FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Engineering Science

Computer Vision in an Automotive Context: High Dynamic Range Imaging with a Dual Camera System

Dominik Urbaniak

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Engineering Science

Computer Vision in an Automotive Context: High Dynamic Range Imaging with a Dual Camera System

Maschinelles Sehen im Automobilen Umfeld: Dual-Kamera-Abbildungen mit Hohem Dynamikbereich

Author:	Dominik Urbaniak
Supervisor:	Prof. Dr.-Ing. habil. Alois Christian Knoll
Advisors:	Dr.sc.nat. Kai Huang, M.Eng. Xiebing Wang
Date:	April 30, 2018

I confirm that this bachelor’s thesis in engineering science is my own work and I have documented all sources and material used.

Garching, April 30, 2018

(Dominik Urbaniak)

Abstract

Cameras are commonly used in autonomous vehicles and are irreplaceable when it comes to the differentiation of colors or complex object classifications. Additionally, distances can be computed with a dual camera system. Hence, cameras represent an important data source for autonomous driving. There are conditions, however, in which cameras fail to provide as many information as necessary to guarantee safe automated driving. One main challenge is the representation of the real world's dynamic range which describes the difference in luminosity of the darkest and lightest parts of a scene. As most of the currently used cameras only take images with a low dynamic range (LDR), details in the darkest and lightest regions are lost. Autonomous vehicles face situations with high dynamic range when strong light sources like sunlight or headlights are present. Furthermore, the reflections of sunlight on plane objects, like streets and windows often appear in the camera's view. This thesis proves that high dynamic range (HDR) imaging from multiple differently exposed images improves the camera vision in those scenarios and specifically illustrates these results by comparing disparity maps from LDR and HDR images. Despite these enhancements, in this project, HDR imaging is not capable of replacing Lidar sensors at distance measurements, due to the camera's general dependence on external light and therefore difficulties in dark environments. It is recommended to test HDR imaging on object recognition algorithms and tasks that only cameras can take. This thesis encourages doing research on high dynamic range imaging to display the world more realistically which will also be beneficial for computer vision and autonomous driving.

List of Abbreviations

C

CRF - Camera Response Function 4

D

DSLR - Digital Single-Lens Reflex camera 1

F

FPGA - Field-Programmable Gate Array 1

fps - frames per second 29

H

HDR - High Dynamic Range VII

L

LDR - Low Dynamic Range VII

Lidar - Light detection and ranging VII

O

OpenCV - Open Source Computer Vision Library 1

P

PCB - Printed Circuit Board 7

R

Radar - Radio detection and ranging 1

ROI - Region Of Interest 7

S

SGBM - Semi-Global Block Matching 15

Table of Contents

1. Introduction

High dynamic range imaging recently gained popularity in photography and is by now an established tool in image improvement for DSLRs as well as smartphone cameras. The ability of presenting colors more naturally and amplifying contrasts in the darkest and lightest regions of the image is also beneficial for the computer vision in automated vehicles. So far, camera vision is the technology that is being influenced the most by environmental conditions. Hence, vision data from Radar and Lidar sensors are added to account for safety requirements in automated driving. Radar frequencies are not disturbed by fog, rain or snow, which makes it irreplaceable as a unit in the automated vehicle. Same holds for the camera and its ability to distinguish colors and for object classification. Lidar vision is superior to camera vision especially in difficult light situations with back light or in dark environments, as well as the accuracy and simplicity in distance measurement. In case, camera vision improves for those difficult situations, lidar technology might become redundant.

In the following, this thesis first states the limitations of conventional imaging in dynamic range and depicts its consequences for everyday situations of autonomous cars. To improve the car's vision, HDR imaging from multiple exposures is introduced and the developing process with OpenCV is explained. The next chapter names the components of the dual camera system and the software used for the project and illustrates the FPGA programming of the automatic exposure time adjustment as well as the alternation of exposure times which is specifically critical for the creation of HDR videos in the postprocessing. Chapter four discusses the settings of the hardware and the attributes chosen for the HDR computation. Afterwards, the development of disparity maps is outlined which display the distances of objects from the camera. Its accuracy is assessed in chapter six. The results of disparity maps from LDR and HDR images represent the main basis of assessment in this thesis. Hence, chapter seven compares the results under varying light conditions and concludes that in situations with strong backlight, HDR imaging clearly improves the vision. Despite its advantage over LDR images, more processing power is necessary to function with the same frame rate and in real time which is described in chapter nine. At the end, several ideas are stated to find solutions for the remaining challenges as well as other applications for HDR imaging in an automotive context. Concluding that even though it cannot replace Lidar sensors, HDR imaging is superior to LDR imaging and will play an important role for computer vision applications.

2. Dynamic Range in Images

2.1. Conventional Imaging

An image that is taken with only one exposure time is referred to as illustrating low dynamic range (LDR). Those images compress the information to eight bits per pixel. Hence, only 256 different intensities are possible per color channel [1, p. 1].

Conventional screens with maximum luminance of 300 to 400 cd/m² can display these contents. Due to real world conditions with a luminance up to 10^6 cd/m² [2, pp. 1/2], LDR images cannot display any contrasts within the darkest and lightest regions from scenes with large differences in brightness. Hernandez-Juarez et al. [3] published a video of sample footage. Figure 1 displays three frames from that video. Each containing the LDR image at the top and the disparity map at the bottom. Disparity maps represent the distance measurements from dual camera systems. Red color implicates objects at close range, blue objects are further in the distance.

In the first frame, the environment is evenly exposed by the sun. The disparity map hardly shows any distortions. In the second frame, the dual camera faces the sun directly. The disparity map cannot illustrate the correct distances in the regions around the sun and its reflections on the street and windscreen. These regions are too bright for the camera sensor to display on a LDR image. In the third frame, an image from the exit of a tunnel is presented. Due to the camera exposing for the darker surroundings inside the tunnel, the daylight outside is again too light to be displayed correctly. Hence, before exposing for the daylight, the camera only receives an image with a big, white region in the center. Even though in the most situations LDR images will be sufficient to create a decent disparity map, there are too many exceptions in which the autonomous car cannot consult data from the camera. Therefore, improvements are critical.

1: Evenly exposed 2: Backlight 3: Tunnel exit

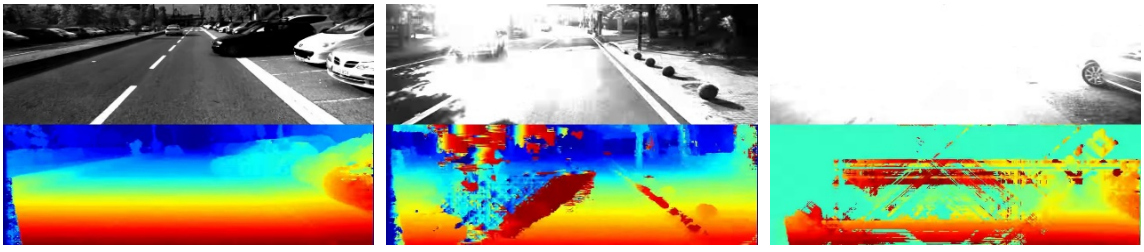

Figure 1 Camera view and disparity map for three different light situations

2.2. High Dynamic Range Imaging

Multiple LDR images with different exposure times are combined to one HDR image. In each of the differently exposed images, a certain group of pixels is well exposed. The first image is taken with the shortest shutter speed and has therefore the average pixel underexposed, while the last image with the longest shutter speed has the average pixel overexposed [1, p. 148]. As illustrated in Figure 2, the shifting in exposure times increases the total number of well exposed pixels when combining the multi-exposed images. However, the contrast decreases, as the higher dynamic range must be compressed to 256 intensity values again.

2.2.1. Alignment

As the images are taken after another, small movements of the camera might lead to "misalignments that blur the results" [1, p. 153]. Therefore, aligning the images is important to receive a sharp HDR result. The MTB-alignment converts each LDR image to a median threshold bitmap (1 for pixels brighter than median luminance and 0 otherwise) and then aligns the resulting bitmaps using the exclusive-or operator [5]. This method is especially developed for HDR creation, as it gives nearly identical results for varying exposure times, whereas the conventional use of edge-detection filters is ill-suited, due to the exposure-dependent appearing and disappearing of edges [1, pp. 156/157].

Figure 2 Three LDR images combined to one HDR image

2.2.2. Derivation of the Camera Response Function

The Camera is not a "perfectly linear light-measurement" device, hence, an individual camera response function (CRF) is computed [1, p. 150]. The Debevec and Malik technique finds irradiance values that minimize a "quadratic objective function" for a set

of the same pixels from differently exposed LDR images to estimate the camera response function f which is explained more detailed in [6].

2.2.3. Merging

With knowledge of the camera response, the HDR radiance map can be constructed.

$$\ln E_i = \frac{\sum_{j=1}^{P} w(Z_{ij})\left(\ln f^{-1}(Z_{ij}) - ln\,\Delta t_j\right)}{\sum_{j=1}^{P} w(Z_{ij})} \quad (1)$$

As declared by Debevec and Malik [6], E_i expresses "film irradiance values", Z_{ij} denotes the pixel value of one specific pixel i in image j, $f^{-1}()$ is the inverted response function and Δt_j represents the exposure time of one LDR image. P is the number of LDR images that is taken to compute the HDR image, in this case two. The weighting function $w()$ is a simple hat function that smooths out the result at the extreme values Z_{min}/Z_{max}.

2.2.4. Tone Mapping

"The Dynamic range of illumination in a real-world scene is […] of the order of 10,000 to 1 from highlights to shadows, and higher if light sources are directly visible". On the other hand, "most display devices […] come with only […] a useful dynamic range of less than 100 to 1" [1, p. 233]. Hence, "contrast experienced in a real environment may greatly exceed the contrast range that can be reproduced by those display devices" [1, pp. 233/234]. Tone mapping is the term that describes the compression of the created HDR image to fit conventional screens while preserving the detail [1, p. 236], [2, p. 4]. Reinhard and Devlin developed an algorithm that "employ[s] a model of photoreceptor adaption" [7]. Therefore, a mixture between the adaption to the intensity of a particular pixel and the average scene is achieved by interpolation. In OpenCV, four parameters are offered to adjust the HDR image before normalizing. Figure 3 illustrates the impact of gamma correction (b), intensity (c) and light adaption (d) on the HDR image (a), which has the properties used throughout the test series in this thesis.

- **Gamma correction:** This value is set to 1.5 in the test series and mainly influences the contrast of the output. Larger value means lower contrast.

- **Intensity:** Within a range [-8, 8] lower values lead to darker results. In this thesis, the intensity is set to two.

- **Light adaption:** This is the parameter, that determines the mixture between pixel and global adaption. Here, global adaption is chosen (no mixture, value equals zero).

- **Color correction:** No color correction is used in the test series (value equals zero).

(a) (b)

(c) (d)

Figure 3 Influence of parameters compared to test settings (a): increasing gamma (b), decreasing intensity (c), adaption to pixel value (d)

3. Preparation of the Dual Camera System

3.1. Hardware

The dual camera system used in the project consists of a single PCB in a rigid case with two lenses mounted on two image sensors with a baseline of 270 mm and one FPGA with integrated soft microprocessor. In the following, specifications to those major parts are given.

- **Kowa LM8XC:** This lens has a focal length of 8.5 mm without fisheye-effect and Its aperture can be adjusted from f/2.8 to f/22.

- **e2v EV76C570:** This CMOS image sensor with optical format 1/1.8" outputs images with a resolution of 1600 x 1200 pixels, offers good performance in low-light situations and multi ROI functions. Its dynamic range reaches approx. 52 dB in global shutter mode [11, p. 2].

- **Altera Cyclone IV EP4CE22 with Nios ii:** This FPGA from Altera offers more than 22 thousand logic elements at low-cost and requires few power.

3.2. Software

Computer vision algorithms are implemented with OpenCV in a Visual Studio 2015 environment utilizing C/C++ and Python. The installation process is explained in a tutorial of "learnopencv.com" [4]. The FPGA of the dual camera system is programmed with the Altera Quartus Prime 16.1 and its Nios ii microprocessor with Eclipse. Videos are recorded with the Windows' camera app. MATLAB is used for the stereo calibration of the cameras.

