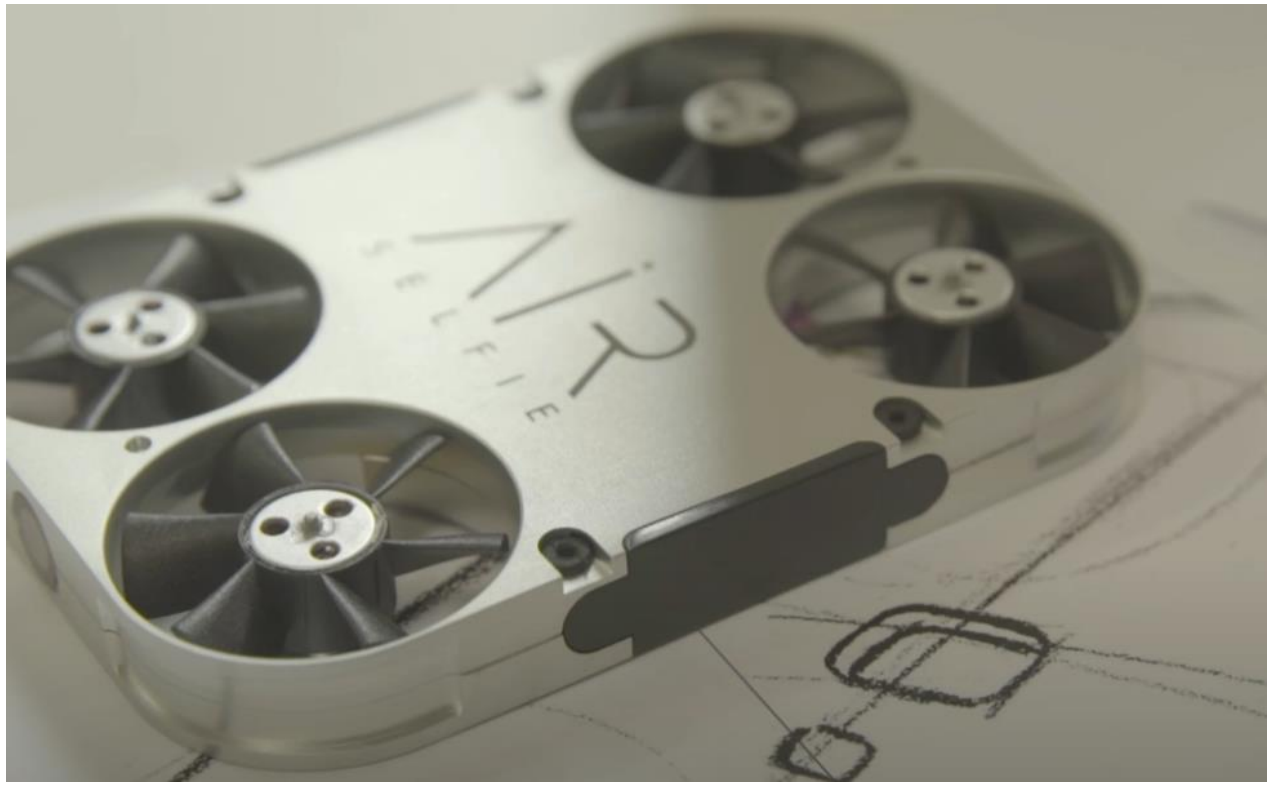
Sensortechnik

Lerneinheit 6: Smart AirCam – Bildverarbeitung und Face Detection

Email: joerg.dahlkemper@haw-hamburg.de



https://www.youtube.com/watch?v=5sgi3A6OwWM

Inhalt der Vorlesung

Sensorgestützte Funktionen

1 IoT-Sensornetze

Machine Learning mit Sensordaten

2 Analyse von Sensordaten
3 Klassifizierung von Sensordaten

Sensorsignalvorverarbeitung

4 Sensordatenfusion
5 Analyse der Sensordatenfusion

Einführung in die Bildverarbeitung

6 Bildverarbeitung und Face Detection

Review

7 Review, Feedback und Fragen

Labor 1: IoT-Sensornetzwerk

Labor 2 + 3:
Maschinelles Lernen zum Betriebszustandsmonitoring

Labor 4: Lagegesteuerter Quadcopter

Labor 5 + 6:
Smart AirCam

Motivation

Idee: Einführung in die Bildverarbeitung am Beispiel

Kameranachführung

http://business.panasonic.de/professional-kamera/remote-kameras/integrierte-pt-kameras/AW-SF100G

Automatischer Fokus

https://www.olympus.ch

Robotersteuerung

http://www.ipa.fraunhofer.de/en/care-o-bot4.html

Motivation

https://www.youtube.com/watch?v=5sgi3A6OwWM

Zielsetzung

Entwickeln Sie eine Smart AirCam auf Basis der zur Verfügung gestellten Drohne und einem PC für den folgenden Use Case:

- Bei Start hebt die Drohne auf eine definierte Höhe von etwa 1,5 m ab.
- Das Video der Drohne wird auf den PC gestreamt.
- Die Drohne dreht auf der Stelle, bis ein Gesicht gefunden wurde.
- Die Drohne verfolgte genau dieses Gesicht allein durch Drehung.
- Wenn sich die Position des Gesichts 3 Sekunden unverändert in der Mitte des Bildes befindet, nimmt die Drohne im Abstand von 2 Sekunden ein Foto auf und landet wieder selbständig.
- Die beiden Fotos werden auf dem PC angezeigt und gespeichert.

Infrastruktur > Bildverarbeitung

1. Bildverarbeitungsanwendungen sind rechenaufwändig und zeitkritisch

2. Hohe Rechenkapazität erforderlich
   - CPU-spezifische parallele Ausführung von Rechenoperationen
   - Auslagerung von Berechnungen in die Graphikkarte (GPU)

3. Gleiche Algorithmen für sehr unterschiedliche Aufgaben benötigt

$\Rightarrow$ Einsatz von Softwarebibliotheken für die industrielle Bildverarbeitung

Infrastruktur > Bildverarbeitung > OpenCV

- OpenCV ist der de-facto-Standard für Open Source BV
  - umfassende und state-of-the-art C und C++ Bibliotheken
  - durch Entwicklerteam von Intel 2006 ins Leben gerufen
  - maßgeblich durch Willow Garage weiterentwickelt und gepflegt

http://opencv.org/

- Vorteile
  - sehr performante Implementierung bei optimaler Nutzung der Hardware
  - aktuelle Entwicklungen der Bildverarbeitung werden schnell in OpenCV umgesetzt
  - sehr umfangreiche Funktionalität (u.a. auch Klassifizierung mit neuronalen Netzen, Gesichtserkennung, optisches Tracking, 3D-Bildverarbeitung)
  - verfügbare Java Wrapper, Versionen auch für Android und IOS verfügbar
  - in Verbindung mit Python sehr effiziente Entwicklungsumgebung
- Nachteile
  - sehr rudimentäre Interaktion mit User – beschränkt auf Bild- und Videoanzeige

Installation

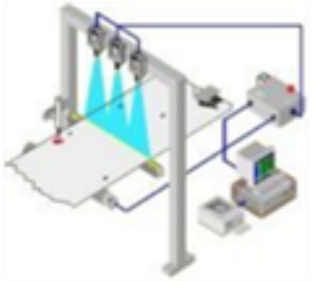
- Anaconda-Umgebung ST aktivieren
- `pip install opencv-contrib-python` eingeben
- Kamerazugriff auf eine Webcam mit folgendem Programm testen, das mit Drücken von <ESC> beendet wird.

```
import cv2
cap = cv2.VideoCapture(0)
while cv2.waitKey(15) != 27: # solange nicht <escape> gedrueckt
    ret, frame = cap.read()  # Statusflag ret und Bild frame auslesen
    cv2.imshow('Webcam', frame)
cap.release()                # Kamera wieder freigeben
cv2.destroyAllWindows()      # Fenster wieder schliessen
```

Bildverarbeitungskette

- Ziel:
- Automatisierte Verarbeitung von Bildern zur Erzielung von Informationen

