Photogrammetric Computer Vision

Exercise 6

Winter semester 24/25

(Course materials for internal use only!)

Computer Vision in Engineering – Prof. Dr. Rodehorst
M.Sc. Mariya Kaisheva
mariya.kaisheva@uni-weimar.de

Agenda

Topics

Assignment 1.	Points and lines in the plane, first steps in MATLAB / Octave
Assignment 2.	Projective transformation (Homography)
Assignment 3.	Camera calibration using direct linear transformation (DLT)
Assignment 4.	Orientation of an image pair
Assignment 5.	Projective and direct Euclidean reconstruction
Assignment 6.	**Stereo image matching**
Final Project	- will be announced later -

Agenda

	Start date		Deadline
Assignment 1.	~~21.10.24~~	~~–~~	~~03.11.24~~
Assignment 2.	~~04.11.24~~	~~–~~	~~17.11.24~~
Assignment 3.	~~18.11.24~~	~~–~~	~~01.12.24~~
Assignment 4.	~~02.12.24~~	~~–~~	~~15.12.24~~
Assignment 5.	~~16.12.24~~	~~–~~	~~12.01.25~~
Assignment 6.	**13.01.25**	**–**	**26.01.25**
Final Project.	27.01.25	–	16.03.25

Assignment 5 – sample solution

Bauhaus-
Universität
Weimar

Assignment 5 Part 1: *Projective reconstruction*

Input: Homologeous Image Coordinates

↓

1. Computation of Fundamental Matrix F

↓

2. Definition of Projection Matrices

↓

3. Linear Triangulation

↓

Output: Projective 3D Reconstruction

Sample code: Part 1

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');            % Read corresponding image points
F = linear_fund(x1, x2);                        % Compute relative orientation
```

```
function [x1, x2] = read_matches(name)               % Read image point matches
%                  ==================
fh = fopen(name, 'r');
A = fscanf(fh, '%f%f%f%f', [4 inf]);                  % Format: x1, y1, x2, y2
fclose(fh);
x1 = A(1:2, :); x1(3, :) = 1;                   % Homogeneous image coordinates
x2 = A(3:4, :); x2(3, :) = 1;
```

Sample code: Part 1

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');          % Read corresponding image points
F = linear_fund(x1, x2);                     % Compute relative orientation
[P1, P2] = define_cameras(F);                  % Define projection matrices


function [P1, P2] = define_cameras(F)          % Define projection matrices
%                  ==================
[e1, e2] = get_epipols(F);
P1 = eye(3, 4);                                        % Normalized camera
P2 = [skew_mat(e2)*F + [e2 e2 e2], e2];    % Projective camera using F-matrix


function [e1, e2] = get_epipols(F)        % Extract epipols from the F-matrix
%                  ==============
[U, D, V] = svd(F);                             % Singular value decomposition
e1 = V(:, 3);                                           % right nullvector
e2 = U(:, 3);                                            % left nullvector


function M = skew_mat(v)                          % Build skew symmetric matrix
%              ===========
M = [0    -v(3) v(2);
     v(3) 0    -v(1);
    -v(2) v(1) 0   ];
```

Linear Triangulation

- Solve linear equation system for all points
- One Sytem for each point (A is a 4x4 matrix)

$$\mathbf{A}\mathbf{X}=\mathbf{0}, \quad \mathbf{A}=\begin{bmatrix} x\,\mathbf{p}^{3}-\mathbf{p}^{1} \\ y\,\mathbf{p}^{3}-\mathbf{p}^{2} \\ x'\mathbf{p}'^{3}-\mathbf{p}'^{1} \\ y'\mathbf{p}'^{3}-\mathbf{p}'^{2} \end{bmatrix}, \quad \mathbf{X}=\begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix}$$

where $\mathbf{p}^{i}$ denotes the row i of $\mathbf{P}$

- Normalize 3D points **X** using W

Sample code: Part 1

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');           % Read corresponding image points
F = linear_fund(x1, x2);                        % Compute relative orientation
[P1, P2] = define_cameras(F);                     % Define projection matrices
X = linear_tri(P1, P2, x1, x2);                         % Spatial intersection
```

