

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

October 30, 2022

**THE HAGUE**  
UNIVERSITY OF  
APPLIED SCIENCES

Document version 1.0

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Assignment 1</b>	
	Setup	<b>2</b>
<b>3</b>	<b>Assignment 2</b>	
	Object in front of Dark Background	<b>3</b>
3.a	Object with dark background . . . . .	3
3.b	Optimal exposure . . . . .	3
3.c	Sketch of setup . . . . .	4
3.d	Calculation of ‘angle of view’ . . . . .	4
<b>4</b>	<b>Assignment 3</b>	
	Moving Object	<b>5</b>
4.a	Optimal exposure . . . . .	5
4.b	Sketch of setup . . . . .	5
4.c	Calculation of ‘angle of view’ . . . . .	5
<b>5</b>	<b>Assignment 4</b>	
	Salt and Pepper Noise	<b>6</b>
5.a	Manipulate own image from assignment 2 . . . . .	6
5.b	Write C/C++ code for Brightness correction . . . . .	6
5.c	Write C/C++ code for ‘Salt and Pepper Noise’ correction . . . . .	6
<b>6</b>	<b>Assignment 5</b>	
	Convolution	<b>8</b>
<b>A</b>	<b>Appendix A</b>	<b>10</b>
<b>B</b>	<b>Appendix B</b>	<b>17</b>

# 1 Introduction

we work from the lab assignment [\[1\]](#). They state:

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 they were solved by the students.

The students used the virtual machine and project template provided by the teacher.

## 2 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

## 3 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

#### 3.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 3.c.

#### 3.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.

### 3.c Sketch of setup

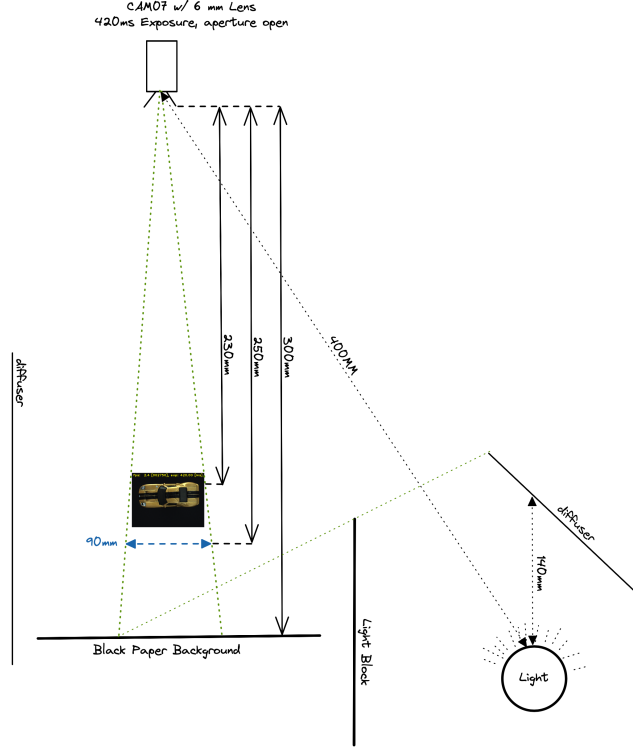


Figure 3: Sketch of setup

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



Figure 4: Angle of view reference image

The (horizontal) angle of view is the angle between the left and right part of the frame, and the camera lens. In order to calculate this we took a picture of a measuring tape stretching the full width of the frame at a known distance, 250 mm. See figure 4 for this image. Now we calculate the angle of view:

$$\text{framewidth} = 90\text{mm}$$

$$\text{distance} = 250\text{mm}$$

$$\theta = 2 \cdot \arctan \frac{\frac{\text{framewidth}}{2}}{\text{distance}} = 2 \cdot \arctan \frac{\frac{90}{2}}{250} = 2 \cdot \arctan \frac{45}{250} = 2 \cdot 19.8 = 39.6 \text{ degrees} \quad (1)$$

## 4 Assignment 3

### Moving Object

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

#### 4.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 5 was handpicked and the other images were ignored.

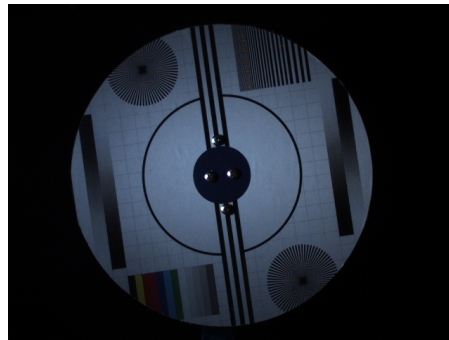


Figure 5: Image of moving object

#### 4.b Sketch of setup

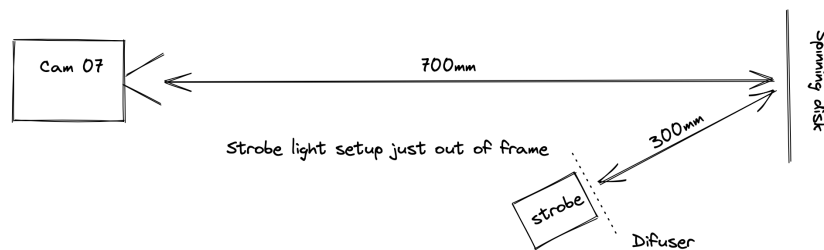


Figure 6: Sketch of setup

#### 4.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 3.d.

## 5 Assignment 4

### Salt and Pepper Noise

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

We applied a salt and pepper noise filter to the image from assignment 2. The image is shown in figure 7.