- Synonyme: „Maschinelles Sehen“, „Computer Vision“, „Robot Vision“, „AOI“



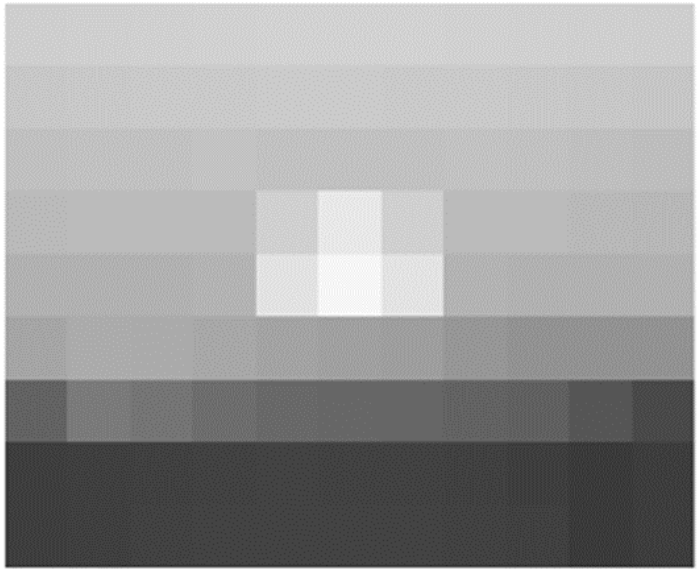

Bildaufnahme	Vorverarbeitung	Segmentierung	Merkmals-extraktion	Klassifizierung
Fotografieren	Rauschminderung	Fehler finden	Fehler vermessen	Fehlertyp bestimmen

Bildaufnahme > Was ist ein Bild?

- aus Bildaufnahme resultiert bei Monochromkamera ein **Graustufenbild**
- Mathematisch ist dies eine Abbildung B mit g: Grauwert, r: row, c:column
- $B: (r, c) \in \{1, \ldots, N, 1 \ldots M\} \rightarrow g(r, c)$
- Ein Bild ist eine **Matrix,** ein Element der Matrix ist das **Pixel** (Picture element)

„Digitaler Sonnenuntergang“

```
[[207 203 206 215 217 211 208 211 208 207 206]
 [199 201 203 200 196 196 202 209 197 196 196]
 [193 196 192 186 190 198 198 190 196 194 192]
 [187 187 184 189 213 232 215 182 187 185 182]
 [177 182 183 191 218 238 220 185 179 177 177]
 [157 168 170 162 165 174 168 151 150 147 144]
 [110 120 120 107  98  98  98  93  92  88  82]
 [ 59  64  66  64  66  70  68  62  64  62  59]
 [ 63  64  66  67  68  67  66  65  65  60  59]]
```



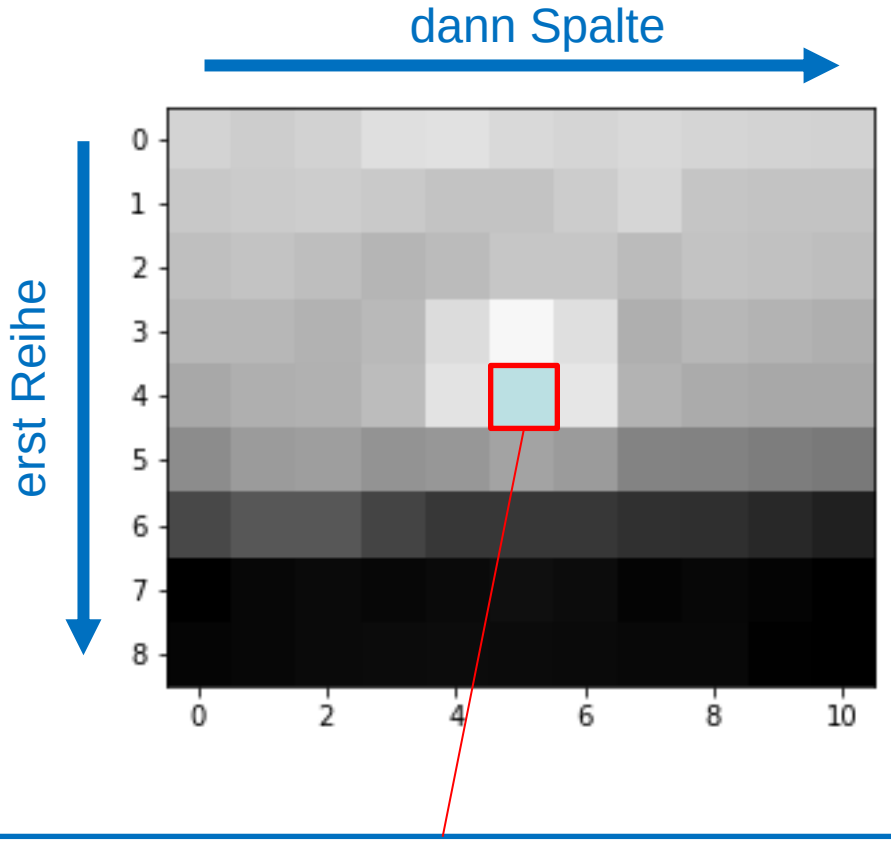
0: schwarz … 255: weiß

Bildaufnahme > Zugriff auf Pixel

- Zugriff auf Elemente der Matrix über eckige Klammern: `frame[row, column]`

`frame =`

```
[[207 203 206 215 217 211 208 211 208 207 206]
 [199 201 203 200 196 196 202 209 197 196 196]
 [193 196 192 186 190 198 198 190 196 194 192]
 [187 187 184 189 213 232 215 182 187 185 182]
 [177 182 183 191 218 238 220 185 179 177 177]
 [157 168 170 162 165 174 168 151 150 147 144]
 [110 120 120 107  98  98  98  93  92  88  82]
 [ 59  64  66  64  66  70  68  62  64  62  59]
 [ 63  64  66  67  68  67  66  65  65  60  59]]
```



Frage:
Wie lesen Sie den Grauwert dieses Pixels aus?

`frame[4, 5]`

Bildaufnahme > Zugriff auf Pixel

- Zugriff auf Teilbereiche der Matrix über Doppelpunkt (Colon operator):

Hinweis: anders als bei matlab

- `frame[:, c]` Spalte c ausgeben
- `frame[r, :]` Reihe r ausgeben
- `frame[:3, :]` die ersten 3 Reihen ausgeben (0, 1, 2)
- `frame[1:3, :]` Start bei Reihe 1 und bis Reihe 2 ausgeben
- `frame[-1, :]` letzte Reihe ausgeben
- `frame[::2, :]` Schrittweite 2, also jede 2. Reihe ausgeben
- `frame[::-1, c]` Schrittweite -1 => Reihenfolge umdrehen

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
frame[:, 2]
```

```
array([ 2,  6, 10, 14])
```

```
frame[:, 2].reshape(-1, 1)
```

```
array([[ 2],
       [ 6],
       [10],
       [14]])
```

- Hinweis:
  Es wird bei der Rückgabe eines Vektors nicht zwischen Spalten- und Zeilenvektoren unterschieden.
  Wenn erforderlich, muss der Vektor in ein Array umgeformt werden:
- `frame[:, 2].reshape(-1, 1)  # -1 = beliebige Anzahl`

Bildaufnahme > Zugriff auf Pixel

- Wie lautet der Befehl, um die markierten Bereich anzusprechen:

```
A =                    A =                    A =
    1    2    3    4        1    2    3    4        1    2    3    4
    5    6    7    8        5    6    7    8        5    6    7    8
    9   10   11   12        9   10   11   12        9   10   11   12
   13   14   15   16       13   14   15   16       13   14   15   16
```

Binde die Bibliothek zum Rechnen mit Matrizen ein (wie matlalb)