3.3. FPGA Programming

The FPGA and its microprocessor are programmed to function as a camera. It reads the information from both image sensors and outputs the data as one image of 3200 x 1200 pixels with both camera views included (Figure 4).

Figure 4 Original camera output

3.3.1. Automatic Exposure Adjustment

The camera system adjusts the exposure time automatically, according to the sum of all pixel intensities. For 256 values per pixel from black to white, a range from 122 (I_{low}) to 124 (I_{up}) is desired. Thus, the overall upper and lower limits are the sum of all pixels' intensities from both image sensors.

$$\sum I_{up/low} = N_{sensors} \times N_{pixels,h} \times N_{pixels,v} \times I_{up/low} \quad (2)$$

In case that range is exceeded, the exposure time is being shortened or enlarged. The greater the change of brightness in contrast to the previous frame, the greater the change of the exposure time. These values are split to an integer and fractional part and translated to hexadecimal numbers. To ensure an equal adjustment, changes are dependent on the current exposure time, as long exposure times need a larger change value than small exposure times to output the same effect in actual darkening or lightening the image. Hence, the fractional part is necessary for small exposure times to allow for small changes. When the maximum exposure time is reached, analog and digital gain are added to further correspond to the light conditions.

3.3.2. Images with Alternating Exposure Times

The second main programming part creates a video which records frames with alternating exposure times. This is solved by exploiting the "regions of interest" (ROI) that the image sensor offers. Up to four ROIs can be activated [11, p. 47] and a different exposure time can be set for each ROI individually. As a result, the frames from all activated ROI are displayed sequentially. For this project, two ROI are activated. First, the fractional part is activated for each ROI [11, pp. 50/52]. Then, the integral and fractional part are combined by multiplying t_{int} with a_{int} (= 1000) and adding t_{frac} to receive one integer number, as shown in equation (3). The fractional part t_{frac} must be

first converted from a hexadecimal range (0x0 – 0x88) to a three-digit decimal number with a range from 0 to 999. Here, the maximum value for the fractional part $t_{frac,max}$ is 136 (equals 0x88). The difference factor x_{diff} is editable and adjusts the difference of the light intensity between both LDR images. Therefore, a factor of “one” results in a conventional output with both ROI having the same exposure time. The value sixteen represents the maximum value of the test series which is illustrated in 4.2.1. This factor is multiplied to the darker exposure time in (4). In the postprocessing every two LDR frames are merged to one HDR image.

$$t_{exp\_dark} = t_{int} \times a_{int} + t_{frac} \times \frac{a_{int} - 1}{t_{frac,max}} \tag{3}$$

$$t_{exp\_bright} = t_{exp\_dark} \times x_{diff} \tag{4}$$

The complete code of the function that adjusts the exposure time is attached to the appendix (p.39).

4. Project Attributes

4.1. Settings

For automotive applications, fast shutter speeds are crucial. The camera inside a car is moving rapidly as well as objects in the surroundings. Moreover, light conditions vary enormously. To function in dark environments as well, all camera configurations are set to be light sensitive while maintaining good image quality.

1. Aperture is set to the minimum of f/2.8, which means that the lenses let through as much light to the image sensor as possible, allowing for shorter exposure times. Low apertures also lead to larger depth in field, which is not desired as focus is fixed to a certain distance. However, test images show sharp lines throughout distances greater than one meter, when setting the focus distance to 0.5 m, such that there is no impact on the disparity map creation.

2. Analog and digital gain influence the sensitivity to light of the image sensors. According to the datasheet from the image sensor [11, p. 51], analog gain (ag) can be adjusted from one to eight and digital gain (dg) can be added with a factor up to 15.875. Best image quality is received with analog gain equaling two and without digital gain (dg = 1). Figure 5 illustrates the impact on the image quality when increasing the parameters. A grey overlap is created from analog gain and graining effects from digital gain. To maintain a sufficient quality, maximum analog gain is set to four, maximum digital gain to a factor of 3.675.

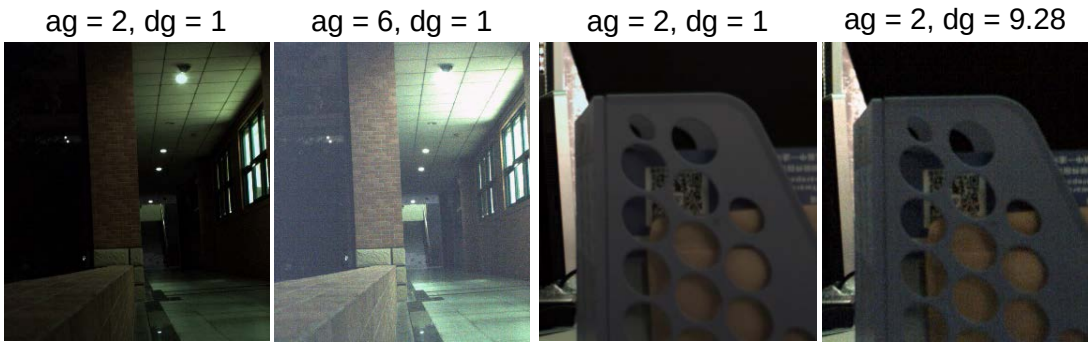

Figure 5 Impact of analog and digital gain on image quality

3. Maximum exposure time is set to 1/24th of a second, in order to be able to record a video with 22 fps. In case, the image still is too dark, analog gain and digital gain are added until all three attributes reach their maximum values. When a scene lightens

up, first analog and digital gain are being set to their default values two and one, respectively.

4.2. Testing Procedure

Video footage is recorded and processed offline with OpenCV as real-time processing is not possible due to required processing time of HDR- and disparity map computations and a deficiency in memory on the PCB to store an LDR image. High dynamic range imaging is achieved with two differently exposed frames, creating a HDR video with eleven fps.

4.2.1. Exposure Time Difference for the HDR Computation

Test results were performed with a different factor in between the exposure times, such that the lighter image has either four-, eight- or sixteen times longer exposure time than the darker one. The larger the difference, the better it is suited for situations with strong light contrasts. However, as seen in Figure 7, maximum exposure time is reached faster. Therefore, LDR images are well below maximum exposure time at 22 m distance from the tunnel exit when HDR images with a factor eight need to add analog and digital gain. Furthermore, when creating the HDR image with only two LDR images, the maximum difference factor is limited. Figure 6 shows two LDR images with a difference factor of sixteen. Even though in the dark image outside areas are still overexposed, a larger difference factor cannot be taken. The bigger the difference factor, the fewer areas are visible in both images which makes it difficult to align them.

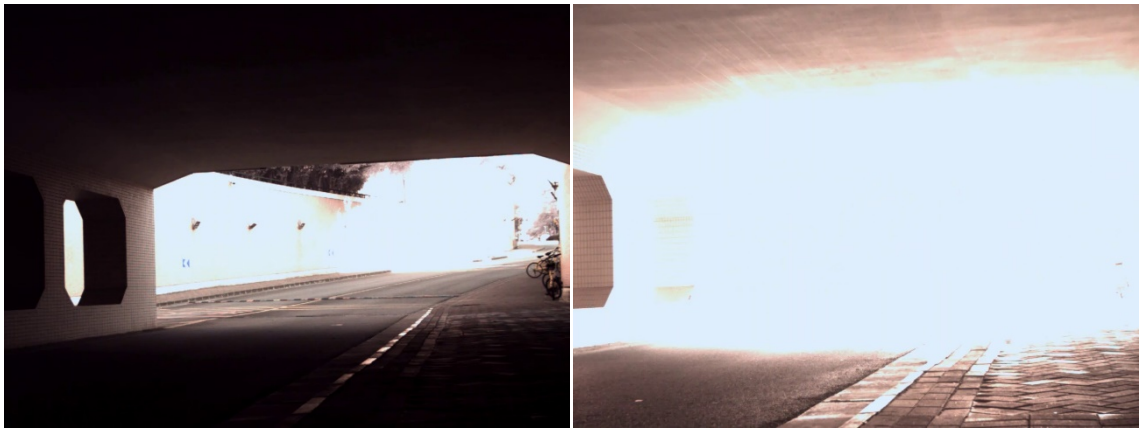

Figure 6 Bright image (right) with 16 times longer exposure time than the dark image (left)

4.2.2. Exposure Time Limits

In order to obtain an objective indication of the camera's capability in low light situations, light intensities are measured with a lux meter. While passing through the tunnel the luminance data were collected from the lux meter and the camera sensors as well as the exposure time and the digital and analog gain. The lux meter is positioned in the same

direction as the lenses. Figure 7 shows a graph that illustrates the relation of the exposure time and the light intensities measured by the lux meter. Being more than 30 m into the tunnel and facing towards the exit, about 40 lux reach the camera. At 27.5 m, digital and analog gain are supporting the image sensors to achieve the average pixel value of 123. For LDR images that target value is reached at any position in the tunnel. For HDR imaging however, the lighter frame must reach a value above the target value, which is not possible with given parameters.

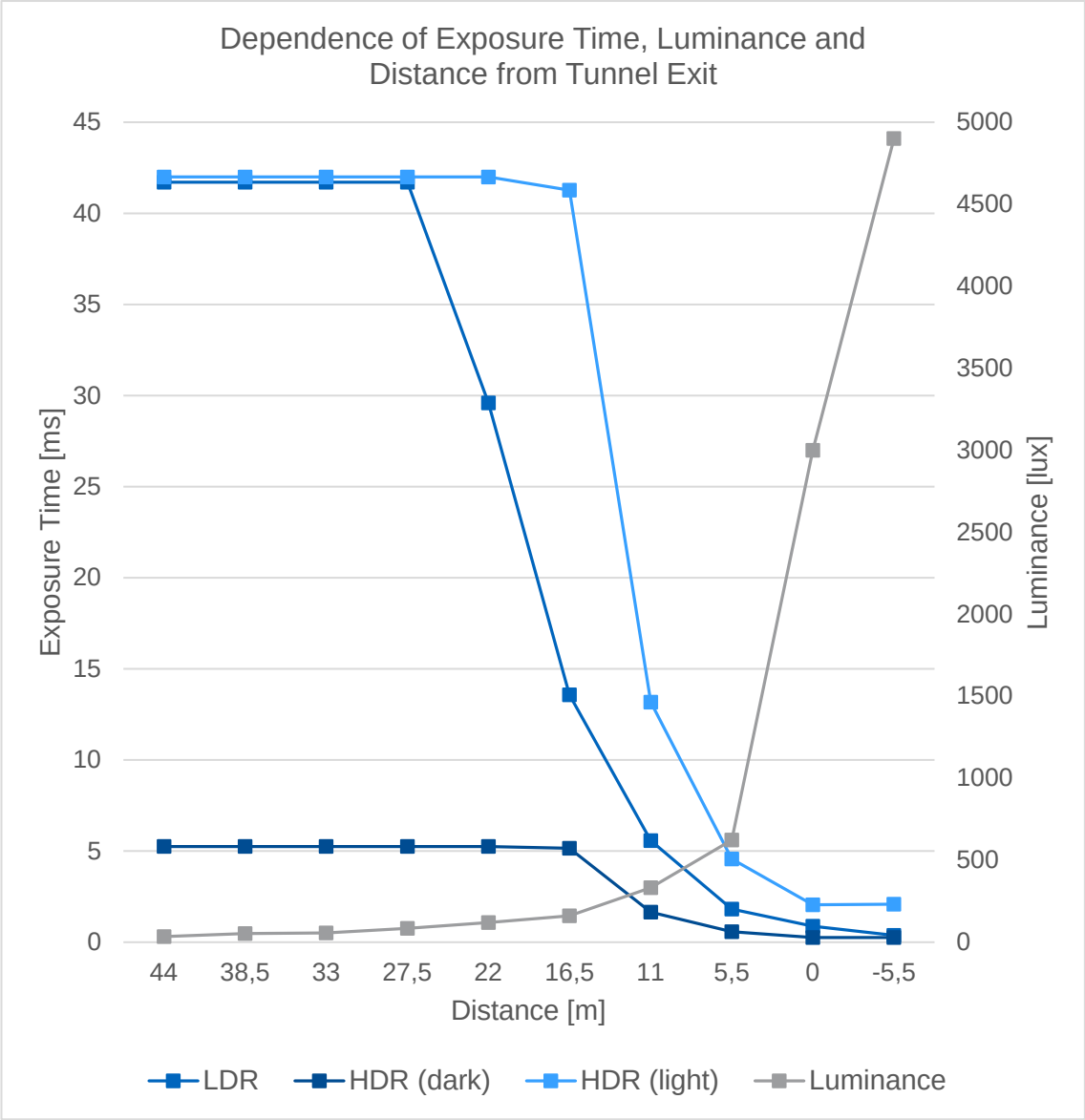

Figure 7 Dependence of exposure time, distance from tunnel exit and measured luminance

5. 3D-Imaging

In order to assess the influence of HDR imaging on three dimensional applications, disparity maps are analyzed. OpenCV offers an approach with two algorithms. Semi Global Block Matching (SGBM) is preferred due to its better accuracy even though it comes with longer runtime. The SGBM algorithm was developed by Hirschmüller in [8].

