

Tracking und Computer Vision für Augmented Reality

Thomas Neumann

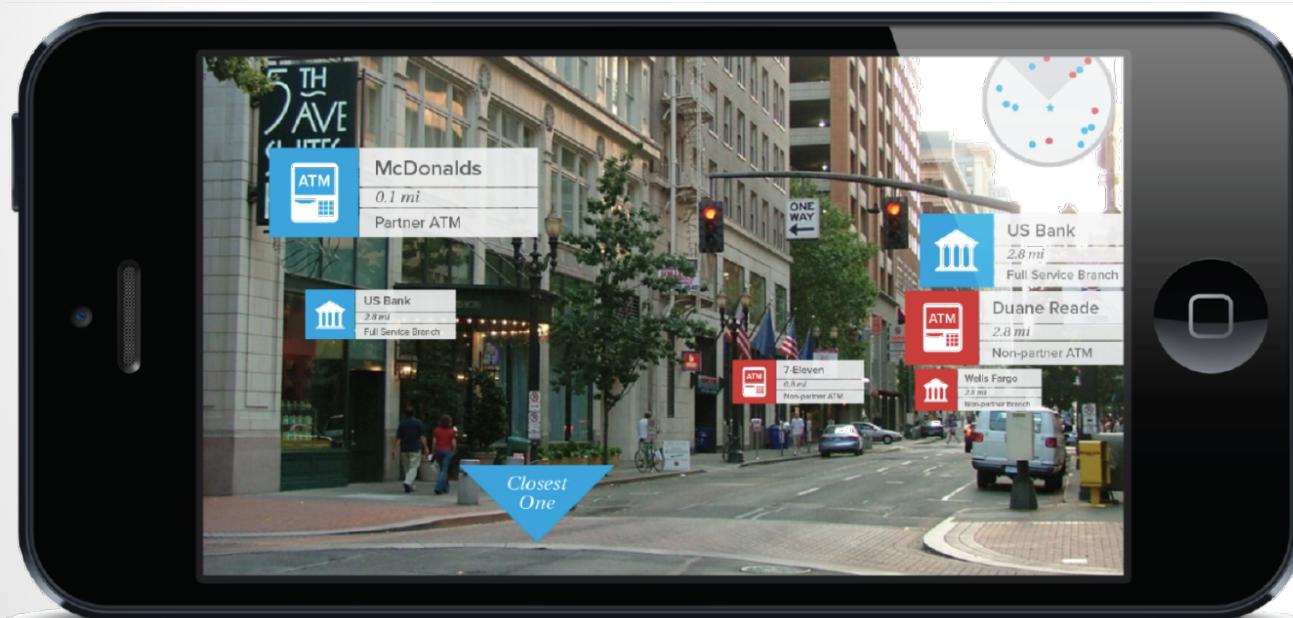
tneumann@htw-dresden.de

Übersicht

- Aufgabe des Trackings
- Tracking von Markern
- Grundlagen Kameratransformation
- Anwendung: Live-Augmentierung mit OpenGL
- Markerloses Tracking
 - Feature-Detector, Feature-Descriptor
 - Übersicht weitergehende Verfahren

Aufgabe des Trackings in AR

- Reale Umgebung erfassen:
 1. Blickwinkel + Position des Betrachters
z.B. einer Kamera im Raum
 2. Objekte, deren Lage (z.B. aus einer Kamera mittels Marker)



1: Wo bin ich? Wo schau ich hin?



2: Wo sind best. Objekte?

- Muss in Echtzeit funktionieren!

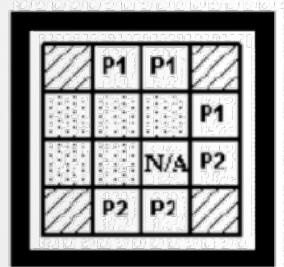
Möglichkeiten des Trackings

- Nichtvisuelles Tracking
 - Kompass (über „Magnetometer“ bzw. “geomagnetic sensor”)
 - Trägheitssensoren
(Neigung über Gyroskop, Bewegung durch Beschleunigungssensor)
 - GPS (Bestimmung einer ungefähren Position auf der Erdkugel)
 - Optoelektrische Sensoren (z.B. Lage zu Sendestation über Infrarot)
 - Ultraschallsensoren (Laufzeitmessung zu Sendestationen)
 - RFID (aktiv/passiv)
- Visuelles Tracking – direkt aus dem Kamerabild
 - Über künstliche Marker / Muster / Fiducials
 - Markerlos - „die Umgebung ist der Marker“
- Kombinierte Verfahren

Künstliche Marker – AR Marker



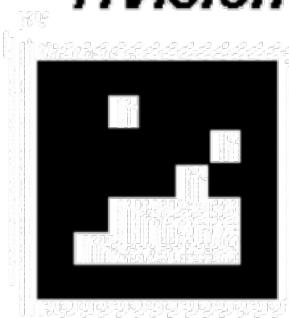
Intersense



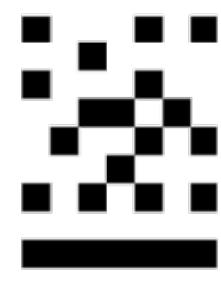
**Binary
Square
Marker**



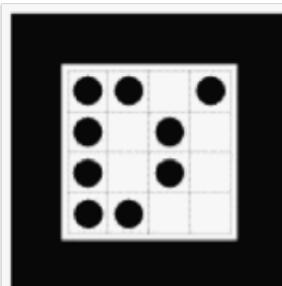
**Reac-
TIVision**



Matrix



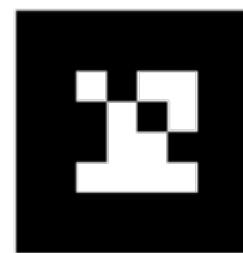
**Cyber-
code**



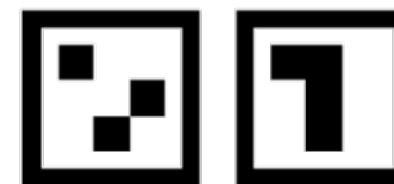
SCR



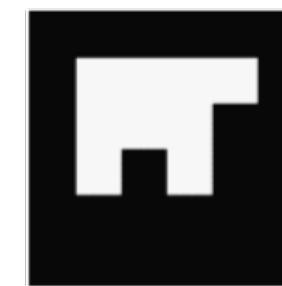
**Visual
Code**



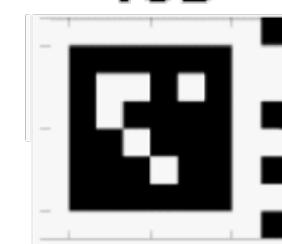
BinARyID



Canon Markers



IGD



HOM



ARToolkit

(2 examples)

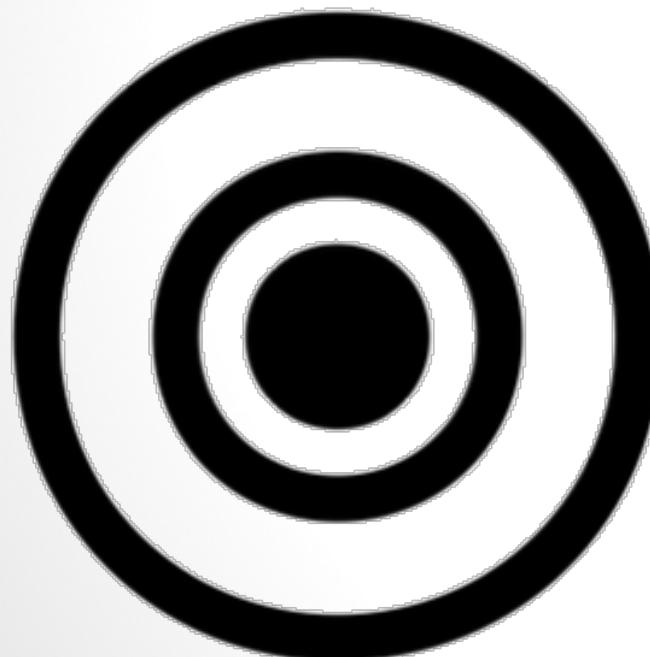


ARTag

(2 examples)

Anforderungen an AR Marker

- Einfach & Automatisch zu erkennen in Bildern
- Für 3D-Augmentierung: min. 4 Punkte
- Nicht verwechselbar (keine Fehldetektion)
- Der Marker muss identifizierbar sein (bzw. Daten tragen)



0 bits

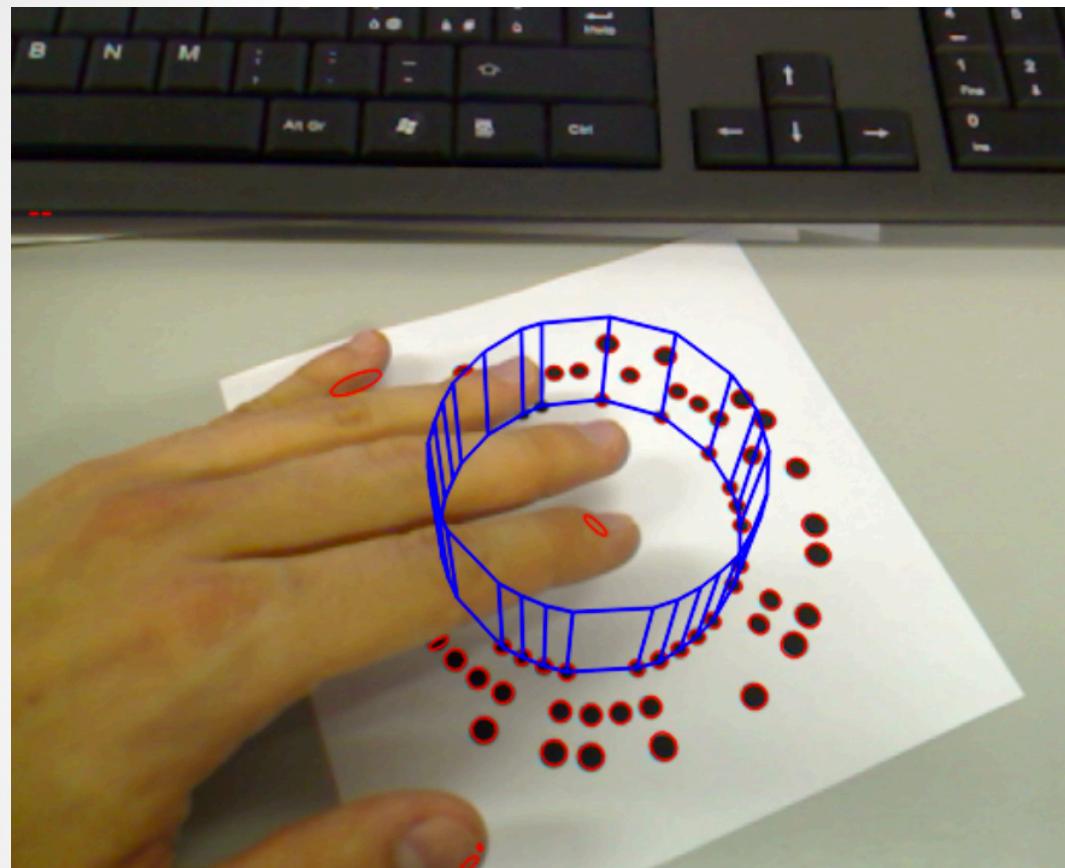


ARTag
10 bits



QR Code
Tausende Bits
(je nach Größe)

Anwendungsspezifische Anforderungen



Kann der Marker partiell
verdeckt sein?
→ RuneTAG

Ist der Marker von
Menschen lesbar?
→ ARToolkit

Hands-On: Wie detektiere ich einen ARTag Marker

- Beispiel-Code in Python
- prototypisch in einem Jupyter Notebook umgesetzt
- Eigene Installation:
 - Anaconda (enthält numpy, matplotlib, scipy, jupyter, ipython), ich nutze Anaconda für Python 2.7
 - zusätzlich installieren: `conda install -c menpo opencv=2.4.11`

Erinnerung: homogene Koordinaten

Kartesische Koordinaten

$$\hat{\mathbf{p}} = \begin{bmatrix} \hat{x} \\ \hat{y} \end{bmatrix} \quad \hat{\mathbf{p}} \in \mathbb{R}^2$$

Homogene Koordinaten

$$\mathbf{p} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} \quad \mathbf{p} \in \mathbb{P}^2$$