```
import numpy as np
frame = np.arange(1, 17).reshape((4, 4))
```

Erzeuge einen Vektor von 1, 2, ..., 16

Forme den Vektor in eine 4 x 4 Matrix um

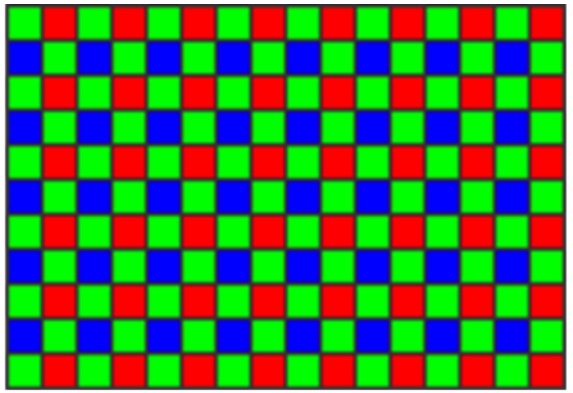
Bildaufnahme > Farbbild

- Bildsensoren sind im gesamten sichtbaren Bereich empfindlich

⇒ Pixel kann keine Farbe wahrnehmen

- Idee: Farbfilter der Grundfarben blau, grün und rot vor jedes Pixel setzen

⇒ Pixel reagiert dann auf die Farbe des Filters



Bayer-Pattern

Wie ordnet man 3 Grundfarben bei einer rechteckigen Matrix an?

- Bayer-Pattern, US Patent 3.971.065 vom 20.06.1976

Was fällt auf?

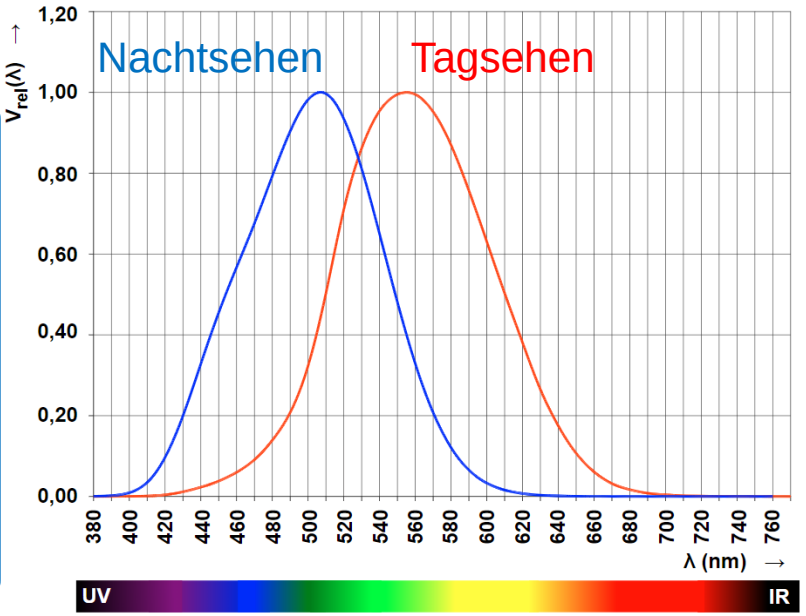
- doppelt so viele grüne Pixel

Warum?

- bei grün genau messen, da das Auge hier besonders empfindlich

Wieviel Pixel hat eigentlich ein 1 MPixel Farbsensor?

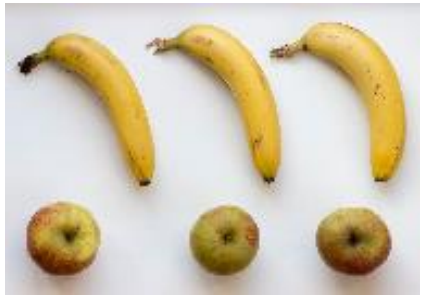
- 1024 x 1024 Pixel => Interpolation notwendig (Demosaicing)



Hellempfindlichkeitskurve, Autor: H. Hahn
https://de.wikipedia.org/wiki/V-Lambda-Kurve

Bildaufnahme > Farbbild > Beispiel Obstsortierung

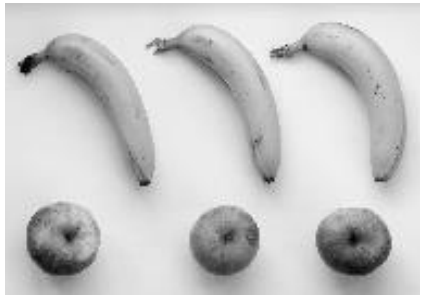
Eingangsbild



Analyse des Farbbildes als Voruntersuchung zur Klassifikation:

Rot

Grün



Blau

Unterscheidung von Apfel und Banane über Grauwerte möglich

hoher Kontrast hilft zur Unterscheidung von Vorder- und Hintergrund (Segmentierung)

Bildaufnahme > OpenCV

(rows, cols, channels)

Kamera liefert als Bild eine 3D-Matrix:

```
frame.shape = (480, 640, 3)
```

3. Dimension ist der Farbkanal:
0: blau – 1: grün – 2: rot (BGR)

Hinweis:
BGR ist eher unüblich. Die Bibliothek matplotlib und matlab arbeiten in dem üblicheren RGB-Format.

Demonstration:

Originalbild und jeder Farbkanal in eigenem Fenster „R“, „G“, „B“ separat sowie Grauwertbild
Interpretieren Sie die Bilder.

Hinweise:

- Zur Auswahl des jeweiligen Bildkanals nutzen Sie die Doppelpunkt-Notation für Array-Indizes.
- Das Grauwertbild kann über die Farbkonvertierungsfunktion `cvtColor` wie folgt erzeugt werden:
  `frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)`

Bildaufnahme > Wichtige Funktionen

Bilder einlesen

- Bild aus Datei einlesen:
```
image = cv2.imread("opencv.png")
```
- Farbbild als Grauwertbild einlesen:
```
image = cv2.imread("opencv.png", 0)
```

Videokamera einlesen

```
cap = cv2.VideoCapture(0)          # Standardkamera öffnen (0: default, auch 1, 2, ...)
ret, frame = cap.read()            # Bild einlesen, ret ist True, wenn erfolgreich
cv2.imshow('Webcam', frame) # Bild in Fenster zur Anzeige vorbereiten
cv2.waitKey(0)                     # Bild anzeigen und warten (0: bis Taste gedrückt)
cap.release()                      # Kamera wieder freigeben
cv2.destroyAllWindows()            # Alle mit imshow erzeugten Fenster schließen
```

Farbraum konvertieren

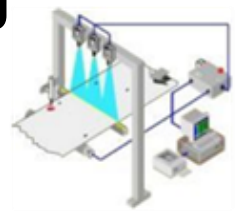
- BGR-Farbbild in Grauwertbild konvertieren:
```
image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```
- BGR-Farbbild in RGB-Farbbild konvertieren:
```
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

Zugriff auf einen Videostream (siehe opencv_tello.py)


```
for tello access
TELLO_IP = '192.168.10.1'
TELLO_PORT = 8889
tello_address = (TELLO_IP, TELLO_PORT)

# for receiving from tello
VS_UDP_IP = '0.0.0.0'
VS_UDP_PORT = 11111

# Create a socket for communication Address family: AF_INET (IPv4), Socket type: SOCK_DGRAM (UDP)
socket = socket.socket (socket.AF_INET, socket.SOCK_DGRAM)
socket.bind (('', TELLO_PORT))

# Throw 'command' text to use command mode and start video streaming
socket.sendto ('command'.encode (' utf-8 '), tello_address)
socket.sendto ('streamon'.encode (' utf-8 '), tello_address)
udp_video_address = 'udp://' + TELLO_IP + ':' + str (VS_UDP_PORT)

video_stream = cv2.VideoCapture (udp_video_address)
while True:
    ret, frame = video_stream.read()
    if ret:
        …
video_stream.release ()
cv2.destroyAllWindows ()