5.1. Depth information from stereo views

To extract depth information from a setting, dual camera systems employ the same principles as the human vision system. Two lenses view the same scene from slightly different perspectives. By matching the positions of an object in each view, its distance can be estimated. Holding a finger closly in front of one's eyes, when shutting each eye alternately, one can observe that the finger "shifts left and right relative to the background" [18, p. 535]. This horizontal motion is called *disparity* and is "inversly proportional to the distance of the observer" [18, p. 535].

5.2. Calibration of the Dual Camera

Intrinsic and extrinsic information of the camera is derived by stereo calibration in MATLAB. Main advantage to the calibration process in OpenCV is the capability to recognize chessboard corners without the necessity of a white margin on all chessboard edges. Hence, a large chessboard with 10 x 10 cm square size can be used. Several images from different perspectives and distances are taken, such that each shot displays the chessboard from the left and the right camera (Figure 8). Inserting those to MATLAB's "Stereo Calibration App" gives distortion values, principal point and focal length of each lens as well as the translation and rotation vector of one camera to the other [10].

Figure 8 Rectified image from dual camera with drawn horizontal lines

5.3. Disparity map creation

```
1.  cv::FileStorage  fs2("Stereo_Calibration",  cv::FileStorage::READ);
2.  fs2["K1"]  >>  K1;
3.  fs2["D1"]  >>  D1;
4.  fs2["K2"]  >>  K2;
5.  fs2["D2"]  >>  D2;
6.  fs2["R"]  >>  R;
7.  fs2["T"]  >>  T;
8.  fs2.release();
9.
10. stereoRectify(K1,  D1,  K2,  D2,  imgSize,  R,  T,  R1,  R2,  P1,  P2,  Q,  CV
    _CALIB_ZERO_DISPARITY,  -1.0,  imgSize,   & roi1,   & roi2);
11.
12. initUndistortRectfyMap(K1,  D1,  R1,  P1,  Size(w / 2,  h),  CV_16SC2,  map1x,
      map1y);
13. remap(left,  left_rec,  map1x,  map1y,  INTER_LINEAR);
14.
15. Ptr < StereoSGBM >  sgbm1  =  StereoSGBM::create(0,  16,  3);
16. sgbm1 - > compute(right_new,  left_new,  disp);
17.
18. double  min,  max;
19. minMaxIdx(disp,   & min,   & max);
20. double  diff  =  max  -  min;
21. disp  =   (disp  +  160)  /  (diff  /  255);
22.
23. applyColorMap(disp8,  dispCol,  COLORMAP_JET);
```

Listing 1 Essential code for the disparity map creation with OpenCV

In the next step, the received values are used to create the camera and distortion matrix for each camera. These matrices and the translation and rotation matrix are included in OpenCV's `stereoRectify` function with a text file which is being read out by the `FileStorage` method from OpenCV (Listing 1**,** ll. 1-8). To rectify the frames from left and right camera, such that all pixels that are present in both images are located at the same height on the y-axis as illustrated in Figure 8, the functions `stereoRectify` (l. 10) is called, as well as `initUndistortRectifyMap` (l. 12) and `remap` (l.13) for the left and right image. Those rectified images are the basis of successfully creating a disparity map with the `StereoSGBM`-function (ll. 15/16) in OpenCV. After adjusting all parameters (see appendix) a clear disparity map with reduced noise is received (Figure 9). In OpenCV's tutorial section, the article in [9] with a full code example on GitHub contributed to the disparity map creation in this thesis. To reduce computation effort, instead of computing a disparity map for the "left view" and the "right view" plus combining these with a filter computation, here only one disparity map is computed. As the disparity maps from "left- "and "right view" display different distances, the combination results in an image that maps a wide range. However, when analyzing all pixel values from one computation before compressing to eight bits, one can find the values from the other computation with a negative sign. By converting all values manually to eight-bit integers before compression, the resulting disparity map shows the combined outcome with the same wide range (ll.

18-21). Eventually, color mapping (l. 23) is performed to achieve a better contrast in contrast to the original greyscale output (Figure 10). The full code of this disparity map creation is attached to the appendix (p. 43).

Figure 9 Original “left-view” disparity map

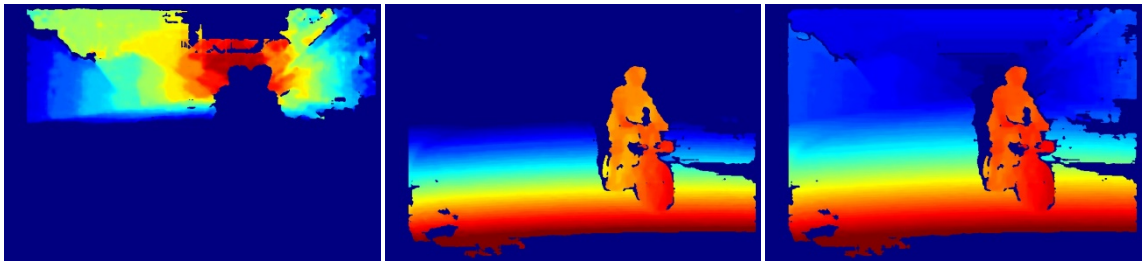

Figure 10 Colored "right view" (left), "left-view" (middle) and self-compressed (right) disparity maps

6. Distance Measurement

```
1.  Mat img3D(dM.size(), CV_32FC3);
2.  reprojectImageTo3D(dM, img3D, Q, true, CV_32F);
3.
4.  void printDistance(Mat in , int x, int y) {
5.      Point3f p = in .at < Point3f > (y, x);
6.      cout<<"Distance at ("<<x<<","<<y<<"): "<<-2*(p.z)<<" meter"<<endl;
7.  }
```

Listing 2 Essential code for distance calculation

When evaluating the disparity maps, it is meaningful to derive the actual distance in meters. Results can be easily compared to real measurements and the reasonability assessed. The function `reprojectImageTo3D` in OpenCV computes the distances for each pixel in the disparity map (Listing 2, l. 2). Apart from the greyscale disparity map, this function takes the “perspective transformation matrix” Q [11] as input, which is obtained from the `stereoRectify` - function (Listing 1, line 10). The output “img3D” is a matrix of the same size as the input disparity map. Inserting that result in the `printDistance` function (ll. 4-7), for each given (x, y)-position, the corresponding z-value multiplicated with minus two, equals the distance in meters (l. 6). The program draws circles on the colored disparity map to display the evaluated pixel, outputs its distance value and allows for navigating the circle through the image with the keyboard. It is attached to the appendix (p.49). In a test series, a plane object (with chessboard look) is being photographed from different distances (Figure 11). The real distances are measured and compared to the outcomes from the disparity map calculation. Figure 12 shows the results of the 19 test images and their deviation in percentage of the actual distance. One can see that the minimum distance is at 4.5 m and that the measurement in accurate. Only a maximum of 4% deviation at distances up to 25 m.

Figure 11 Test environment of distance measurement of chessboard

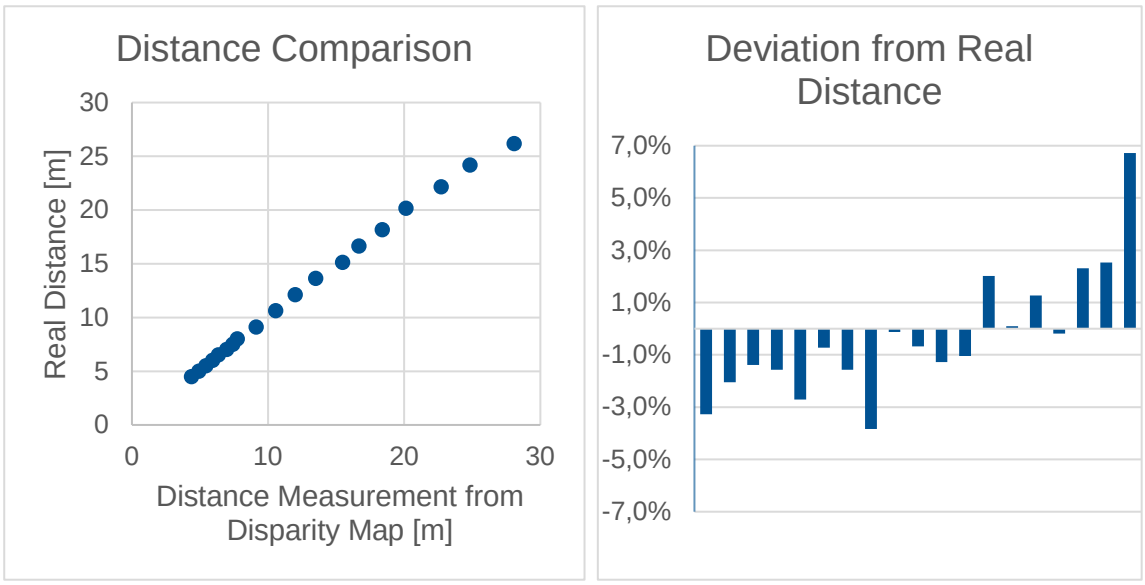

Figure 12 Real distance vs. disparity distance (left) and its deviation (right)

7. Comparison of Disparity Maps from HDR and LDR Images

7.1. Normal Light Situation

The first test is performed to compare the disparity maps from HDR- and conventional images in a situation with good light. In Figure 13, beside the conventional image, HDR footage with a difference factor of four and eight are shown. The result from the conventional method performs best as its disparity map gives the smoothest transitions of depth changes. The disparity maps from HDR images show increasingly noise and areas on the street without depth information. Hence, due to better quality of disparity information and less necessity for computations, conventional imaging is preferred in good light situations.

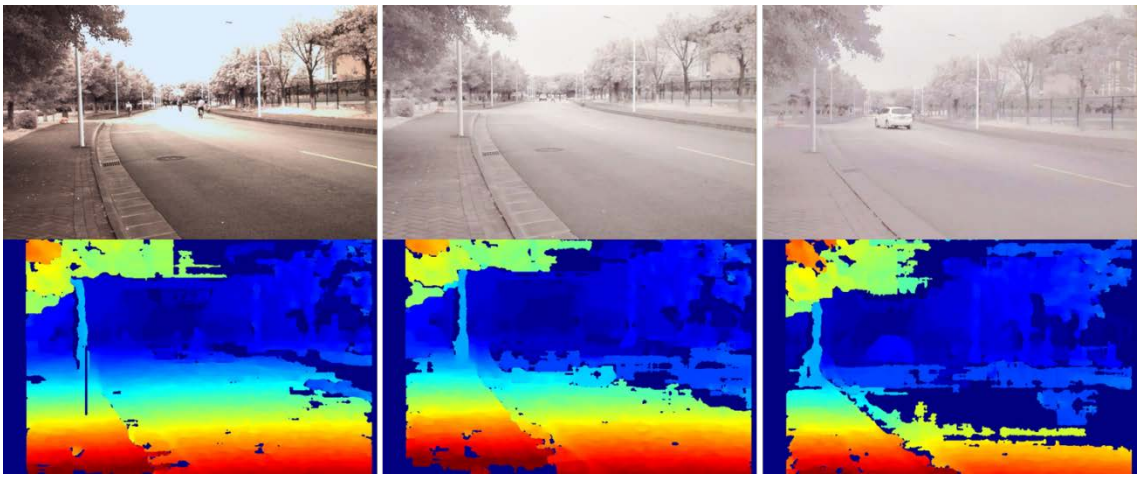

Figure 13 LDR image (left) compared to HDR images with difference factor of 4 (middle) and 8 (right) and their disparity maps

7.2. Tunnel Situation

When passing through a tunnel the camera exposes first for the light environment outside the tunnel. Approaching the tunnel entrance, the darker tunnel region increases on the image until the camera exposes for the dark tunnel environment after entering the tunnel. Then, the bright tunnel exit is approaching until ultimately the camera adjusts the exposure time to the daylight again. Most interesting for the thesis' topic are the situations where the light situation changes. As shown in Figure 1, the camera couldn't display any events that were only a few meters ahead. Even though the situation of a tunnel exit is more difficult due to the backlight, a tunnel entrance is considered to evaluate the HDR performance in medium light conditions as well.