Umwandlung

Kartesische \rightarrow Homogene
durch „Eins Hinzufügen“

$$\mathbf{p} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix}$$

Umwandlung

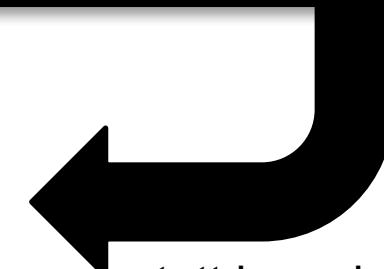
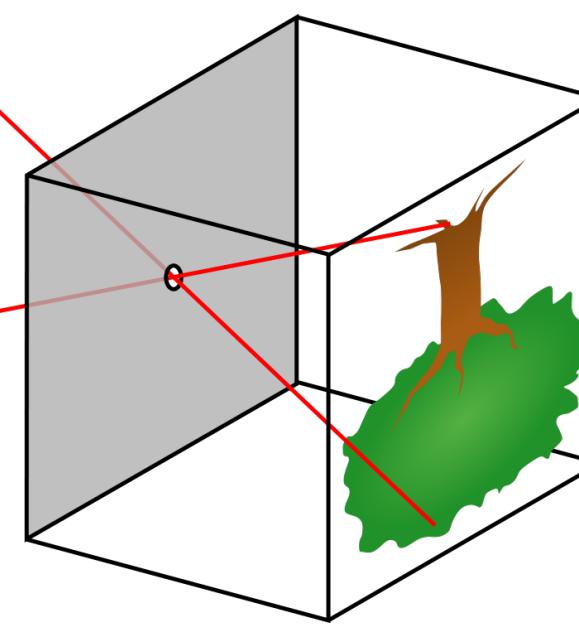
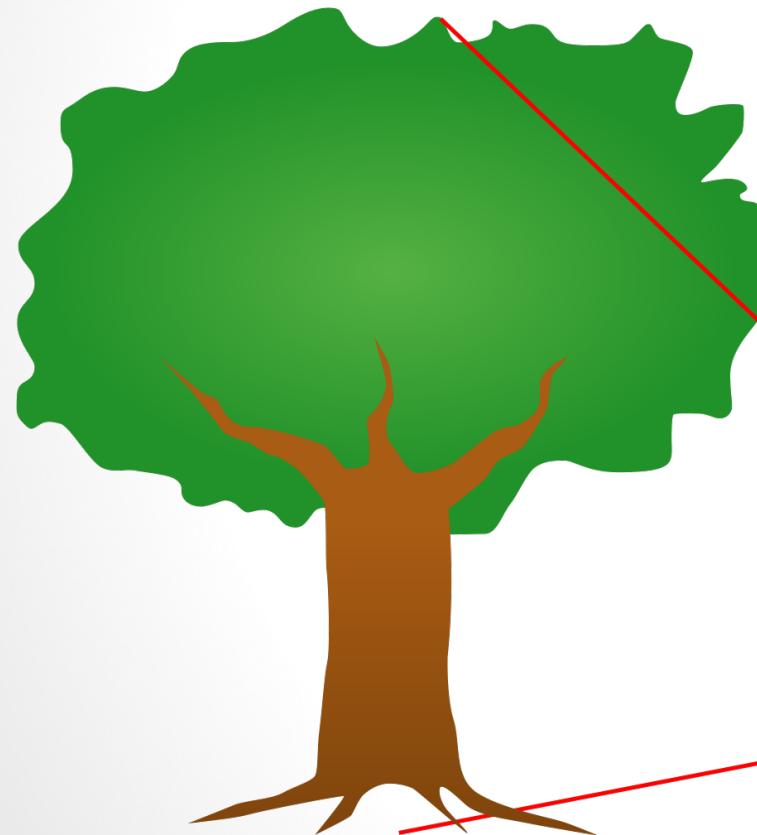
Homogene \rightarrow Kartesische
durch “Normalisierung”:

$$\hat{\mathbf{p}} = \begin{bmatrix} \left(\frac{x}{w}\right) \\ \left(\frac{y}{w}\right) \end{bmatrix}$$

Kamera: wandelt 3D in 2D Information



Vereinfachung: Lochkamera



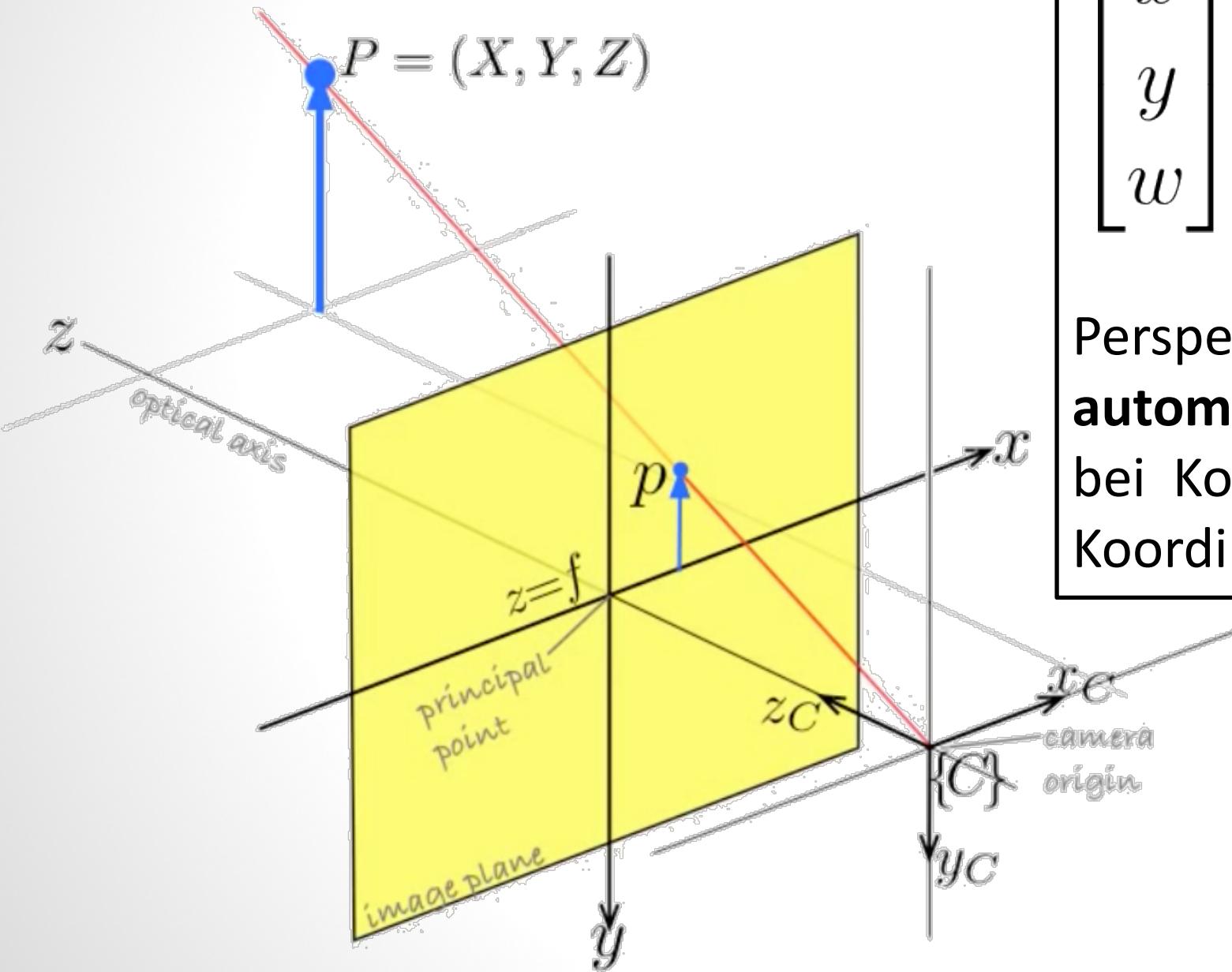
statt komplexe Linsensysteme
mathematisch zu beschreiben...

Zentralprojektion (homogene Koordinaten)

Darstellung in Matrixform:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Perspektivische Division passiert **automatisch** durch Normalisierung bei Konvertierung zu kartesischen Koordinaten



Konvertierung in Pixelkoordinaten

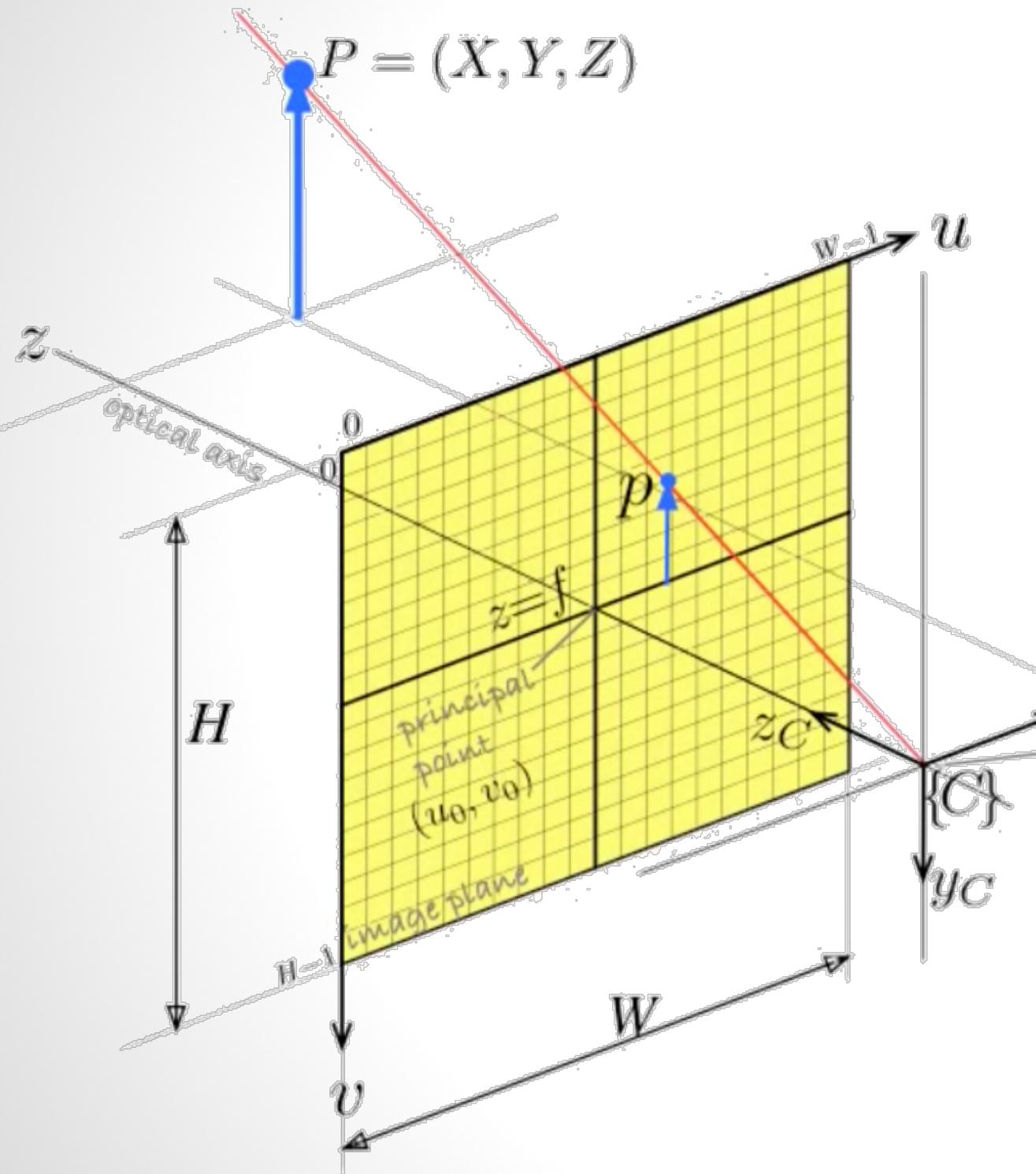


Image by Peter Corke - robotacademy.net.au

Darstellung in Matrixform:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & 0 & c_x \\ 0 & \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

K ist die **intrinsische Kameramatrix**.
Oben vereinfacht, generell aber:

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

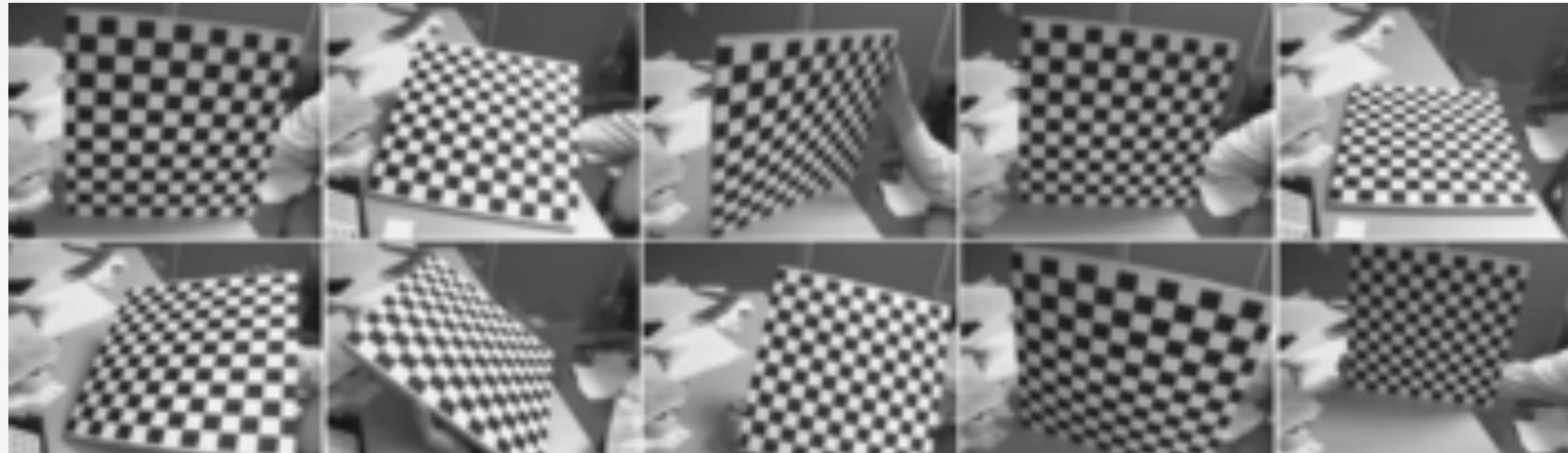
α_x, α_y ... Brennweite in Pixel

c_x, c_y ... Bildzentrum (principal point)

s ... Pixelschiefe (skew)