# Stop video streaming
socket.sendto ('streamoff'.encode (' utf-8 '), tello_address)
socket.close()
```

Inhalt

1. Einführung
2. Bildaufnahme
3. **Bildvorverarbeitung**
4. Bildsegmentierung
5. Merkmalsextraktion
6. Klassifizierung



Kamera
Objektiv
Beleuchtung

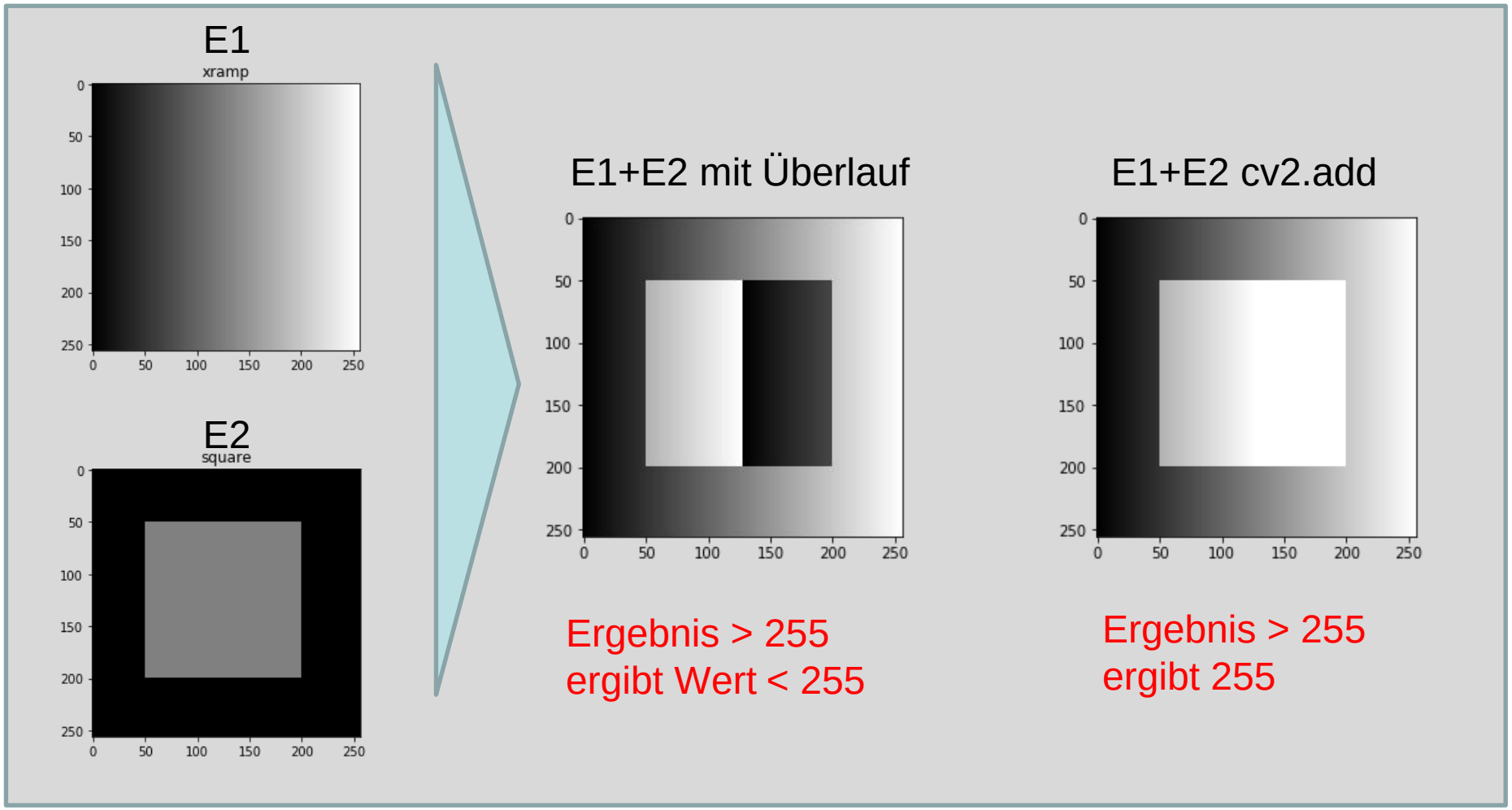
Kontrast optimieren,
Rauschminderung

Fehler
finden

Fehler
vermessen

Fehlertyp
bestimmen

- Datentyp: uint8 = 8-Bit unsigned mit Wertebereich 0 ... 255
  ⇒ Addition/Subtraktion führt ohne Warnung zu Überlauf



statt der üblichen Operatoren +, -, *, / OpenCV-Funktionen verwenden:

```
cv2.add
cv2.subtract
cv2.absdiff
cv2.divide
cv2.multiply
cv2.log
cv2.exp
cv2.pow
```

HAW Hamburg
Fakultät TI
Technik und Informatik

Demonstration:

- Grauwertbild der KameraErweiter
- Fenster „plus“: per „+“-Operator den Wert 128 auf das Bild addiert
- Fenster „add“: 128 per `cv2.add`

Analysieren Sie, wo im Bild ein Überlauf stattfindet.

Hinweis: Die meisten Rechenoperationen arbeiten nach dem folgenden Schema:

```
cv2.add(src1, src2[, dst[, mask[, dtype]]]) → dst
```

`src1`	first input array **or a scalar**
`src2`	second input array **or a scalar**
`dst`	output array that has the same size and number of channels as the input array(s); depth is defined by dtype or src1/src2.
`mask`	optional operation mask - 8-bit single channel array, that specifies elements of the output array to be changed.
`dtype`	optional depth of the output array

Quelle: https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html

Aufgabe: Finden Sie den Fehler. Wo unterscheiden sich die Bilder?

E1

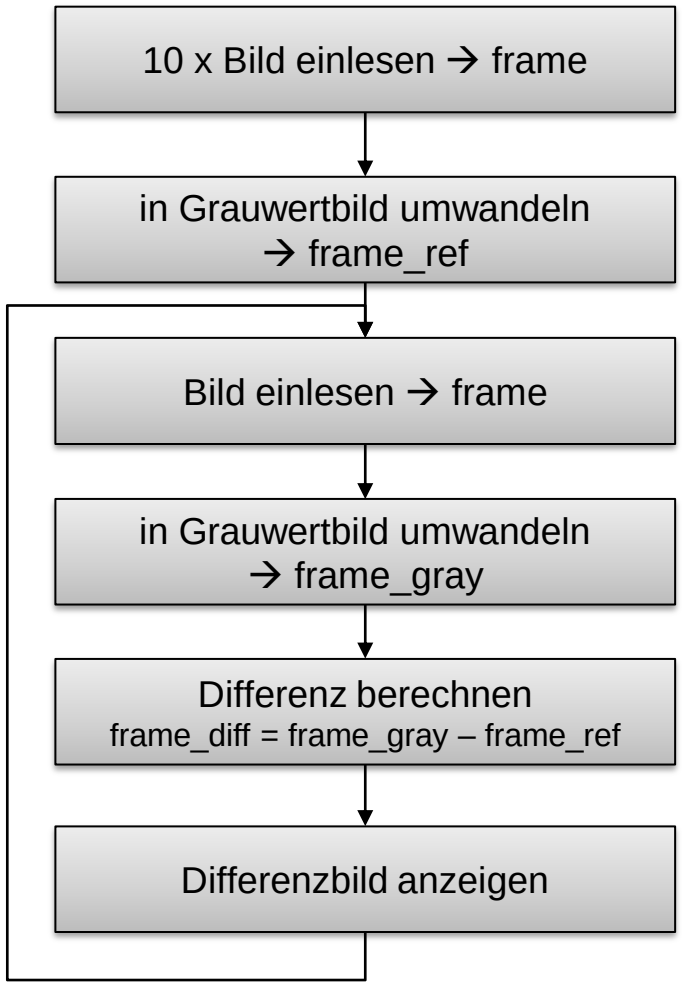
E2

Wichtige Begriffe:

Segmentierung: Aufteilung des Bildes in relevante Segmente und den unwichtigen Hintergrund.

Background-Subtraction: Segmentierung bei der von dem Bild ein Referenzbild abgezogen wird.

Bildvorverarbeitung > Beispiel zur Bewegungserkennung



Aufgabe

Analysieren Sie den dargestellten Programmablauf.

Hinweise:
Zu Beginn passt die Kamera die Helligkeit an. Hierzu werden als Zeitschleife 10 Bilder aufgenommen und die ersten 9 verworfen.

Wenn ein zurückgegebener Wert nicht benötigt wird, kann dieser der temporären Variablen _ zugewiesen werden.

Implementierungsbeispiel

```python
import cv2

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)   # set width
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)  # set height

