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 28, 2022

THE HAGUE UNIVERSITY OF APPLIED SCIENCES

Document version 1.0

Contents

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 tamplate provided by the teacher.

2 Assignment 1
Setup

For this assignment we connected the Camera to the Virtual Machine. And to the project tempalte a case was addet 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 wich the program was run.

```
    case 's':
        cout << "Saving..." << endl;
        // save image using openCV API
        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 apature 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 exposuretime 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.

```
    case ' ':
        if (imgSave(image, "output.png")) {
            cout << "Image saved succesfully!" << endl;
        } else {
            cout << "Error saving file." << endl;
        }
        break;
        case ',':
        cam0.setExpoMs(--cfg.exposureMS);
        cout << "Exposure adjusted to " << cfg.exposureMS << endl;
        break;
        case '.':
        cam0.setExpoMs(++cfg.exposureMS);
        cout << "Exposure adjusted to " << cfg.exposureMS << endl;
        break;
        case '[':
        cam0.setExpoMs(cfg.exposureMS -= 10);
        cout << "Exposure adjusted to " << cfg.exposureMS << endl;
        break;
        case ']':
        cam0.setExpoMs(cfg.exposureMS += 10);
        cout << "Exposure adjusted to " << cfg.exposureMS << endl;
        break;
```

Listing 2: save image to file

3.a Object with dark background

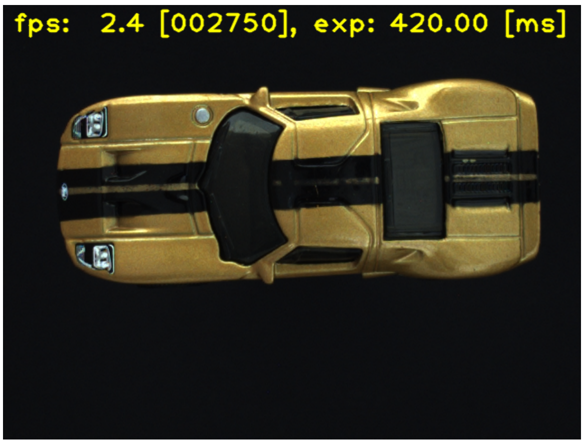

Figure 2: Object in front of Dark Background

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

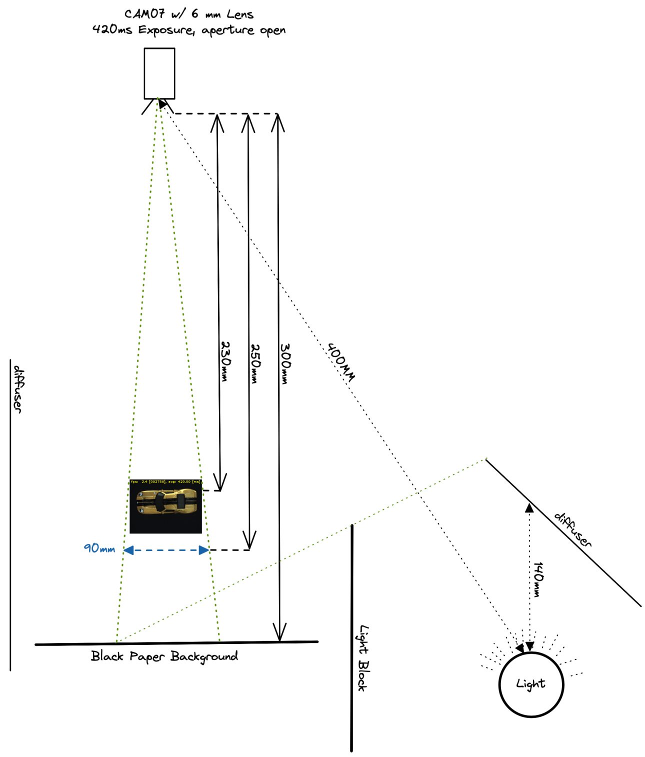

Figure 3: Sketch of setup

3.d Calculation of 'angle of view'

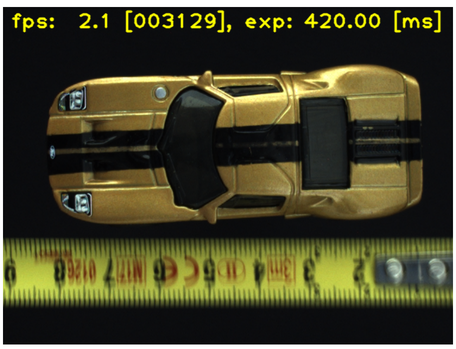

Figure 4: Angle of view reference image

The (horisontal) 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:

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

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

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

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]

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 brightnes. We took 50 images, and saved them to the file system.

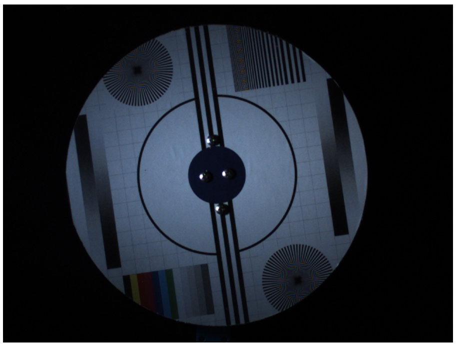

Figure 5: Image of moving object

4.b 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

6 Assignment 5 Convolution

7 Assignment 6 Demosaicing Filter

References

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

A Appendix A