7.2.1. Facing Tunnel Entrance

After examining image material from different distances, a test series 30 m away from the tunnel exit is taken. The goal of making objects inside the tunnel more visible is achieved as presented in Figure 15 for LDR image and HDR image with exposure-difference factor of four, eight and sixteen. Those shots show the tunnel section of the original output in Figure 14 and its color mapping is shifted to stress the changes in distance from 30 m and further. For comparability, a person is approaching the camera in all tests the same way and identified on the disparity maps with a drawn circle (◎). The person seems to be displaced to the left in the disparity maps compared to the image, due to the display from the right camera only. The results from HDR images are slightly better. Especially on the HDR images themselves, the person is easier to discover. However, also the disparity maps from the LDR image show the person in the third frame even though it is not visible on the image. Furthermore, the person cannot be identified in any first frame's disparity maps. At more than 50 m, LDR images still receive enough information to confidently display the environment at a tunnel entrance situation and hence, a disparity map creation from HDR images is not critical.

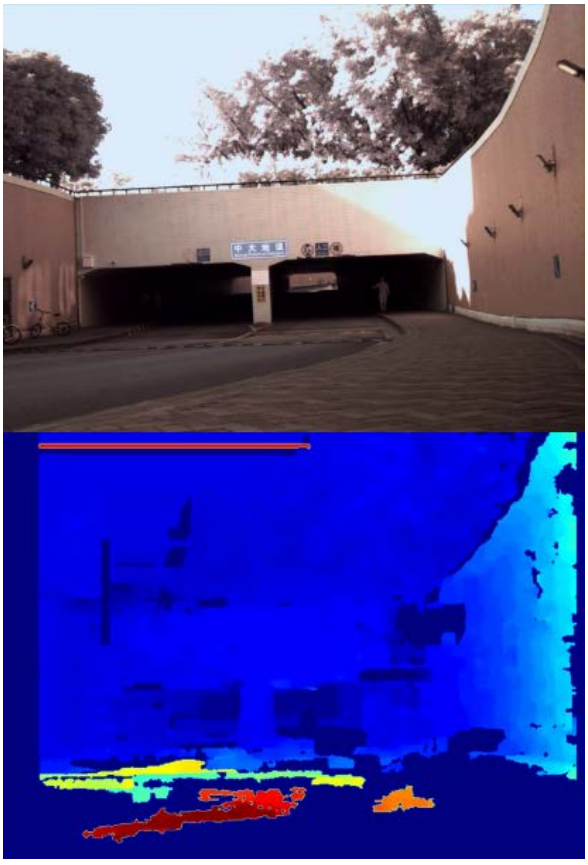

Figure 14 Original camera view and disparity map coloring

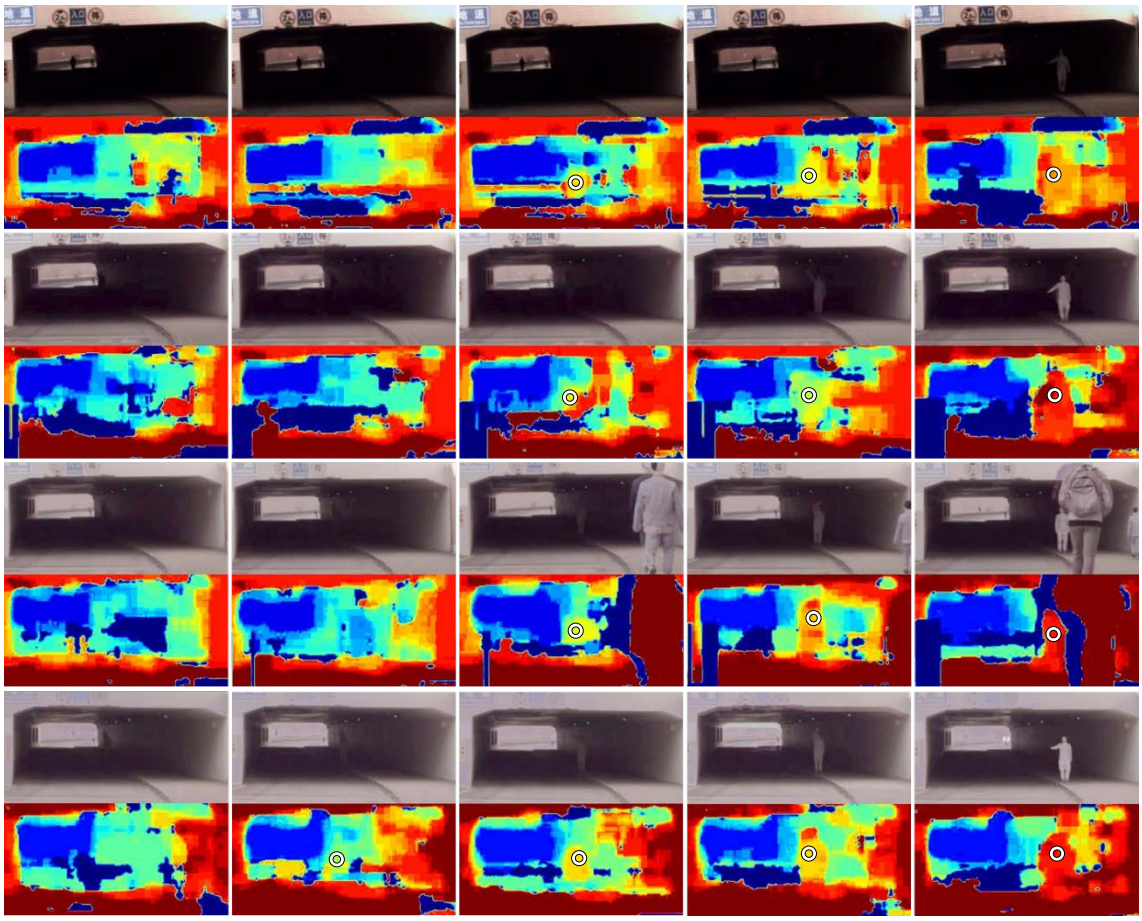

Figure 15 Comparison of LDR and HDR images and disparity maps at tunnel entrance situation

7.2.2. Facing Tunnel Exit

The more challenging scenario is performed similarly. In Figure 16, results from 14 to 35 m distance are displayed for the LDR and HDR images. Only 14 m away from the tunnel exit, LDR images cannot identify the person correctly. Interestingly, when the person stands directly at the exit, the disparity map expands it through the whole exit area. Analogically, the person 20 m ahead is also displayed as an obstacle across the street. In contrast to that, the disparity maps from HDR images achieve a clear enhancement. Especially the HDR image with a difference factor of 16 differentiates the person and the empty street confidently in every disparity map created from the captured video up to the person's distance of 29 m from the camera. It can be observed, that the effect of expanding an obstacle's width in x-direction appears in disparity maps from LDR images as well as from HDR images. First, an obstacle can be differentiated from its environment. When the object's visibility decreases through strong backlight, as seen in the sample outputs, that obstacle is displayed in the disparity map throughout a wide range where the camera doesn't detect other contrasts. Ultimately, no obstacle can be seen on the image and the disparity map displays random distances.

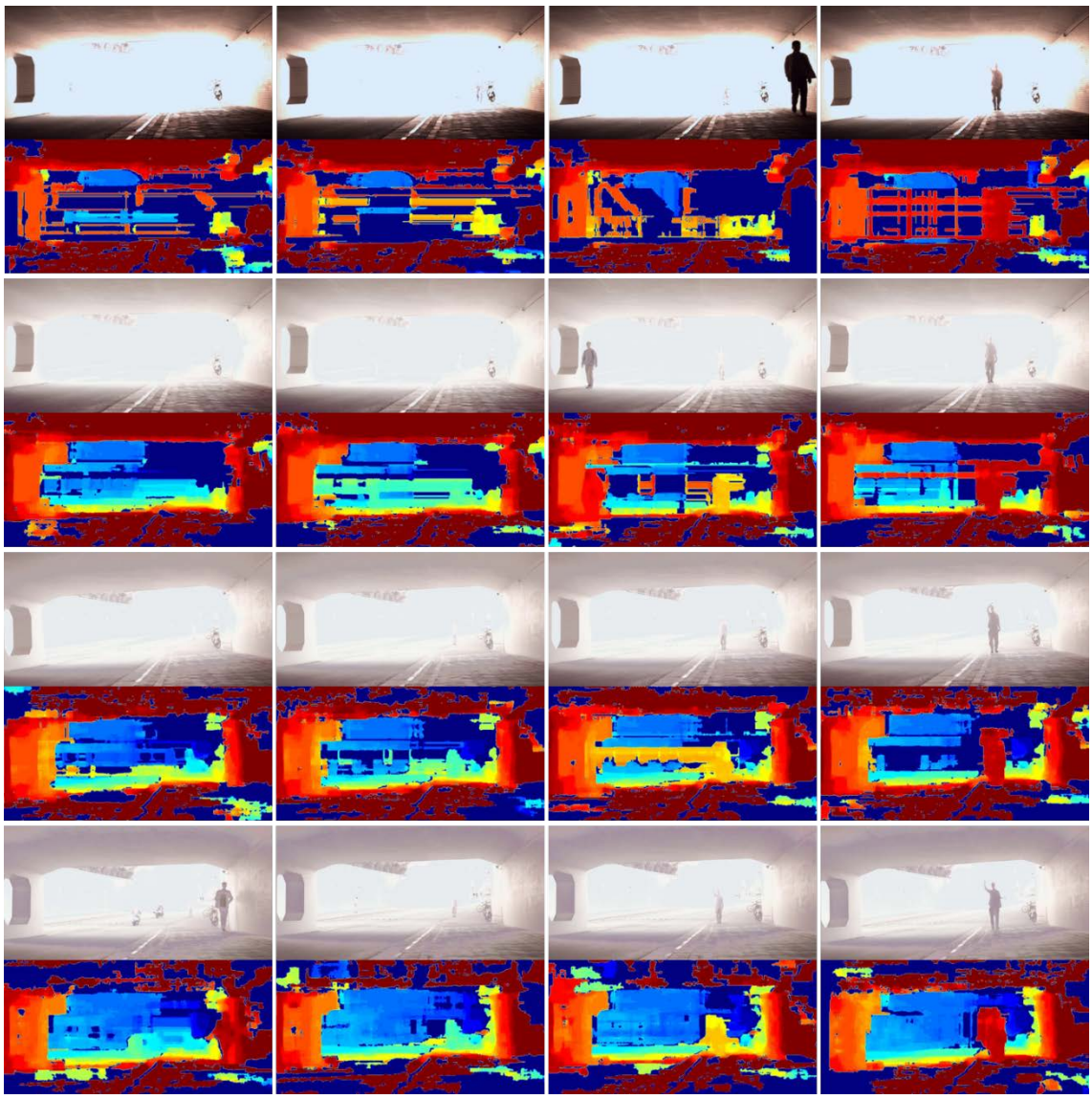

Figure 16 Comparison of LDR and HDR images and disparity maps at tunnel exit situation

The major improvement of the HDR results give an autonomous car information about obstacles that are at least twice as far away as from conventional images in a tunnel exit situation. Figure 17 shows the output from HDR imaging with difference factor of 16 with the originally colored disparity map. An unrecognizable, low-contrast wall, reflecting sunlight, at about 50 m distance and visible background further than 50 m are highlighted in rectangles. As the person only walks in front of the bright wall, it disappears at more than 30 m from the camera. However, with a background's higher contrast HDR results prove the capability of obstacle distinction even at distances of more than 50 m.

