EBU6307 – Visual Computing – 2024/25

Coursework report and exercises

Name: Zheyun Zhao

QM student number: 221170559

BUPT student number: 2022213670

DISCLAIMER

- The students are only allowed to use the provided Anaconda environment.
- All codes will be tested in the provided Anaconda environment.
- All codes should be executable in the terminal. That is, `python <EXERCISE_NAME>.py` should be able to produce all the results.
- Outputs of the implementation must be generated in the **\results** folder
- Zero mark will be given if your code is not executable in the provided Anaconda environment.
- Zero mark will be given if another student's name and ID are written in your own dataset, and **the submitted coursework will be reported as a plagiarism case**.
- If the submitted file does not follow the structure below, **10 marks will be deducted**.
- Make your folder structure as below:

```
EBU6307_FIRSTNAME_FAMILYNAME_QMSTUDNETNUMBER
├── inputs
├── results
└── codes
```

- In the final coursework submission, submit 1) your report and the single 2) zip file of the above folder (`EBU6307_FIRSTNAME_FAMILYNAME_QMSTUDNETNUMBER`) to the QMplus.
- QMplus submission example:
  - EBU6307_SALMAN_HALEEM_1911XXXXX.pdf
  - EBU6307_SALMAN_HALEEM_1911XXXXX.zip

DO NOT WRITE IN THIS TABLE (TA-only)					
	1	**2**	**3**	**4**	**Total**
(a)	**/10**	**/10**	**/10**	**/10**	
(b)	**/10**	**/10**	**/10**	**/10**	
(c)		**/10**	**/10**	**/10**	
(d)	**-**	**/10**			
		/10			
		/10			
Total	**/20**	**/60**	**/30**	**/30**	**/140**

Exercise 1: Data acquisition, filtering and getting familiar with Python/Scipy/Numpy/OpenCV
Exercise 1 (a): Image resolution (Spatial and Intensity)

Input data

- lena.jpg

Output of the implementation

- ex1a_downsample_spatial_4.jpg
- ex1a_upsample_spatial_4.jpg
  (Spec: 256 x 256 resolution, grayscale)
- ex1a_downsample_quant_4.jpg
- ex1a_downsample_quant_16.jpg

Description

Implement spatial scaling and intensity scaling functions using a grayscale lena.jpg as an input. You need to get the original dimensions of the image.

Spatial Scaling:

i. Define the spatial scaling function without using OpenCV.
ii. Downsample the original image by factor of 4 i.e. if original dimension is 512 x 512, the downsampled dimension should be 128 x 128
iii. Take the downsampled image from ii as input and upsample back to original dimension as i.

Intensity Scaling:

i. Define the intensity scaling function without using OpenCV.
ii. Downsample the original image upto 4 intensity levels
iii. Downsample the original image upto 16 intensity levels

Note. You are not allowed to use OpenCV library except image I/O.

Display the results in the boxes below:

Original image	Spatial scaling (ii)	Spatial scaling (iii)
		
Original image	**Intensity scaling (ii)**	**Intensity scaling (iii)**
		

Discuss the results (in 10 lines):

Spatial scaling reduces the spatial resolution of the image, causing details and edges to become less sharp after downsampling. When the downsampled image is upsampled back to the original size, lost details cannot be recovered, resulting in a blocky and blurry appearance. This demonstrates that aggressive downsampling leads to irreversible information loss.

Intensity scaling decreases the number of grayscale levels in the image. Reducing to 4 intensity levels causes strong posterization, where smooth gradients are replaced by visible bands. With 16 intensity levels, the result appears smoother but still lacks subtle gradations seen in the original.

Overall, spatial scaling mainly causes loss of image structure and sharpness, while intensity scaling introduces quantization artifacts. Both processes highlight the trade-off between data reduction and visual quality in digital image processing.

Exercise 1 (b): Image translation, rotation and shearing

Input data

- Synthetic red outlined rectangle

Output of the implementation

- ex1b_translate_50_30.jpg
- ex1b_rotate_30.jpg
- ex1b_shear_x_0_5.jpg
- ex1b_shear_y_0_5.jpg

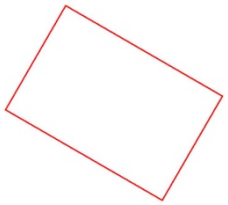
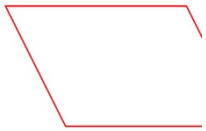
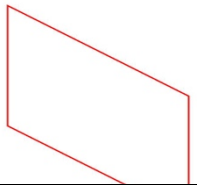
Description

Implement image translation, rotation and shearing. First you need to create a synthetic image i.e. red outlined rectangle using pillow (PIL) package and importing Image and ImageDraw functions which can take width and height of user defined choice. You can then put width=600 pixels and height=400 pixels, Outline width=5, and background colour as white. Red colour rgb code=(255,0,0) and white colour rgb code is (255,255,255).

i. Implement the image translation function and translate by tx=50, ty=30
ii. Implement the image rotation function and rotate by 30 degrees
iii. Implement the image shearing function and shear the image by a) $S_x = 0.5$ and b) $S_y = 0.5$

Note. You are not allowed to use OpenCV library except image I/O

Display the results in the boxes below:

Original image	Image Translation (i)	Image Rotation (ii)
		
	Image Shearing (iii) $S_x = 0.5$	Image Shearing (iii) $S_y = 0.5$
		

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

This experiment demonstrates the effects of translation, rotation, and shearing on a synthetic rectangle. After translation, the rectangle shifts right and downward while retaining its original shape and orientation. Rotation by 30 degrees pivots the rectangle around its center, resulting in a

tilted appearance. Shearing in the x-direction distorts the rectangle horizontally, slanting the vertical edges, while shearing in the y-direction causes a vertical slant, modifying the horizontal edges. Each transformation alters the geometric properties of the shape in a distinct way. Translation preserves proportions, while rotation changes orientation without affecting shape. Shearing, on the other hand, distorts the angles and produces a parallelogram-like figure. These results illustrate how basic geometric transformations can be applied to images, which are fundamental operations in image processing and computer vision tasks.

Exercise 2 (a): Convolution gaussian filtering

Input data

- lena.jpg

Output of the implementation

- ex2a_gf_5_1.jpg
- ex2a_gf_5_10.jpg
- ex2a_gf_21_1.jpg
- ex2a_gf_21_10.jpg

Description

Implement gaussian filtering based on convolution function using a grayscale lena.jpg (you can convert coloured lena into grayscale) as an input. See the results with varying the window size (kernel size), $W = 5, 21$, and the standard deviation, $\sigma_s = 1,10$.