Figure 7: Image with salt and pepper noise

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

#### 5.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 4.

```
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++){
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)
17             {
18                 for (int _y = -1; _y < 2; ++_y)
19                 {
20                     int idx_y = h + _y;
21                     int idx_x = w + _x;
22
23                     if (idx_x < 0 || idx_x > s.width)
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));
30                 }
31             }
32             std::sort(std::begin(median), std::end(median));
33
34             for (auto it = median.begin(); it != median.end(); ++it) {
35                 sum = median.at(median.size()/2);
```



```

36         }
37         out[h][w]=(uint8_t)sum;
38     }
39 }
40 result = Mat(s.height, s.width, CV_8U, out); //or maybe CV_8UC1?
41 Size _s = result.size();
42 std::cout << "done: " << _s.height << " " << _s.width << std::endl;
43 }
44 }

```

Listing 4: Noise correction filter

The result of output image is shown in figure 8.



Figure 8: Filtered image

## 6 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.



Figure 9: 5x5 kernel over image from lab 2

the code we used to make the kernel is shown in listing 5.

```
1 void convolve5 (Mat* inputImg, Mat* outImg, int* kernel5){
2     //I assume the mat is CV_8UC1 since I want to process BGR channels individually
3     size s = inputImg->size();
4     const uint8_t kernel_size = 5;
5     uint8_t out[s.height][s.width];
6     //int kernel[kernel_size][kernel_size] =
7     {{1,1,1,1,1},{1,1,1,1,1},{1,1,1,1,1},{1,1,1,1,1},{1,1,1,1,1}};
8
9     int x ,y, h, w, i, j, sum;
10
11     for (h=0; h<s.height; h++){
12         for(w=0; w<s.width; w++){
13             sum = 0;
14             for(i=0 ;i<kernel_size; i++){
15                 for(j=0; j<kernel_size; j++){
16                     y=h-i+1; x=w-j+1;
17                     if(y<0) y=0;
18                     if(y>s.height-2) y=s.height-2;
19                     if(x<0) x=0;
20                     if(x>s.width-2) x=s.width-2;
21                     //sum+=kernel[i][j]*inputImg->at<uint8_t>(y,x);
22                     sum += *kernel5[i][j]*inputImg->at<uint8_t>(y,x);
23                 }
24             }
25             sum /= kernel_size*kernel_size;
26             if(sum<0) sum=0;
27             if(sum>255) sum=255;
28             std::memcpy(outImg->data, out, s.height*s.width*sizeof(uint8_t));
29         }
30     }
```

Listing 5: 5x5 kernel

## 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
63 DEFINITIONS OF GLOBALS
64
65 *****/
66
67 int ShowFPS = true;
68 int DisplayMode = COLMODE_COL;
69
70 struct ImgConf {
71     int resolution;
72     int camMode;
73     double exposureMS;
74 };
75
76 //
77 *****/
78 int main(int argc, char *argv[]) {
79     //
80     *****/
81     double t;
82     int key;
83     int cntframe = 0;
84
85     string camName;
86     char countxt[90];
87
88     struct ImgConf cfg;
89
90     // set default values
91     cfg.resolution = CAM_RES_640_480;
92     cfg.camMode = CAM_MODE_COL;
93     cfg.exposureMS = 12.34; // setting default exposure time in milliseconds
94
95     // set the configuration according to commandline-parameters
96     Config(argc, argv, &cfg);
97
98     // call the constructor and open default camera
99     // if this does NOT succeed the program will abort here (see: constructor)
100     dh cam0(0);
101
102     // declare the matrix where our image is stored
103     Mat image;
104
105     // set camera-mode and exposure-time
106     cam0.setMode(cfg.resolution, cfg.camMode);
107     cam0.setExpoMs(cfg.exposureMS);
108
109     // get camera name
110     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++){
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)
26             {
27                 for (int _y = -1; _y < 2; ++_y)
28                 {
29                     int idx_y = h + _y;
30                     int idx_x = w + _x;
31
32                     if (idx_x < 0 || idx_x > s.width)
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));
39                 }
40             }
41             std::sort(std::begin(median), std::end(median));
42
43             for (auto it = median.begin(); it != median.end(); ++it) {
44                 sum = median.at(median.size()/2);
45             }
46             out[h][w]=(uint8_t)sum;
47         }
48     }
49     result = Mat(s.height, s.width, CV_8U, out); //or maybe CV_8UC1?
50     Size _s = result.size();
51     std::cout << "done: " << _s.height << " " << _s.width << std::endl;
52 }
53
54
55 int main() {
56     // Read the image (in BGR)
57     Mat img = imread("fordgt_test.png", IMREAD_COLOR);
58     if(img.empty())

```

```

59     {
60         std::cout << "Could not read the image: " << std::endl;
61         return 1;
62     }
63
64     // Split the image into 3 new images for blue, green and red.
65     std::cout << "Splitting channels: " << std::endl;
66     Mat bands[3];
67     split(img, bands);
68
69
70     Mat bandsFiltered[3];
71     filter(bands[0], bandsFiltered[0]);
72     filter(bands[1], bandsFiltered[1]);
73     filter(bands[2], bandsFiltered[2]);
74
75     // Display the image until q is pressed
76     std::cout << "Displaying result: " << std::endl;
77     imshow("Display window", bands[0]);
78     waitKey(0); // Wait for a keystroke in the window
79     imshow("Display window", bands[1]);
80     waitKey(0); // Wait for a keystroke in the window
81     imshow("Display window", bands[2]);
82     waitKey(0); // Wait for a keystroke in the window
83     imshow("Display window", bandsFiltered[0]);
84     waitKey(0); // Wait for a keystroke in the window
85     imshow("Display window", bandsFiltered[1]);
86     waitKey(0); // Wait for a keystroke in the window
87     imshow("Display window", bandsFiltered[2]);
88     waitKey(0); // Wait for a keystroke in the window
89     return 0;
90 }
91

```