (w=0; w<s.width; w++){ //loop over the image,
21         for(h=0; h<s.height; h++){
22             sum = 0;
23             std::vector<uint8_t> median;
24
25             for (int _x = -1; _x < 2; ++_x) //for every pixel, loop over every
pixel in a 3x3 kernel
26             {
27                 for (int _y = -1; _y < 2; ++_y)
28                 {
29                     int idx_y = h + _y; //sum the incrementor with the kernel's, so we
can identify borders
30                     int idx_x = w + _x;
31
32                     if (idx_x < 0 || idx_x > s.width) //the kernel goes outside of the
image, therefore break and ignore that pixel.
33                         break;
34
35                     if (idx_y < 0 || idx_y > s.height)
36                         break;
37
38                     median.push_back(input->at<uint8_t>(idx_y,idx_x)); // Add all the
pixels from the kernel into a vector
39                 }
40             }
41             std::sort(std::begin(median), std::end(median)); //sort all pixel
values from high to low
42
43             for (auto it = median.begin(); it != median.end(); ++it) {
44                 sum = median.at(median.size()/2); //The pixel at the y,x coordinate
is now the median from our 3x3 sliding window
45             }
46             out[h][w]=(uint8_t)sum;
47         }
48     }
49     // this did not work, since the out array was never copied to memory,
causing image data pointing to nowhere!
50     //result = Mat(s.height, s.width, CV_8U, out);
51     // Instead, the data from out needs to be copied directly to Mat result
...

```

```

with the correct size
52     std::memcpy(result->data, out, s.height*s.width*sizeof(uint8_t));
53 }
54
55 void gammaCorrection(Mat* input, Mat* output, float gamma) {
56     Size s = input->size();
57     long h, w;
58     float sum;
59
60     std::cout << "Correcting gamma" << std::endl;
61     uint8_t out[s.height][s.width];
62     uint8_t lut[256];
63
64     for (int i = 0; i < 256; i++) { //create a lookup table, with the gamma
correction curve.
65         lut[i] = saturate_cast<uint8_t>(pow((float)(i / 255.0), gamma) * 255.0f);
//saturate cast negative values to 0, and higher values to 255 (uint8_t or
unsigned char)
66         //std::cout << unsigned(lut[i]) << " "; //print the function for testing.
cout prints uint8_t as chars so we cast it.
67     }
68
69
70     for (w=0; w<s.width; w++){ //loop over the image,
71         for(h=0; h<s.height; h++){
72             sum = lut[(input->at<uint8_t>(h,w))]; //the original output value will
be scaled to the value in the LUT.
73             out[h][w]=(uint8_t)sum;
74         }
75     }
76     std::memcpy(output->data, out, s.height*s.width*sizeof(uint8_t)); //copy
our standard 2D array to a new buffer that OpenCV understands
77 }
78
79
80 int main() {
81     // Read the image (in BGR)
82     Mat img = imread("pixerror.png", IMREAD_COLOR);
83     if(img.empty())
84     {
85         std::cout << "Could not read the image: " << std::endl;
86         return 1;
87     }
88     Size imgsize = img.size();
89
90     // Split the image into 3 new images for blue, green and red.
91     std::cout << "Splitting channels: " << std::endl;
92     Mat bands[3];
93     split(img, bands);
94
95     Mat bandsFiltered[3];
96     Mat bandsCorrected[3];
97     bandsFiltered[0] = Mat(imgsize.height, imgsize.width, CV_8U);
98     bandsFiltered[1] = Mat(imgsize.height, imgsize.width, CV_8U);
99     bandsFiltered[2] = Mat(imgsize.height, imgsize.width, CV_8U);
100
101     bandsCorrected[0] = Mat(imgsize.height, imgsize.width, CV_8U);
102     bandsCorrected[1] = Mat(imgsize.height, imgsize.width, CV_8U);
103     bandsCorrected[2] = Mat(imgsize.height, imgsize.width, CV_8U);

```