```
function X = linear_tri(P1, P2, x1, x2)                   % Linear triangulation
%            ==========================
for i = 1 : size(x1, 2)                                   % For all image points
    A = [x1(1,i)*P1(3,:) - P1(1,:);                              % Design matrix
         x1(2,i)*P1(3,:) - P1(2,:);
         x2(1,i)*P2(3,:) - P2(1,:);
         x2(2,i)*P2(3,:) - P2(2,:)];
    X(:, i) = enorm(solve_dlt(A));                               % Object points
end

function y = enorm(x)               % Euclidean normalization (x, y, ..., 1)^T
%            ========
for i = 1 : size(x, 2)
    y(:, i) = x(:, i) / x(end, i);
end
```

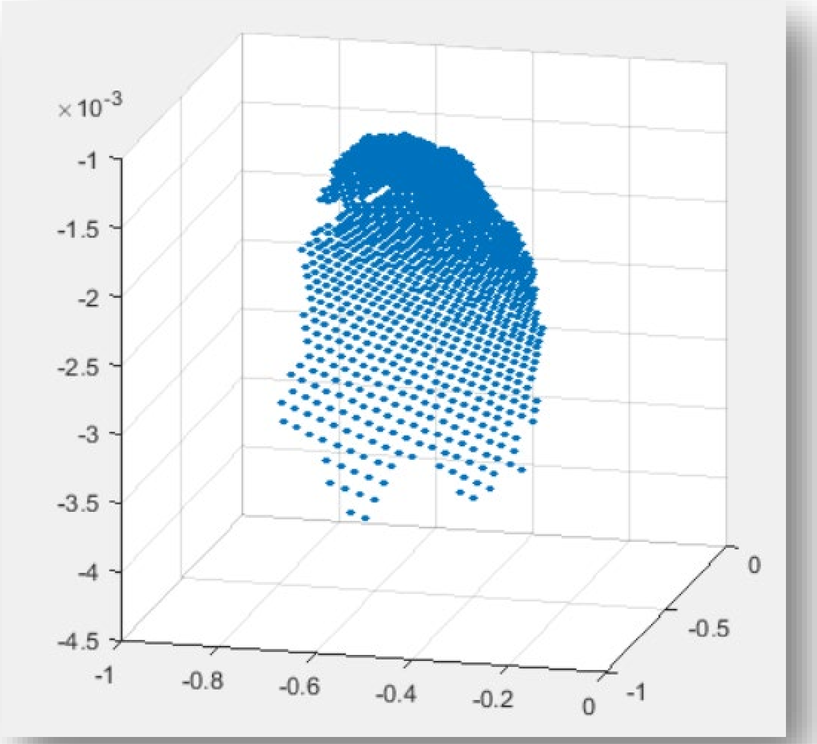
Sample code: Part 1

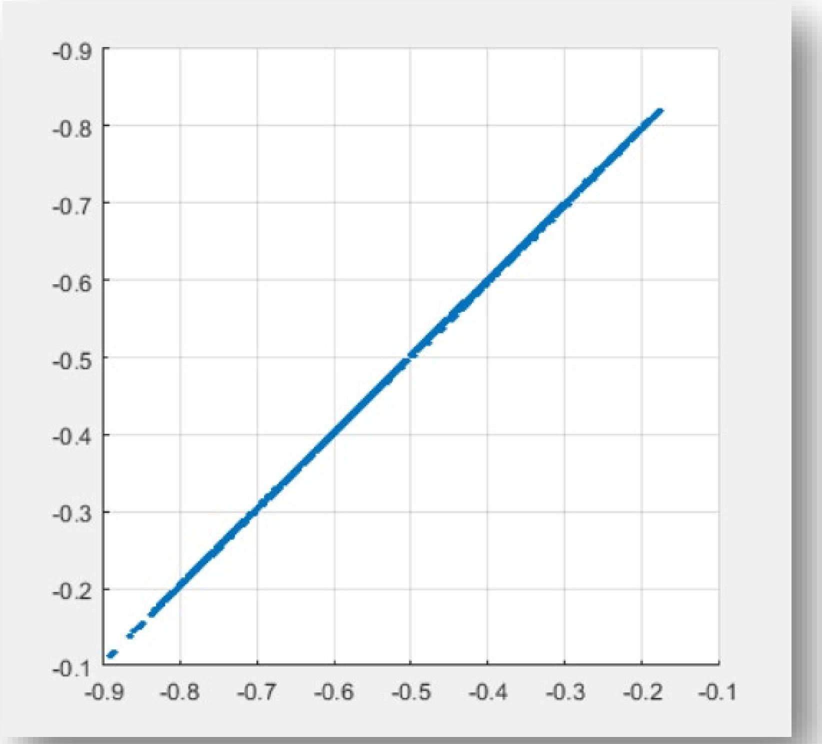
```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');           % Read corresponding image points
F = linear_fund(x1, x2);                       % Compute relative orientation
[P1, P2] = define_cameras(F);                    % Define projection matrices
X = linear_tri(P1, P2, x1, x2);                        % Spatial intersection
plot_3d(X);                                    % Draw projective point cloud
```