Note. You are not allowed to use OpenCV library except image I/O.

Display the results in the boxes below:

Original image	$W = 5, \sigma_s = 1$	$W = 5, \sigma_s = 10$
		
	$W = 21, \sigma_s = 1$	$W = 21, \sigma_s = 10$
		

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

Gaussian filtering smooths the image by reducing noise and detail, with the effect depending on the kernel size (W) and standard deviation (σs). When W = 5 and σs = 1, the image appears only slightly blurred, retaining most edges and details. Increasing σs to 10 with the same small window produces a more noticeable blur, but the effect is still localized. When the kernel size increases to W = 21, even with σs = 1, the blur covers a larger area, making the image appear softer and less detailed. With both W = 21 and σs = 10, the blurring is most significant, leading to strong loss of fine detail and very smooth transitions. Larger window sizes and higher standard deviations both contribute to

a stronger smoothing effect. Overall, Gaussian filtering is highly effective for noise reduction but at the cost of detail loss, especially with large kernel sizes or high sigma values.

Exercise 2: Low Vision, Image filtering, Image features
Exercise 2 (b): Image Aliasing

Input data

- Squares.jpg

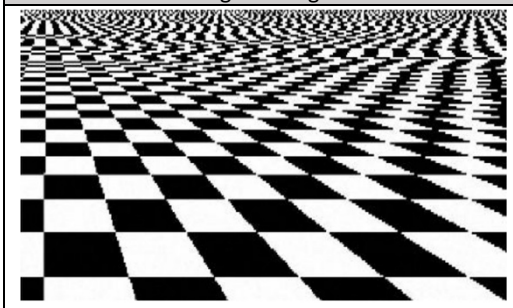
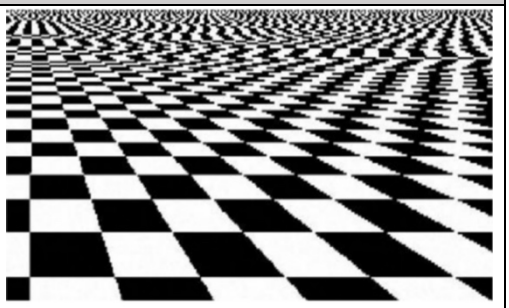
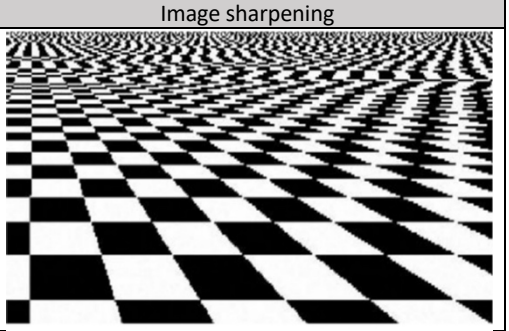
Output of the implementation

- ex2b_Alias.jpg
- ex2b_Sharp.jpg

Description

i. Using a Squares.jpg as an input and implement anti-aliasing filter
ii. Using a Squares.jpg as an input and implement image sharpening filter on anti-aliased image
Note. You are not allowed to use OpenCV library except image I/O.

Display the results in the boxes below:

Original image	Anti-alias
	
	Image sharpening
	

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

The original image shows a checkerboard pattern, which is highly susceptible to aliasing effects such as jagged edges and moiré patterns, especially in the distant and fine grid regions. After applying an anti-aliasing filter, the image appears smoother, with reduced jaggedness and less visible distortion in the high-frequency areas. The anti-aliasing process helps to suppress unwanted artifacts by averaging neighboring pixel values, resulting in a cleaner appearance. When a sharpening filter is applied to the anti-aliased image, the edges of the squares become more pronounced and crisp, restoring some of the lost clarity while maintaining the reduced aliasing. This sequence demonstrates that anti-aliasing effectively minimizes artifacts in high-frequency patterns, and sharpening can be used to selectively enhance edge definition without reintroducing significant aliasing.

Exercise 2 (c): SIFT (Scale Invariant Feature Transform) and feature matching

---

Input data

- sift_input1.jpg
- sift_input2.jpg

Output of the implementation

- ex2c_sift_input1.jpg
- ex2c_sift_input2.jpg
- ex2c_matches_least10.jpg
- ex2c_matches_most10.jpg

Description

Run the SIFT descriptor and nearest neighbour (NN) feature matching using the function provided by OpenCV.

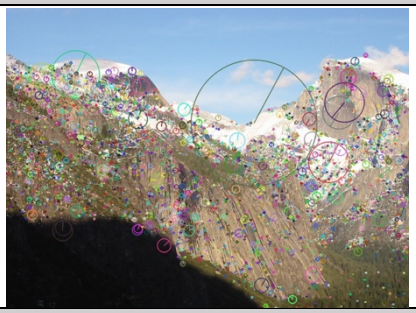
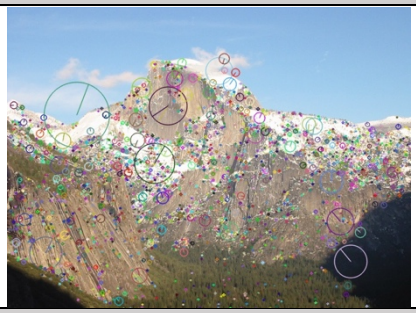
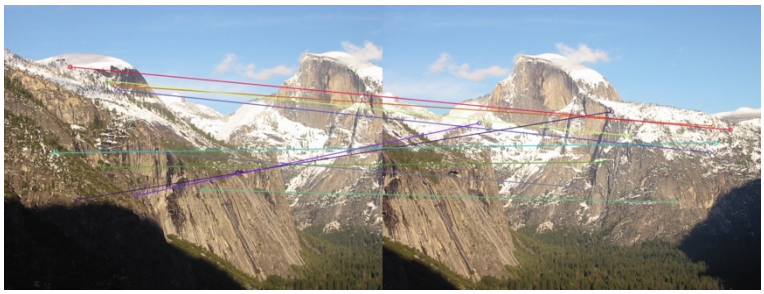
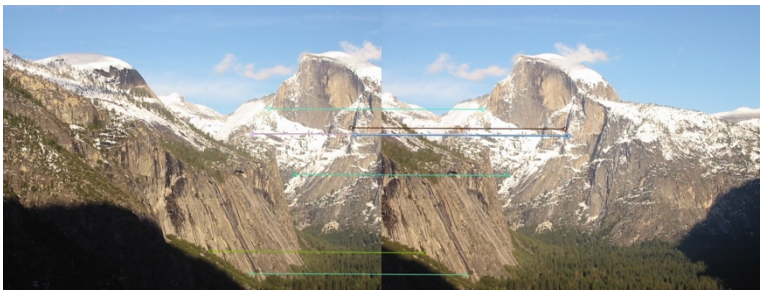
Task 1.

