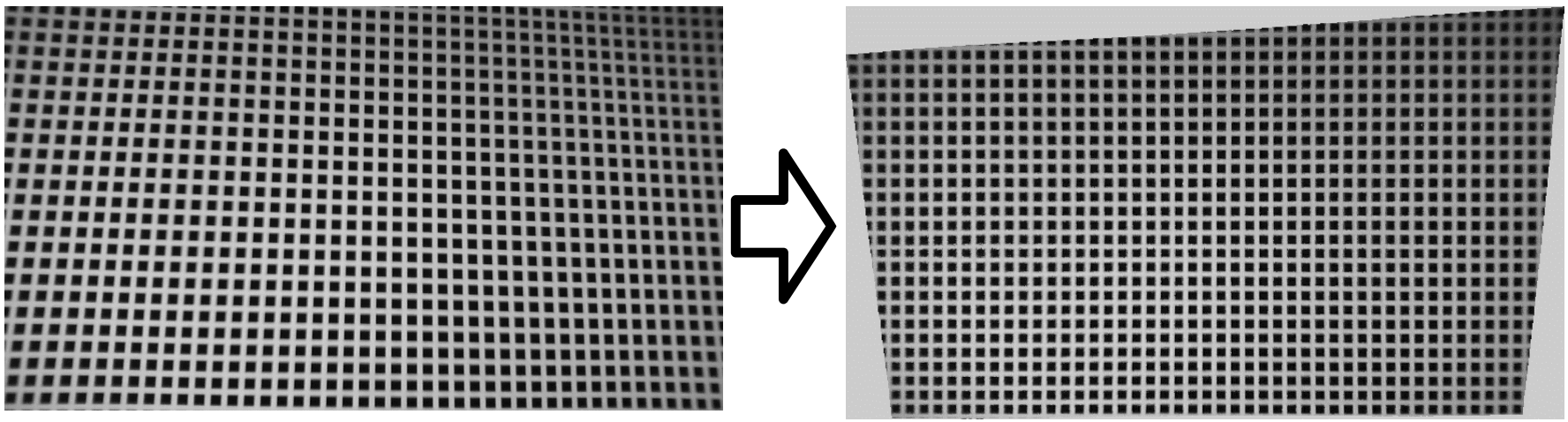
Automatic distortion calibration

Ahmed Ashraf, Nils Hamacher, Linghan Qian, Vivica Wirth

TECHNISCHE UNIVERSITÄT DARMSTADT



Task description

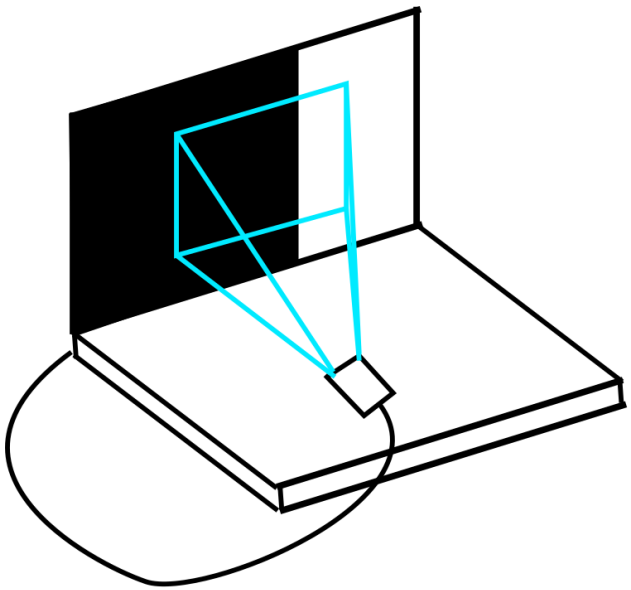


Figure: System

Task:
Calibrate camera distortion automatically
Advantage:
Efficiency
No human-interaction
Dense model
1. Precise position
2. Non-parametric