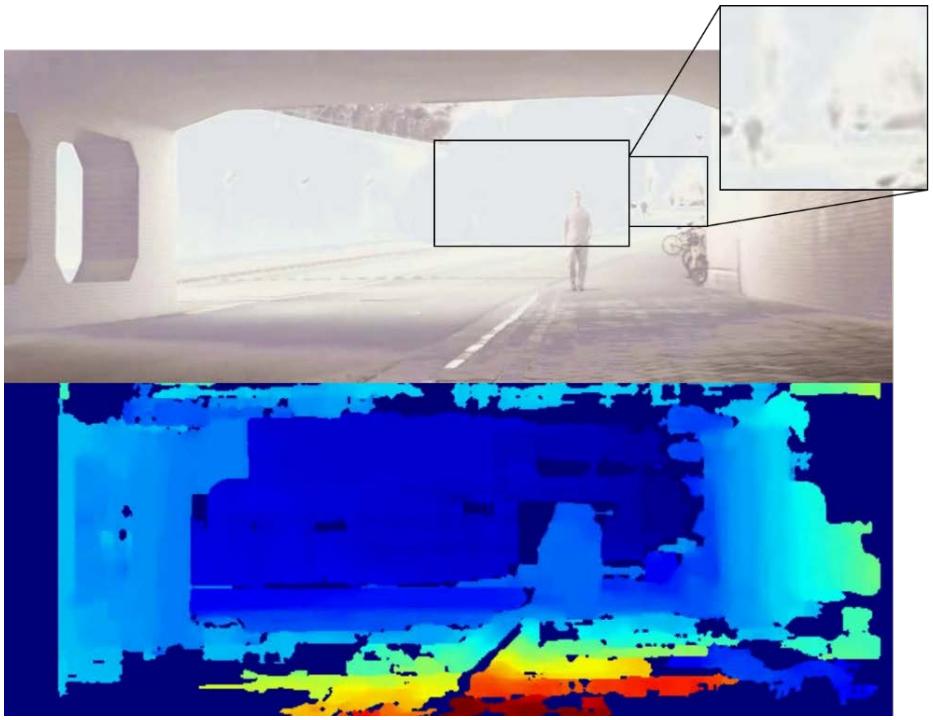

Figure 17 HDR output with difference factor of 16 with bright wall (left rectangle) and zoomed in background

7.3. Significance for the Breaking Distance

An autonomous car's breaking distance is calculated by equating its kinetic energy and necessary friction work to stop the car [17]:

$$D_{break} = \frac{v^2}{2g\mu} \tag{5}$$

Assuming no perception time and dry conditions, the friction coefficient between asphalt and the tire is 0.72 [17]. Therefore, when a vehicle that relies on LDR mapping needs to be able to stop within 14 m, its maximum velocity might not exceed 50.6 km/h. With a visual range of 28 m from HDR imaging, the vehicle can reach almost 40% higher speed or weaken the breaking intensity and improving the driving experience.

8. Comparison to Lidar Sensors

Lidar technology is commonly used to support the realization of autonomous driving. Lidars emit strong and focused pulses of laser beams. Objects that are within a certain range reflect those beams and are being detected again by the lidar sensor. It then calculates the object's distance from the travel time between emission and absorption. For this thesis, the same test is conducted with a Velodyne Puck (VLP-16) with 16 channels, facing the tunnel exit (compare 7.2.2). Figure 18 depicts the lidar result with the person standing 14 m away from the sensor. In contrast to Figure 17, the lighting conditions don't have any impact. It can be well observed that three lasers encounter with the person. That is because the VLP-16 emits its 16 lasers within 30 degrees, which leads to gaps of two degrees between two beams [14]. At 14 m these gaps evaluate to about 50 cm. Hence, the Velodyne Puck might completely miss a car of 1.4 m height in 40 m distance. Here the resolution of a camera (1200 pixels) is much higher.

As a result, both approaches provide reliable data for shorter distances (< 30 m) only.

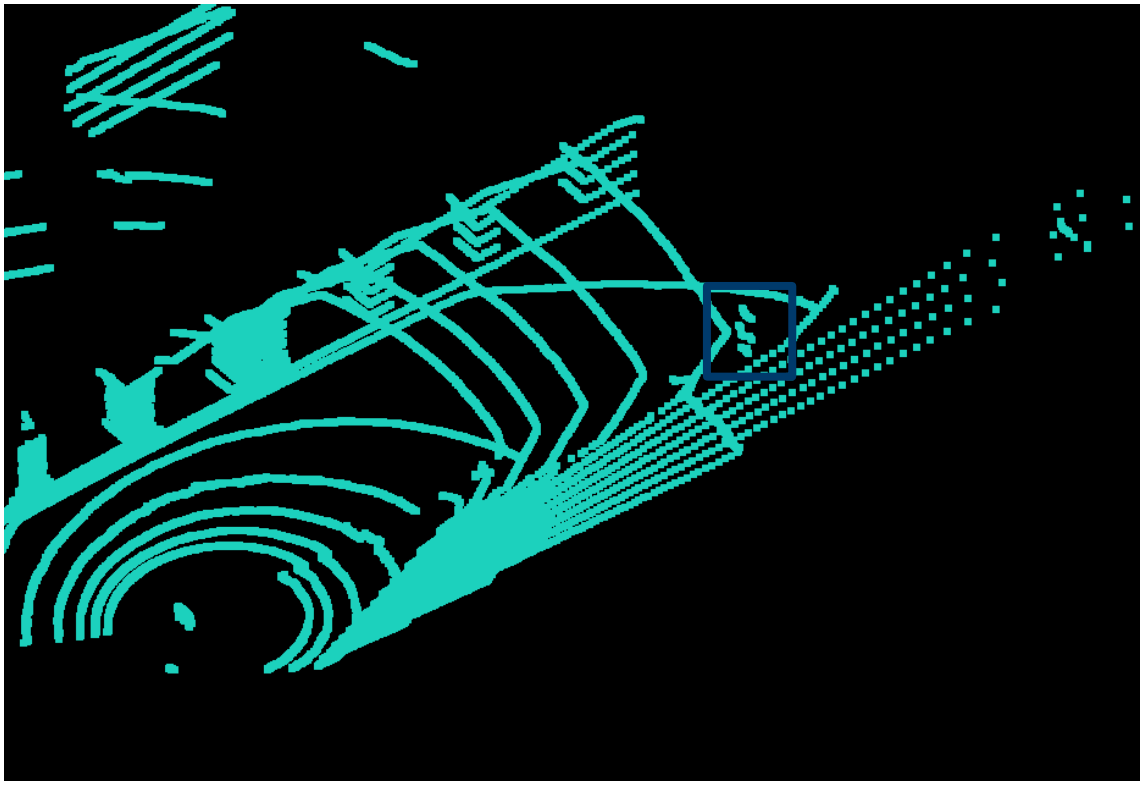

Figure 18 Lidar result from test series with person in blue square

9. Challenges of HDR Imaging

9.1. Ghosting

Ghosting describes the artifact that is created when combining multiple images which contain a moving object. That object is located at a different position in each image and is displayed translucently in the combined image at all positions [1, p. 185]. Following measures can avoid ghosting.

- Alignment of all images
- Reducing shutter speed and the time in between shots
- Avoiding fast movements of camera and environment

For an automotive purpose, fast moving objects cannot be avoided. Hence, alignment and short image capturing times are essential to limit the influence of ghosting artifacts. Figure 19 illustrates the ghosting effect in a HDR image and its impact on the disparity map. As a result, the object shown in the disparity map appears wider as all positions of the object in each LDR image is displayed.

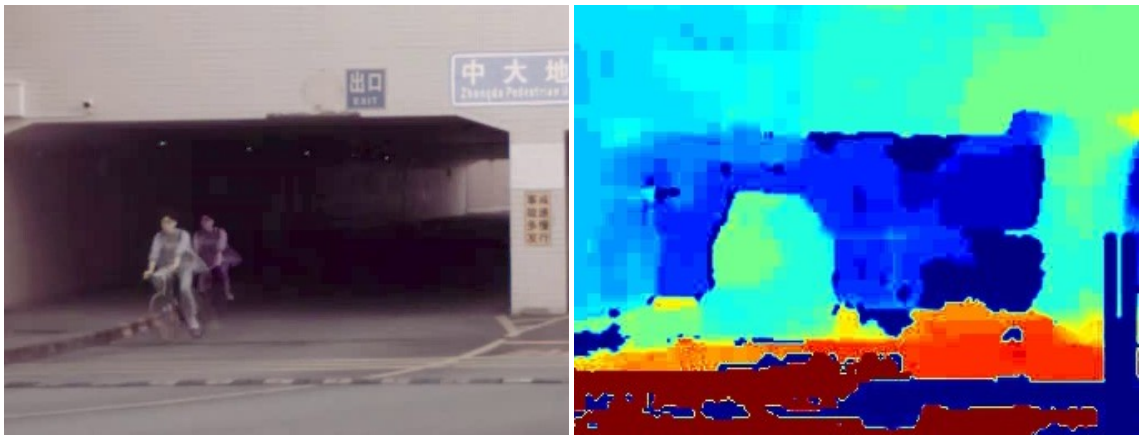

Figure 19 Ghosting effect (left) and impact on disparity map (right)

9.2. Decrease in Frame Rate

In a fast-moving environment, movements of objects must be detected precisely. The capability of capturing many frames per second (fps) assure that fast objects are easy to track. When HDR imaging requires multiple images to form only one, the resulting fps reduce depending on the number of images that are combined:

$$fps_{HDR} = \frac{fps_{original}}{N_{LDR}} \quad (6)$$

9.3. Real Time Application

The HDR and disparity map calculation are performed after recording the video in this thesis. However, in automotive applications real-time capability is critical and there are solutions for the HDR and disparity map creation.

- The commercially available stereo camera from “Stereolabs” offer the creation of real-time disparity maps for $449 at hand-sized dimensions [16].
- The “EyeTap welding helmet” tha enables HDR imaging for “extreme dynamic range scenes” [15]. Therefore, Mann et al. developed a HDR camera that processes 30 fps_{HDR} from four LDR images ($N_{LDR} = 4$) in real-time.

Combining the HDR and disparity map creation seems possible, will occupy high computing capacity though.

9.4. Dark Environment

Most challenging is the disparity map creation in dark environments. Unlike Lidars, cameras are passive “remote sensing” systems, that “detect radiation that is generated by an external source of energy” [14, p. 3]. Hence, when there is weak light provided by the external source, cameras have difficulties picturing the environment with required detail. HDR computation makes it even more difficult (see Figure 7). There are measures that improve the light limitations for cameras:

- Lenses with low aperture
- Larger and more light sensitive sensors
- Limit use of HDR imaging

The challenge of dark surroundings can still be overcome for automotive purposes by investigating in appropriate external lighting.

10. Future Work

Object Tracking

In a subjective assessment, the quality of disparity maps from HDR images are clear superior under difficult lighting conditions. For autonomous applications however, performing tests on object tracking give a more detailed indication in which situations HDR imaging is feasible.

Real-Time Implementation

Another important aspect for autonomous driving is the real-time computation. For that reason, the code implementation directly onto the hardware is advisable. In [15, Sec. 4] hardware requirements for real-time HDR videos are described. Especially additional memory on the board is necessary to store LDR frames.

HDR Imaging for Color Detection / Object Classification

Competing with Lidar sensors at distance measurement and 3D-mapping is difficult for cameras, particularly when dark situations play an important role as well. As this is the case for autonomous driving, it may be interesting to assess the HDR performance at tasks that cameras are best capable of, e.g. color detection, object classification. In Figure 20 is illustrated that the HDR image can distinguish the sun from the sky, whereas the LDR image depicts the sun and most of the sky only as a homogenous white area.

Figure 20 Comparison of LDR (left) and HDR (right) image in terms of colors and edges

Sensor Fusion

Since the correct representation of the environment is of upmost importance for autonomous vehicles, the lidar's advantage of being independent of lighting conditions and its accuracy and the camera's advantage of being cheap and having high resolution might further increase the quality when fusing both sensors types.