Run SIFT descriptor for two images I_1 (sift_input1.jpg) and I_2 (sift_input2.jpg).
Draw the keypoints on I_1 (ex2c_sift_input1.jpg) and I_2 (ex2d_sift_input2.jpg).

Task 2.

Perform the feature matching using NN for $I_1 \rightarrow I_2$.
Show the worst 10 matches (ex2c_matches_least10.jpg) and best (nearest) 10 matches (ex2c_matches_most10.jpg) for $I_1 \rightarrow I_2$.

Display the results:

ex2c_sift_input1.jpg	ex2c_sift_input2.jpg
	
ex2c_matches_least10.jpg	ex2c_matches_most10.jpg
	

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

The SIFT algorithm detects and matches keypoints between two images of the same scene. The keypoint visualizations show that SIFT identifies many distinct features, especially along edges and textured regions. The “worst 10 matches” image reveals some mismatches, with lines connecting unrelated points in the two images. In contrast, the “best 10 matches” image shows accurate correspondences, with lines connecting similar features such as mountain edges. This demonstrates that SIFT is effective at robustly matching keypoints, but not all matches are reliable. Filtering out poor matches is important for tasks like image stitching or object recognition. Overall, SIFT combined with nearest neighbor matching offers strong performance in feature detection and matching.

Exercise 2 (d): Canny Edge Detection

Input data

- building.jpg

Output of the implementation

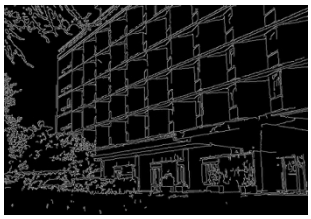
- ex2d_canny1.jpg
- ex2d_canny2.jpg
- ex2d_canny3.jpg
- ex2d_canny4.jpg
- ex2d_canny5.jpg
- ex2d_ canny6.jpg

Description

Convert building.jpg into grayscale and perform

i. Perform Gaussian Smoothing using sigma=1.4
ii. Perform Sobel operator and calculate gradient magnitude(you are not allowed to use OpenCV library or scipy function)
iii. Perform the non-maxima suppression (you are not allowed to use OpenCV library or scipy function)
iv. Apply double threshold to detect strong, weak and non edges (use the range 0.5<weak edge<1 and strong edge>1) (you are not allowed to use OpenCV library or scipy function)
v. Perform edge tracking by converting weak edge to strong edge if connected to strong edge. (you are not allowed to use OpenCV library or scipy function)
vi. Determine edge map using OpenCV canny edge detection function and compare your results

Display the results in the boxes below:

Original image	i	ii	iii
			
	iv	v	vi
			

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

The Canny edge detection process effectively highlights the structural features of the building. After Gaussian smoothing, noise is reduced while preserving important details. Applying the Sobel operator and non-maxima suppression enhances the clarity and precision of edges, making the building's outlines more pronounced. Double thresholding separates strong, weak, and non-edges, helping to distinguish prominent architectural lines from less significant details. Edge tracking further connects weak edges to strong ones, resulting in continuous and complete boundaries around windows and other features. The final Canny edge map clearly outlines the major geometric shapes and repetitive patterns of the building. Overall, the process demonstrates how Canny edge detection isolates meaningful structural information while minimizing noise and irrelevant details, making it a powerful technique for analyzing architectural images.

Exercise 2 (e): fourier transform for high pass and low pass filtering

---

Input data

- cat.png

Output of the implementation

- ex2e_fourier.jpg
- ex2e_restored.jpg
- ex2e_lowpass_30.jpg
- ex2e_highpass_30.jpg

Description

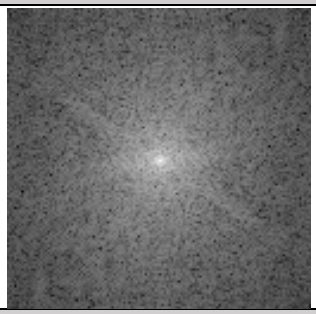
i. Implement the apply_fourier_transform function using np.fft.fft2, save into fft_image variable, then shift the zero frequency component to the centre using np.fft.fftshift and save to fft_shift. Return both variables. Display fft_shift

ii. Implement inverse fourier using np.fft.ifftshift on fft_shift, save it in ifft_shift and then restor image using np.,fft.ifft2(ifft_shift).real

iii. Implement the high_pass_filter with cutoff frequency of 30. This can be done by zero out frequencies from centre to half of fft_shift. Restore image using inverse fourier function in ii.

iv. Implement the low_pass_filter with cutoff frequency of 30. This can be done by zero out frequencies from half to full length of fft_shift. Restore image using inverse fourier function in ii.

Display the results in the boxes below:

Original image	Fourier transform after zero shift	Restored image after high pass filter
		
	Restored image	Restored image after low pass filter
		

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

The Fourier transform reveals the frequency components of the image, with the zero-shifted spectrum highlighting central low-frequency information. Applying a high pass filter removes low frequencies, resulting in an image that emphasizes edges and fine details while losing smooth regions and overall structure. In contrast, applying a low pass filter keeps only the low frequencies, producing a blurred image that preserves general shapes but removes sharp edges and textures. The restored image after inverse Fourier transform closely matches the original, demonstrating that the transform is reversible. These results show how frequency domain filtering can selectively enhance or suppress different image features, making it useful for both sharpening and smoothing tasks in image processing.