Intrinsische Kameramatrix K ermittelt durch:

- Bekanntes Kameramodell (nicht genau!)
- Kamerakalibrierung – mehrere Aufnahmen einer Ebene



- Auto-Calibration
 - Structure from Motion (SfM)
 - Simultaneous Localization and Mapping (SLAM)

Extrinsische Kameramatrix

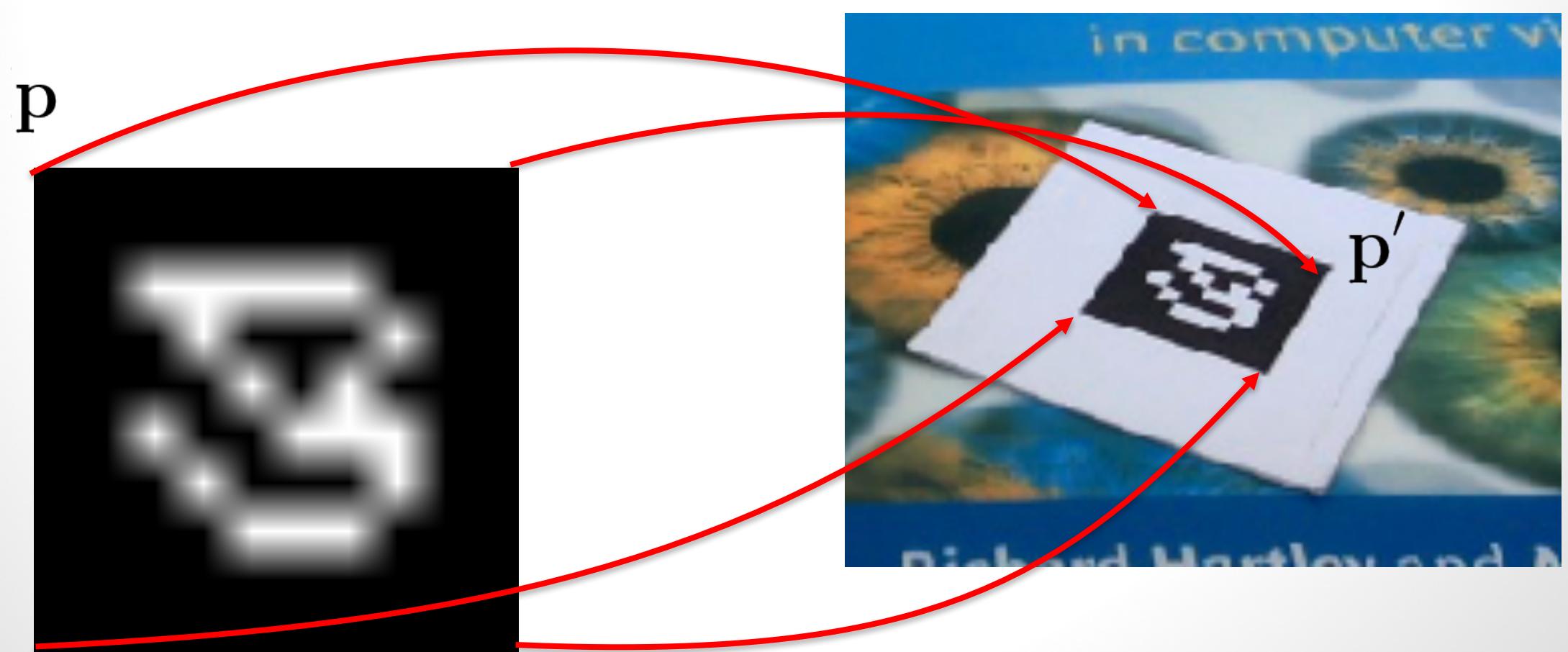
- Rotation und Verschiebung des Koordinatensystems durch die extrinsischen Kameramatrix \mathbf{M} :

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & 0 & c_x \\ 0 & \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_z \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\mathbf{P}}$$

- Komplette Abbildung durch Projektionsmatrix \mathbf{P}

Projektion einer Ebene = Homographie

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & 0 & c_x \\ 0 & \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_z \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \mathbf{p}' = \mathbf{H}\mathbf{p}$$

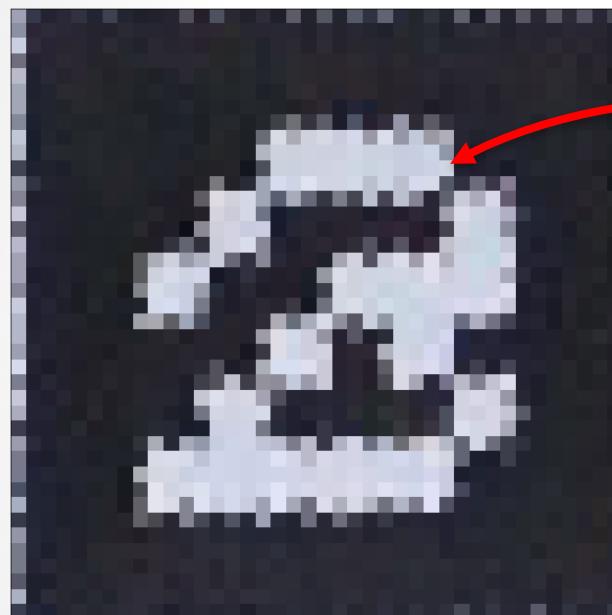


Finden einer Homographie: DLT (Direct Linear Transform)

- Bestimmung der Homographie \mathbf{H} über OpenCV:
`cv2.findHomography(quellPunkte, zielPunkte)`
- Ansatz:
 - Umstellung der Gleichung: $\mathbf{p}' = \mathbf{H}\mathbf{p} \Leftrightarrow \mathbf{A}\mathbf{h} = \mathbf{0}$
(\mathbf{h} enthält Komponenten von \mathbf{H} , \mathbf{A} enthält die Punkte \mathbf{p} & \mathbf{p}')
 - GLS hat 9 Unbekannte, 4 Punkte (je 2 Koordinaten) = 8
Bekannte \rightarrow aber wegen homogenen Koordinaten ist $H_{33}=1$
 - Löse daher (mittels Singulärwertzerlegung SVD) ersatzweise
$$\min_{\mathbf{h}} \|\mathbf{A}\mathbf{h} - \mathbf{0}\|_2^2 \quad \text{subject to } \|\mathbf{h}\|_2^2 = 1$$
 - siehe auch Seiten 91 und 592f in „Multiple View Geometry in Computer Vision“ (Hartley & Zisserman)

Verifizierung des Markers

1. Jeden Pixel im Bild mit Homographie zu warpen



$$H^{-1}$$



2. Mit bekanntem Musterbild vergleichen

→ Demo

Homographie aus Punktkorrespondenzen bestimmen:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & 0 & c_x \\ 0 & \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}}_K \underbrace{\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_z \end{bmatrix}}_M \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} H_{1,1} & H_{1,2} & 0 & H_{1,3} \\ H_{2,1} & H_{2,2} & 0 & H_{2,3} \\ H_{3,1} & H_{3,2} & 0 & H_{3,3} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \rightarrow p' = Hp$$

über: `cv2.findHomography (quellPunkte, zielPunkte) -> H`

PnP-Problem um Kamerapose (bzw. Markerpose) zu bestimmen:

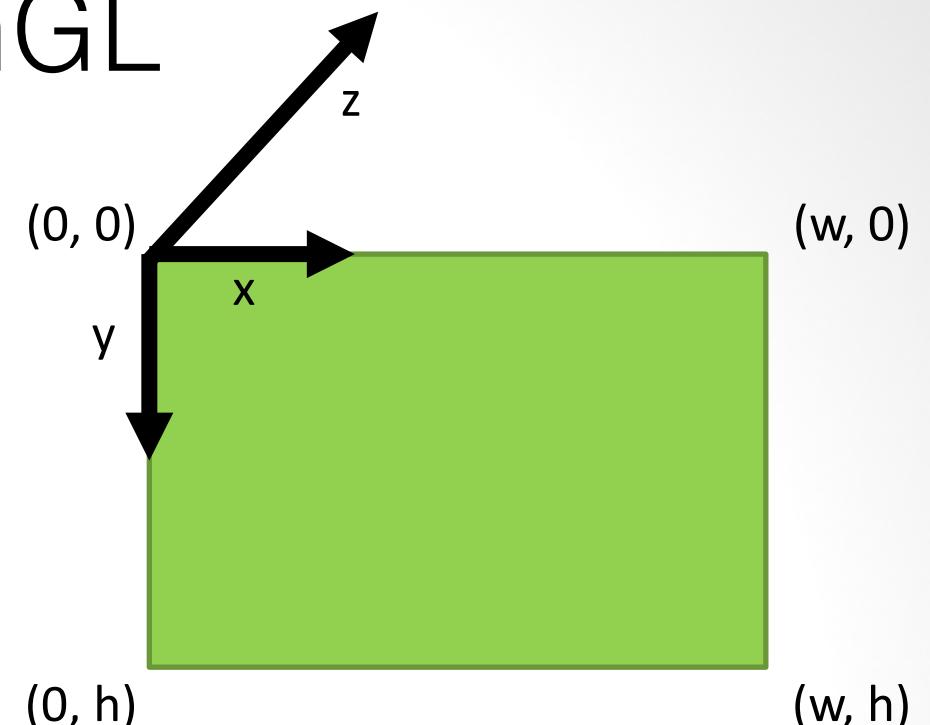
$$\begin{bmatrix} \hat{u} \\ \hat{v} \\ \hat{w} \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & 0 & c_x \\ 0 & \alpha & c_y \\ 0 & 0 & 1 \end{bmatrix}}_K \underbrace{\begin{bmatrix} R_{1,1} & R_{1,2} & R_{1,3} & t_x \\ R_{2,1} & R_{2,2} & R_{2,3} & t_y \\ R_{3,1} & R_{3,2} & R_{3,3} & t_z \end{bmatrix}}_M \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \rightarrow p' = K [R | t] p$$

über: `cv2.solvePnP (punkte3d, punkte2d, K) -> R, t`

Augmentierung mit OpenGL

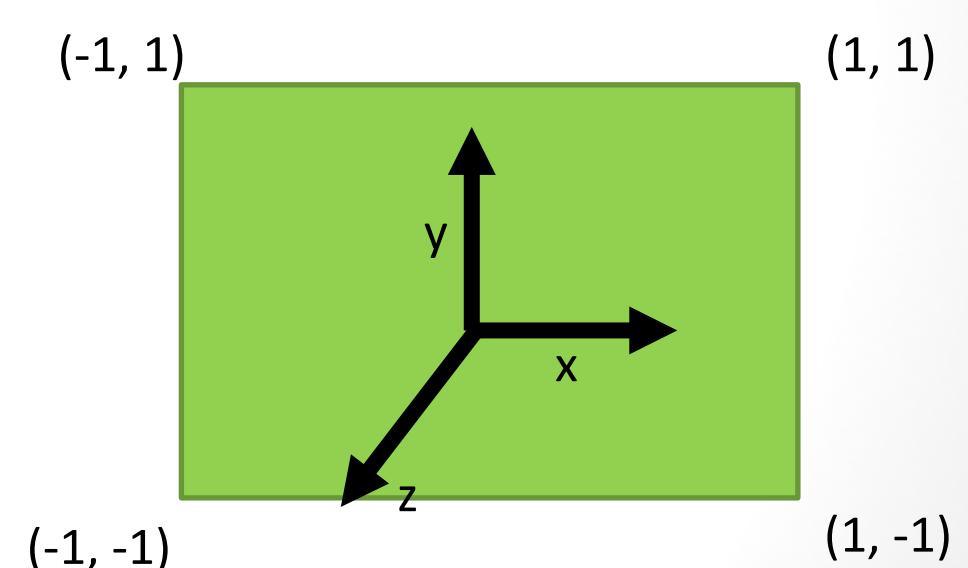
- Unser Koordinatensystem:

- Positive Z-Achse
 - x-Achse: 0 – Bildbreite
 - y-Achse: 0 – Bildhöhe

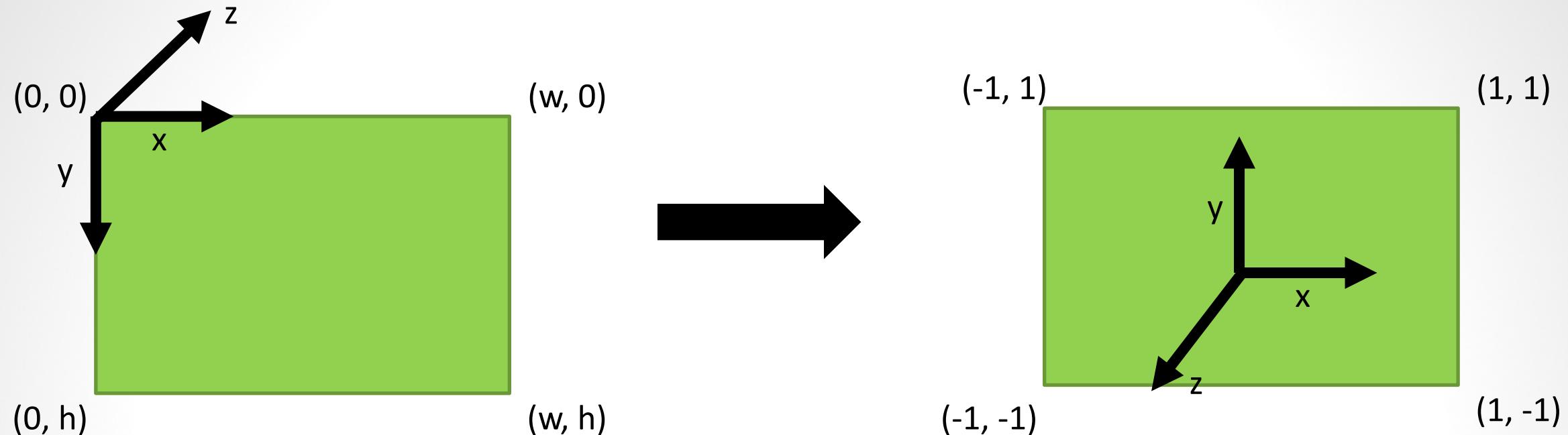


- OpenGL Koordinatensystem:

- Negative Z-Achse
 - x-Achse von -1 – 1
 - y-Achse von -1 – 1 (invertiert!)



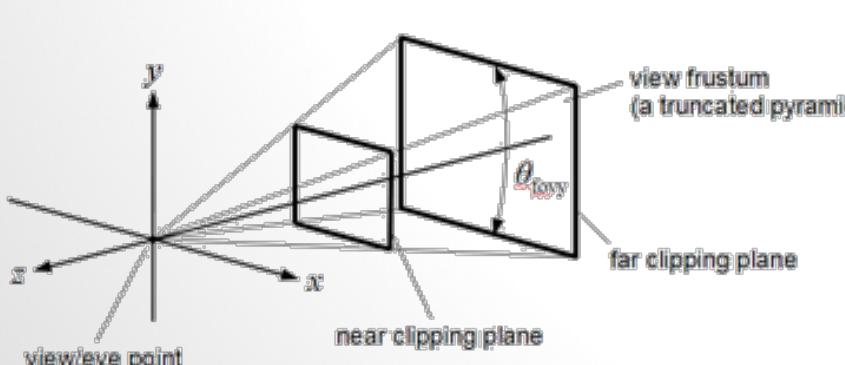
Koordinatentransformation zu OpenGL



Funktioniert nur wenn $c_x = \frac{w}{2}$

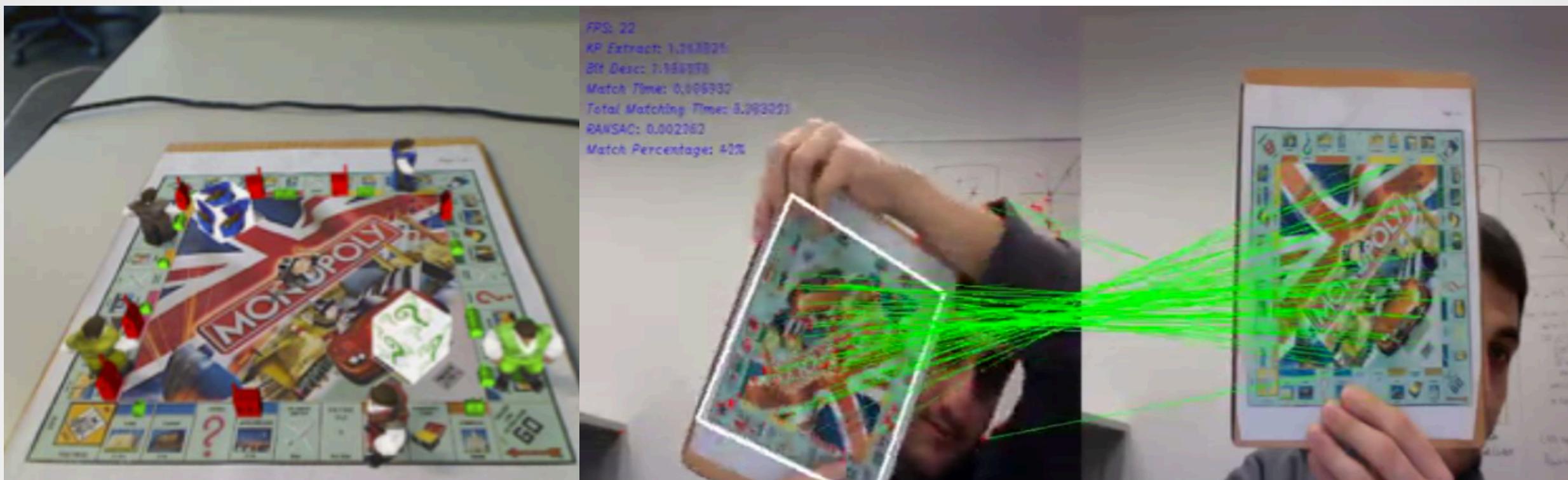
Y-Achse invertieren

$$\mathbf{K}_{\text{gl}} = \begin{bmatrix} \frac{f}{c_x} & 0 & 0 & 0 \\ 0 & -\frac{f}{c_y} & 0 & 0 \\ 0 & 0 & \frac{z_{\text{far}} + z_{\text{near}}}{z_{\text{far}} - z_{\text{near}}} & -\frac{2z_{\text{far}}z_{\text{near}}}{z_{\text{far}} - z_{\text{near}}} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



Z-Achse korrekt konvertieren und in „normalized device coordinates“ umwandeln

Markerless AR: Beispiel



Problem: Ausreichend viele Punkte auf dem Spielbrett müssen im Bild detektiert und zugeordnet werden. Wie finden wir solche Punkte automatisch?

→ Feature Points

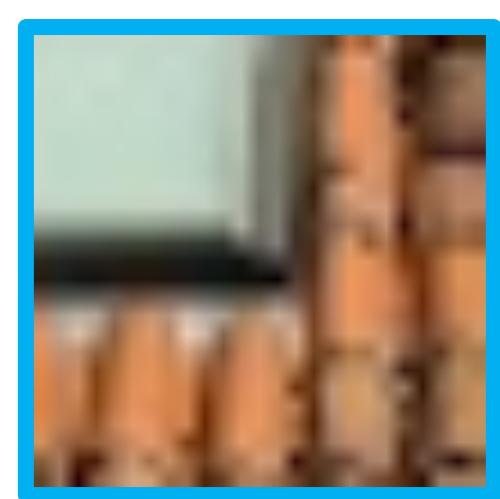
Welche Punkte im linken Bild sind gut im rechten Bild wiederzufinden?



1



2



3

Feature Detectors

... extrahieren aus einem Bild mehrere “interest points” bzw. „Featurepunkte“:

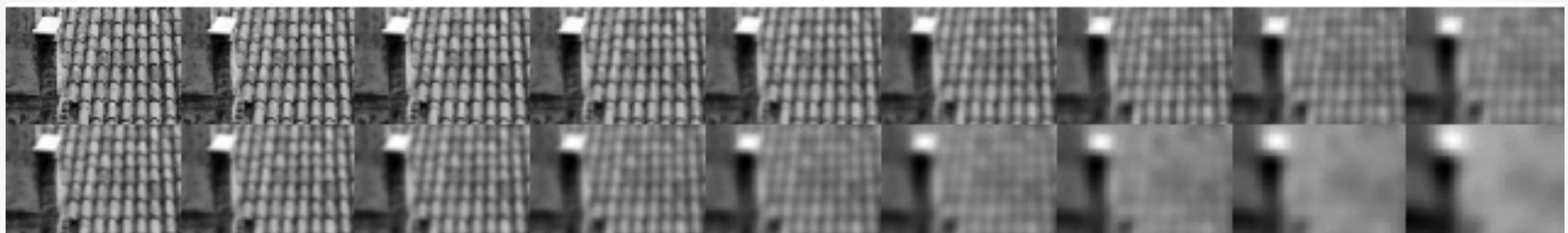
- unterscheidbar, auch wenn sich Blickpunkt / Licht ändert
- robust gegen Verdeckungen → lokale Regionen
- eindeutig identifizierbar
- invariant unter
Rotation & Skalierung



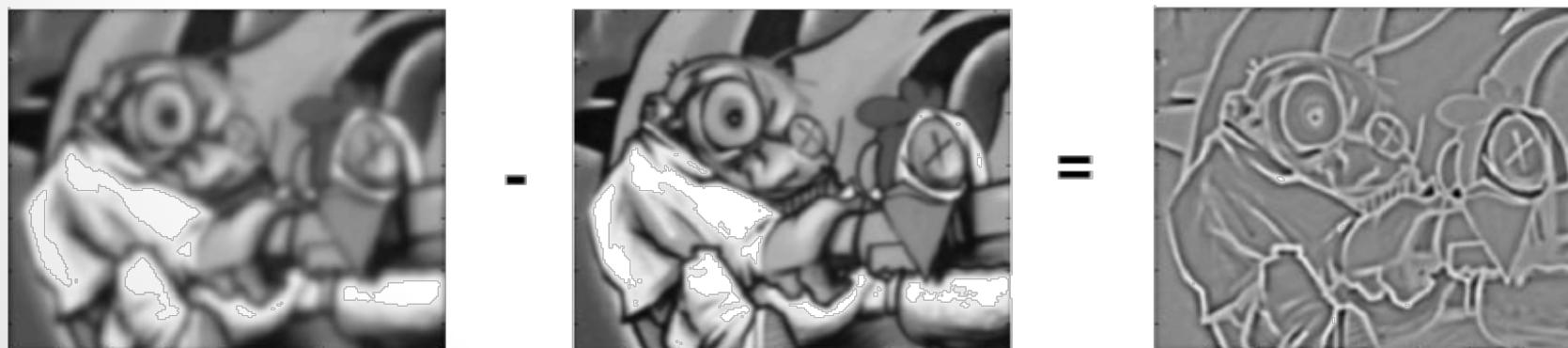
In einem Bild gefundene SIFT Featurepunkte