```

104
105     filter(&bands[0],&bandsFiltered[0]); //filter all channels from noise
individually
106     filter(&bands[1],&bandsFiltered[1]);
107     filter(&bands[2],&bandsFiltered[2]);
108
109     gammaCorrection(&bandsFiltered[0],&bandsCorrected[0],0.33);
110     gammaCorrection(&bandsFiltered[1],&bandsCorrected[1],0.33);
111     gammaCorrection(&bandsFiltered[2],&bandsCorrected[2],0.33);
112
113     Mat merged;
114     std::vector<Mat> channels =
{bandsCorrected[0],bandsCorrected[1],bandsCorrected[2]};
115     merge(channels, merged);
116
117     // Display the image until q is pressed
118     std::cout << "Displaying result: " << std::endl;
119     imshow("Display window", bands[0]);
120     waitKey(0); // Wait for a keystroke in the window
121     imshow("Display window", bands[1]);
122     waitKey(0); // Wait for a keystroke in the window
123     imshow("Display window", bands[2]);
124     waitKey(0); // Wait for a keystroke in the window
125     imshow("Display window", bandsFiltered[0]);
126     waitKey(0); // Wait for a keystroke in the window
127     imshow("Display window", bandsFiltered[1]);
128     waitKey(0); // Wait for a keystroke in the window
129     imshow("Display window", bandsFiltered[2]);
130     waitKey(0); // Wait for a keystroke in the window
131     imshow("Display window", merged);
132     waitKey(0); // Wait for a keystroke in the window
133     return 0;
134 }
135

```

## C Appendix C

```

1 #include <iostream>
2
3 #include <opencv2/core.hpp>
4 #include <opencv2/imgcodecs.hpp>
5 #include <opencv2/highgui.hpp>
6 #include <opencv2/opencv.hpp>
7
8 using namespace cv;
9
10 void filter(Mat* input, Mat* result) {
11     Size s = input->size();
12     long h, w;
13     long sum;
14
15     //std::cout << "Input type was : " << input->type() << std::endl;
16     uint8_t out[s.height][s.width];
17
18     std::cout << s.height << " " << s.width << std::endl;
19
20     for (w=0; w<s.width; w++){ //loop over the image,
21         for(h=0; h<s.height; h++){
22             sum = 0;
23             std::vector<uint8_t> median;
24
25             for (int _x = -1; _x < 2; ++_x) //for every pixel, loop over every
pixel in a 3x3 kernel
26             {
27                 for (int _y = -1; _y < 2; ++_y)
28                 {
29                     int idx_y = h + _y; //sum the incrementor with the kernel's, so we
can identify borders
30                     int idx_x = w + _x;
31
32                     if (idx_x < 0 || idx_x > s.width) //the kernel goes outside of the
image, therefore break and ignore that pixel.
33                         break;
34
35                     if (idx_y < 0 || idx_y > s.height)
36                         break;
37
38                     median.push_back(input->at<uint8_t>(idx_y,idx_x)); // Add all the
pixels from the kernel into a vector
39                 }
40             }
41             std::sort(std::begin(median), std::end(median)); //sort all pixel
values from high to low
42
43             for (auto it = median.begin(); it != median.end(); ++it) {
44                 sum = median.at(median.size()/2); //The pixel at the y,x coordinate
is now the median from our 3x3 sliding window
45             }
46             out[h][w]=(uint8_t)sum;
47         }
48     }
49     // this did not work, since the out array was never copied to memory,
causing image data pointing to nowhere!
50     //result = Mat(s.height, s.width, CV_8U, out);
51     // Instead, the data from out needs to be copied directly to Mat result
...

```

```

with the correct size
52     std::memcpy(result->data, out, s.height*s.width*sizeof(uint8_t));
53 }
54
55 void gammaCorrection(Mat* input, Mat* output, float gamma) {
56     Size s = input->size();
57     long h, w;
58     float sum;
59
60     std::cout << "Correcting gamma" << std::endl;
61     uint8_t out[s.height][s.width];
62     uint8_t lut[256];
63
64     for (int i = 0; i < 256; i++) { //create a lookup table, with the gamma
correction curve.
65         lut[i] = saturate_cast<uint8_t>(pow((float)(i / 255.0), gamma) * 255.0f);
//saturate cast negative values to 0, and higher values to 255 (uint8_t or
unsigned char)
66         //std::cout << unsigned(lut[i]) << " "; //print the function for testing.
cout prints uint8_t as chars so we cast it.
67     }
68
69
70     for (w=0; w<s.width; w++){ //loop over the image,
71         for(h=0; h<s.height; h++){
72             sum = lut[(input->at<uint8_t>(h,w))]; //the original output value will
be scaled to the value in the LUT.
73             out[h][w]=(uint8_t)sum;
74         }
75     }
76     std::memcpy(output->data, out, s.height*s.width*sizeof(uint8_t)); //copy
our standard 2D array to a new buffer that OpenCV understands
77 }
78
79
80 int main() {
81     // Read the image (in BGR)
82     Mat img = imread("pixerror.png", IMREAD_COLOR);
83     if(img.empty())
84     {
85         std::cout << "Could not read the image: " << std::endl;
86         return 1;
87     }
88     Size imgsize = img.size();
89
90     // Split the image into 3 new images for blue, green and red.
91     std::cout << "Splitting channels: " << std::endl;
92     Mat bands[3];
93     split(img, bands);
94
95     Mat bandsFiltered[3];
96     Mat bandsCorrected[3];
97     bandsFiltered[0] = Mat(imgsize.height, imgsize.width, CV_8U);
98     bandsFiltered[1] = Mat(imgsize.height, imgsize.width, CV_8U);
99     bandsFiltered[2] = Mat(imgsize.height, imgsize.width, CV_8U);
100
101     bandsCorrected[0] = Mat(imgsize.height, imgsize.width, CV_8U);
102     bandsCorrected[1] = Mat(imgsize.height, imgsize.width, CV_8U);
103     bandsCorrected[2] = Mat(imgsize.height, imgsize.width, CV_8U);

```