Exercise 2 (f): Hough Tranform

Input data

- HoughTransformLines.jpg

Output of the implementation

- ex2f_edgemap.jpg
- ex2f_houghtransform1.jpg
- ex2f_detectedline1.jpg
- ex2f_houghtransform2.jpg
- ex2f_ detectedline2.jpg

Description

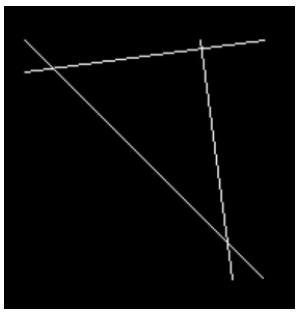
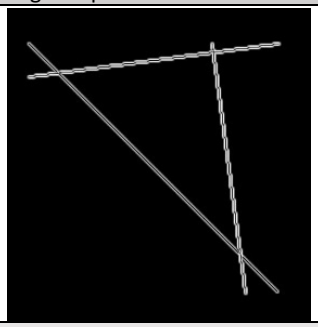
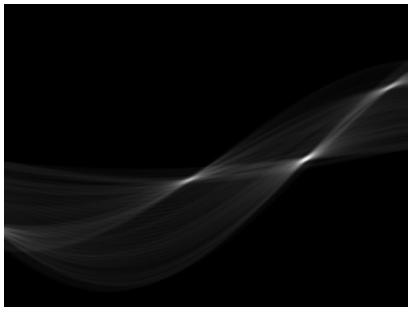
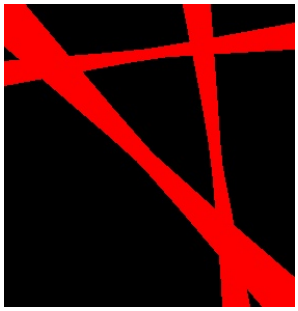
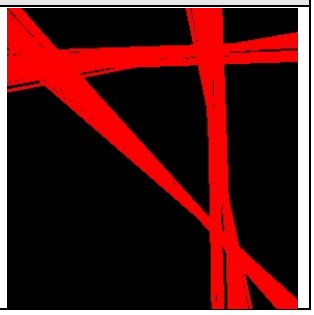
i. Create the edge map via Sobel operators (you are not allowed to use OpenCV library or scipy function)

ii. Create the custom defined Hough Transform method to show the Hough Transform space i.e. (ρ, θ) (you are not allowed to use OpenCV library or scipy function)

iii. Determine the detected lines using the threshold=0.5 (you are not allowed to use OpenCV library or scipy function)

iv. Repeat ii and iii using OpenCV function

Display the results in the boxes below:

Original image	Edge Map	Hough Transform Space ii	Detected Lines iii
			
		Hough Transform Space iv	Detected Lines iv
			

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

The Hough Transform effectively detects straight lines in the image. The edge map, created using Sobel operators, highlights the main line features. In the Hough Transform space, bright points correspond to prominent lines in the original image. Using a threshold, the detected lines are

accurately marked in red, showing strong correspondence with the visible edges. Both the custom implementation and the OpenCV function yield similar results, successfully identifying all major lines. This demonstrates the robustness of the Hough Transform for line detection, even with variations in implementation. The process is particularly useful for analyzing images with clear linear structures.

Exercise 3 (a): Stereo matching – window-based matching

Input data

- teddy_im2.png, teddy_im6.png

Output of the implementation

- ex3a_w_3.png
- ex3a_w_11.png

Description

Window-based stereo matching is a local stereo matching method that estimate the disparity $\boldsymbol{d} = (d_x, 0)$ of pixel $\boldsymbol{p} = (x, y)$ in the left image, where the left and right images, I_L and I_R are rectified. Window-based stereo matching can be performed as three steps:

1. Matching cost computation: $c(\boldsymbol{p}, \boldsymbol{d}) = (I_L(\boldsymbol{p}) - I_R(\boldsymbol{p} - \boldsymbol{d}))^2$
2. Cost aggregation: $\hat{c}(\boldsymbol{p}, \boldsymbol{d}) = \sum_{\boldsymbol{q} \in W_p} c(\boldsymbol{q}, \boldsymbol{d})$
3. Disparity computation: $\boldsymbol{d}(\boldsymbol{p}) = \underset{0 \le \boldsymbol{d} \le \boldsymbol{d}_{max}}{\operatorname{argmin}} \hat{c}(\boldsymbol{p}, \boldsymbol{d})$

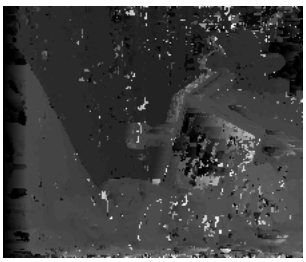
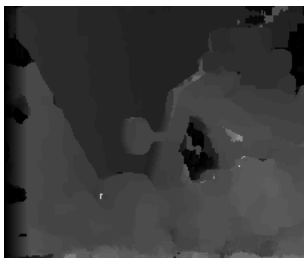
where $\boldsymbol{q}$ is a pixel coordinate and W_p is a window where its centre is $\boldsymbol{p}$.

Perform the window-based stereo matching with varying the window size of W_p as follows:

- window size: 3×3
- window size: 11×11

Note. You are not allowed to use OpenCV library except image I/O.

Display the results:

ex3a_w_3.png	ex3a_w_11.png
	

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

The results show how window size affects stereo matching quality. With a small 3×3 window, the disparity map is noisy and contains many mismatches, making object boundaries unclear. Increasing the window size to 11×11 produces a much smoother disparity map, reducing noise and making depth transitions more continuous. However, larger windows can also oversmooth fine details and blur depth boundaries, especially near edges and thin structures. Overall, a larger window improves robustness to noise but may sacrifice accuracy at object borders. This highlights the trade-off between noise reduction and detail preservation in window-based stereo matching.