SIFT Detector 1/3

- Bild mit schrittweise größerem Gauss-Kern filtern (blurren) → Scale Space

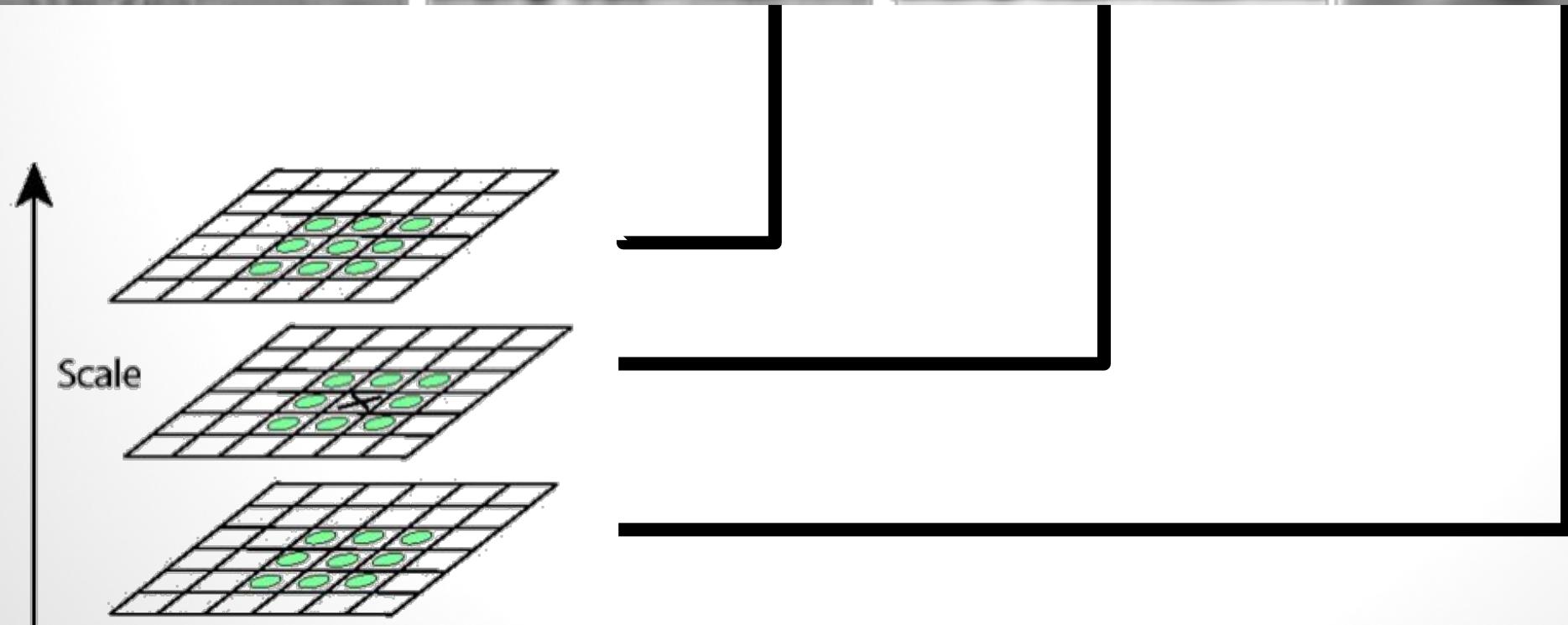
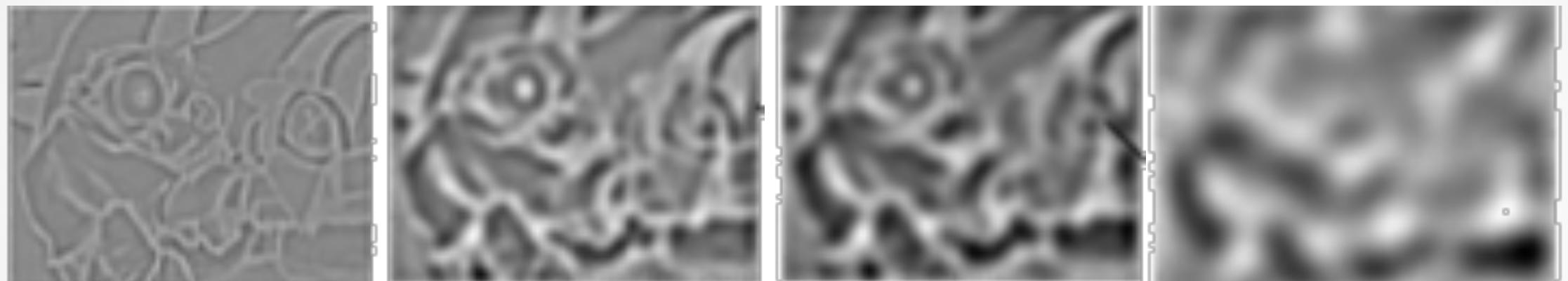


- Differenz der Gauss-gefilterten (DoG) springt gut auf Keypoints an



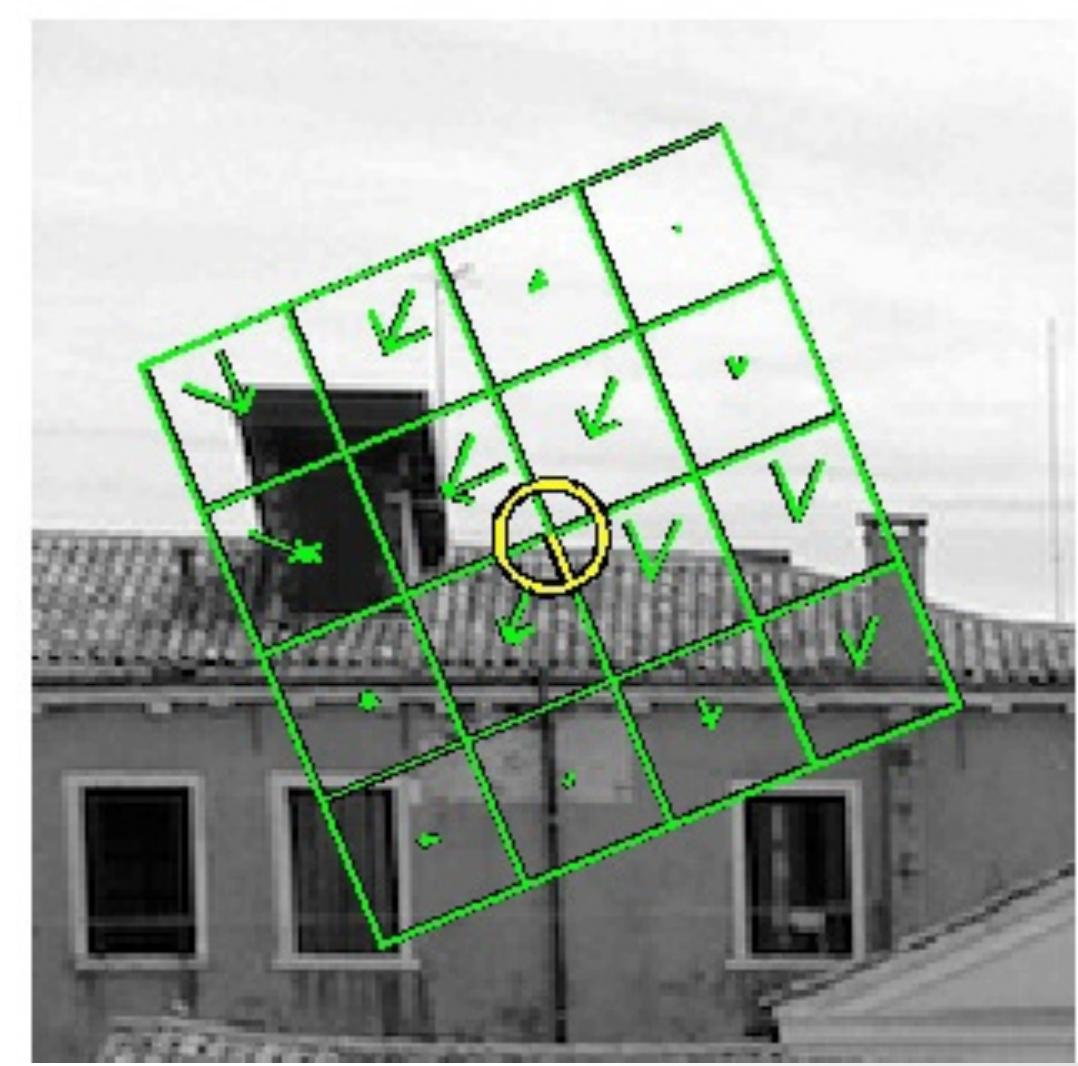
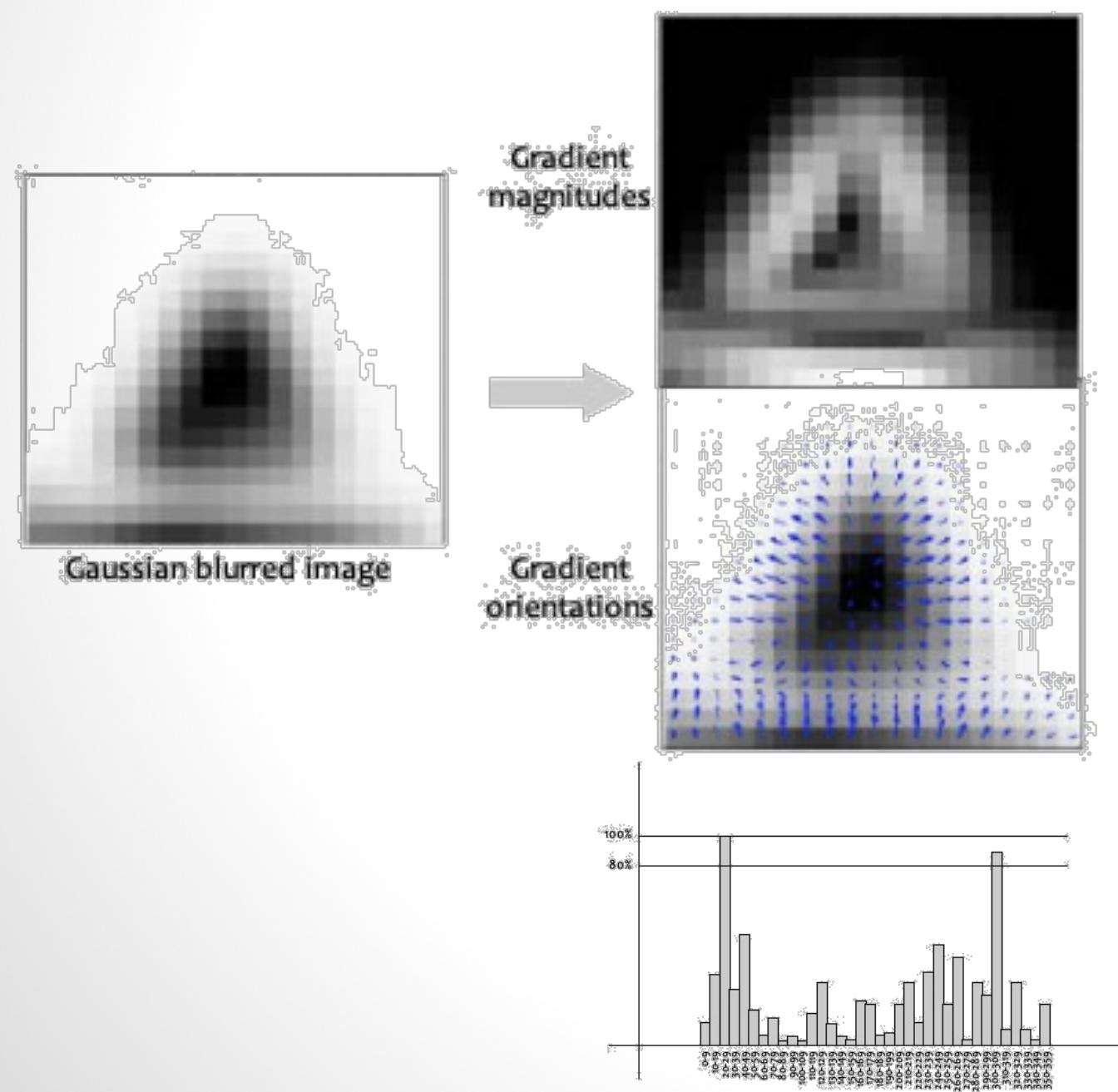
SIFT Detector 2/3

- Extrempunkte im Set der DoG Bilder suchen
(Scale-Space Maxima finden)