```

104     filter(&bands[0],&bandsFiltered[0]); //filter all channels from noise
105 individually
106     filter(&bands[1],&bandsFiltered[1]);
107     filter(&bands[2],&bandsFiltered[2]);
108
109     gammaCorrection(&bandsFiltered[0],&bandsCorrected[0],0.33);
110     gammaCorrection(&bandsFiltered[1],&bandsCorrected[1],0.33);
111     gammaCorrection(&bandsFiltered[2],&bandsCorrected[2],0.33);
112
113     Mat merged;
114     std::vector<Mat> channels =
115     {bandsCorrected[0],bandsCorrected[1],bandsCorrected[2]};
116     merge(channels, merged);
117
118     // Display the image until q is pressed
119     std::cout << "Displaying result: " << std::endl;
120     imshow("Display window", bands[0]);
121     waitKey(0); // Wait for a keystroke in the window
122     imshow("Display window", bands[1]);
123     waitKey(0); // Wait for a keystroke in the window
124     imshow("Display window", bands[2]);
125     waitKey(0); // Wait for a keystroke in the window
126     imshow("Display window", bandsFiltered[0]);
127     waitKey(0); // Wait for a keystroke in the window
128     imshow("Display window", bandsFiltered[1]);
129     waitKey(0); // Wait for a keystroke in the window
130     imshow("Display window", bandsFiltered[2]);
131     waitKey(0); // Wait for a keystroke in the window
132     imshow("Display window", merged);
133     waitKey(0); // Wait for a keystroke in the window
134     return 0;
135 }

```



D    Appendix D

```

1 #include <iostream>
2
3 #include <opencv2/core.hpp>
4 #include <opencv2/imgcodecs.hpp>
5 #include <opencv2/highgui.hpp>
6 #include <opencv2/opencv.hpp>
7
8 #define BGR_BLUE 0
9 #define BGR_GREEN 1
10 #define BGR_RED 2
11
12
13 using namespace cv;
14
15 void deBayer(Mat *rawImg, Mat *outImg) {
16     // the Daheng camera datasheet specifies a GRBG pattern (page 88). Our
17     // input values will be 8 bit
18     // Split 1 channel image into 3 channels according to bayer pattern
19     // 0 1
20     //0 G R
21     //1 B G
22
23     //I will create a new "Mat" with OpenCV that contains three channels. The
24     //splitting and processing is done by hand.
25     if (rawImg->type() != CV_8UC1)
26         throw("Sorry, only 1 8-bit channel should be used");
27
28     (*outImg) = cv::Mat::zeros(rawImg->rows, rawImg->cols, CV_8UC3); // Fill
29     // output buffer with zeros with the correct geometry
30     MAT outImgMID = cv::Mat::zeros(rawImg->rows, rawImg->cols, CV_8UC3); //
31     // Fill output buffer with zeros with the correct geometry
32     //Size s = rawImg->size(); I will be using rows and columns rather than
33     //height and width
34     long row, col;
35
36     std::cout << "Bayer splitting to 3 channels" << std::endl;
37
38     for(row=0; row<rawImg->rows; row++){ //todo: make this evaluation smaller
39     // to increase speed
40         for(col=0; col<rawImg->cols; col++){
41             if (row % 2 == 0 && col % 2 == 0) //odd row, odd column
42                 outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>
43                 (row,col);
44             if (row % 2 == 0 && col % 2 == 1) //odd row, even column
45                 outImgMID->at<Vec3b>(row, col).val[BGR_RED] = rawImg->at<uint8_t>
46                 (row,col);
47             if (row % 2 == 1 && col % 2 == 0) //even row, odd column
48                 outImgMID->at<Vec3b>(row, col).val[BGR_BLUE] = rawImg->at<uint8_t>
49                 (row,col);
50             if (row % 2 == 1 && col % 2 == 1) //even row, even column
51                 outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] = rawImg->at<uint8_t>
52                 (row,col);
53             }
54         }
55
56     // Interpolate green channel by taking the value of the nearest neighbour
57     // that is green
58     std::cout << "Interpolating green channel" << std::endl;
59 }

```