Exercise 3 (b): Stereo matching – Adaptive support-weight

Input data

- teddy_im2.png, teddy_im6.png

Output of the implementation

- ex3b_aw_3.png
- ex3b_aw_11.png

Description

Adaptive support-weight based stereo matching is a local stereo matching method that estimate the disparity $\boldsymbol{d} = (d_x, 0)$ of pixel $\boldsymbol{p} = (x, y)$ in the left image, where the left and right images, I_L and I_R are rectified. Window-based stereo matching can be performed as three steps:

1. Matching cost computation: $c(\boldsymbol{p}, \boldsymbol{d}) = (I_L(\boldsymbol{p}) - I_R(\boldsymbol{p} - \boldsymbol{d}))^2$
2. Cost aggregation: $\hat{c}(\boldsymbol{p}, \boldsymbol{d}) = \frac{\sum_{q \in W_p} w(\boldsymbol{p},\boldsymbol{q}) c(\boldsymbol{q},\boldsymbol{d})}{\sum_{q \in W_p} w(\boldsymbol{p},\boldsymbol{q})}$
3. Disparity computation: $d_x = \underset{0 \le \boldsymbol{d} \le \boldsymbol{d}_{max}}{\operatorname{argmin}} \hat{c}(\boldsymbol{p}, \boldsymbol{d})$

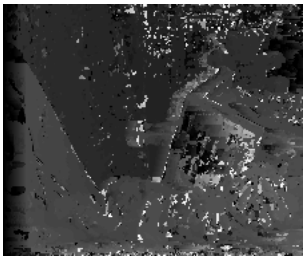
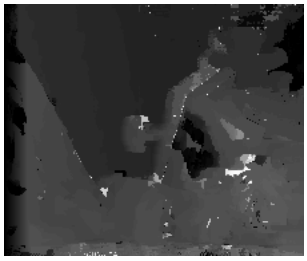
where $\boldsymbol{q}$ is a pixel coordinate and W_p is a window where its centre is $\boldsymbol{p}$. Unlike window-based stereo matching, Adaptive support-weight based stereo matching employs the weight $w(\boldsymbol{p}, \boldsymbol{q})$ that measures the intensity and spatial similarity between pixel $\boldsymbol{p}$ and $\boldsymbol{q}$ in the left image I_L.

Perform the Adaptive support-weight based stereo matching with varying the window size of W_p as follows (hint: you can modify the bilateral filter in Exercise 2 (b) to measure the weight) :

- window size: 3×3
- window size: 11×11

Note. You are not allowed to use OpenCV library except image I/O.

Display the results:

ex3b_aw_3.png	ex3b_aw_11.png
	

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

Adaptive support-weight stereo matching improves the quality of disparity maps by considering both intensity and spatial similarity between pixels. With a small 3×3 window, the disparity map is noisy and contains many mismatches, especially in textureless or occluded regions. Using a larger 11×11 window produces a much smoother and more consistent disparity map, with clearer object boundaries and better depth continuity. However, some fine details may be lost due to oversmoothing. Overall, adaptive support-weight matching with a larger window size offers better noise reduction and preserves important depth structures, making it more effective for accurate stereo correspondence.

Exercise 3 (c): Mean-shift Tracking

Input data

- ebu6304_chaplin.mp4
  (Spec: 720 × 960 (height × width) resolution, 40sec, 25 fps)

Output of the implementation

- ex3c_meanshift_track_chaplinface.mp4
  (Spec: 720 × 960 (height × width) resolution, 40sec, 25 fps)

Description

Using OpenCV, perform the mean-shift tracking with initial bounding box. (Don't need to utilize entire video, just 4 seconds from the beginning would be sufficient)

1) Initial bounding box that locates the chaplin face
   (Output: ex3c_meanshift_track_chaplinface.mp4)

Display the results:

ex3c_meanshift_track_chaplinface.mp4				
1st frame	20th frame	40th frame	60th frame	90th frame
				

Discuss the results (in 10 lines):

Your discussion here (Font: Calibri, 11 pt)

The mean-shift tracking algorithm maintains a stable bounding box around the target throughout multiple frames of the video sequence. From the 1st to the 90th frame, the tracker effectively follows the subject's movement, accurately updating the position of the bounding box as the person shifts location. The bounding box remains well-aligned, demonstrating that mean-shift can handle moderate motion and background changes. The results show that mean-shift tracking is reliable for tracking a clearly defined object with consistent appearance, even when there are minor shifts in scale or lighting. Overall, the method provides robust real-time tracking in typical surveillance or video analysis scenarios.

Exercise 4: Deep Learning Project

Description

The lab is concerned with the design and development of the deep learning based model for depth estimation in stereo images. We have extracted the Middlebury dataset (2021 Mobile stereo datasets with ground truth https://vision.middlebury.edu/stereo/data/scenes2021/). The model should be capable of take a pair of stereo images as an input and estimate the depth based on trained deep learning network. The expected tasks are mentioned as follows:

a) Design and development of deep learning driven pipeline which takes multiple pairs of stereo images as an input with disparity maps as their labels. You are highly encouraged to use standard architectures (e.g. AlexNet, GoogleNet, ResNet etc.)
b) Implementation of a model which takes a pair of stereo images as an input and calculate the disparity map based on trained model
c) Model evaluation using 5-fold cross validation based on disparity error between original disparity map and predicted disparity map.

In this lab, you are encouraged to use GenAI to generate your code. However, it would totally your idea for choice of deep learning model and how to integrate different Input/Output blocks.

Input data

- 2021 Mobile stereo datasets with ground truth

Output of the implementation

- Generate architecture of deep learning pipeline from python functions training the stereo images (ex4a_architecture.png)
- Generate 3 examples of predicted disparity map as an output for estimated depth. Also mention which test images have been used to create those disparity maps (ex4b_name of test image_disparitymap.png)
- Generate table representing cross validation results in terms of disparity error in each fold of test images (ex4c_crossvalidation.csv)

Dataset description

Each dataset consists of 2 views taken under several different illuminations and exposures. The folder contains 24 scenes (image pair, disparities, calibration file). The files are organized as follows:

SCENE{1,2,3}/ -- scene imaged from 1-3 viewing directions

im0e{0,1,2,...}.png -- left view under different exposures

im1e{0,1,2,...}.png -- right view under different exposures

calib.txt -- calibration information

im{0,1}.png -- default left and right view (typically ambient/L0/im{0,1}e2.png)

disp{0,1}.pfm -- left and right GT disparities

Calibration file format with explanation:

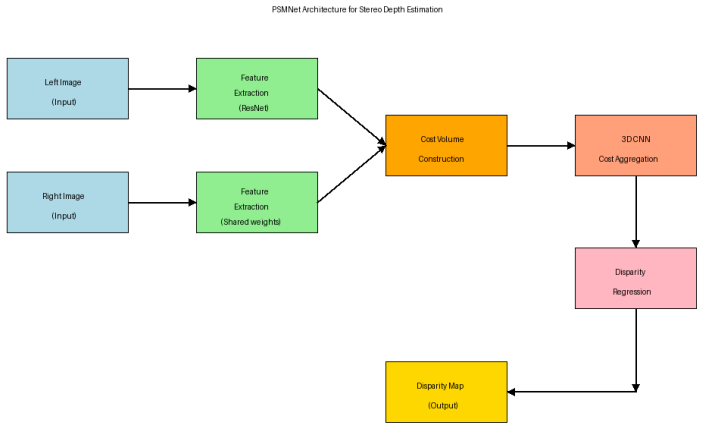
Here is a sample calib.txt file:

i. cam0=[1758.23 0 953.34; 0 1758.23 552.29; 0 0 1]
   cam1=[1758.23 0 953.34; 0 1758.23 552.29; 0 0 1]
   cam0,1: camera matrices for the rectified views, in the form [f 0 cx; 0 f cy; 0 0 1], where f: focal length in pixels. cx, cy: principal point