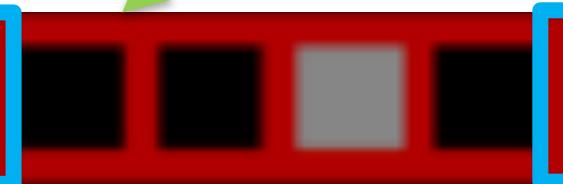
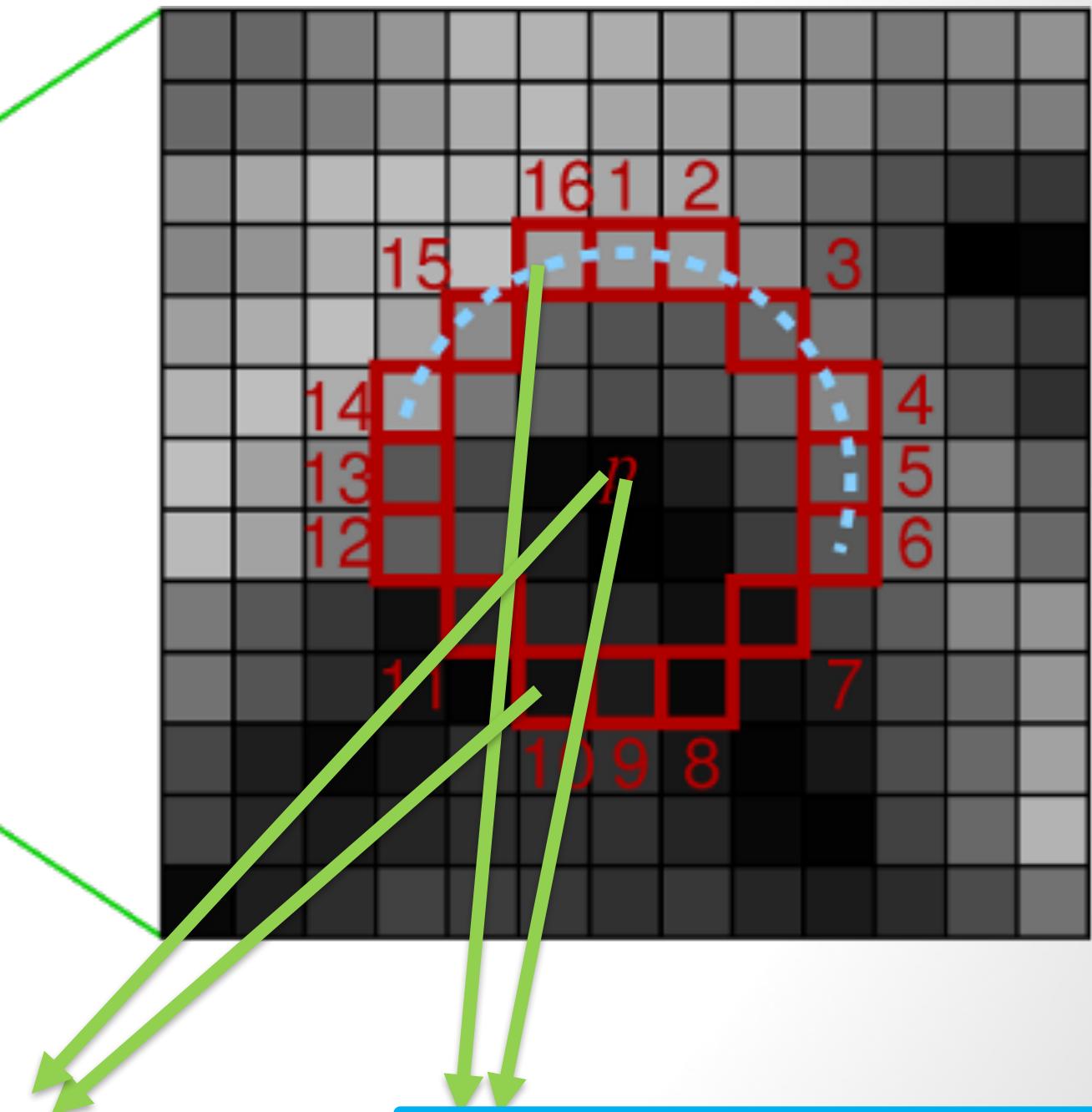
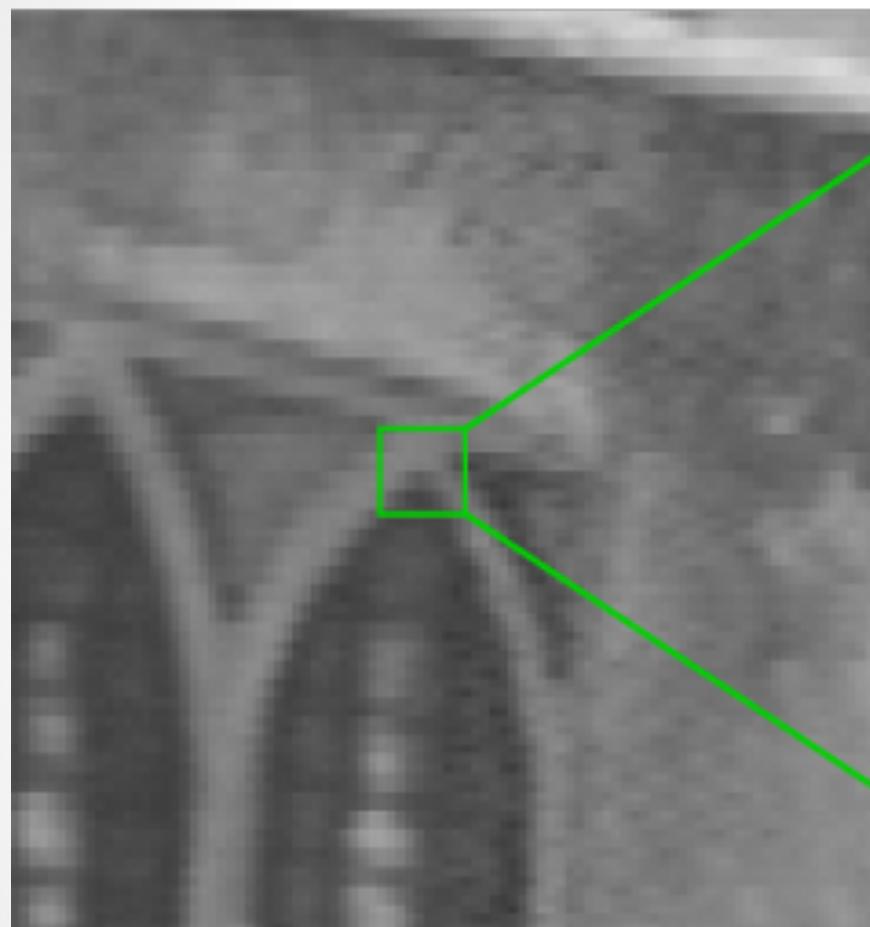


SIFT Detector 3/3

- Orientierung der Keypoints berechnen



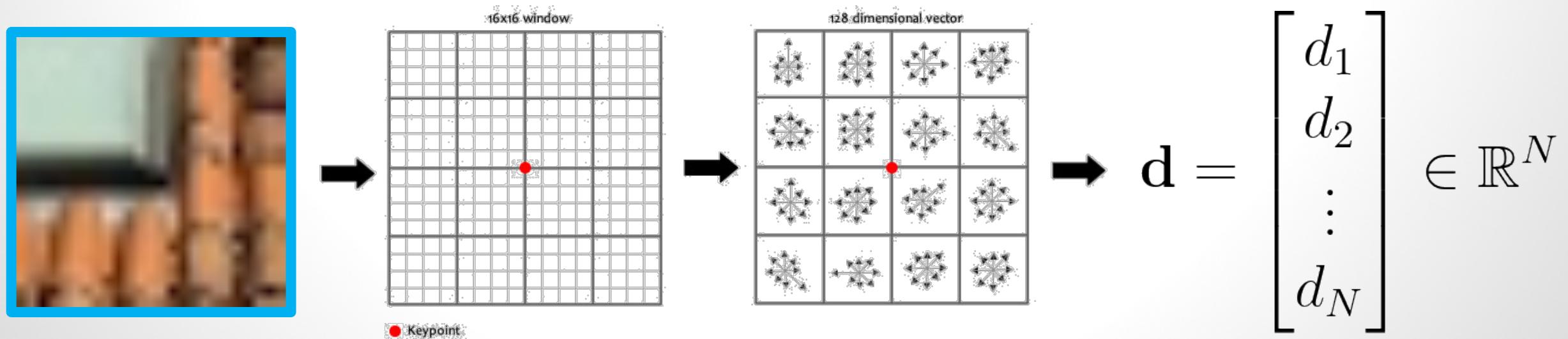
Beispiel: FAST Detector



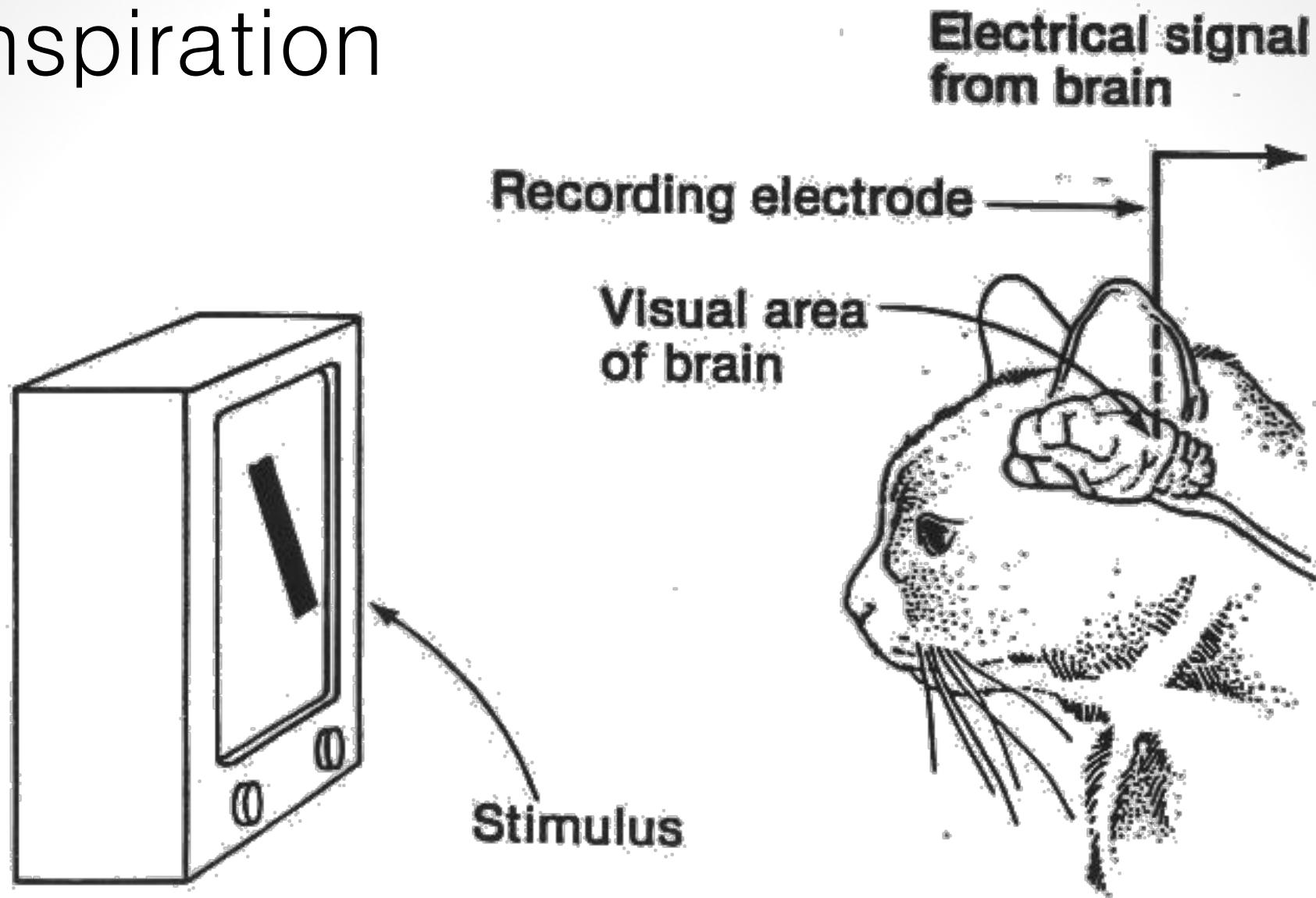
Feature Descriptors

- ... werden aus einer detektierten Bildregion berechnet und räpräsentieren die Bildintensität/-farbe in der Region.
- Sollten invariant sein unter: Blickwinkel, Beleuchtung

Beispiel: Ein SIFT Featuredeskriptor berechnet für eine Bildregion einen 128-dimensionalen Vektor. Ähnlich aussehende Patches resultieren in ähnliche Vektoren:



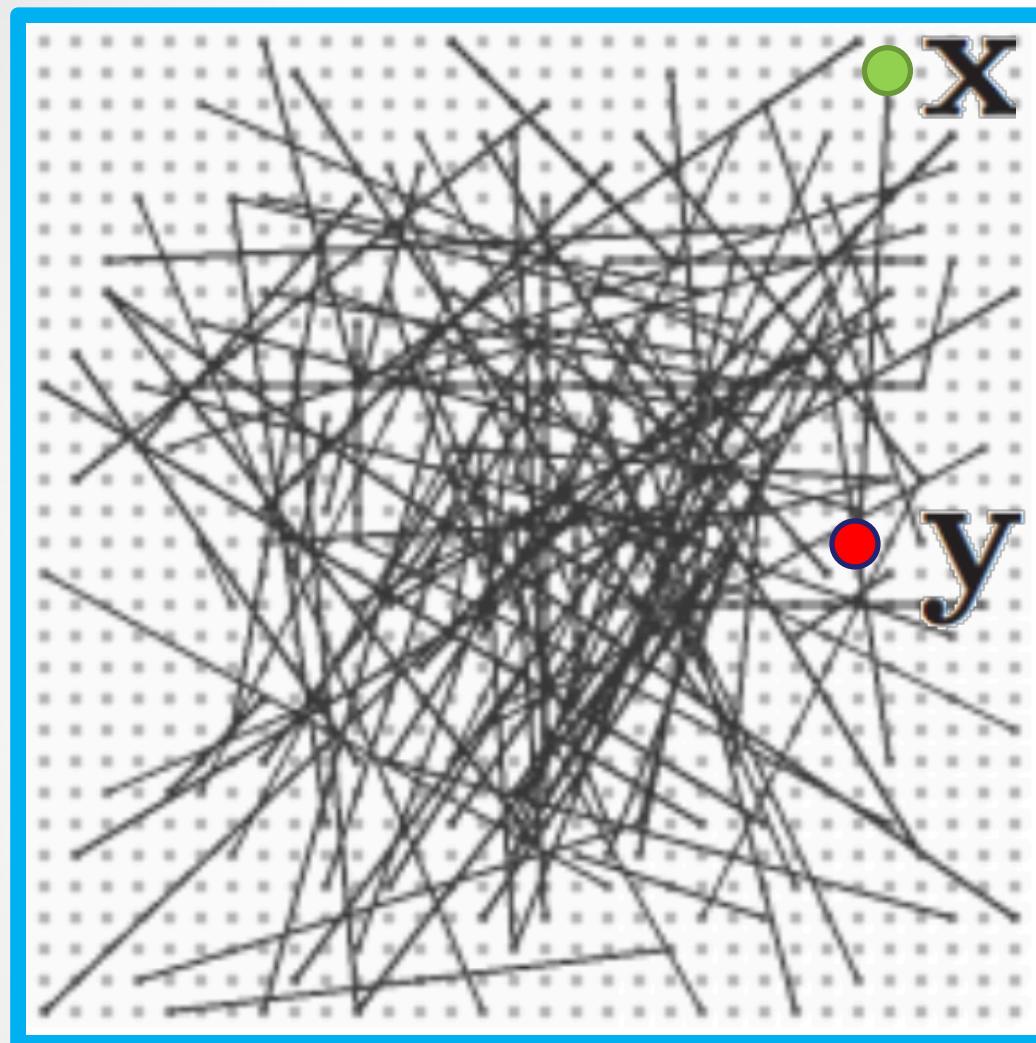
SIFT: Inspiration



1981 Nobelpreis für Physiologie oder Medizin
„für ihre Entdeckungen über Informationsverarbeitung
im Sehwahrnehmungssystem“