```
function plot_3d(X) % Draw point cloud
% ==========
figure; scatter3(X(1, :), X(2, :), X(3, :), 10, 'filled');
axis square; view(32, 75);
```

Sample code: Part 1

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');          % Read corresponding image points
F = linear_fund(x1, x2);                       % Compute relative orientation
[P1, P2] = define_cameras(F);                   % Define projection matrices
X = linear_tri(P1, P2, x1, x2);                       % Spatial intersection
plot_3d(X);                                    % Draw projective point cloud
```





Assignment 5 Part 2: *Euclidean reconstruction*

Input: -Projective 3D Reconstruction
-Projection Matrices P_N, P'
-5 Control Points (Euclidean) $x_1 \leftrightarrow x_2, X_E$

1. Triangulate 2D control points using P_N, P'

Intermediate Result: Projective 3D Coordinates X_{P2} of 5 control points

2. Estimate 3D Homography H using X_E, X_{P2}

3. Transform all 3D points of the projective reconstruction to obtain euclidean reconstruction using H

Output: Euclidean 3D reconstruction

Sample code: Part 2

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');          % Read corresponding image points
F = linear_fund(x1, x2);                       % Compute relative orientation
[P1, P2] = define_cameras(F);                   % Define projection matrices
X = linear_tri(P1, P2, x1, x2);                       % Spatial intersection
plot_3d(X);                                    % Draw projective point cloud

[x1, x2, Xe] = read_control('pp.dat');       % Read control point information
```

```
function [x1, x2, X] = read_control(name)     % Read control point information
%                      ==================
fh = fopen(name, 'r');
A = fscanf(fh, '%f%f%f%f%f%f%f', [7 inf]);  % Format: x1, y1, x2, y2, X, Y, Z
fclose(fh);
x1 = A(1:2, :); x1(3, :) = 1;                 % Homogeneous image coordinates
x2 = A(3:4, :); x2(3, :) = 1;
X  = A(5:7, :);  X(4, :) = 1;                % Homogeneous object coordinates
```

Bauhaus-Universität Weimar

Sample code: Part 2

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');           % Read corresponding image points
F = linear_fund(x1, x2);                        % Compute relative orientation
[P1, P2] = define_cameras(F);                     % Define projection matrices
X = linear_tri(P1, P2, x1, x2);                         % Spatial intersection
plot_3d(X);                                     % Draw projective point cloud

[x1, x2, Xe] = read_control('pp.dat');        % Read control point information
Xp = linear_tri(P1, P2, x1, x2);                  % Triangulate control points
H = homography3(Xp, Xe);                          % Compute spatial homography

function H = homography3(X1, X2)              % General spatial transformation
%            ===================
T1 = condition3(X1); N1 = T1 * X1;             % Conditioning of object points
T2 = condition3(X2); N2 = T2 * X2;
A = design_homo3(N1, N2);                                % Build design matrix
h = solve_dlt(A);                              % Linear least-squares-solution
H = inv(T2) * reshape(h, 4, 4)' * T1;                   % Reverse conditioning

function A = design_homo3(X1, X2)       % Design matrix for spatial homography
%            ====================
A = [];
for i = 1 : size(X1, 2)                                 % For all object points
    A = [ A; -X2(4,i)*X1(:,i)'  0 0 0 0  0 0 0 0  X2(1,i)*X1(:,i)';
             0 0 0 0  -X2(4,i)*X1(:,i)'  0 0 0 0  X2(2,i)*X1(:,i)';
             0 0 0 0  0 0 0 0  -X2(4,i)*X1(:,i)'  X2(3,i)*X1(:,i)'];
end
```