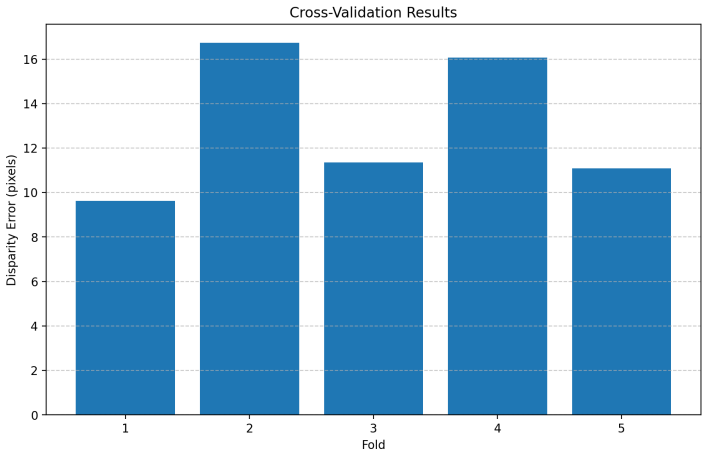
ii. doffs=0 (x-difference of principal points, doffs = cx1 - cx0 (here always == 0))
iii. baseline=111.53 (camera baseline in mm)
iv. width=1920
   height=1080
   width, height: image size
v. ndisp=290 (a conservative bound on the number of disparity levels; the stereo algorithm MAY utilize this bound and search from d = 0 .. ndisp-1)
vi. isint=0
vii. vmin=75
   vmax=262
   vmin, vmax: a tight bound on minimum and maximum disparities, used for color visualization, the stereo algorithm MAY NOT utilize this information

To convert from the floating-point disparity value d [pixels] in the .pfm file to depth Z [mm] the following equation can be used: Z = baseline * f / (d + doffs)