# Referenzbild einlesen
for i in range(10):
    _, frame = cap.read()
    cv2.imshow('Reference', frame)
    cv2.waitKey(100)
frame_ref = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

while cv2.waitKey(100) != 27: # 500 ms per image for 2 fps
    _, frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame_diff = cv2.absdiff(frame_gray, frame_ref)
    cv2.imshow('Differenzbild', frame_diff)

cap.release()
cv2.destroyAllWindows()
```

Bildvorverarbeitung > Wichtige Funktionen

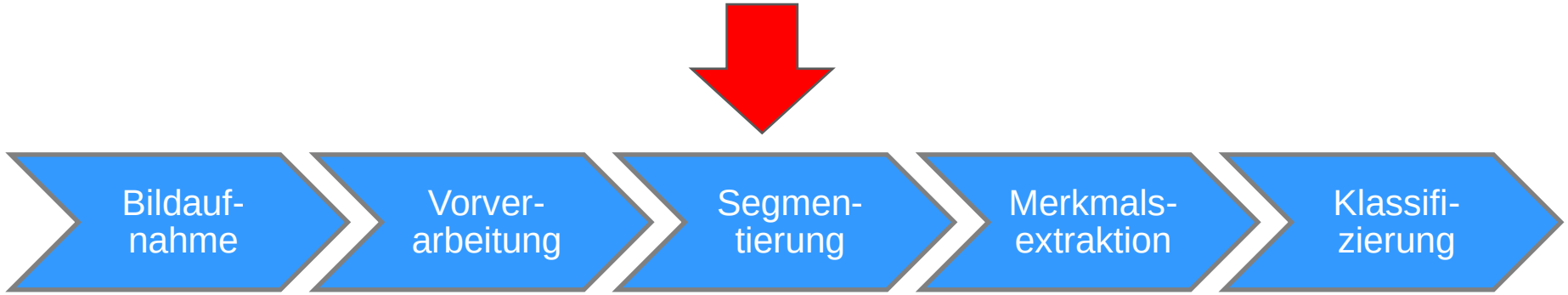
- **Rechenoperationen**
- `cv2.add, cv2.subtract, cv2.absdiff`
- `cv2.divide, cv2.multiply`
- `cv2.log, cv2.exp`
- `cv2.pow`

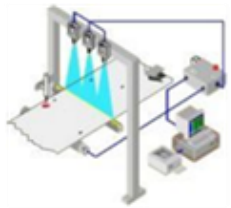
- **Glättungsfilter**
- Rechteckfilter `cv2.blur(E, (3,3))`
- Gauß-Filter `cv2.GaussianBlur(E, (3,3), 0)`

- **Kantenfilter**
- `Ex = cv2.Sobel(E, cv2.CV_8U,  1, 0, ksize=3) # als uint8 in x-Richtung`
- `Ey = cv2.Sobel(E, cv2.CV_64F, 0, 1, ksize=3) # als float mit Vorzeichen in y-Richtung`

Inhalt

1. Einführung
2. Bildaufnahme
3. Bildvorverarbeitung
4. **Bildsegmentierung**
5. Merkmalsextraktion
6. Klassifizierung





Kamera
Objektiv
Beleuchtung

Kontrast optimieren,
Rauschminderung

Fehler
finden

Fehler
vermessen

Fehlertyp
bestimmen

Bildsegmentierung > Begriff

Segmentierung ist die Zerlegung eines Bildes in die für die jeweilige Anwendung relevanten Objekte und den nicht relevanten Hintergrund. Im engeren Sinne spricht man nur dann von einer Segmentierung, wenn diese vollständig und überdeckungsfrei ist. (Tönnies, 2005)

⇒ Segmentierung = Trennung von Objekt und Hintergrund

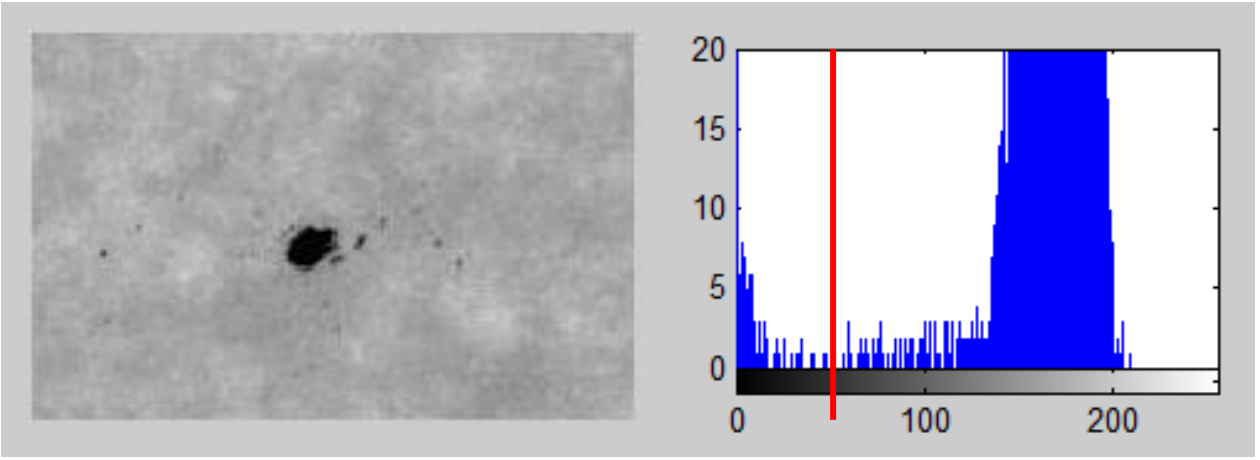
Idealfall: **bimodales Histogramm**

- Objekt und Hintergrund unterscheiden sich deutlich durch Grauwert
- ausgeprägtes Minimum im Histogramm zwischen Häufungen

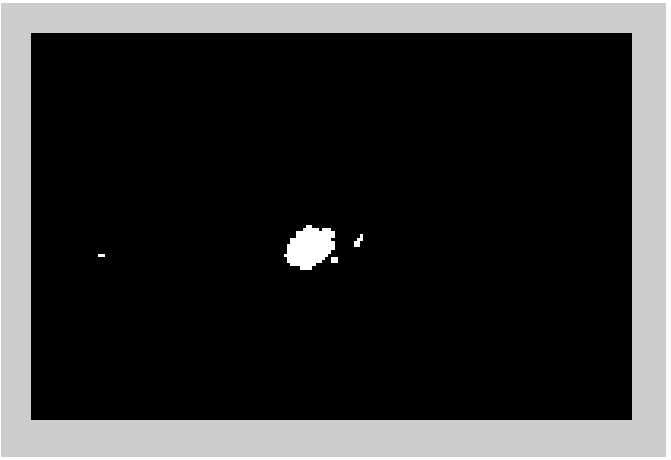
⇒ Prüfen, ob Grauwert über/unter einer Schwelle liegt

Konvention bei Binärbild:

- Objekt = 1 bzw. 255 bei OpenCV (weiß)
- Hintergrund = 0 (schwarz)



Schwellwert 50



(peqlab Biotechnologie GmbH, 2011)

Bildsegmentierung > Umsetzung in openCV

Ergebnis der Segmentierung ist Binärbild mit Hintergrund = 0 und Objektpixel = 1 oder 255 (konfigurierbar)

`level, EBW = cv2.threshold(E,` Schwelle, Wert für Objektpixel, Modus wie `cv2.THRESH_BINARY)`

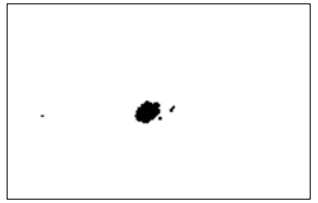
Funktion gibt 2 Parameter zurück

Da man den zurückgegebenen Schwellwert meist nicht braucht, häufig:

```
EBW = cv2.threshold(E, 50, 255, cv2.THRESH_BINARY)[1]
_, EBW = cv2.threshold(E, 50, 255, cv2.THRESH_BINARY)
```

Invertierte Schwellwertbildung: dunkle Bereiche suchen