HDR Imaging with More than Two LDR Images

Situations in autonomous driving, like exiting a tunnel, challenge HDR imaging from two LDR images (illustrated in 4.2.1). Those results can be improved by using more LDR images for the HDR creation, such that the overall dynamic range can be extended without exceeding the maximum difference factor. Therefore, however, more computing power is necessary to achieve the same processing time.

Variable HDR Settings

As illustrated in Figure 13, HDR imaging does not improve the disparity mapping at good light conditions. Hence, an algorithm that adjusts the HDR settings to the present light condition might be helpful to always receive the most meaningful disparity map. This could be achieved by analyzing the distribution of the pixel intensities from the histogram.

Measure Dynamic Range with a Spot Meter

The dynamic range of a scene can be measured with a spot meter by pointing it to the darkest and brightest regions. The difference will give an objective measure to compare different situations and evaluate the necessity and requirements for HDR imaging.

HDR Cameras

A solution to the required processing power to computer HDR images are HDR cameras that produce a higher dynamic range from the beginning. At the moment, those cameras are used primarily in professional video productions and prices don't make it feasible for automotive applications. Nevertheless, Reinhard expects an increasing interest and demand for HDR cameras in [1, pp. 203/204], such that they might be an option for autonomous vehicles in some years.

11. Conclusion

This thesis demonstrated the improvement of disparity maps under difficult light conditions using HDR imaging. In everyday situations, vehicles often face difficult light situations, mostly caused by the sun directly or through reflections, that conventional cameras' dynamic range exceed by far. Consequently, wide areas around the light source may not be visible which makes safe autonomous driving upon that data impossible. In dark environments with backlight, like a tunnel, conventional images fail to produce any meaningful mapping beyond a close distance, whereas HDR imaging is capable of at least doubling the visible range. Nevertheless, even HDR images are inferior to Lidar sensors. Especially in situations at night, technologies that are dependent of external light sources, like cameras, cannot match those that emit their own light, like lidars do. Hence, best practice will be to fuse the data from multiple sensors.

The more exact illustrations of the environment with HDR imaging under difficult light conditions can improve object detection and simplify object classification. Finally, creating more dynamic range in images to map the real world more realistically will be a goal for future technologies which also improves computer vision for autonomous vehicles.

Bibliography

[1] E. Reinhard, *High dynamic range imaging*, 2nd ed. Burlington, MA, USA: Elsevier, 2010.

[2] F. Banterle, *Advanced high dynamic range imaging*. Boca Raton, FL, USA: CRC Press, 2011.

[3] D. Hernandez-Juarez, A. Chac´on, A. Espinosa, D. V´azquez, J. C. Moure, and A. M. L´opez, "Embedded real-time stereo estimation via Semi-Global Matching on the GPU," *ICCS Procedia Comput. Sci.*, vol. 80, pp. 143-153, Jun. 2016.

[4] S. Mallick, "Install OpenCV 3 on Windows," Big Vision LCC, San Diego, CA, USA, May 2017. Available: https://www.learnopencv.com/install-opencv3-on-windows/**.** Accessed on: Apr. 05, 2018.

[5] G. Ward, "Fast, Robust Image Registration for Compositing High Dynamic Range Photographs from Handheld Exposures," *Journal of Graphics Tools*, vol. 8, pp. 17-30, 2003.

[6] P. E. Debevec and J. Malik, "Recovering High Dynamic Range Radiance Maps from Photographs," in *Proc. of SIGGRAPH 97, Annu. Conf. Series*, Los Angeles, 1997, pp. 369-378.

[7] E. Reinhard and K. Devlin, "Dynamic range reduction inspired by photoreceptor physiology," *IEEE Trans. Vis. Comput. Graph.*, vol. 11, no. 1, pp. 13-24, Jan. 2005.

[8] H. Hirschmüller, "Stereo Processing by Semiglobal Matching and Mutual Information," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol.30, no. 2, Feb. 2008, pp. 328-341.

[9] N.U., "Disparity map post-filtering," OpenCV, May 2015. Available: https://docs.opencv.org/3.1.0/d3/d14/tutorial_ximgproc_disparity_filtering.html, https://github.com/opencv/opencv_contrib/blob/master/modules/ximgproc/samples/disparity_filtering.cpp. Accessed on: Apr. 08, 2018.

[10] MATLAB. MathWorks Inc., Natick, MA, USA, "Stereo Camera Calibrator App," Available: https://de.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html?requestedDomain=true, Accessed on: Apr. 08, 2018.

[11] *EV76C570 2 Mpixels B&W and Color CMOS Image sensor*, Teledyne e2v, Chelmsford, UK, Mar. 29, 2013.

[12] N.U., "Camera Calibration and 3D Reconstruction", OpenCV, Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html. Accessed on: Apr. 16, 2018.

[13] National Oceanic and Atmospheric Administration (NOAA) Coastal Services Center, "Lidar 101: An Introduction to Lidar Technology, Data, and Applications." Revised. Charleston, SC, USA, 2012.

[14] *User's Manual and Programming Guide VLP-16*, Velodyne LiDAR, Morgan Hill, CA, USA, 2016.

[15] S. Mann, R. C. H. Lo, K. Ovtcharov, S. Gu, D. Dai, C. Ngan, T. Ai, „Realtime HDR (High Dynamic Range) Video for Eyetap Wearable Computers, FPGA-Based Seeing Aids, and Glasseyes (Eyetaps)," in IEEE CCECE, Montral, Canada, 2012, pp. 1-6.

[16] Stereolabs Inc., San Francisco, CA, USA, "The camera that senses space and motion", Available: https://www.stereolabs.com/zed/. Accessed on: Apr. 17, 2018.

[17] Engineering ToolBox, “Friction and Friction Coefficients”, 2004. Available: https://www.engineeringtoolbox.com/friction-coefficients-d_778.html Accessed on: Apr. 18, 2018.
[18] R. Szeliski, *Computer Vision: Algorithms and Applications*. London, UK, Springer, 2011.

List of Figures

A. Exposure Time Adjustment Code

```
1.  const int PXSUM_TARGET = 2 * 1600 * 1200 / 10 * 1230;
2.  const int PXSUM_UPPER_LIMIT = 2 * 1600 * 1200 * 124;
3.  const int PXSUM_LOWER_LIMIT = 2 * 1600 * 1200 * 122;
4.  const int PXSUM_CAM_DIFFERENCE_THRESHOLD = 1600 * 1200 * 10;
5.  const alt_u16 EXPOSURE_MAX = 0x4e2; //1250 ~1/24s
6.  const alt_u16 EXPOSURE_MIN_INT = 0x0000; //fractional part: integer = 0x0000
7.  const alt_u16 EXPOSURE_MIN_FRAC = 0x0020;
8.  const alt_u16 FRACTIONAL_MAX = 0x0088; //~0.99
9.  const int EXP_STEP_FACTOR = 22;
10. const int DGAIN_STEP_FACTOR = 24;
11. const alt_u16 DGAIN_MAX = 0x3f;
12. const alt_u16 DGAIN_STEP = 1;
13. const alt_u16 AGAIN_MAX = 4;
14. const alt_u16 AGAIN_MIN = 2;
15. const int HDR_DIFF_FACTOR = 16; //exposure time difference between two hdr ima
    ges
16. const int step_value = 200;
17. int combined_sum1 = 1600 * 1200 * 120; // /2
18. int combined_sum2 = 1600 * 1200 * 120; // /2
19. int combined_sum_mean;
20. int exposure_mean = 0;
21. int maxExp_mean = 0;
22. int minExp_mean = 0;
23. int dgain_mean = 0;
24. int again_mean = 0;
25. int brightness_mean = 0;
26. int sec = 0;
27.
28. //Algorithm to auto-adjust the exposure
29. void correct_exposure_time() {
30.     const alt_u32 cam0_sum = IORD(EV76C570_AVALON_SLAVE_BASE, 1);
31.     const alt_u32 cam1_sum = IORD(EV76C570_AVALON_SLAVE_BASE, 2);
32.     const alt_u32 combined_sum = cam0_sum + cam1_sum;
33.     const alt_u32 cam0_thresh = IORD(EV76C570_AVALON_SLAVE_BASE, 0x10);
34.     const alt_u32 cam1_thresh = IORD(EV76C570_AVALON_SLAVE_BASE, 0x14);
35.     const alt_u32 combined_thresh = cam0_thresh + cam1_thresh;
36.     if ((frame_count & 0x01) == 0) {
37.         combined_sum1 = combined_sum;
38.     } else {
39.         combined_sum2 = combined_sum;
40.     }
41.     combined_sum_mean = (combined_sum1 + combined_sum2) / 2;
42.     exposure_time_mean = (exposure_time_int + exposure_time_int2) / 2;
43.     alt_32 new_exposure_time_int = exposure_time_int;
44.     alt_32 new_exposure_time_frac = exposure_time_frac;
45.     alt_32 new_dgain = common_dgain;
46.     alt_32 new_again = common_again;
47.
48.     if (combined_sum_mean > PXSUM_UPPER_LIMIT) //too bright
49.     {
50.         const int diff = combined_sum_mean - PXSUM_UPPER_LIMIT;
51.         const int expstep = (diff >> EXP_STEP_FACTOR);
52.         const int expstep_valued = expstep * (exposure_time_int * 1000 + expos
    ure_time_frac * 999 / 136);
53.         int expstep_valued_int = expstep_valued / 1000 / step_value;
54.         const int expstep_valued_frac = (expstep_valued / step_value - expstep
    _valued_int * 1000) * 136 / 999;
55.         const int gainstep = (diff >> DGAIN_STEP_FACTOR);
56.
57. //first: reduce digital and analog gain to default minimum (dgain=0, again=2)
58.         if (common_dgain > 0 || common_again > 2) {
59.             new_dgain = (common_dgain - gainstep > 0) ? common_dgain - gainste
    p : 0;
```

```
60.             if (common_dgain < 10 && common_again > 2) {
61.                 new_again = new_again - 1;
62.                 new_dgain = new_dgain + 26;
63.             }
64.         } else if (exposure_time_int == EXPOSURE_MIN_INT) {
65.             if (exposure_time_frac > EXPOSURE_MIN_FRAC) {
66.                 new_exposure_time_frac = exposure_time_frac - expstep_valued_f
    rac;
67.                 if (new_exposure_time_frac < EXPOSURE_MIN_FRAC) {
68.                     new_exposure_time_frac = EXPOSURE_MIN_FRAC;
69.                 }
70.             }
71.         } else {
72.             if (expstep_valued_frac > exposure_time_frac) {
73.                 new_exposure_time_frac = FRACTIONAL_MAX + 1 - (expstep_valued_
    frac - exposure_time_frac);
74.                 expstep_valued_int = +1;
75.             } else {
76.                 new_exposure_time_frac = exposure_time_frac - expstep_valued_f
    rac;
77.             }
78.             new_exposure_time_int = exposure_time_int - expstep_valued_int;
79.         }
80.
81.     } else if (combined_sum_mean < PXSUM_LOWER_LIMIT) //not bright enough
82.     {
83.         const int diff = PXSUM_LOWER_LIMIT - combined_sum_mean;
84.         const int expstep = (diff >> EXP_STEP_FACTOR);
85.         const int expstep_valued = expstep * (exposure_time_int * 1000 + expos
    ure_time_frac * 999 / 136);
86.         int expstep_valued_int = expstep_valued / 1000 / step_value;
87.         const int expstep_valued_frac = (expstep_valued / step_value - expstep
    _valued_int * 1000) * 136 / 999;
88.         const int gainstep = (diff >> DGAIN_STEP_FACTOR);
89. //first increase exposure if not at maximum, then analog and digital gain
90.         if (exposure_time_int * HDR_DIFF_FACTOR < EXPOSURE_MAX) {
91.             if ((exposure_time_frac + expstep_valued_frac) > FRACTIONAL_MAX) {

92.                 new_exposure_time_frac = (exposure_time_frac + expstep_valued_
    frac) - (FRACTIONAL_MAX + 1);
93.                 expstep_valued_int = +1;
94.             } else {
95.                 new_exposure_time_frac = exposure_time_frac + expstep_valued_f
    rac;
96.             }
97.             new_exposure_time_int = exposure_time_int + expstep_valued_int;
98.             if (new_exposure_time_int * HDR_DIFF_FACTOR > EXPOSURE_MAX) {
99.                 new_exposure_time_int = EXPOSURE_MAX / HDR_DIFF_FACTOR;
100.                if (common_dgain < DGAIN_MAX) {
101.                    new_dgain = (common_dgain + gainstep < DGAIN_MAX) ? com
    mon_dgain + gainstep : DGAIN_MAX;
102.                    if (new_dgain > 26 && common_again < AGAIN_MAX) {
103.                        new_again = new_again + 1;
104.                        new_dgain = new_dgain - 25;
105.                    }
106.                }
107.            }
108.        } else {
109.            if (common_dgain < DGAIN_MAX) {
110.                new_dgain = (common_dgain + gainstep < DGAIN_MAX) ? common_
    dgain + gainstep : DGAIN_MAX;
111.                if (new_dgain > 26 && common_again < AGAIN_MAX) {
112.                    new_again = new_again + 1;
113.                    new_dgain = new_dgain - 25;
114.                }
115.            }
```

```
            }
        }
        exposure_time_int = new_exposure_time_int;
        exposure_time_frac = new_exposure_time_frac;
        const int exposure_time_bright_all = (exposure_time_int * 1000 + exposu
    re_time_frac * 999 / 136) * HDR_DIFF_FACTOR;
        exposure_time_int2 = exposure_time_bright_all / 1000;
        exposure_time_frac2 = (exposure_time_bright_all - (exposure_time_int2 *
     1000)) * 136 / 999;
        if (exposure_time_int == EXPOSURE_MIN_INT) {
            if (exposure_time_frac < EXPOSURE_MIN_FRAC) {
                exposure_time_frac = EXPOSURE_MIN_FRAC;
                exposure_time_frac2 = EXPOSURE_MIN_FRAC * HDR_DIFF_FACTOR;
                if (HDR_DIFF_FACTOR > 3) //here frac rises higher than 0x88
                {
                    int i = exposure_time_frac2 * 999 / 136;
                    exposure_time_int2 = i / 1000;
                    exposure_time_frac2 = (i - exposure_time_int2) * 136 / 999;

                }
            }
        }

        common_again = new_again;
        common_dgain = new_dgain;
        slave_select(CAMS_BOTH);
        spi_write(0x0e, (exposure_time_int & 0xff00) >> 8, exposure_time_int &
    0xff);
        spi_write(0x0f, (0x0000 & 0xff00) >> 8, exposure_time_frac & 0xff);
        spi_write(0x1b, (exposure_time_int2 & 0xff00) >> 8, exposure_time_int2
    & 0xff);
        spi_write(0x1c, (0x0000 & 0xff00) >> 8, exposure_time_frac2 & 0xff);
        spi_write(0x11, common_again, common_dgain);
        spi_write(0x1e, common_again, common_dgain);

      // prints average values once per second (easier to monitor)
        if ((frame_count % 22) == 0) {
            sec++;
            printf("Seconds: %d\n", sec);
            printf("Mean Exposure MIN: %d\n", minExp_mean / 22);
            printf("Mean Exposure MAX: %d\n", maxExp_mean / 22);
            printf("Mean Brightness: %d\n", brightness_mean / 22);
            printf("Mean digital gain: %d\n", dgain_mean / 22);
            printf("Mean analog gain: %d\n\n", again_mean / 22);
            minExp_mean = exposure_time_int * 1000 + exposure_time_frac * 999 /
     136;
            maxExp_mean = exposure_time_int2 * 1000 + exposure_time_frac2 * 999
     / 136;
            brightness_mean = combined_sum_mean / 192000;
            dgain_mean = common_dgain;
            again_mean = common_again * 10;
        } else {
            minExp_mean = minExp_mean + (exposure_time_int * 1000 + exposure_ti
    me_frac * 999 / 136);
            maxExp_mean = maxExp_mean + (exposure_time_int2 * 1000 + exposure_t
    ime_frac2 * 999 / 136);
            brightness_mean = brightness_mean + combined_sum_mean / 192000;
            dgain_mean = dgain_mean + common_dgain;
            again_mean = again_mean + common_again * 10;
        }
    }
```