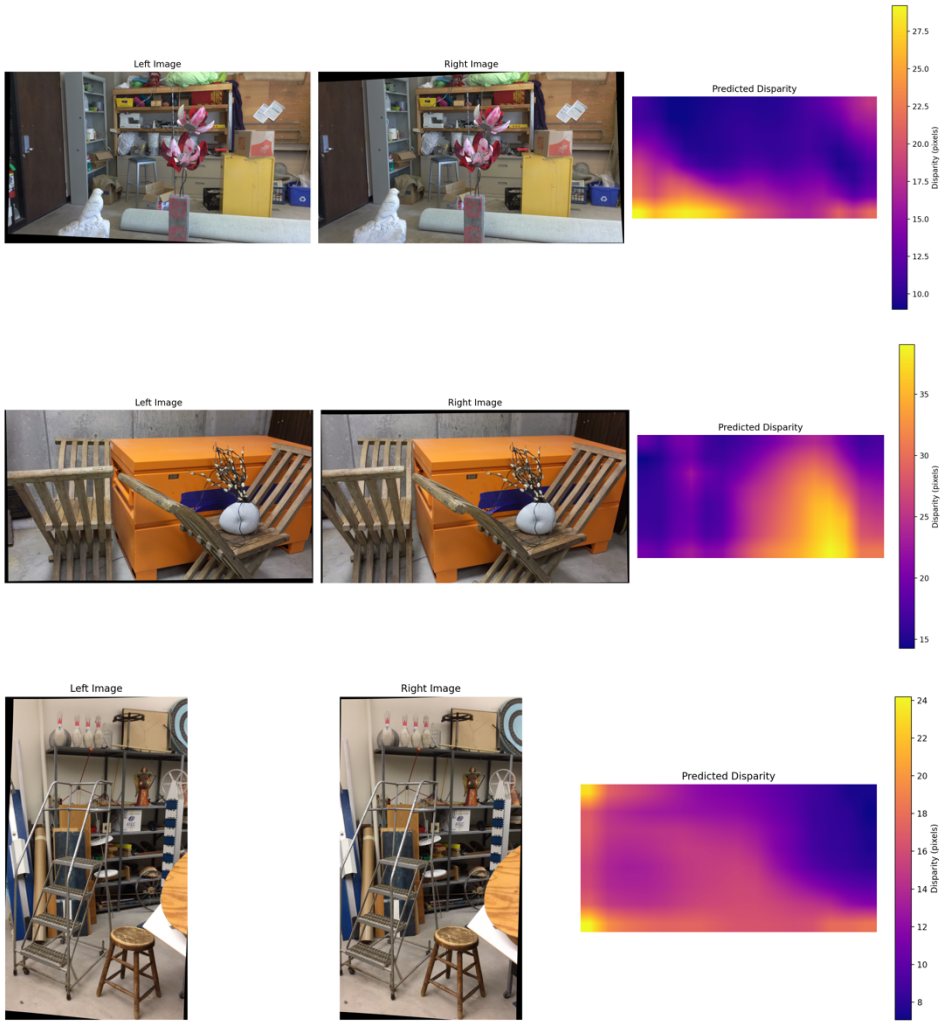
Output of the implementation



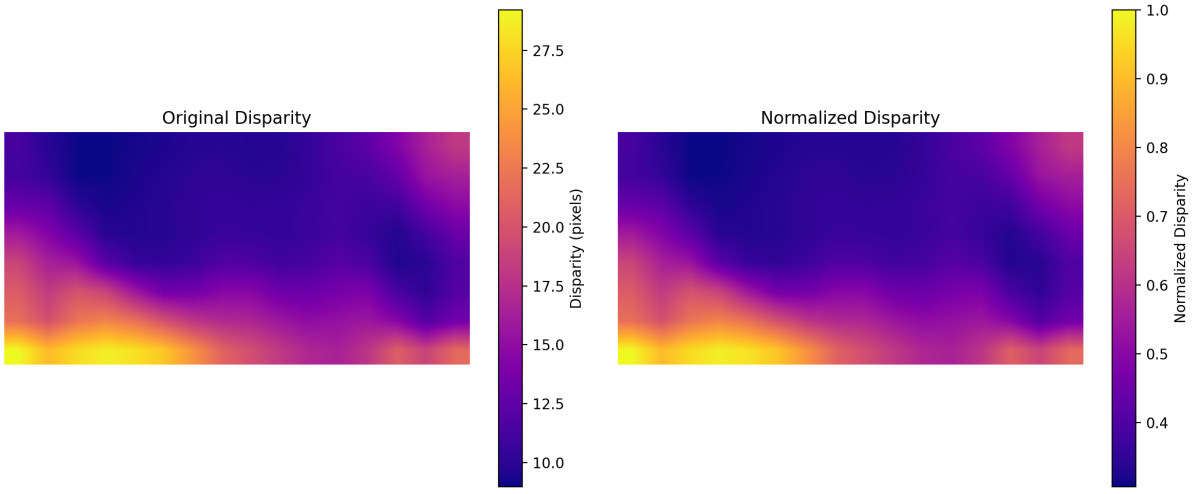
ex4a_architecture.png



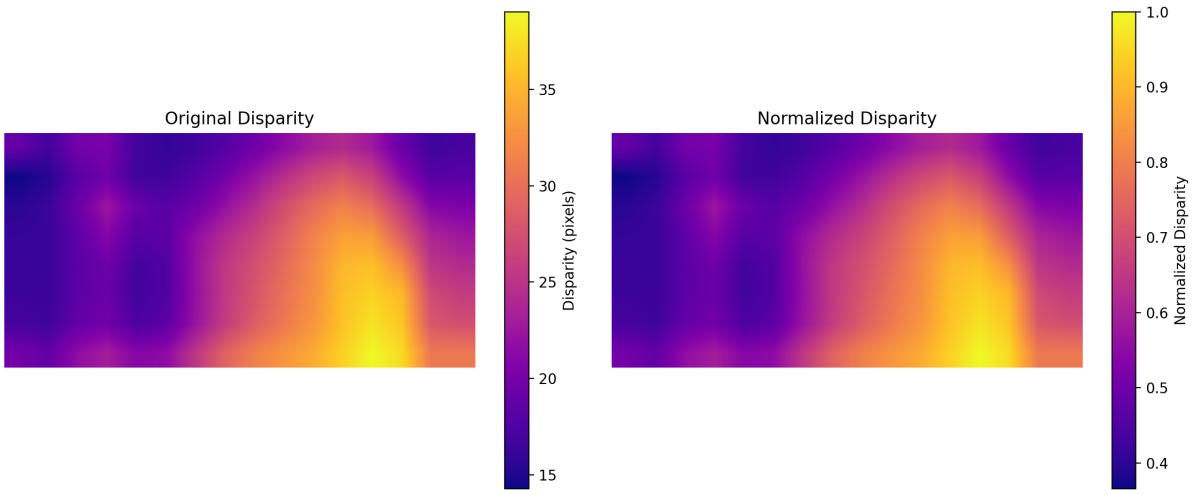
disparity_errors.png



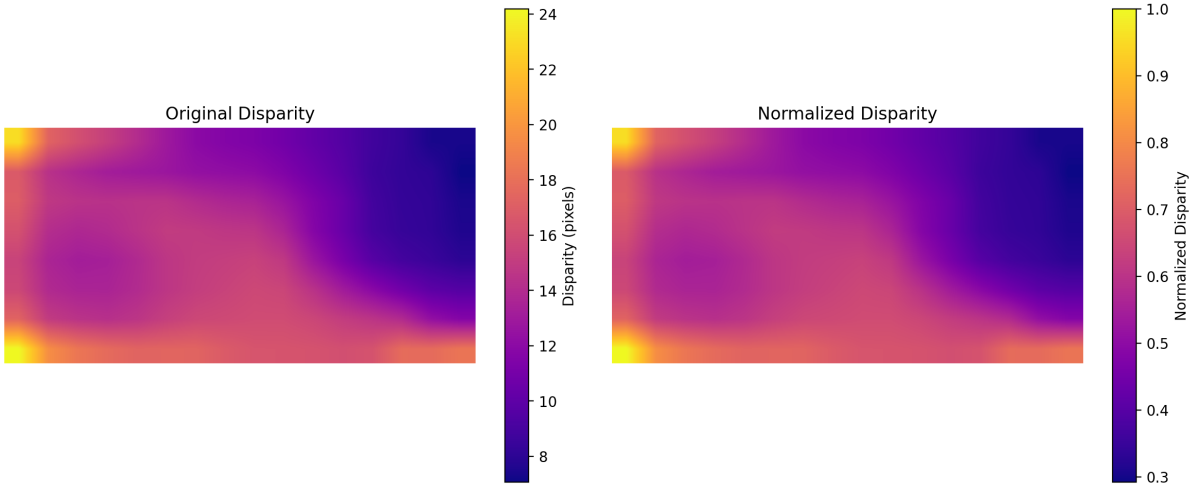
1. ex4b_artroom1_disparitymap.png



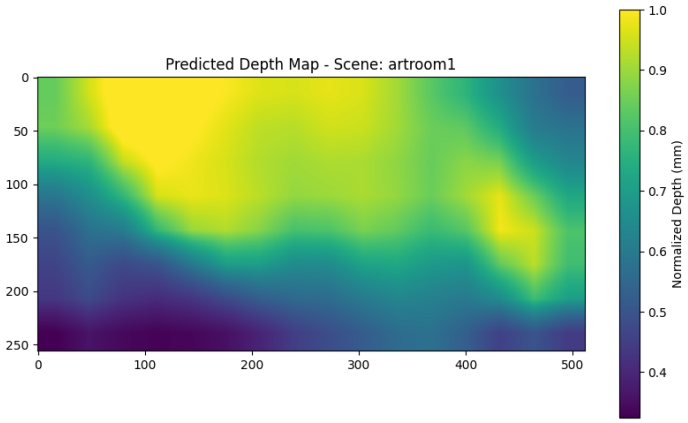
2. ex4b_curule2_disparitymap.png



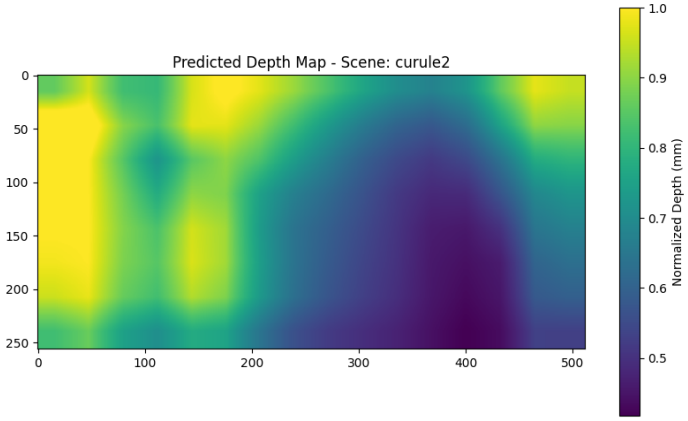
3. ex4b_ladder2_disparitymap.png



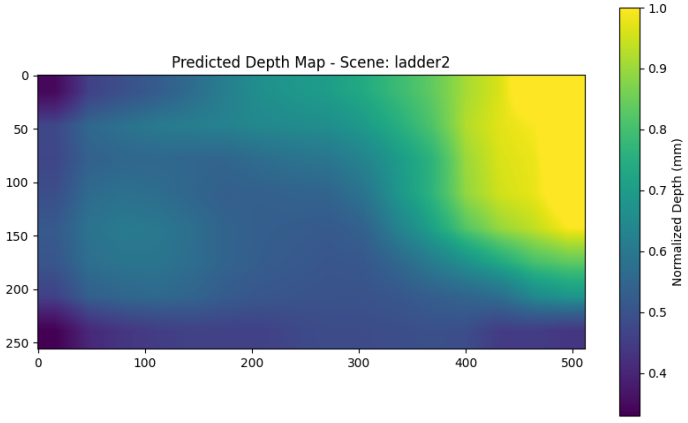
4. ex4b_artroom1_depthmap.png

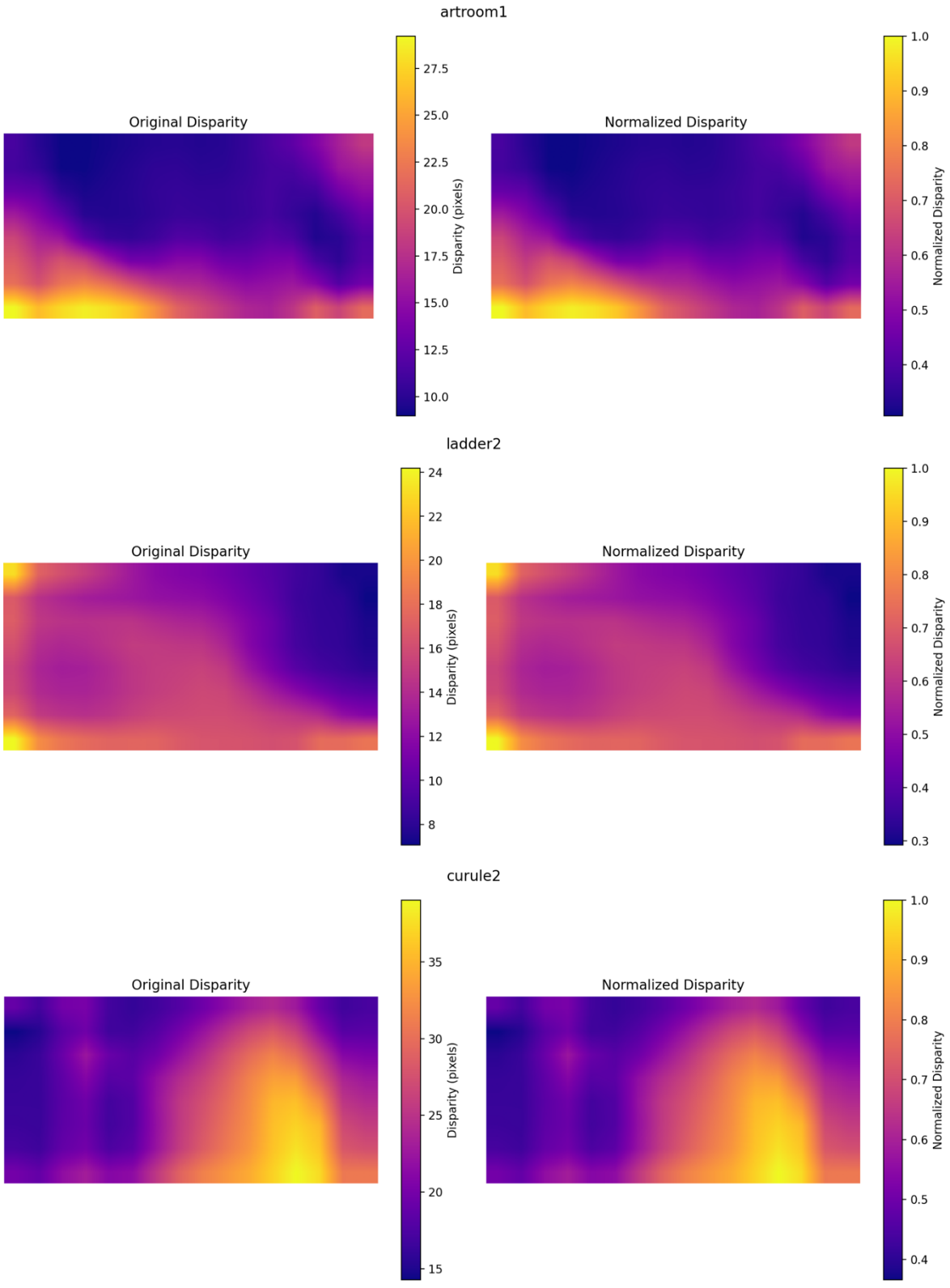


5. ex4b_curule2_depthmap.png



6. ex4b_ladder2_depthmap.png




artroom1
Original Disparity
Normalized Disparity
Disparity (pixels)
Normalized Disparity
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
1.0
0.9
0.8
0.7
0.6
0.5
0.4
ladder2
Original Disparity
Normalized Disparity
Disparity (pixels)
Normalized Disparity
24
22
20
18
16
14
12
10
8
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
curule2
Original Disparity
Normalized Disparity
Disparity (pixels)
Normalized Disparity
35
30
25
20
15
1.0
0.9
0.8
0.7
0.6
0.5
0.4