- **Given:** At least $n \geq 5$ point pairs in space $\mathbf{X}_i \leftrightarrow \mathbf{X}'_i$
- **Wanted:** 4×4 homography matrix $\mathbf{H}$ (15 DOF) for which $\mathbf{X}'_i = \mathbf{H}\mathbf{X}_i$ holds
- **Conditioning** of the object points $\mathbf{X}_i = (U_i, V_i, W_i, T_i)^\mathsf{T}$ and $\mathbf{X}'_i$ by translation to the origin and scaling to a mean distance of $\sqrt{3}$
- Assemble the **design matrix**:

$$\mathbf{A}_i = \begin{bmatrix} -\tilde{T}'_i \tilde{\mathbf{X}}_i^\mathsf{T} & \mathbf{0}^\mathsf{T} & \mathbf{0}^\mathsf{T} & \tilde{U}'_i \tilde{\mathbf{X}}_i^\mathsf{T} \\ \mathbf{0}^\mathsf{T} & -\tilde{T}'_i \tilde{\mathbf{X}}_i^\mathsf{T} & \mathbf{0}^\mathsf{T} & \tilde{V}'_i \tilde{\mathbf{X}}_i^\mathsf{T} \\ \mathbf{0}^\mathsf{T} & \mathbf{0}^\mathsf{T} & -\tilde{T}'_i \tilde{\mathbf{X}}_i^\mathsf{T} & \tilde{W}'_i \tilde{\mathbf{X}}_i^\mathsf{T} \end{bmatrix}$$

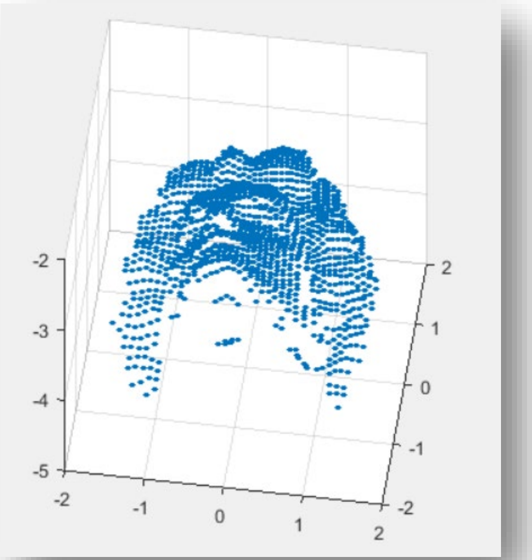
- **Solution** of $\mathbf{A}\mathbf{h} = \mathbf{0}$ using SVD
- **Reshape** vector $\mathbf{h} = (h_1, \ldots, h_{16})^\mathsf{T}$ in matrix form $\tilde{\mathbf{H}}$ and finally
- **Reverse conditioning** with $\mathbf{H} = \mathbf{T}'^{-1}\tilde{\mathbf{H}}\mathbf{T}$