Distortion example

Source [4]

$$x = x_d + (x_d - x_c)(1 + K_1 r^2 + K_2 r^4) + P_1 \left(r^2 + 2(x_d - x_c)^2\right) + 2P_2(x_d - x_c)(y_d - y_c)$$

$$y = y_d + (y_d - y_c)(1 + K_1 r^2 + K_2 r^4) + 2P_1(x_d - x_c)(y_d - y_c) + P_2 \left(r^2 + 2((y_d - y_c)^2\right)$$

Radial distortion:

$K_n = n^{th}$ radial distortion coefficient

Tangential distortion:

$P_n = n^{th}$ tangential distortion coefficient

$(x_d, y_d) =$ distorted image point as projected on image plane,

$(x, y) =$ undistorted image point as projected on image plane,

$(x_c, y_c) =$ distortion center,

$r = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2}$,

coefficients bigger 2 were not considered.

Distortion example

$$x = x_d + (x_d - x_c)(1 + K_1 r^2 + K_2 r^4) + P_1 \left(r^2 + 2(x_d - x_c)^2\right) + 2P_2(x_d - x_c)(y_d - y_c)$$

$$y = y_d + (y_d - y_c)(1 + K_1 r^2 + K_2 r^4) + 2P_1(x_d - x_c)(y_d - y_c) + P_2 \left(r^2 + 2((y_d - y_c)^2\right)$$

Radial distortion:

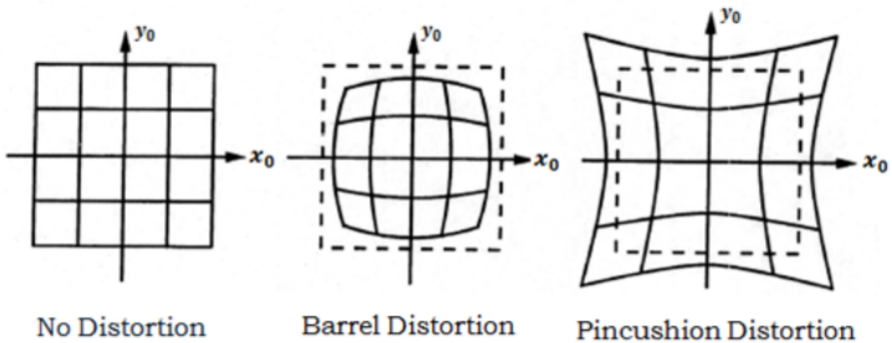


Figure: radial distortions [3]

Tangential distortion:

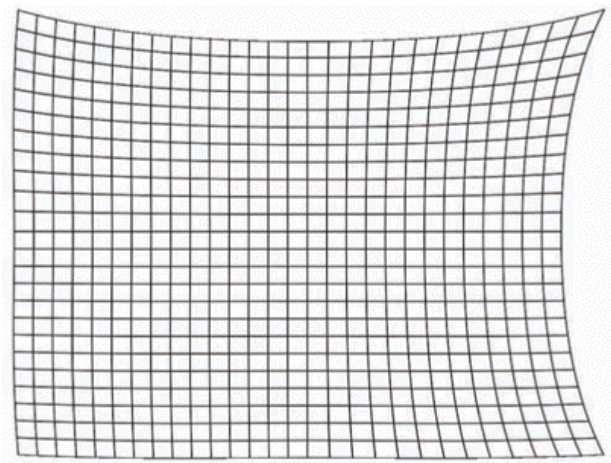


Figure: first order tangential distortion [2]

Applying directional filters to extract edges

Sobel operator

Figure: Original image [1]

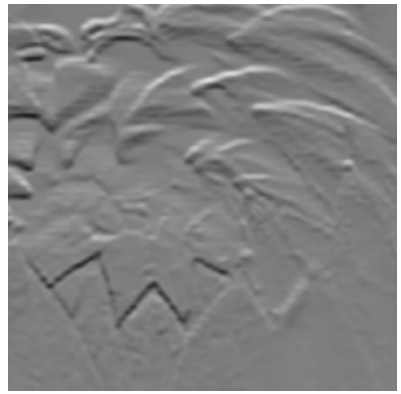


Figure: Vertical edges [1]