```cpp
/*
 hhs_cam.cpp

 Get frames from DaHeng USB3.0 camera and
 display them in window
 Created on: 2022 / 07
 Author: Fidelis Theinert
 Reading DaHeng cameras with OpenCV 4.5
 Version 1.0
 */

#include <iostream>
#include <string>
#include <stdio.h>

#include <opencv.hpp>
#include <highgui.hpp>

#include "dh0.h"

// Namespace for using cout.
using namespace std;

// Namespace for OpenCV
using namespace cv;


/******************************************************************************

 DEFINITIONS AND MACROS

 *****************************************************************************/

// blue green red is order used in openCV
#define COL_BLUE                  0
#define COL_GREEN                 1
#define COL_RED                   2

#define COLMODE_COL               0
#define COLMODE_GREY              1

/******************************************************************************

 PROTOTYPES OF NOT EXPORTED FUNCTIONS

 *****************************************************************************/

//int imgSave(int cnt, Mat img, string fname);
int Config(int argc, char **argv, struct ImgConf *camCfg);
int PrintHelp(void);
void InitWindows(string);
void ConvertGrey(Mat*);

/****************************************************************************
```

```
**

 PROTOTYPES OF EXPORTED FUNCTIONS

 ****************************************************************************
*/

/****************************************************************************
**

 DEFINITIONS OF GLOBALS

 ****************************************************************************
*/

int ShowFPS = true;
int DisplayMode = COLMODE_COL;

struct ImgConf {
  int resolution;
  int camMode;
  double exposureMS;
};

//
****************************************************************************
int main(int argc, char *argv[]) {
//
****************************************************************************
  double t;
  int key;
  int cntframe = 0;

  string camName;
  char countxt[90];

  struct ImgConf cfg;

  // set default values
  cfg.resolution = CAM_RES_640_480;
  cfg.camMode = CAM_MODE_COL;
  cfg.exposureMS = 12.34; // setting default exposure time in milliseconds

  //  set the configuration according to commandline-parameters
  Config(argc, argv, &cfg);

  //  call the constructor and open default camera
  //  if this does NOT succeed the program will abort here (see: constructor)
  dh cam0(0);

  //  declare the matrix where our image is stored
  Mat image;

  //  set camera-mode and exposure-time
  cam0.setMode(cfg.resolution, cfg.camMode);
  cam0.setExpoMs(cfg.exposureMS);

  //  get camera name
  cam0.getName(&camName);
```

```
  cout << "using device '" << camName << "' " << endl;

  //  initialize our OpenCV display window
  InitWindows(camName);

  // discard first image to let camera settle
  cam0.captureFrame(&image);

  //  get systemtime to calculate frame-rate later on
  t = (double) getTickCount();

  // get actual exposuretime
  cam0.getExpoMs (&cfg.exposureMS);
  cout << "Using resolution: " << image.cols << " by " << image.rows
      << ", exposuretime: " << cfg.exposureMS << " ms" << endl;

  //  here the main-loop starts, read one frame
  while (cam0.captureFrame(&image) == CAM_OK) {
    //  increment frame counter
    cntframe++;