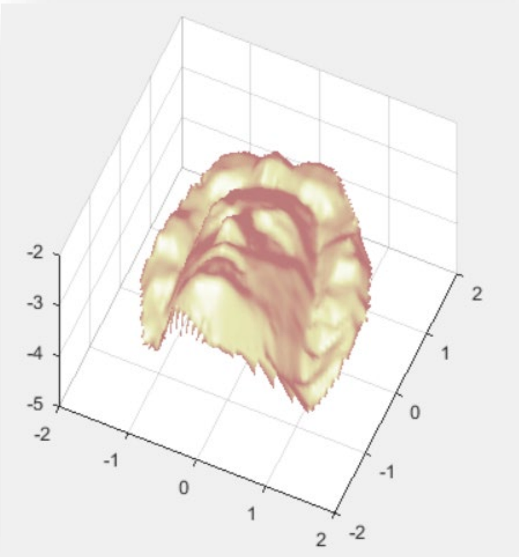
Sample code: Part 2

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');          % Read corresponding image points
F = linear_fund(x1, x2);                      % Compute relative orientation
[P1, P2] = define_cameras(F);                   % Define projection matrices
X = linear_tri(P1, P2, x1, x2);                       % Spatial intersection
plot_3d(X);                                    % Draw projective point cloud

[x1, x2, Xe] = read_control('pp.dat');        % Read control point information
Xp = linear_tri(P1, P2, x1, x2);                % Triangulate control points
H = homography3(Xp, Xe);                        % Compute spatial homography
X = enorm(H * X);                   % Upgrade from projective to Euclidean space
plot_3d(X);                                     % Draw Euclidean point cloud
plot_surface(X);                                % Draw shaded object surface


function plot_surface(X)                        % Draw shaded object surface
%        ===============
t = -2:0.05:2;                                          % Generate raster points
[XI, YI] = meshgrid(t, t);
ZI = griddata(X(1,:), X(2,:), X(3,:), XI, YI, 'cubic');     % Interpolate depth
figure; surfl(XI, YI, ZI); axis square; view(67, 50);
shading interp; colormap(pink);
```

Sample code: Part 2

```
function exercise5
%        =========
[x1, x2] = read_matches('bh.dat');          % Read corresponding image points
F = linear_fund(x1, x2);                     % Compute relative orientation
[P1, P2] = define_cameras(F);                % Define projection matrices
X = linear_tri(P1, P2, x1, x2);              % Spatial intersection
plot_3d(X);                                  % Draw projective point cloud