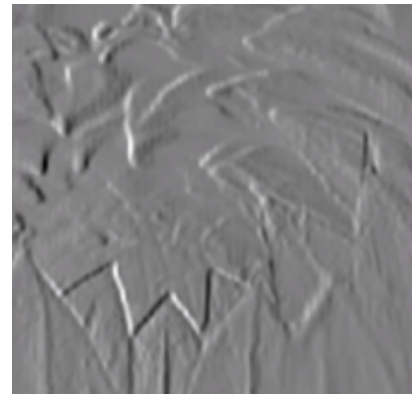


Figure: Horizontal edges [1]

$$Kernel_{vertical} = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$Kernel_{horizontal} = Kernel_{vertical}^{T}$$

Applying directional filters to extract edges

Canny operator

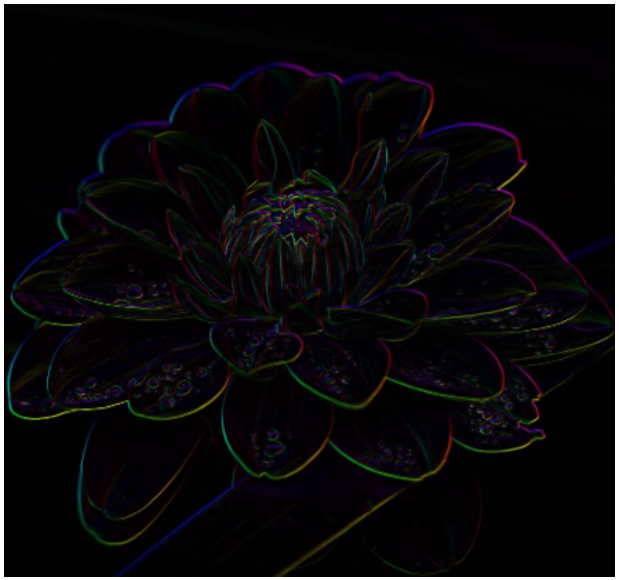


Figure: Orientation image [1]

Figure: Canny image

$$|grad(I)| = \sqrt[2]{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$$

$$\theta_{orientation} = \arctan\left(\frac{\partial I}{\partial y}, \frac{\partial I}{\partial x}\right)$$

Applying directional filters to extract edges

Algorithms comparison [1]