```
level, EBW = cv2.threshold(E, 50, 255, cv2.THRESH_BINARY_INV)
```



Automatisches Finden der Schwelle nach Otsu

- Varianz von Vordergrund und Varianz von Hintergrund bestimmen und Schwelle so legen, dass beide minimal

```
level, EBW = cv2.threshold(E,  0, 255, cv2.THRESH_OTSU)
```

gefundener Schwellwert

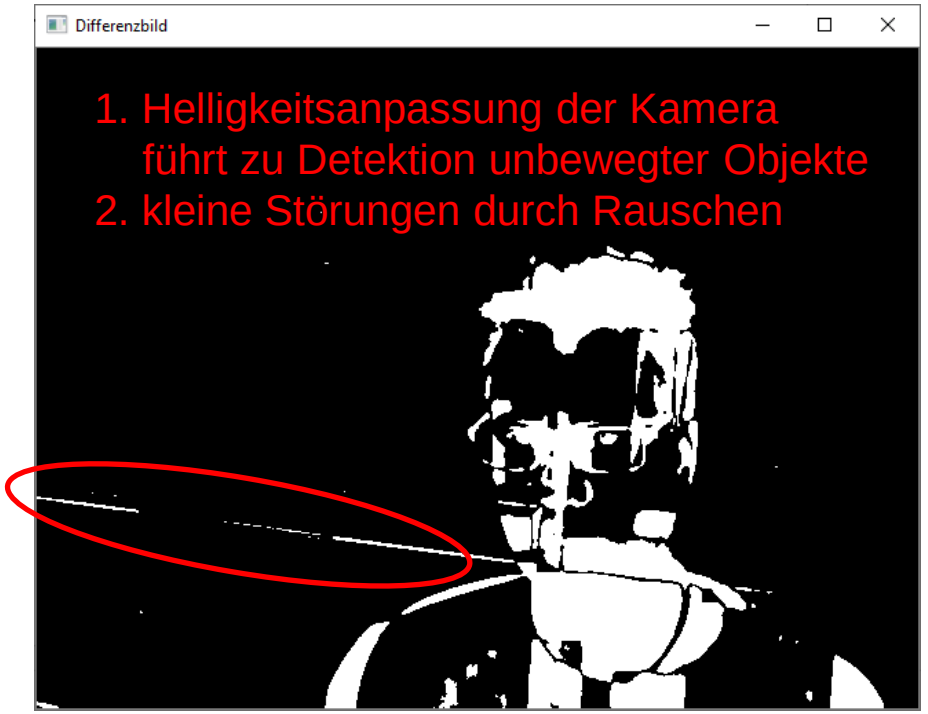
Die Vorgabe einer Schwelle wird einfach ignoriert.

Bildsegmentierung > Anwendungsbeispiel

Demonstration

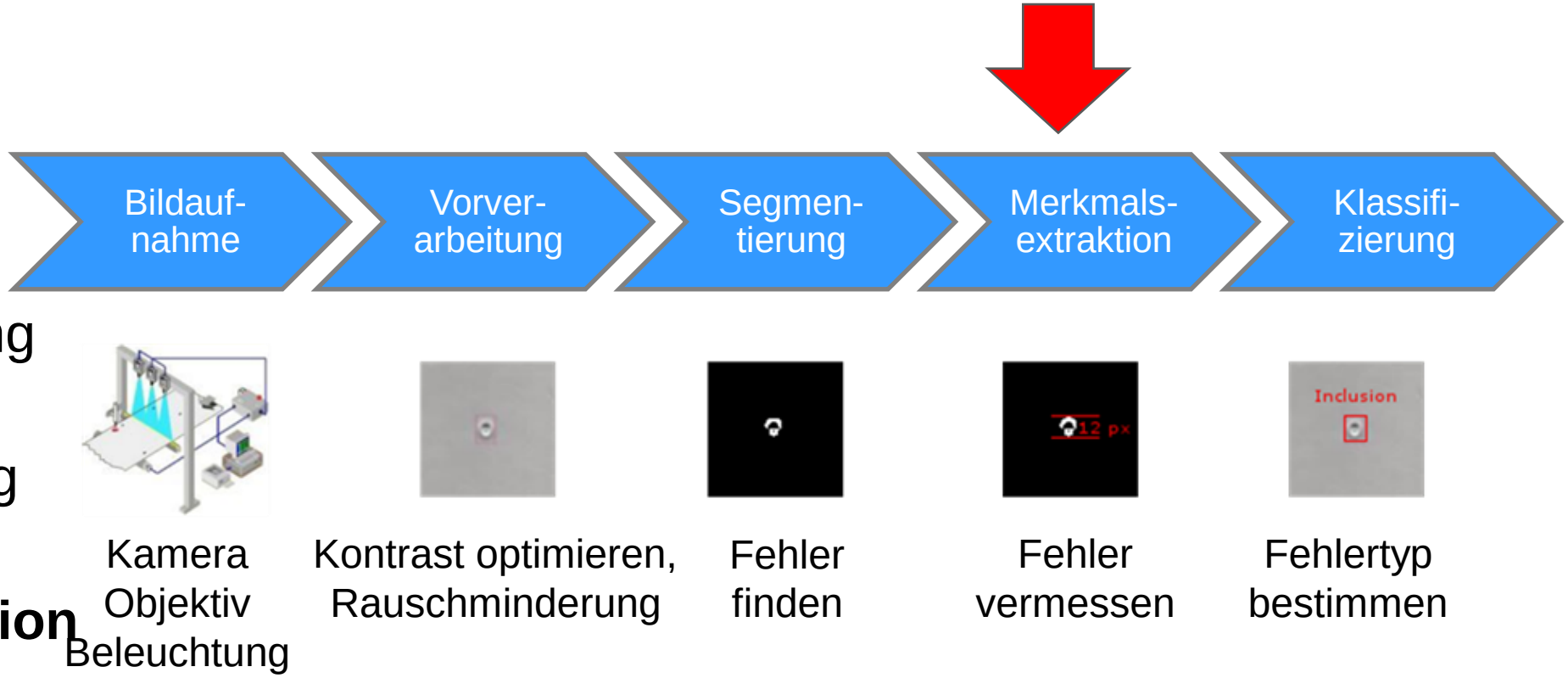
Erweiterung der Bewegungserkennung, so dass das Ergebnis als Binärbild angezeigt wird, so dass Pixel mit Bewegung den Wert 255 aufweisen.
Welche Probleme ergeben sich insbesondere bei geringem Schwellwert?

```python
import cv2

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)   # set width
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)  # set height

# Referenzbild einlesen
for i in range(20):
    _, frame = cap.read()
    cv2.imshow('Reference', frame)
    cv2.waitKey(100)
frame_ref = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
frame_ref = cv2.GaussianBlur(frame_ref, (5, 5), 0)

while cv2.waitKey(100) != 27: # 500 ms per image for 2 fps
    _, frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    frame_gray = cv2.GaussianBlur(frame_gray, (5, 5), 0)
    frame_diff = cv2.absdiff(frame_gray, frame_ref)
    frame_thresh = cv2.threshold(frame_diff, 30, 255, cv2.THRESH_BINARY)[1] # binary image
    cv2.imshow('Differenzbild', frame_thresh)