```

48
49     for(row=0; row<outImgMID->rows; row++){
50         for(col=0; col<outImgMID->cols; col++){
51             if (outImgMID->at<Vec3b>(row, col).val[BGR_GREEN] == 0){
52                 if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_GREEN] != 0)
53                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
54 (row, col-1).val[BGR_GREEN];
55                 else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
56 col+1).val[BGR_GREEN] != 0)
57                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
58 (row, col+1).val[BGR_GREEN];
59                 else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_GREEN]
60 != 0)
61                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
62 (row-1, col).val[BGR_GREEN];
63                 else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
64 col).val[BGR_GREEN] != 0)
65                     outImg->at<Vec3b>(row, col).val[BGR_GREEN] = outImgMID->at<Vec3b>
66 (row+1, col).val[BGR_GREEN];
67             }
68         }
69     }
70
71     // Interpolate red channel by taking the value of the nearest neighbour
72     that is red
73     std::cout << "Interpolating red channels" << std::endl;
74
75     for(row=0; row<outImgMID->rows; row++){
76         for(col=0; col<outImgMID->cols; col++){
77             if (outImgMID->at<Vec3b>(row, col).val[BGR_RED] == 0){
78                 if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_RED] != 0)
79                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
80 (row, col-1).val[BGR_RED];
81                 else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
82 col+1).val[BGR_RED] != 0)
83                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
84 (row, col+1).val[BGR_RED];
85                 else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_RED] !=
86 0)
87                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
88 (row-1, col).val[BGR_RED];
89                 else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
90 col).val[BGR_RED] != 0)
91                     outImg->at<Vec3b>(row, col).val[BGR_RED] = outImgMID->at<Vec3b>
92 (row+1, col).val[BGR_RED];
93             }
94         }
95     }
96
97     // Interpolate blue channel by taking the value of the nearest neighbour
98     that is blue
99     std::cout << "Interpolating blue channels" << std::endl;
100
101     for(row=0; row<outImgMID->rows; row++){
102         for(col=0; col<outImgMID->cols; col++){
103             if (outImgMID->at<Vec3b>(row, col).val[BGR_BLUE] == 0){
104                 if (col > 0 && outImgMID->at<Vec3b>(row, col-1).val[BGR_BLUE] != 0)
105                     outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
106 (row, col-1).val[BGR_BLUE];

```

```

90     else if (col < outImgMID->cols-1 && outImgMID->at<Vec3b>(row,
91 col+1).val[BGR_BLUE] != 0)
92         outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
93 (row, col+1).val[BGR_BLUE];
94     else if (row > 0 && outImgMID->at<Vec3b>(row-1, col).val[BGR_BLUE] !=
95 0)
96         outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
97 (row-1, col).val[BGR_BLUE];
98     else if (row < outImgMID->rows-1 && outImgMID->at<Vec3b>(row+1,
99 col).val[BGR_BLUE] != 0)
100         outImg->at<Vec3b>(row, col).val[BGR_BLUE] = outImgMID->at<Vec3b>
101 (row+1, col).val[BGR_BLUE];
102     }
103 }
104
105 // done?
106 std::cout << "Done?" << std::endl;
107
108 }
109
110 int main() {
111     // Read the image (in 8bit grayscale)
112     Mat img = imread("test_RAW.png", CV_8UC1);
113     if(img.empty()) {
114         std::cout << "Could not read the image: " << std::endl;
115         return 1;
116     }
117
118     Mat result;
119     deBayer(&img, &result);
120
121     imshow("Display window", result);
122     waitKey(0); // Wait for a keystroke in the window
123     return 0;
124 }

```