--> Zellen in V1 reagieren auf Ecken/Kanten mit best. Winkel

Beispiel BRIEF Descriptor



$$\tau(p; x, y) := \begin{cases} 1 & \text{if } p(x) < p(y) \\ 0 & \text{otherwise} \end{cases}$$



10010111010111011...110001

Bitstring als Descriptor
(Vergleich über Hamming-Distanz)

Matching zweier Bilder mittels Features

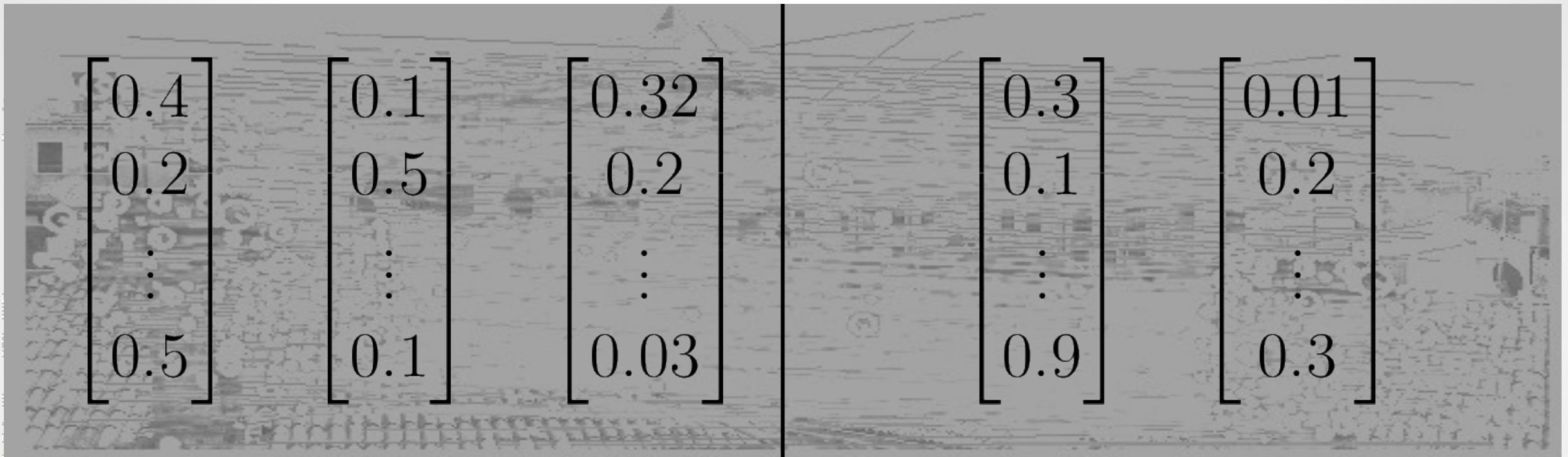


Bild 1

Bild 2

- Für jeden Feature-Descriptor in Bild 1, suche die nächsten Nachbarn in den Feature-Descriptoren in Bild2
 - evtl. über Suchbaum (KDTree, z.B. FLANN)
- Non-maxima Suppression: akzeptiere nur Matches mit viel näherem ersten als zweitnächsten Nachbar

Feature Detectors und Descriptors

Harris Corners & Shi-Tomasi Detector

(finde kleine Bildausschnitte die sich unter Verschiebung stark ändern)

MSER – Maximally Stable Extremal Regions

(keine Punkte, sondern Regionen)

FAST - Features from Accelerated Segment Test

(schnelle Pixelvergleiche um „corners“ zu finden. Machine Learning)

SIFT – Scale Invariant Feature Transform

(bekanntestes Verfahren. Langsam. Vom menschlichen Sehen inspiriert)

BRIEF - Binary Robust Independent Elementary Features

(schnelles Verfahren, Deskriptor ist eine Bitstring)

ORB – Oriented FAST and Rotated BRIEF

(sehr schnelle Kombination aus FAST und BRIEF)

LIFT – Learned Invariant Feature Transform

(neu & hip: großes Neuronales Netz, über Deep Learning trainiert)

LIFT: Learned Invariant Feature Transform

K.M. Yi, E. Trulls, V. Lepetit, P. Fua
ECCV 2016

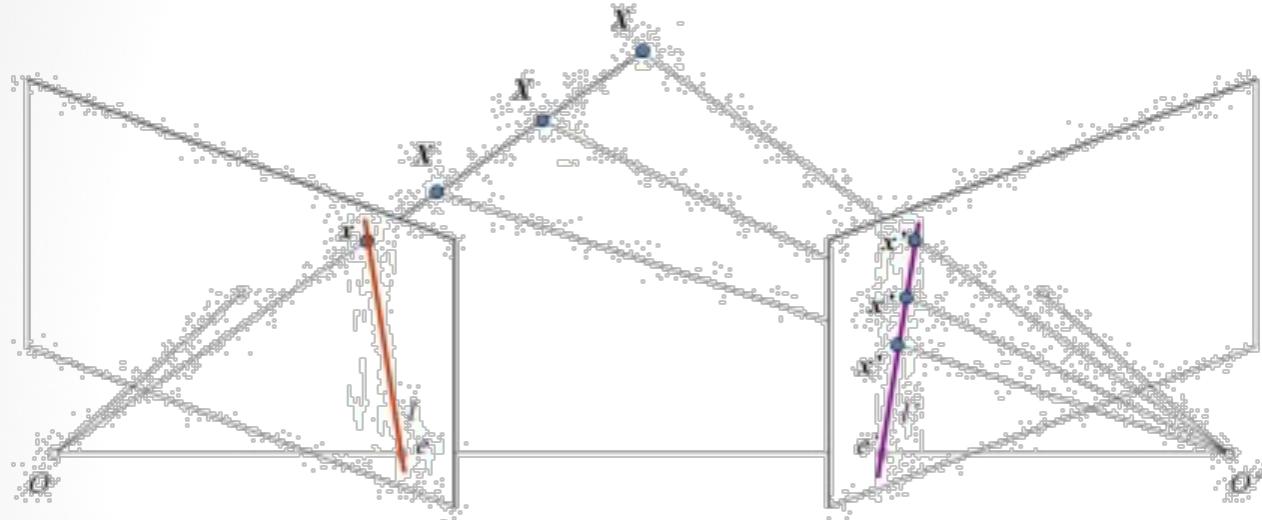
Tracken der Kamerabewegung: Epipolargeometrie

Funktioniert für: Statische Szene, Kamera bewegt



Epipolargeometrie, Fundamentalmatrix

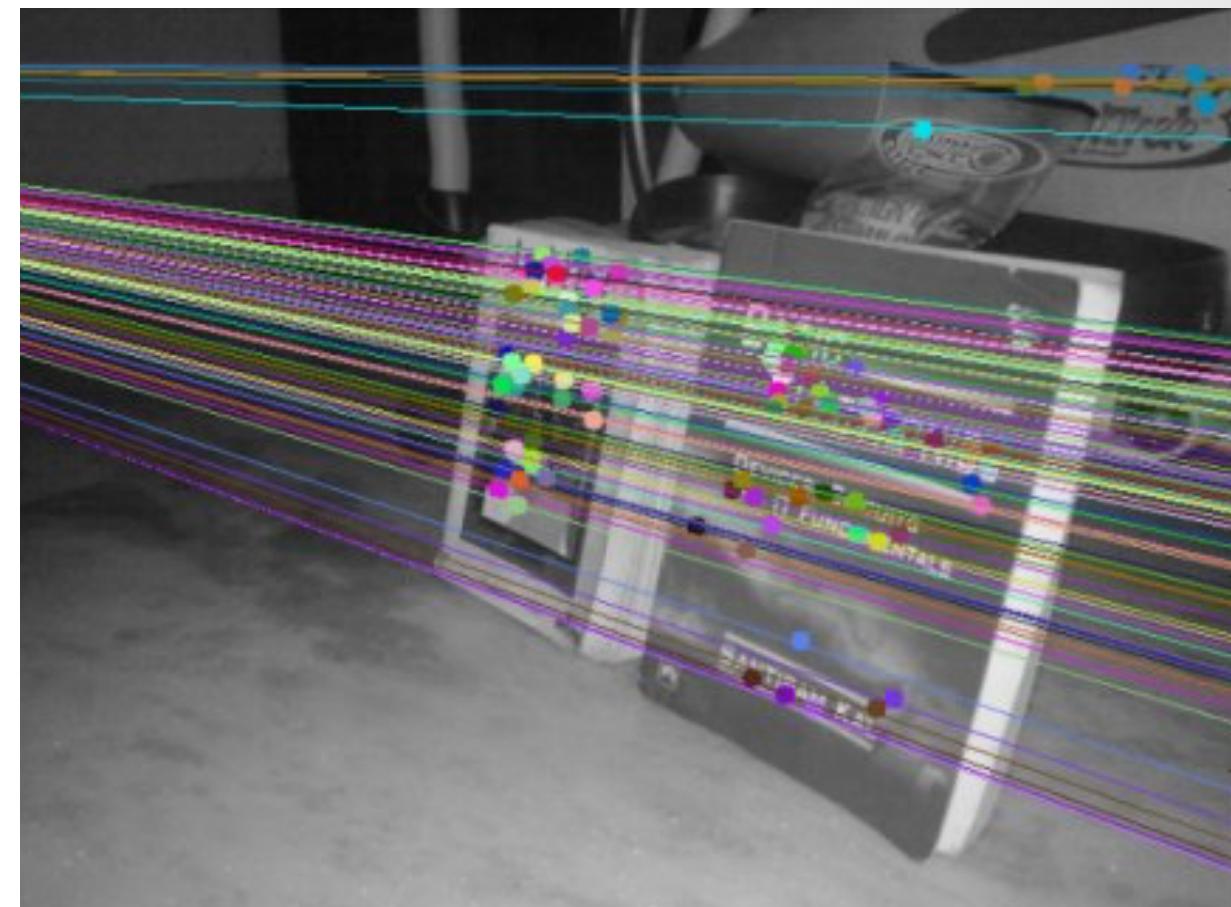
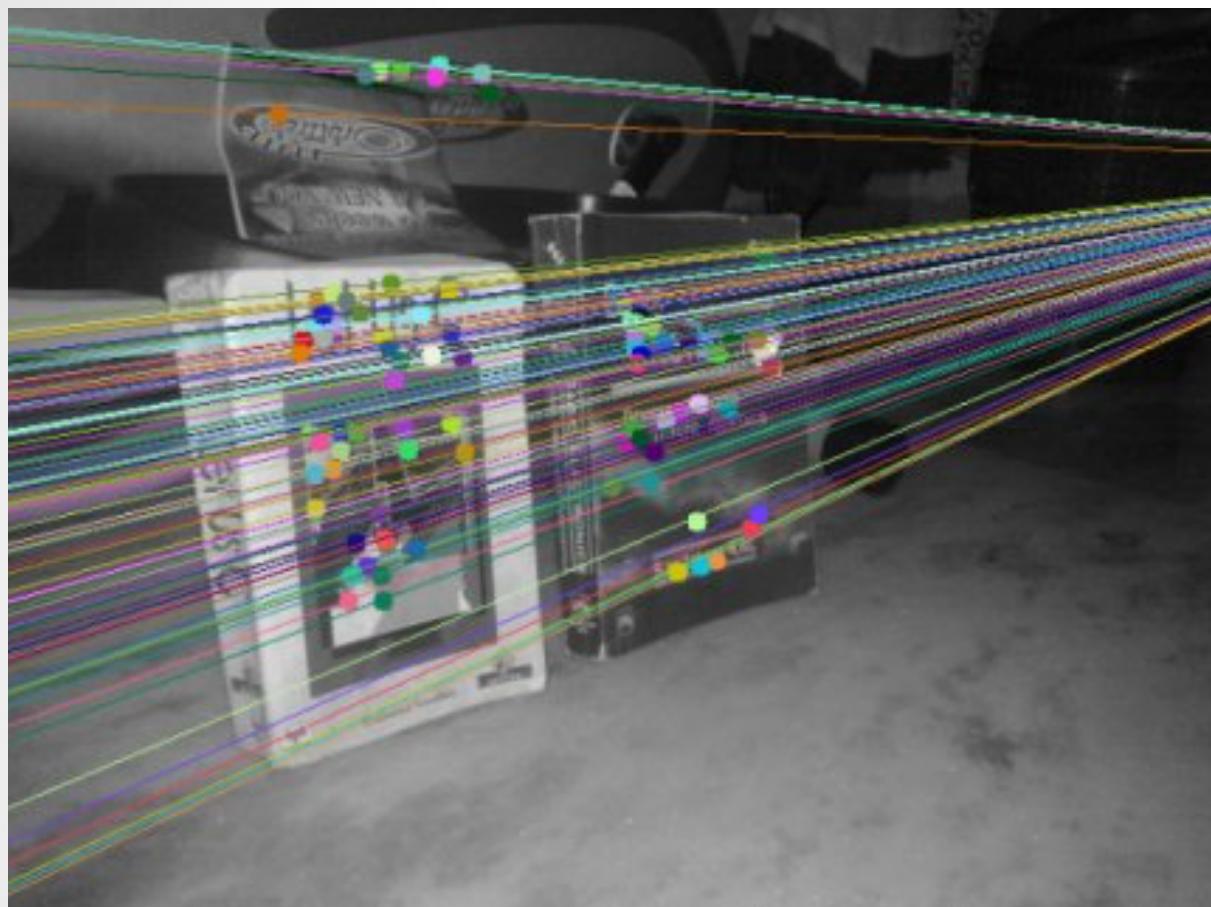
- Jeder 2D-Punkt im linken Bild korrespondiert zu einer Linie im rechten Bild (und umgekehrt)