cap.release()
cv2.destroyAllWindows()
```



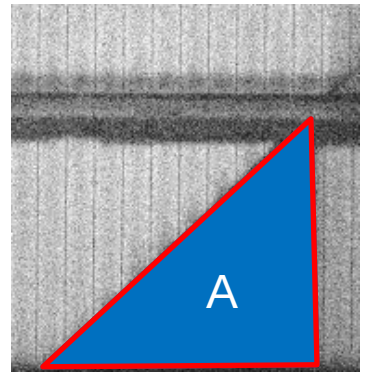
Inhalt

1. Einführung
2. Bildaufnahme
3. Bildvorverarbeitung
4. Bildsegmentierung
5. **Merkmalsextraktion**
6. Klassifizierung



Merkmalsextraktion > Begriffe

- Merkmal (engl. feature)
  eine in der Regel numerische Kennzahl, die aus den Bildpunkten des betrachteten Segments berechnet wird (auch Merkmalsausprägung)
  Beispiele: Größe des Segments, mittl. Grauwert



- Merkmalsvektor (engl. feature vector)
  zur Unterscheidung von Objekten fasst man in der Regel mehrere unterschiedliche Merkmale zu einem Merkmalsvektor zusammen

$$\vec{x} = \begin{cases} A\ (Fläche) \\ \bar{g}\ (mittlerer\ Grauwert) \end{cases}$$

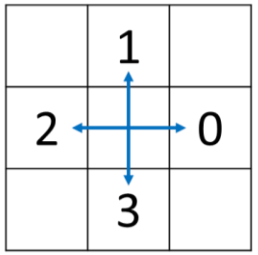
- Merkmalsextraktion (engl. feature extraction, feature engineering)
  Vorgang der Berechnung der Merkmale

Merkmalsextraktion > Kontur

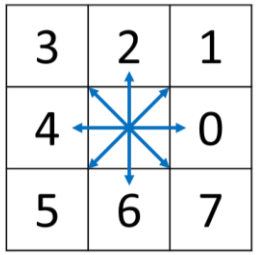
Definition: Die Kontur (auch Rand) beschreibt die Pixel,

- die selbst zum Objekt gehören und
- in deren Nachbarschaft sich mindestens ein Hintergrundpixel befindet.

bei 4er-Nachbarschaft



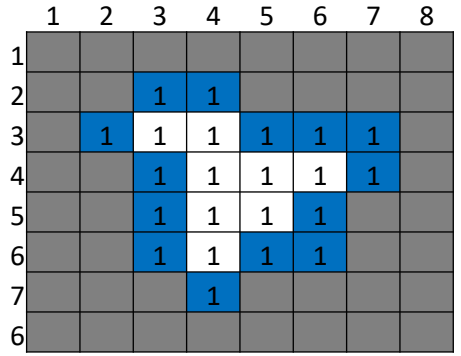
bei 8er-Nachbarschaft



Die Kontur ist die zentrale Grundlage zur Bildverarbeitung bei OpenCV (anders als bei Matlab).

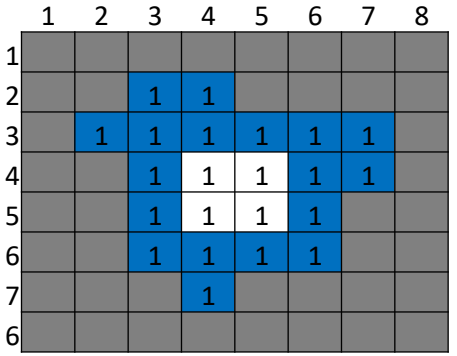
Bestimmen Sie zu dem gegebenen Objekt die Kontur, und zwar die ...

Kontur mit Hintergrund in 4er-Nachbarschaft



⇒ Pfad in 8er-Nachbarschaft

Kontur mit Hintergrund in 8er-Nachbarschaft



⇒ Pfad in 4er-Nachbarschaft

Besondere Bedeutung von Konturen bei OpenCV

Bestimmung der Konturen erfolgt auf einem Binärbild mit einem oder mehreren BLOBS (Binary Large OBjectS):

- Vorteil: Effizientere Speicherung als vollständiges Segment (vgl. region labelling in Matlab)

1. Konturen finden
   `cnts, h = cv2.findContours(EBW, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)`
   liefert eine Liste von Konturen `[cnt1, cnt2, cnt3, ...]` oder eine leere Liste (wenn keine gefunden)
2. Anzeige der Konturen – sinnvollerweise im zugehörigen farbigen BGR-Eingangsbild E
   `cv2.drawContours(E, cnts, -1, (0, 0, 255), 2)`

Liste von Konturen

Auswahl: alle Konturen

Farbe (hier rot), Strichstärke

Wichtig

Häufig Kontur mit größter Fläche gesucht:

```
cnt = max(cnts, key=cv2.contourArea)
```

oder iterieren

```
for cnt in cnts:
    cv2.drawContours(E, [cnt], -1, (0, 0, 255), 2)
```

Einer-Liste von Konturen

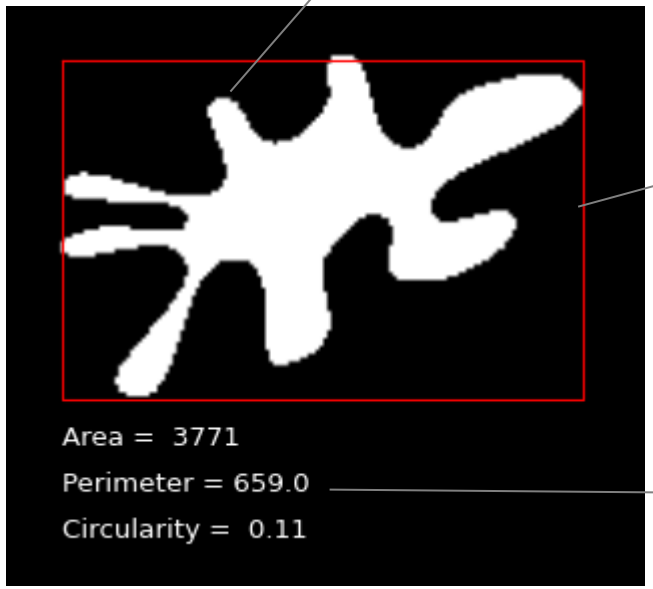
```
cnts, h = cv2.findContours(frame_thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
if cnts:                                                    # leere Kontur ausschliessen
    cnt = max(cnts,  key=cv2.contourArea)                   # finde groesste Kontur
    cv2.drawContours(frame, [cnt], -1, (0, 0, 255), 2)      # Farbe ins Farbbild malen
cv2.imshow("Groesste Kontur", frame)
```

Demonstration:
Konturen bewegter Objekte

Merkmalsextraktion > Vermessung und Anzeige von BLOBs

Unter **Bounding Box** versteht man das minimale achsenparallele Rechteck, dass alle gesetzten Pixel des betrachteten Segments einschließt.

Kontur
`cv2.drawContours()`

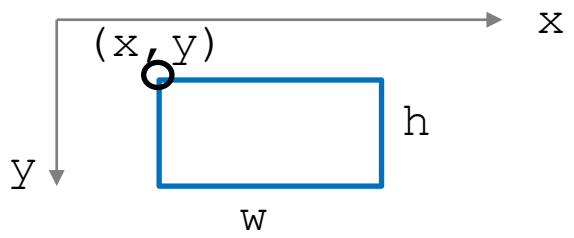


Bounding Box
`cv2.rectangle()`

Text
`cv2.putText()`

Umsetzung in openCV

`Bounding Box = [x y w h]`

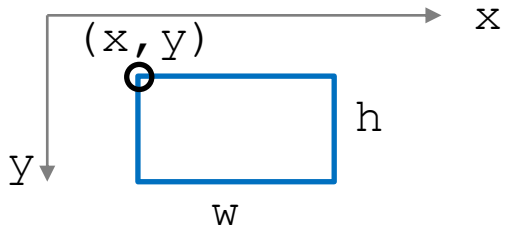



```
x,y,w,h = cv2.boundingRect(cnt)
```

Merkmalsextraktion > Bounding Box

Bounding box finden

```
x, y, w, h = cv2.boundingRect(cnt)
```



Bounding box zeichnen

```
cv2.rectangle(frame_bgr, (x, y), (x + w, y + h), (0, 0, 255), 1)
```

Farbe

Strichstärke

Text in zugehöriges Farbbild schreiben

```
cv2.putText(frame_BGR, 'Text', (cx, cy), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0))
```

Farbe

Position

akzeptiert keinen Zeilenumbruch

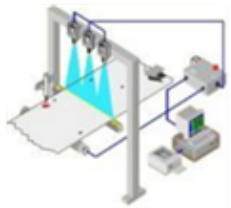
Fontgröße

Demonstration: Bounding Box