    //  check if retrieving image was successful
    if (!image.empty()) {

      //  check is we have to convert the image to grey-scale
      switch (DisplayMode) {
      case COLMODE_COL: // normal color
        break;

      case COLMODE_GREY: // grey-scale
        ConvertGrey(&image);
        break;
      }

      //  check if we have to display frame-rate
      if (ShowFPS == true) {
        //  define location where to display the frame-rate
        Point org;
        org.x = 10;
        org.y = 30;

        //  calculate the expired time since last acquisition of frame
        t = ((double) getTickCount() - t) / getTickFrequency();

        sprintf(countxt, "fps: %4.1f [%06d], exp: %6.2f [ms]",
            (1.0 / t), cntframe, cfg.exposureMS);
//        sprintf(countxt, "fps: %4.1f [%06d], exp: %6.2f [ms]",
//            (1.0 / t), cntframe, cam0.getExpoMs());

        //  get new time
        t = (double) getTickCount();

        //  print string to image-buffer
        putText(image, countxt, org, 1, 2, Scalar(0, 255, 255), 2, 16,
            false);
      }

      // display frame in standard window
```

```
      imshow(camName, image);
    }

    // make frame visible
    key = waitKey(1);

//    if (key != -1)
//      cout << "key: '" << key << "'  " << endl;

// check for 'Esc' (or 'backspace' or 'enter') to stop
    if ((key == 0x1b) || (key == 0x08) || (key == 0x0d)) {
      cout << "Stopping Cam!" << endl;
      cam0.close();
      break;
    } else {
      // check for keyboard commands
      switch (key) {

      case '?':
        cout << "ROI width = " << image.cols << ", height = "
            << image.rows << endl;
        break;

      case 'e':
        cout << "Exposure time set to: " << cfg.exposureMS << " ms"
            << endl;
        break;

      case ' ':
        // save image using openCV API
        break;
      }
    }
  }

  return 0;
}

//
****************************************************************************
void ConvertGrey(Mat *image) {
//
****************************************************************************
  //  go through all cols and rows and convert each pixel to gray value
  //  grey = 0.299 * red + 0.587 * green + 0.114 * blue
  for (int r = 0; r < image->rows; r++) {
    for (int c = 0; c < image->cols; c++) {
      Vec3b &rgb = image->at<Vec3b>(r, c);

      rgb[COL_RED] = (unsigned char) (0.299 * (float) rgb[COL_RED]
          + 0.587 * (float) rgb[COL_GREEN]
          + 0.114 * (float) rgb[COL_BLUE]);
      rgb[COL_GREEN] = rgb[COL_RED];
      rgb[COL_BLUE] = rgb[COL_RED];
    }
  }
}

//
```

```
// *****************************************************************************
int Config(int argc, char **argv, struct ImgConf *camCfg) {
// *****************************************************************************
  //  read commandline-parameters one by one
  if (argc > 1) {
    for (int i = 1; i < argc; i++) {
      if (argv[i][0] == '-') {

        //  check for help
        if (argv[i][1] == '?') {
          PrintHelp();
        }

        //  check for frames per second display
        if (argv[i][1] == 'F') {
          cout << "show FPS!" << endl;
          ShowFPS = true;
        }

        //  check for grey-scale display
        if (argv[i][1] == 'G') {
          cout << "show grey-scale image" << endl;
          DisplayMode = COLMODE_GREY;
        }
      }
    }
  } else {
    PrintHelp();
  }

  return 0;
}

// *****************************************************************************
int PrintHelp(void) {
// *****************************************************************************
  cout << "DaHeng USB3 Camera-Framework, V1.0" << endl;
  cout << "(c) F. Theinert 2022" << endl;
  cout << "Commandline options: -F -G -?" << endl;
  cout << "  -F show frames per second" << endl;
  cout << "  -G grey-scale image" << endl;
  cout << "  -? this help-screen" << endl;

  return 0;
}

// ****************************************************************************
void InitWindows(string camName) {
// ****************************************************************************

  // make HighGui OpenCV window for display
  namedWindow(camName, WINDOW_AUTOSIZE | WINDOW_GUI_NORMAL);
}
```

```
///* EOF hhs_cam.cpp  */
```