- Kann mittels der Fundamentalmatrix ausgedrückt:
 $p' F p = 0 \quad F \in \mathbb{R}^{3 \times 3}$
- Sind beide Kameras kalibriert bzw. haben bekannte intrinsische Matrix K : „Essential Matrix“

$$p'^\top E p = 0, \quad E = R [R^\top t]_\times$$

Beispiel für Epipolarlinien

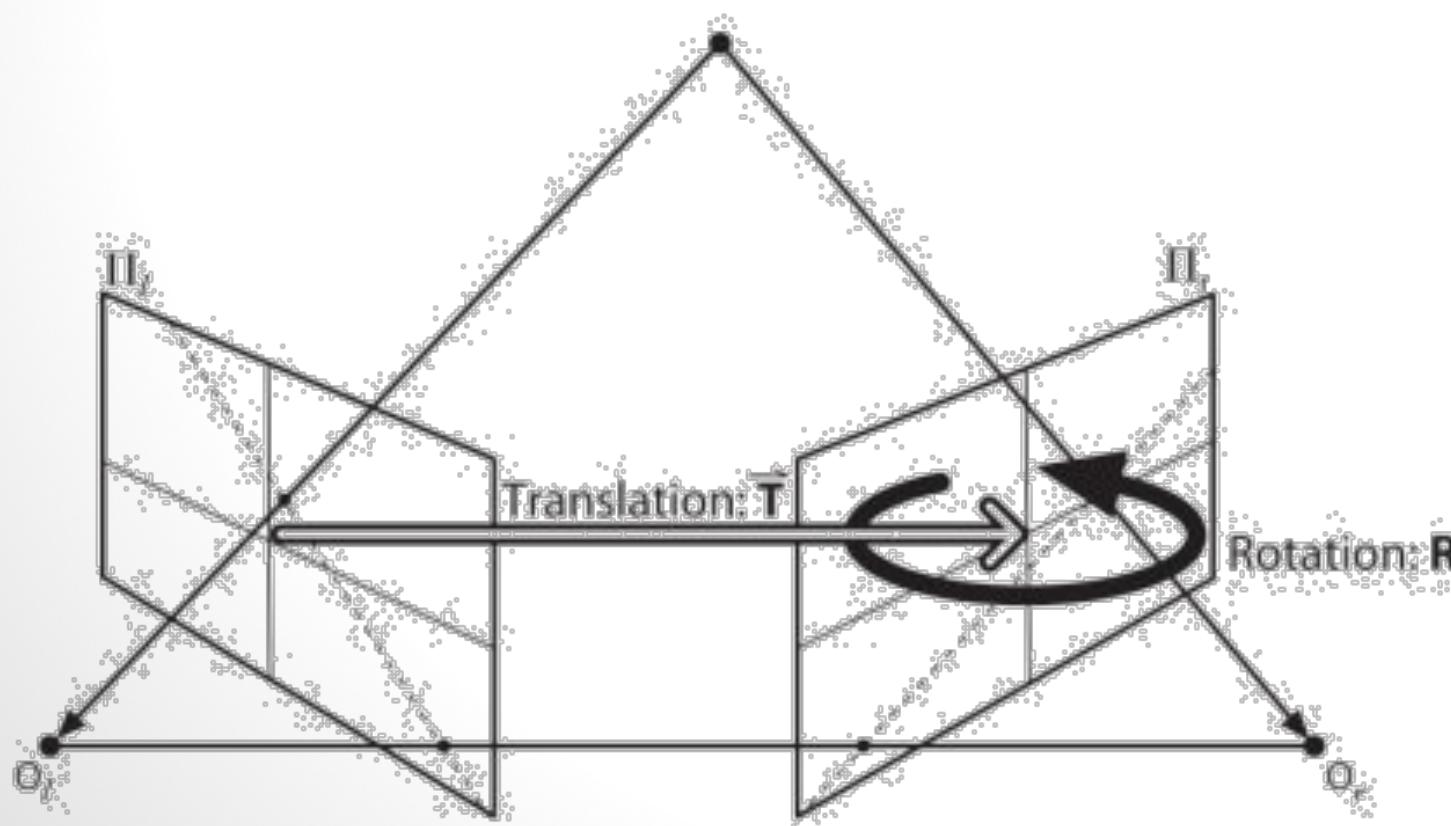


- Korrespondierende Punkte im Bild rechts liegen auf der Epipolarlinie im linken Bild

Ermittlung der Kameratranslation und Rotation

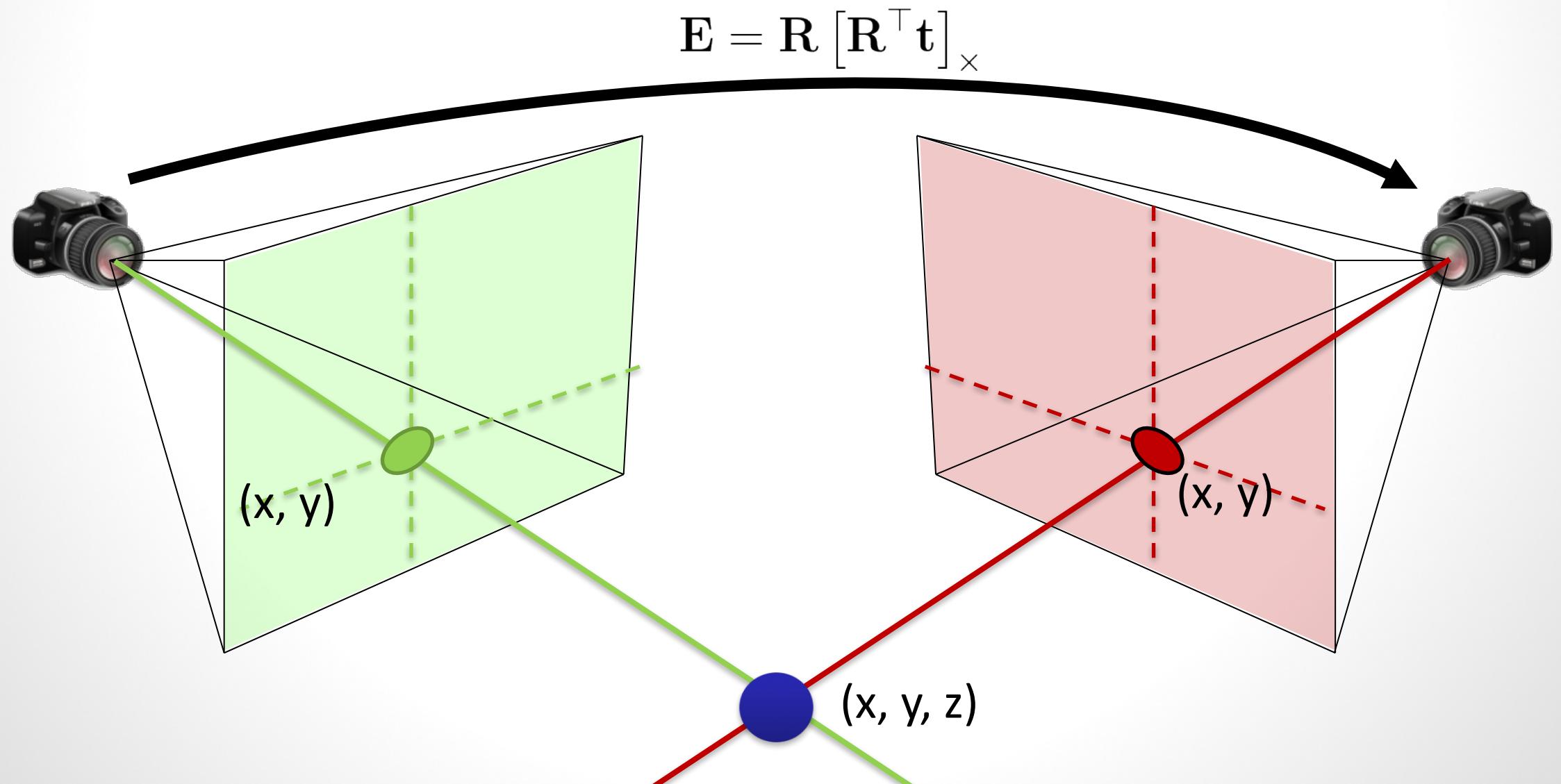
Gegeben: min. 8 2D-zu-2D Punktkorrespondenzen & intrinsische Parameter K :

1. Essential Matrix berechnen
(8-point Algorithm, evtl. RANSAC)
2. Rotation und Translation der Kamera ausfaktorisieren



Funktionsprinzip: „Stereorekonstruktion“

- Rekonstruktion 2D aus 3D: Mindestens 2 Kameras (oder 1 Kamera und 2 Bilder an unterschiedlichen Orten / statische Szene)
- Korrespondenz zwischen 2D-Punkten (z.B. über SIFT)



Prinzip SLAM

Initialisierung:

- Neues Bild → Features → Matchen mit vorigem Bild
- Kamerapose anhand Features berechnen
- Featurepunkte 3D-rekonstruieren und dem Modell (der Map) hinzufügen

Danach:

- Neues Bild → Features → Matchen mit bekannten 3D-Punkten aus der Map
- Kamerapose anhand der Features berechnen
- Map verfeinern (3D-Punkte korrigieren/hinzugügen)

ORB-SLAM2 Open Source Tracking System



Instituto Universitario de Investigación
en Ingeniería de Aragón
Universidad Zaragoza

ORB-SLAM2: an Open-Source SLAM System
for Monocular, Stereo and RGB-D Cameras

Raúl Mur-Artal and Juan D. Tardós

raulmur@unizar.es

tardos@unizar.es

Zusammenfassung

Wir haben uns heute angeschaut wie wir:

- ✓ Marker, 3D-Umgebung, Kamerapose tracken
- ✓ Dreidimensional in das Bild rendern

Was noch fehlt:

- Beleuchtung der Szene schätzen
- Integration zusätzlicher Sensoren beim Tracking
- Objekterkennung (z.B. automatisch Ebenen erkennen)
- Überlappungen zwischen virtuellen und echten Objekten korrekt behandeln (→ Segmentierung)
- Dynamische Szenen/Objekte tracken

Wingnut AR – ARKit + Unreal Engine



Beispiel: Dynamische Oberflächen augmentieren

Makeup Lamps: Live Augmentation of Human Faces via Projection

A. H. Bermano^{1,2}, M. Billeter³, D. Iwai⁴, A. Grundhöfer¹



Tracking und Computer Vision für Augmented Reality

Thomas Neumann

tneumann@htw-dresden.de

Code + Slides demnächst im OPAL (oder auf github, mal sehen)