TECHNISCHE UNIVERSITÄT DARMSTADT

Original image

Figure: Original image

Sobel operator

Figure: Image with Sobel operator

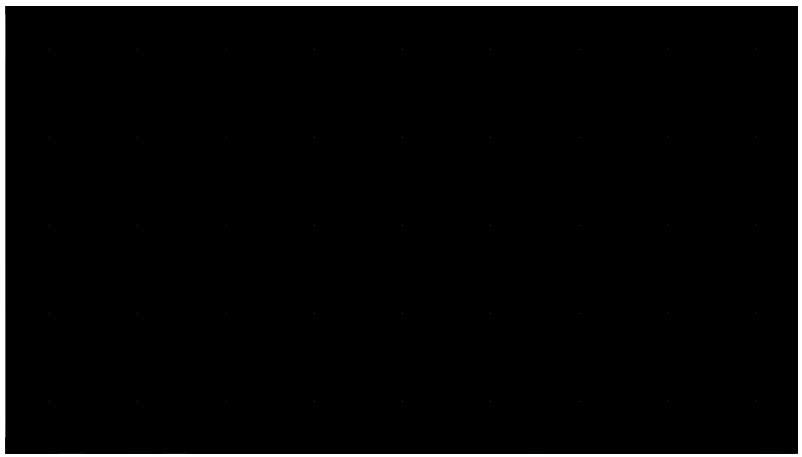
Canny operator

Figure: Image with Canny operator

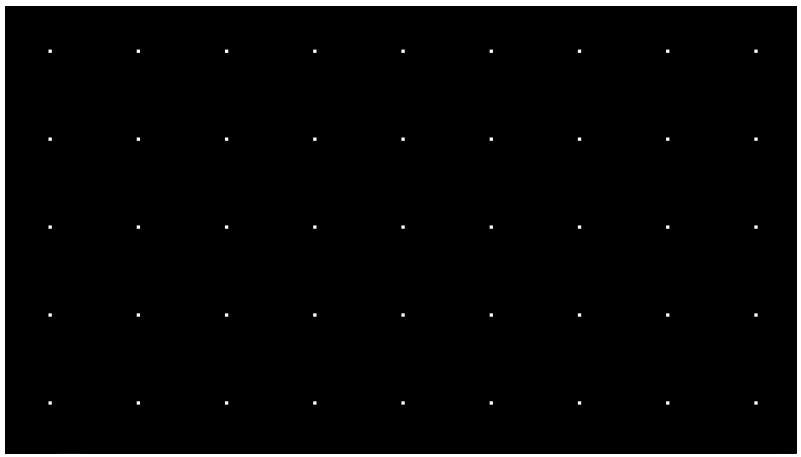
pixel size detection

TECHNISCHE UNIVERSITÄT DARMSTADT

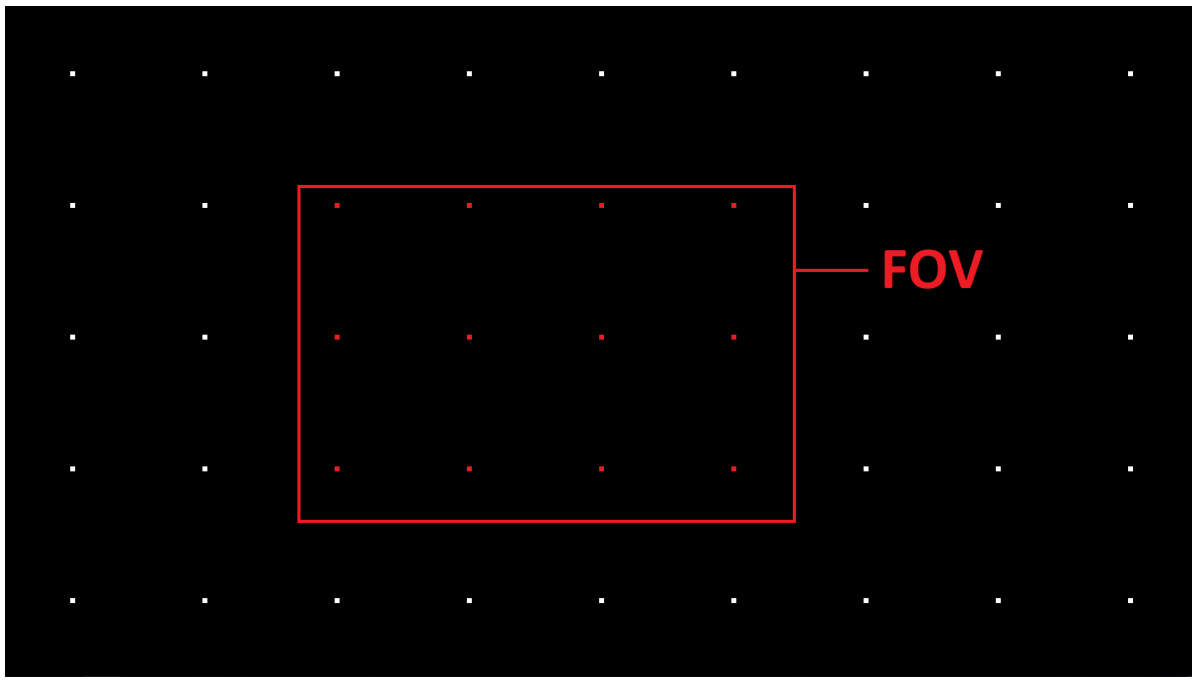
pixelSize = 1:



pixelSize = 8:



Center point estimation



$$\mathbf{x}_c = \frac{\sum_{k=1}^{n} \mathbf{x}_k}{n}$$

n : number of seen pixels

$\mathbf{x}_k$: position of seen pixel

Runtime

TECHNISCHE UNIVERSITÄT DARMSTADT

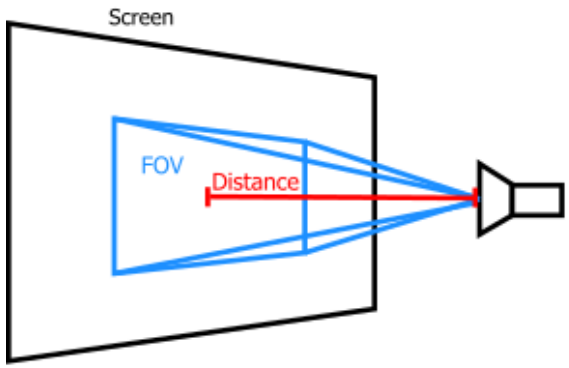
- Map gaining:
  - Depends on approach used
  - Usual dependencies
    - # images
    - Field of View (FOV)
    - Turnover rate of images (3 images per seconds)
    - Framerate (21 frames per second)
- Interpolate Map $<$ 100 *ms*
- Correct distortion $<$ 100 *ms*

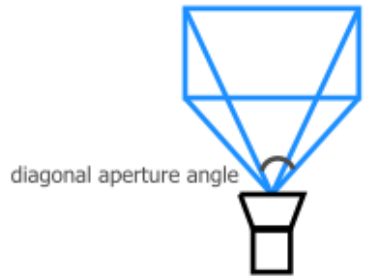
Runtime

FOV

Received Info:

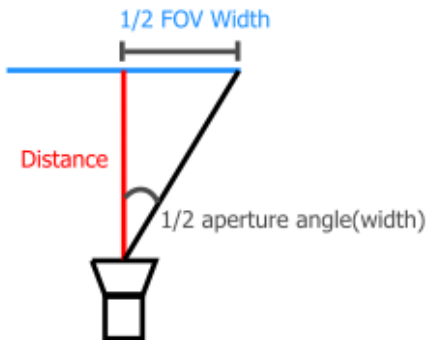
- $\alpha_{width||height}$
- α_{diag}

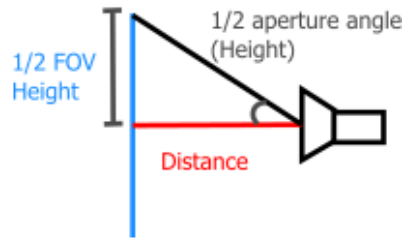




Our FOV:

- 670x395 pixels
- α_{diag} = 68.46°
- camera specs:
  $\alpha_{diag,s}$ = 68.5°





Runtime

Line-based

TECHNISCHE UNIVERSITÄT DARMSTADT

naive:

$$rt(w_s, h_s) = \frac{1}{3}(w_s + h_s) = 790sec$$

w_s : width of screen

h_s : height of screen

w_{FOV} : wdith of FOV

h_{FOV} : height of FOV

Runtime

Line-based

TECHNISCHE UNIVERSITÄT DARMSTADT

Center point & FOV based:

1. $rt_{cp}(w_s, h_s, s) = \frac{w_s \cdot h_s}{3s^2}$

2. $rt_{map}(w_{FOV}, h_{FOV}, s, m, j) = \frac{1}{3j}\,(w_{FOV} + 2s + 2m + h_{FOV} + 2s + 2m)$