B. Disparity Map Creation

```
1.  void rawToDisp(Mat Input, Mat & DispMap, Mat & DispCol) {
2.      Mat frame; //, disp, dispCol
3.      Mat K1, K2, D1, D2, R1, R2, P1, P2; //, F, E;
4.      cv::FileStorage fs2("out_file_19_01", cv::FileStorage::READ);
5.      fs2["K1"] >> K1;
6.      fs2["D1"] >> D1;
7.      fs2["K2"] >> K2;
8.      fs2["D2"] >> D2;
9.      fs2["R1"] >> R1;
10.     fs2["R2"] >> R2;
11.     fs2["P1"] >> P1;
12.     fs2["P2"] >> P2;
13.     fs2.release();
14.     printf("splitting...\n");
15.     Size size = Input.size();
16.     int w = size.width;
17.     int h = size.height;
18.     Rect ROI_l(0, 0, w / 2, h);
19.     Rect ROI_r(w / 2, 0, w / 2, h);
20.     Mat left = Input(ROI_l);
21.     Mat right = Input(ROI_r);
22.     Mat map1x, map1y, map2x, map2y;
23.     printf("start undistorting frames...\n");
24.     initUndistortRectifyMap(K1, D1, R1, P1, Size(w / 2, h), CV_16SC2, map1x,
    map1y);
25.     initUndistortRectifyMap(K2, D2, R2, P2, Size(w / 2, h), CV_16SC2, map2x,
    map2y);
26.     Mat left_rec, right_rec;
27.     printf("start remapping...\n");
28.     remap(left, left_rec, map1x, map1y, INTER_LINEAR);
29.     printf("right..\n");
30.     remap(right, right_rec, map2x, map2y, INTER_LINEAR);
31.     Rect ROI_new(1, 21, 1598, 1179);
32.     Mat left_new = left_rec(ROI_new);
33.     Mat right_new = right_rec(ROI_new);
34.     Mat temp1, temp2;
35.     printf("resizing...\n");
36.     left_new.convertTo(left_new, CV_8UC1);
37.     right_new.convertTo(right_new, CV_8UC1);
38.     resize(left_new, temp1, Size(), 0.4, 0.4);
39.     resize(right_new, temp2, Size(), 0.4, 0.4);
40.     left_new = temp1;
41.     right_new = temp2;
42.     printf("start desparity Map creation:\n");
43.     Ptr < StereoSGBM > sgbm1 = StereoSGBM::create(0, 16, 3);
44.     sgbm1 - > setNumDisparities(48);
45.     sgbm1 - > setMinDisparity(-9);
46.     sgbm1 - > setPreFilterCap(31);
47.     sgbm1 - > setBlockSize(13);
48.     sgbm1 - > setUniquenessRatio(9);
49.     sgbm1 - > setSpeckleWindowSize(1000);
50.     sgbm1 - > setSpeckleRange(1);
51.     sgbm1 - > setDisp12MaxDiff(-1);
52.     sgbm1 - > setP1(8 * 13 * 13); // 13 = blocksize!
53.     sgbm1 - > setP2(32 * 13 * 13);
54.     sgbm1 - > setMode(StereoSGBM::MODE_SGBM_3WAY);
55.     Mat disp, dispCol, disp8, disp82;
56.     sgbm1 - > compute(right_new, left_new, disp);
57.     double min, max;
58.     minMaxIdx(disp, & min, & max);
59.     double diff = max - min;
60.     disp = (disp + 160) / (diff / 255);
61.     disp.convertTo(disp8, CV_8U);
```

```
62.      applyColorMap(disp8, dispCol, COLORMAP_JET);
63.      resize(dispCol, DispCol, Size(960, 720));
64.      resize(disp8, DispMap, Size(960, 720));
65.  }
66.  void hdrR(vector < Mat > images, int diff, Mat & out) {
67.      vector < float > times;
68.      static
69.      const float timesArray[] = {
70.          1.0, diff
71.      };
72.      times.assign(timesArray, timesArray + 2);
73.      Mat responseDebevec, hdrDebevec;
74.      Ptr < CalibrateDebevec > calibrateDebevec = createCalibrateDebevec();
75.      calibrateDebevec - > process(images, responseDebevec, times);
76.      Ptr < MergeDebevec > mergeDebevec = createMergeDebevec();
77.      mergeDebevec - > process(images, hdrDebevec, times);
78.      Ptr < TonemapReinhard > tonemapReinhard = createTonemapReinhard(1.5, 2.0
     , 0, 0);
79.      tonemapReinhard - > process(hdrDebevec, out);
80.      out = out * 255;
81.      out.convertTo(out, CV_8U);
82.  }

83.  void mergeNhdrNdisps(cv::VideoCapture vid, string dir_out, int diff, int sta
     rtFrame, int endFrame, bool makeVid, int fps, int fourcc) {
84.      cout << "started MERGE and HDR..." << endl;
85.      Mat merge, mDispCol, mDispMap;
86.      Mat hdr, hDispCol, hDispMap;
87.      Mat frame, left1, left2;
88.      Mat result_merge, result_hdr;
89.      int counter = 0;
90.      int count = 0, countFractions = 0, startingFrame = -1;
91.      vector < Mat > frames;
92.      int bright_dark_diff = 10;
93.      bool ready = false;
94.      bool save = false;
95.      Scalar a, b, c;
96.      int co = 0;
97.      Mat img1, img2;
98.      if (makeVid) {
99.          string s_out0 = dir_out + "_" + to_string(startFrame / 22) + "_" + t
     o_string(endFrame / 22) + "mergeD.avi";
100.         string s_out1 = dir_out + "_" + to_string(startFrame / 22) + "_" + t
     o_string(endFrame / 22) + "merge.avi";
101.         string s_out2 = dir_out + "_" + to_string(startFrame / 22) + "_" + t
     o_string(endFrame / 22) + "merge_disp.avi";
102.         VideoWriter writerMergeD = VideoWriter(s_out0, fourcc, fps / 2, Size
     (960, 720), 0);
103.         VideoWriter writerMerge = VideoWriter(s_out1, fourcc, fps / 2, Size(
     1600, 1200), 1);
104.         VideoWriter writerMergeDisp = VideoWriter(s_out2, fourcc, fps / 2, S
     ize(960, 720), 1);
105.         string s_out5 = dir_out + "_" + to_string(startFrame / 22) + "_" + t
     o_string(endFrame / 22) + "hdrD.avi";
106.         string s_out3 = dir_out + "_" + to_string(startFrame / 22) + "_" + t
     o_string(endFrame / 22) + "hdr.avi";
107.         string s_out4 = dir_out + "_" + to_string(startFrame / 22) + "_" + t
     o_string(endFrame / 22) + "hdr_disp.avi";
108.         VideoWriter writerHdrD = VideoWriter(s_out5, fourcc, fps / 2, Size(9
     60, 720), 0);
109.         VideoWriter writerHdr = VideoWriter(s_out3, fourcc, fps / 2, Size(16
     00, 1200), 1);
110.         VideoWriter writerHdrDisp = VideoWriter(s_out4, fourcc, fps / 2, Siz
     e(960, 720), 1);
111.         cout << "Start & End: " << startFrame / 22 << " & " << endFrame / 22
      << endl;
```

```
        while (vid.isOpened()) {
            vid >> frame;
            cout << counter / 22 << ", " << counter << endl;
            if (counter >= startFrame && counter <= endFrame) {
                if (count == startingFrame) {
                    ready = true;
                    std::cout << "starting frame: " << count << endl;
                }
                if (ready) {
                    if (counter == ((endFrame + startFrame) / 2)) {
                        save = true;
                    } else {
                        save = false;
                    }
                    if (countFractions == 0) {
                        a = mean(frame);
                        cout << "dark: " << a[0] << endl;
                        if ((a[0] + bright_dark_diff) < b[0]) {
                            frames.push_back(frame.clone());
                            countFractions++;
                            co++;
                            if (save) {
                                imwrite("dark.jpg", frame);
                            }
                        } else {
                            cout << "found two BRIGHT frames in a row" << en
    dl;
                        }
                    } else if (countFractions == 1) {
                        b = mean(frame);
                        cout << "bright: " << b[0] << endl;
                        if ((a[0] + bright_dark_diff) < b[0]) {
                            frames.push_back(frame);
                            countFractions++;
                            co++;
                            if (save) {
                                imwrite("bright.jpg", frame);
                            }
                        } else {
                            cout << "found two dark frames in a row" << endl
    ;
                        }
                    }
                    if (countFractions == 2) {
                        cout << "processing: " << (endFrame - counter) << en
    dl;
                        mergeM(frames, result_merge);
                        cout << "finish merging" << endl;
                        getLeft(result_merge, left1);
                        writerMerge.write(left1);
                        hdrR(frames, diff, result_hdr);
                        getLeft(result_hdr, left2);
                        writerHdr.write(left2);
                        cout << "finish hdr" << endl;
                        if (save) {
                            getLeft(result_merge, left1);
                            getLeft(result_hdr, left2);
                            imwrite("merge.jpg", left1);
                            imwrite("hdr.jpg", left2);
                        }
                        cout << "dispMap..." << endl;
                        rawToDisp(result_merge, mDispMap, mDispCol);
                        rawToDisp(result_hdr, hDispMap, hDispCol);
                        writerMergeDisp.write(mDispCol);
                        writerMergeD.write(mDispMap);
                        writerHdrDisp.write(hDispCol);
```

```
                        writerHdrD.write(hDispMap);
                        if (save) {
                            imwrite("hdispCol.jpg", hDispCol);
                            imwrite("mDispCol.jpg", mDispCol);
                            imwrite("hDispMap.jpg", hDispMap);
                            imwrite("mDispMap.jpg", mDispMap);
                        }
                        countFractions = 0;
                        frames = {};
                        co++;
                    }
                } else if (count == 0) {
                    a = mean(frame);
                } else if (count == 1) {
                    b = mean(frame);
                    if ((a[0] + bright_dark_diff) < b[0]) {
                        startingFrame = 2;
                    } else {
                        b = a;
                        if ((b[0] + bright_dark_diff) < a[0]) {
                            startingFrame = 3;
                        }
                    }
                } else if (count == 2) {
                    c = mean(frame);
                    if ((c[0] + bright_dark_diff) < b[0]) {
                        a = c;
                        startingFrame = 4;
                    } else {
                        b = c;
                        startingFrame = 3;
                    }
                }
                count++;
            }
            counter++;
            if (counter > endFrame) {
                vid.release();
                writerMerge.release();
                writerHdr.release();
                writerMergeDisp.release();
                writerHdrDisp.release();
                writerMergeD.release();
                writerHdrD.release();
                break;
            }
        }
    } else {
        string s_out0 = dir_out + to_string(startFrame / 22) + "merge_d.jpg"
    ;
        string s_out1 = dir_out + to_string(startFrame / 22) + "merge.jpg";

        string s_out2 = dir_out + to_string(startFrame / 22) + "merge_disp.j
    pg";
        string s_out3 = dir_out + to_string(startFrame / 22) + "hdr.jpg";
        string s_out4 = dir_out + to_string(startFrame / 22) + "hdr_disp.jpg
    ";
        string s_out5 = dir_out + to_string(startFrame / 22) + "hdr_d.jpg";

        while (vid.isOpened()) {
            vid >> frame;
            if (counter % 22 == 0) {
                cout << counter / 22 << endl;
            }
            if (counter >= startFrame) {
                if (count == startingFrame) {
```

```
                        ready = true;
                        std::cout << "starting frame: " << count << endl;
                    }
                    if (ready) {
                        if (countFractions == 0) {
                            a = mean(frame);
                            if ((a[0] + bright_dark_diff) < b[0]) {
                                frames.push_back(frame.clone());
                                countFractions++;
                                imwrite("dark.jpg", frame);
                            } else {
                                cout << "found two BRIGHT frames in a row" << en
    dl;
                            }
                        } else if (countFractions == 1) {
                            b = mean(frame);
                            if ((a[0] + bright_dark_diff) < b[0]) {
                                frames.push_back(frame);
                                countFractions++;
                                imwrite("bright.jpg", frame);
                            } else {
                                cout << "found two dark frames in a row" << endl
    ;
                            }
                        }
                        if (countFractions == 2) {
                            cout << "processing: " << (endFrame / 22 - counter /
     22) << endl;
                            mergeM(frames, result_merge);
                            cout << "finish merging" << endl;
                            getLeft(result_merge, left1);
                            imwrite(s_out1, left1);
                            hdrR(frames, diff, result_hdr);
                            getLeft(result_hdr, left2);
                            imwrite(s_out3, left2);
                            cout << "finish hdr" << endl;
                            cout << "dispMap..." << endl;
                            rawToDisp(result_merge, mDispMap, mDispCol);
                            imwrite(s_out2, mDispCol);
                            imwrite(s_out0, mDispMap);
                            rawToDisp(result_hdr, hDispMap, hDispCol);
                            imwrite(s_out4, hDispCol);
                            imwrite(s_out5, hDispMap);
                            cout << "written images successfully" << endl;
                            break;
                        }
                    } else if (count == 0) {
                        a = mean(frame);
                    } else if (count == 1) {
                        b = mean(frame);
                        if ((a[0] + bright_dark_diff) < b[0]) {
                            startingFrame = 2;
                        } else {
                            b = a;
                            if ((b[0] + bright_dark_diff) < a[0]) {
                                startingFrame = 3;
                            }
                        }
                    } else if (count == 2) {
                        c = mean(frame);
                        if ((c[0] + bright_dark_diff) < b[0]) {
                            a = c;
                            startingFrame = 4;
                        } else {
                            b = c;
                            startingFrame = 3;
```

```
                    }
                }
                count++;
            }
            counter++;
            if (counter > (endFrame + count)) {
                vid.release();
                break;
            }
        }
    }
}
int VidToHdrDisp(string name, bool merging, bool hdrImaging, int vidStart, int vidEnd) {
    string dir_out = "result/" + name;
    int fps = 22;
    int diff = 4; //diff for hdr images
    string s_in = name + ".mp4"; //string dir_out = dir_o + "/";
    Mat frame, dispCol;
    Mat merge, mDispCol;
    Mat hdr, hDispCol;
    VideoCapture vid(s_in);
    int numFrames = (int) vid.get(CAP_PROP_FRAME_COUNT);
    printf("Number of Frames: %d\n", numFrames);
    int fourcc = CV_FOURCC('D', 'I', 'V', '3');
    int startFrame = vidStart * fps;
    int endFrame = vidEnd * fps;
    int mode;
    bool makeVid;
    if (endFrame > numFrames) {
        endFrame = numFrames - 5; // cause of first test, whether start frame is a darker one
    }
    if (startFrame >= 0 && endFrame >= 0 && startFrame <= endFrame) {
        if (startFrame < endFrame) {
            makeVid = true;
        } else {
            makeVid = false;
        }
    } else {
        makeVid = false;
        startFrame = numFrames / 2;
        endFrame = startFrame;
    }
    if (merging && hdrImaging) {
        mergeNhdrNdisps(vid, dir_out, diff, startFrame, endFrame, makeVid, fps, fourcc);
    } else if (hdrImaging) {
        hdrNdisp(vid, dir_out, diff, startFrame, endFrame, makeVid, fps, fourcc);
    } else if (merging) {
        mergeNdisp(vid, dir_out, startFrame, endFrame, makeVid, fps, fourcc);
    } else {
        normToDisp(vid, dir_out, startFrame, endFrame, makeVid, fps, fourcc);
    }
    return 0;
}
int main(int argc, char * * argv) {
    Mat frame, disp, dispCol;
    VidToHdrDisp("videoName", 1, 1, 6, 6);
    cout << "finish" << endl;
    return 0;
}
```

