Titel

Namen

TECHNISCHE UNIVERSITÄT DARMSTADT

Distortion example

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

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

Abbildung: Formula to correct tangential and radial distortion

Radial distortion:

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

Tangential distortion:

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

$$\begin{aligned}
(x_d, y_d) &= \text{ distorted image point as projected on image plane,} \\
(x, y) &= \text{ undistorted image point as projected on image plane,} \\
(x_c, y_c) &= \text{ distortion center,} \\
r &= \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2},
\end{aligned}$$

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:

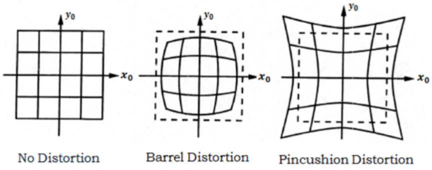


Abbildung: radial distortions

Tangential distortion:

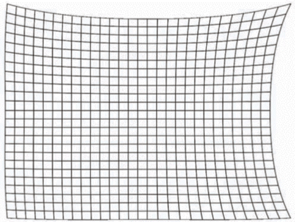
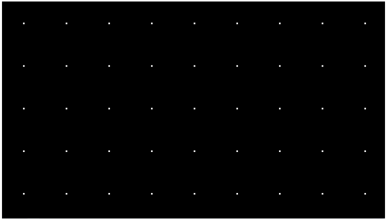


Abbildung: first order tangential distortion

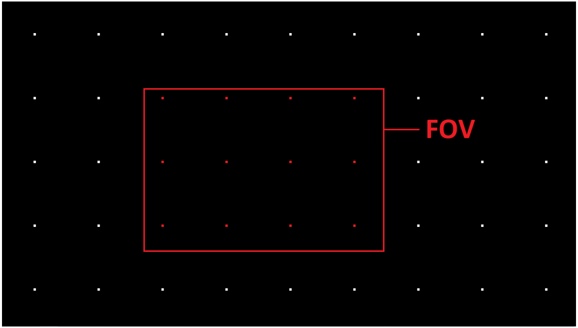
pixel size detection

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

Results

Ground truth:

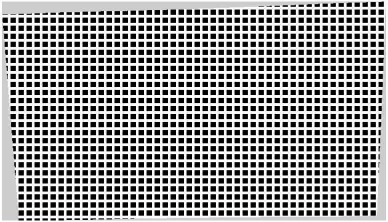


Abbildung: all white lines that we're drawn and seen on the screen

Mapped Image:

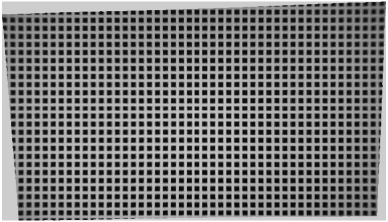


Abbildung: the seen lines after they were mapped by the algorithm

Comparison

Substraction of ground truth and mapped image

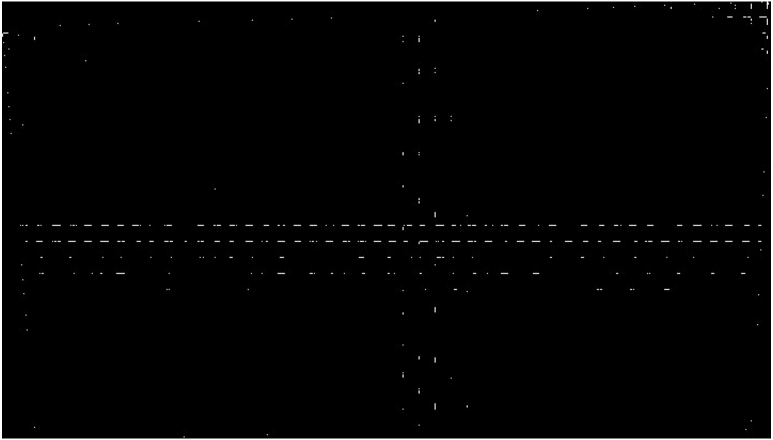
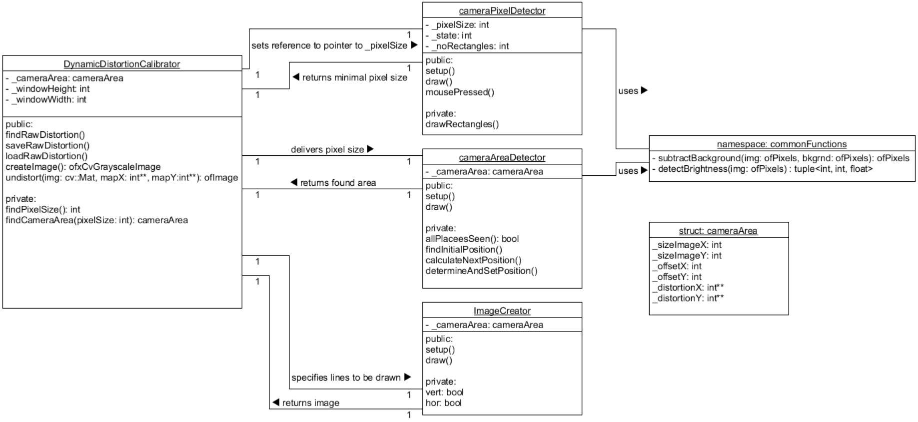


Abbildung: difference of both images

824 of 290,191 pixels do not fit

Overview implementation

UML diagramm



Runtime

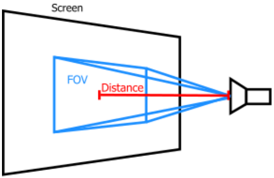
- Map gaining:
  - Depends on approach used
  - Usual dependencies
    - # images
    - Field of View (FOV)
    - Turnover rate of images (3 images per seconds)
    - Framerate (21 frames per second)
- Interpolate Map < 100 *ms*
- Correct distortion < 100 *ms*

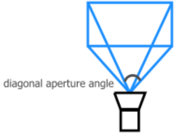
Runtime

FOV

Received Info:

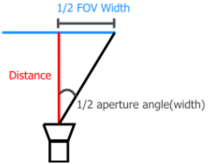
- $\alpha_{width||height}$
- α_{diag}

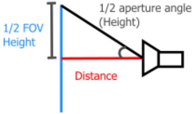




Our FOV:

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





TECHNISCHE UNIVERSITÄT DARMSTADT

Runtime

Line-based

naive:

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

w_s : width of screen

h_s : height of screen

w_{FOV} : wdith of FOV

h_{FOV} : height of FOV

Runtime

Line-based

Center point & FOV based:

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

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

Runtime

Line-based

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

$j = 1$**:**
$s = 70 \quad \Rightarrow \quad rt = 550$

$j = 3$**:**
$s = 101 \quad \Rightarrow \quad rt = 211$

Future Work

- Run mapping in one step with *openframeworks*-version
- detect white threshold
- improvement of line detection ⇒ line counting?
- solve flipping problems seen in substracted picture
- unplausible found mappings
- undo statemachine if internal *openframeworks*-timing works