3. $rt\,(w_s, h_s, w_{FOV}, h_{FOV}, s, m, j) = \frac{1}{3}\left((w_{FOV} + 4s + 4m + h_{FOV})\frac{1}{j} + \frac{w_s \cdot h_s}{s^2}\right)$

s : space between lit pixels in center point estimation
m : safety margin
j : skip range

Runtime

Line-based

TECHNISCHE UNIVERSITÄT DARMSTADT

Find optimum

$$rt'(s) = -\frac{w_s h_s}{3s^3} + \frac{4}{3j} \stackrel{!}{=} 0$$

$$s = \sqrt[3]{\frac{w_s \cdot h_s \cdot j}{4}}$$

For our Setup

$j = 1$:

$s = 70 \quad \Rightarrow \quad rt = 550$

$j = 3$:

$s = 101 \quad \Rightarrow \quad rt = 211$

Overview implementation

UML diagramm

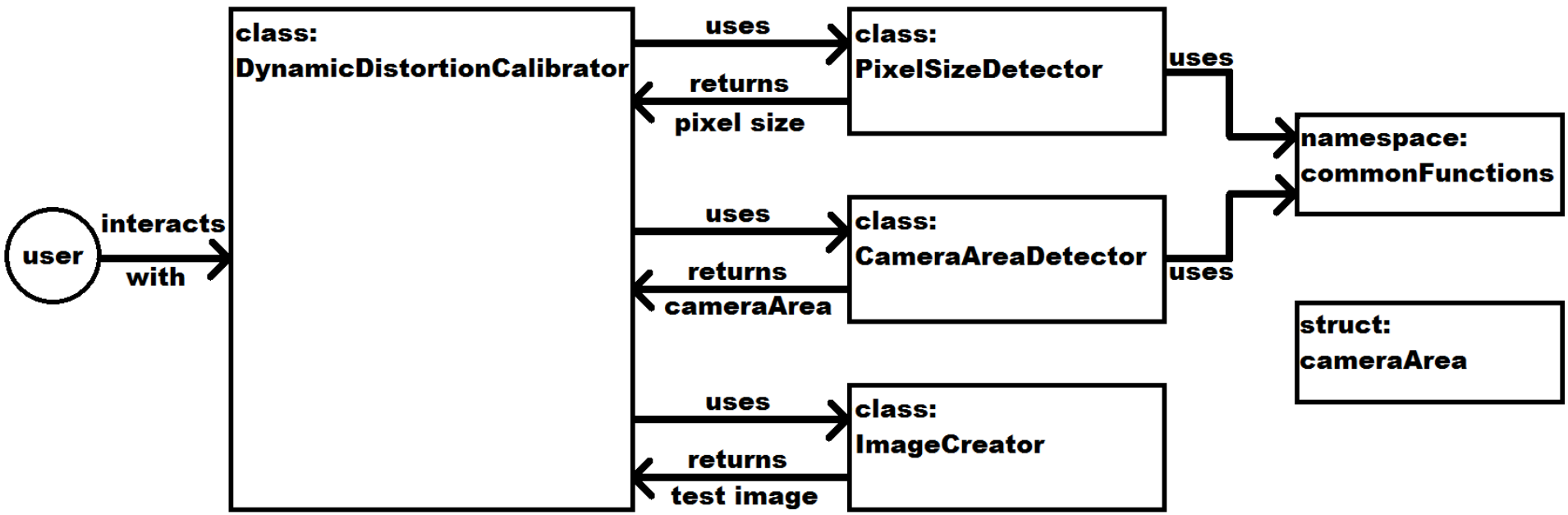


Figure: overview of classes, structs, and namespaces

Results

TECHNISCHE UNIVERSITÄT DARMSTADT

Ground truth:

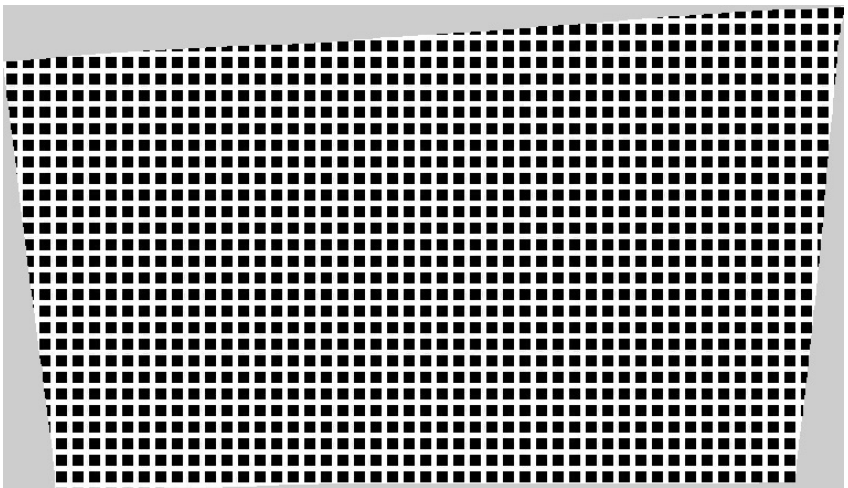


Figure: all white lines in FOV

Mapped Image:

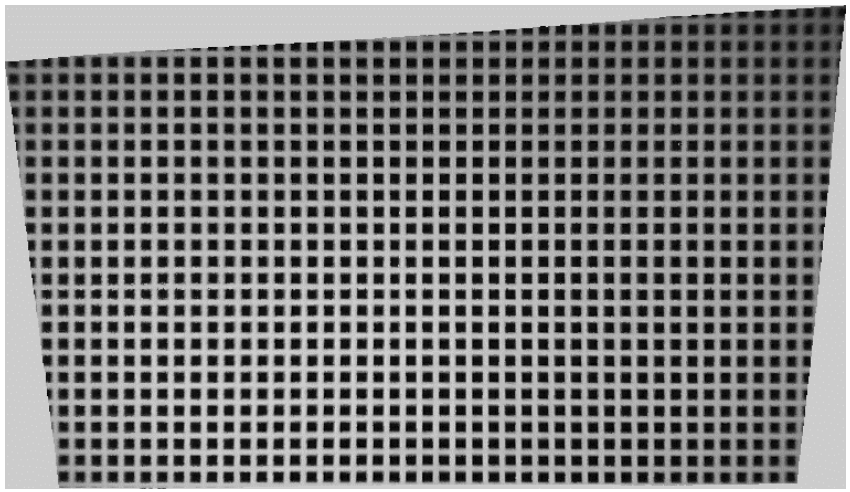


Figure: the seen lines after they were mapped by the algorithm

Comparison

Subtraction of ground truth and mapped image

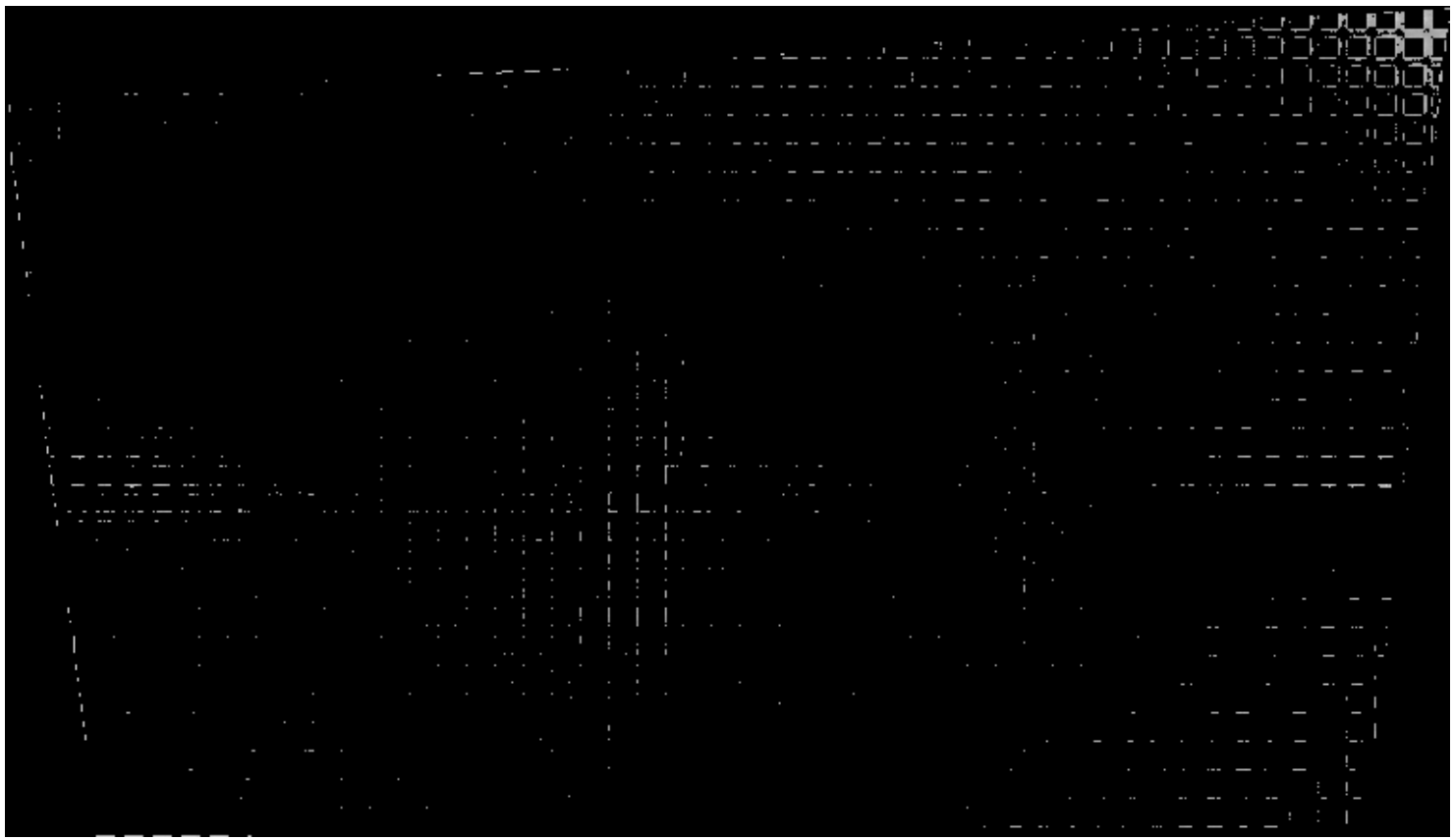


Figure: difference of both images
2,983 of 333,756 pixels (0.89%) do not fit

Future Work

TECHNISCHE UNIVERSITÄT DARMSTADT

- Run mapping in one step with *openframeworks*-version
- detect white threshold
- improvement of line detection $\Rightarrow$ line counting?
- avoid implausible mapping
- undo statemachine, if internal *openframeworks*-timing works

The End

TECHNISCHE UNIVERSITÄT DARMSTADT

Thank you all for your attention!
Are there any questions?

Sources

- Computerphile
  https://www.youtube.com/watch?v=uihBwtPIBxM 15.07.2017.
  https://www.youtube.com/watch?v=sRFM5IEqR2w 15.07.2017.
- B. Ribbens, V.A. Jacobs, C. Vuye1, J.A.N. Buytaert, J.J.J. Dirckx, and S. Vanlanduit. *High-resolution temporal profilometry using Fourier estimation*, 2007
- S. Mannuru. *A fully automated geometric lens distortion correction method*, University of Dayton, Ohio 2011
- J. P. de Villiers, F.W. Leuschnerb, R. Geldenhuys. *Centi-pixel accurate real-time inverse distortion corretion*, University of Pretoria, South Africa 2008