C. Distance Calculation

```
#include < opencv2\ opencv.hpp >
#include < Windows.h >
using namespace cv;
using namespace std;

void drawCircle(Mat in , int x, int y, int fac, Mat & out) {
    Mat in_new = in * fac;
    Mat cM1, cM2, cM3;
    Mat temp1 = in_new.clone();
    Mat temp2 = in_new.clone();
    Mat temp3 = in_new.clone();
    applyColorMap(temp1, cM1, COLORMAP_HSV);
    applyColorMap(temp2, cM2, COLORMAP_JET);
    applyColorMap(temp3, cM3, COLORMAP_COOL);
    circle(cM1, Point(x, y), 2, (0, 255, 0), 2, 8);
    circle(cM1, Point(x, y), 6, (255, 0, 255), 2, 8);
    circle(cM2, Point(x, y), 2, (0, 255, 0), 2, 8);
    circle(cM2, Point(x, y), 6, (255, 0, 255), 2, 8);
    circle(cM3, Point(x, y), 2, (0, 255, 0), 2, 8);
    circle(cM3, Point(x, y), 6, (255, 0, 255), 2, 8);
    imshow("HSV", cM1);
    imshow("JET", cM2);
    out = cM2;
}
void printIntensity(Mat in , int x, int y) {
    Scalar intensity = in .at < uchar > (Point(x, y));
}
void printDistance(Mat in , int x, int y) {
    Point3f p = in .at < Point3f > (y, x);
    cout << "Distance at (" << x << "," << y << "): " << -
2 * (p.z) << " meter" << endl;
}
int main(int argc, char * * argv) {
    Mat dispMap, dM;
    Mat dM1, dM2, dM3;
    Mat cM1, cM2, cM3;
    Mat Q;
    int fac = 2;
    string dir = "";
    string name = "00092.jpg";
    int p1 = 0;
    int p2 = 0; //Scalar intensity;
    cv::FileStorage fs2("out_file_28_01", cv::FileStorage::READ);
    fs2["Q"] >> Q;
    fs2.release();
    string inStr = dir + name;
    dispMap = imread("dist22_disp.jpg");
    dM = imread(inStr, IMREAD_GRAYSCALE);
    dM1 = (dM * fac);
    applyColorMap(dM1, cM1, COLORMAP_HSV);
    applyColorMap(dM1, cM2, COLORMAP_JET);
    applyColorMap(dM1, cM3, COLORMAP_COOL);
    Mat img3D(dM.size(), CV_32FC3);
    reprojectImageTo3D(dM, img3D, Q, true, CV_32F); //imwrite("img3d.png", img
3D);
    namedWindow("HSV", CV_WINDOW_FREERATIO);
    namedWindow("JET", CV_WINDOW_FREERATIO);
    imshow("HSV", cM1);
    imshow("JET", cM2);
    Size size = dispMap.size();
    Size s = dM.size();
    int w = size.width;
    int h = size.height;
```

```
    int w1 = s.width;
    int h1 = s.height;
    cout << size << ", " << s << endl;
    while (1) {
        char c = waitKey(50);
        if (c == 27) {
            destroyAllWindows();
            return 0;
            break;
        }
        if (c == 'j') {
            if (p1 > 5) {
                p1 -= 6;
                drawCircle(dM, p1, p2, fac, cM2);
                printIntensity(dM, p1, p2);
                printDistance(img3D, p1, p2);
            }
        }
        if (c == 'k') {
            if (p2 < h - 5) {
                p2 += 6;
                drawCircle(dM, p1, p2, fac, cM2);
                printIntensity(dM, p1, p2);
                printDistance(img3D, p1, p2);
            }
        }
        if (c == 'l') {
            if (p1 < w - 5) {
                p1 += 6;
                drawCircle(dM, p1, p2, fac, cM2);
                printIntensity(dM, p1, p2);
                printDistance(img3D, p1, p2);
            }
        }
        if (c == 'i') {
            if (p2 > 5) {
                p2 -= 6;
                drawCircle(dM, p1, p2, fac, cM2);
                   printIntensity(dM, p1, p2);
                   printDistance(img3D, p1, p2);
               }
           }
           if (c == '+') {
               fac++;
               cout << "Factor: " << fac << endl;
           } else if (c == '-') {
               if (fac > 1) {
                   fac--;
                   cout << "Factor: " << fac << endl;
               }
           }
           if (c == ' ') {
               imwrite(dir + "fac" + to_string(fac) + "_" + name, cM2);
               cout << "saved" << endl;
           }
       }
       return 0;
   }
```