[x1, x2, Xe] = read_control('pp.dat');       % Read control point information
Xp = linear_tri(P1, P2, x1, x2);             % Triangulate control points
H = homography3(Xp, Xe);                     % Compute spatial homography
X = enorm(H * X);              % Upgrade from projective to Euclidean space
plot_3d(X);                                  % Draw Euclidean point cloud
plot_surface(X);                             % Draw shaded object surface
```





Assignment 6: *Stereo image matching*

Assignment 6: *Stereo image matching*

For the exercise a pair of normal images is taken from the Middlebury stereo vision research page (left.png and right.png).

a) Read the images and convert the gray value intensities to float values (`double`). Implement a procedure in MATLAB for the *normalized cross-correlation* (`mean, sqrt, mean2`) without using the build-in functions (i.e. `std, var, cov, std2, corr2, corrcoef, xcov, xcorr`).

- For each pixel in the left image define a reference window `img(i-r : i+r, j-r : j+r)` and search horizontally in the right image for a window position with maximum correlation. You may have to cope with the image borders (`min, max`).
- Produce a *disparity map* for the left image by registering the horizontal coordinate difference between the reference windows and most similar search windows.

b) Visualize the disparity map as *gray value image* (`imshow(…, [])`).

c) Find the optimal parameters for the *window size* and for the *search range*.

Assignment 6: *Stereo image matching using **normalized cross-correlation***

reference image

search image

$$\rho_{NCC}(a,b) = \frac{\sigma_{ab}}{\sqrt{\sigma_a^2 \cdot \sigma_b^2}}$$

$$= \frac{\frac{1}{n^2}\left(\sum_{i,j=1}^{n} a(i,j)\cdot b(i,j)\right) - \bar{a}\cdot\bar{b}}{\sqrt{\left(\frac{1}{n^2}\left(\sum_{i,j=1}^{n} a(i,j)^2\right) - \bar{a}^2\right)\cdot\left(\frac{1}{n^2}\left(\sum_{i,j=1}^{n} b(i,j)^2\right) - \bar{b}^2\right)}}$$

Bauhaus-Universität Weimar

Assignment 6:
Hints

1. Pre-calculation of mean values
2. Exclude the image border pixels wrt. the chosen window radius r
3. Further reduce the search space by defining a maximum possible search range (task c) in the second image, e.g. $d_{min} = 5$ and $d_{max} = 12$.
4. Extract a window with radius r from array:

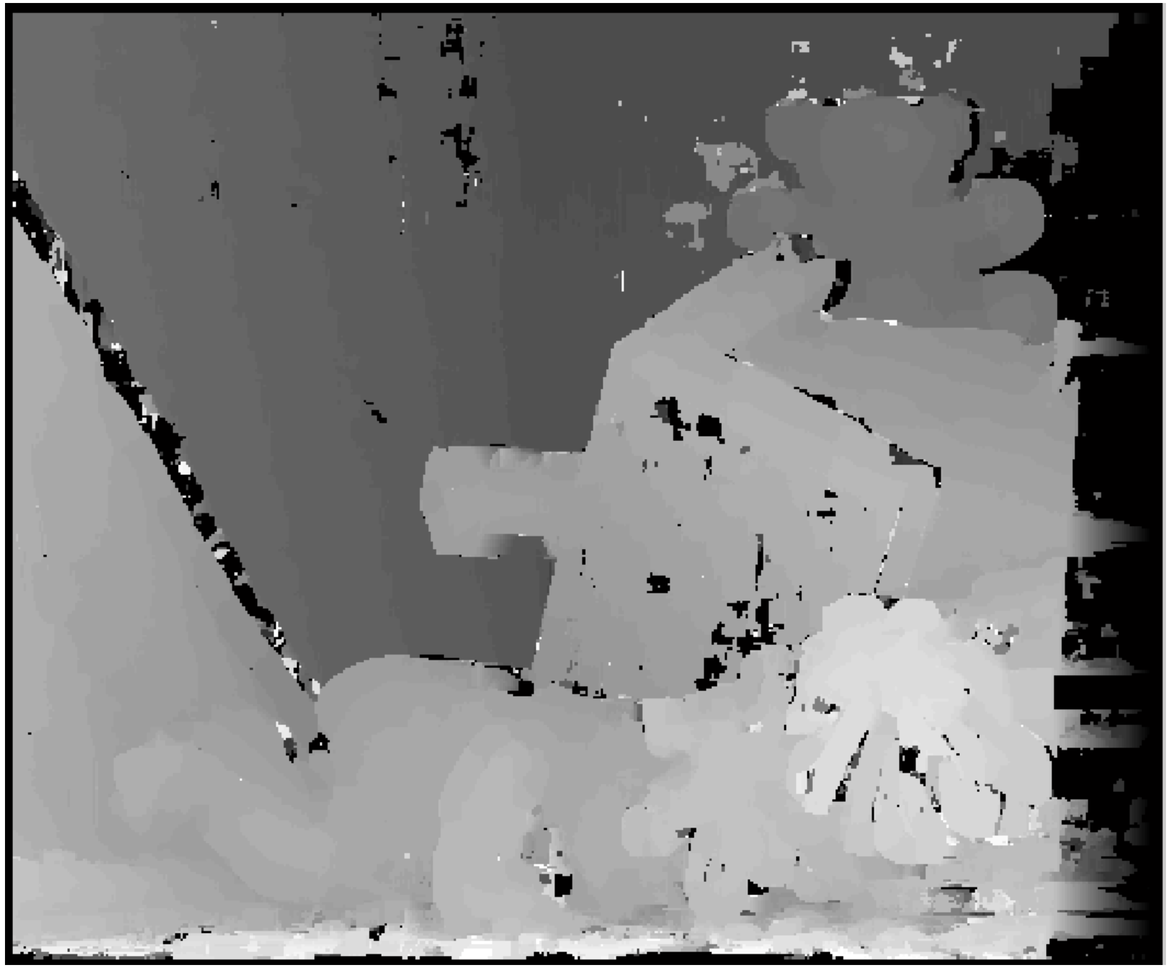
$$W = Image\left(pos_x - r : pos_x + r, pos_y - r : pos_y + r\right)$$

5. NCC is not defined for homogeneous image areas

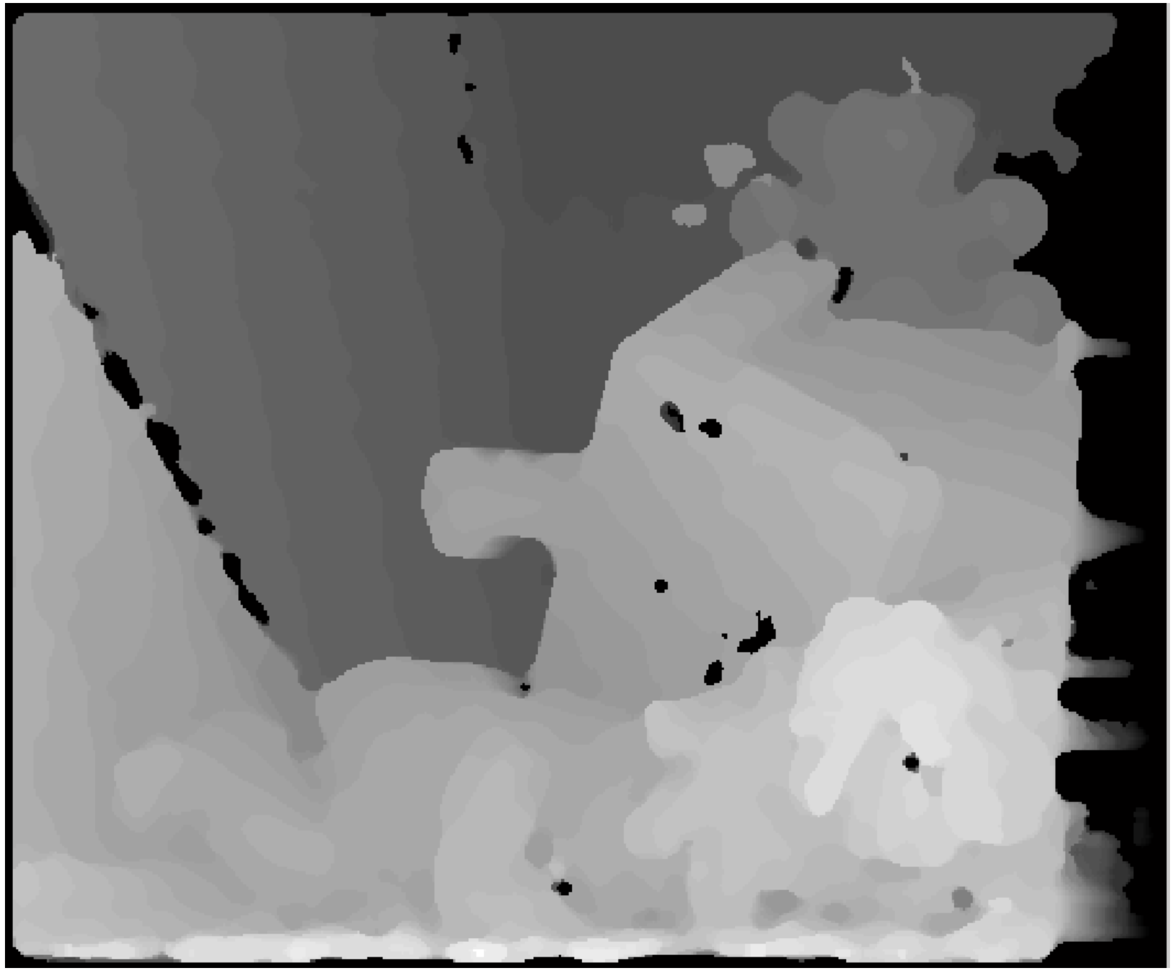
   → Test if region variance > 0

6. You may apply an appropriate filter for depth map smoothing

Assignment 6: Sample results



Depth Map



Smoothed Depth Map

Bauhaus-Universität Weimar