```
25      cnts, h = cv2.findContours(frame_thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
26      if cnts:                                                    # leere Kontur ausschliessen
27          cnt = max(cnts,  key=cv2.contourArea)                   # finde groesste Kontur
28          x, y, w, h = cv2.boundingRect(cnt)                      # Koordinaten des Bounding Rectangles
29          cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 1)   # Rechteck zeichnen
30      cv2.imshow("Groesste Kontur", frame)
```

Inhalt

1. Einführung
2. Bildaufnahme
3. Bildvorverarbeitung
4. Bildsegmentierung
5. Merkmalsextraktion
6. **Klassifizierung → siehe Lerneinheit ST Maschinelles Lernen VL03**



Kamera
Objektiv
Beleuchtung

Kontrast optimieren,
Rauschminderung

Fehler
finden

Fehler
vermessen

Fehlertyp
bestimmen

… mehr dazu im WP AIBV

HAW Hamburg
Fakultät TI
Technik und Informatik

Beispiel: Druckbildkontrolle, Fa. EyeC, Hamburg

Angewandte industrielle Bildverarbeitung

Einführung und Beleuchtung

joerg.dahlkemper@haw-hamburg.de
Raum 6.82

Aufbau der Vorlesung

HAW Hamburg
Fakultät TI
Technik und Informatik

Bildverarbeitung mit neuronalen Netzen		
9 Klassifizierung mit CNN	10 Object Detection und Segmentierung	
Neuronale Netze		
7 Einführung in KNN	8 Training von KNN	
Klassische Bildverarbeitung		
4 Vorverarbeitung	5 Merkmalsextraktion	6 Klassifizierung
Bildverarbeitungsgrundlagen		
1 Einführung + Beleuchtung	2 Objektiv + Kamera	3 Hands-on Workshop

Gesichtserkennung > Einführung

Face Detection

locating faces in a photograph

Wo ist ein Gesicht im Bild?

Face Recognition
auch Face Identification

one-to-many mapping for a given face against a database of known faces

Wer ist diese Person?

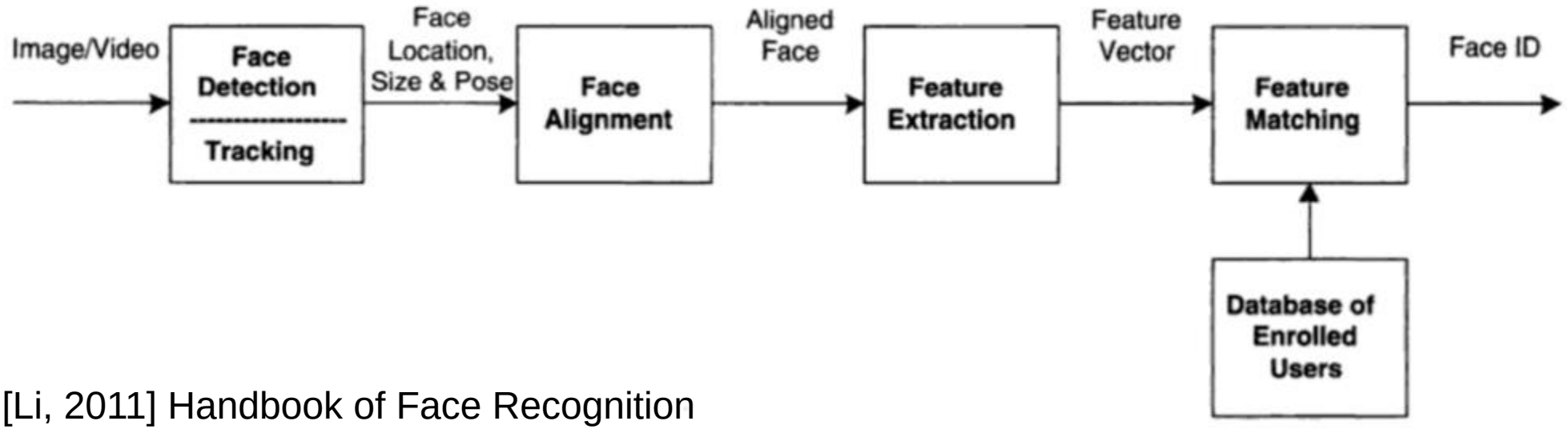
Face Verification
auch Face Authentication

one-to-one mapping of a given face against a known identity

Ist dies tatsächlich die Person?

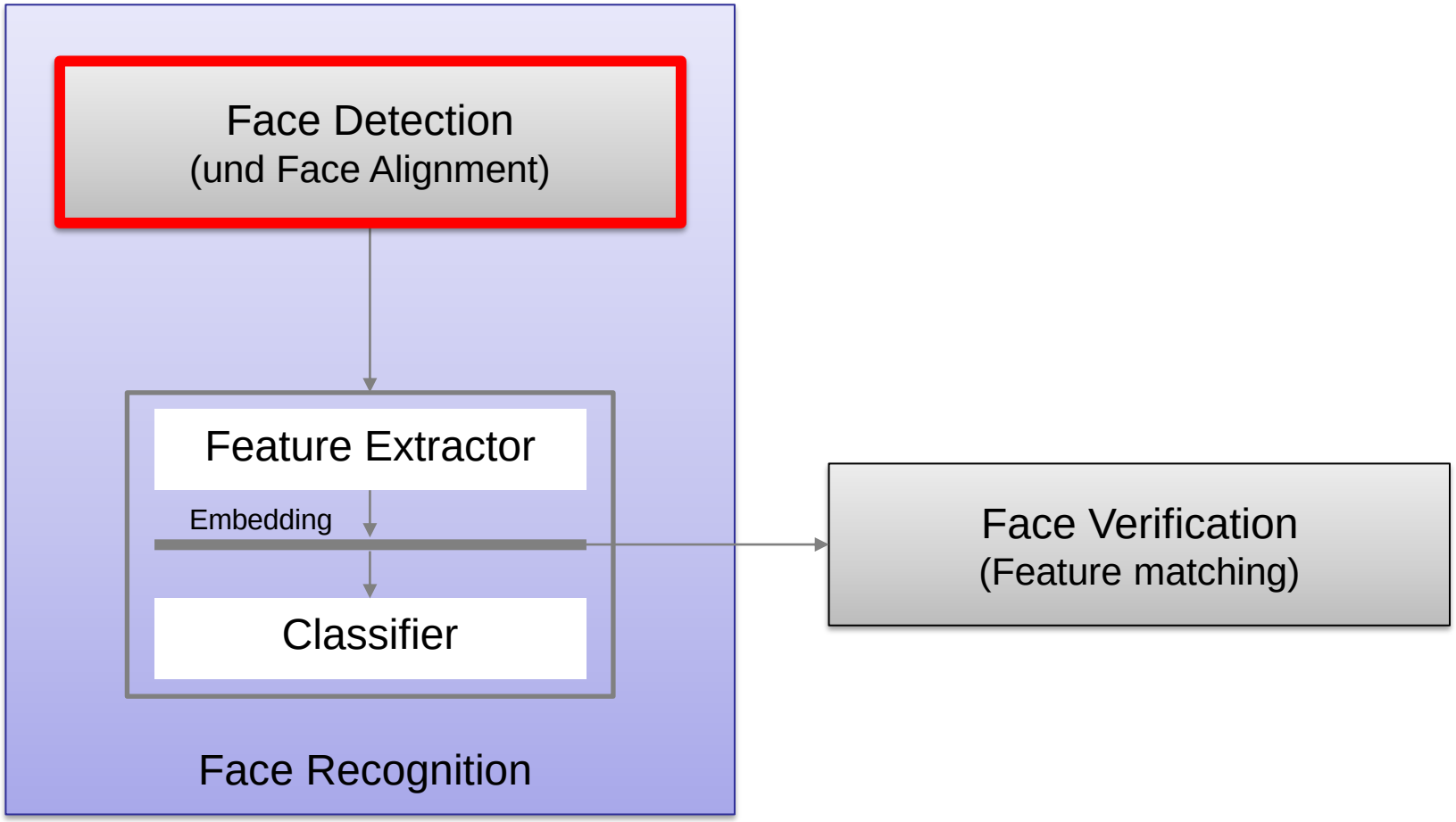
[Quelle für Begriffe: Li; Jain: Handbook of Face Recognition. 2011]

Grundsätzlicher Ablauf von Face Recognition



[Li, 2011] Handbook of Face Recognition

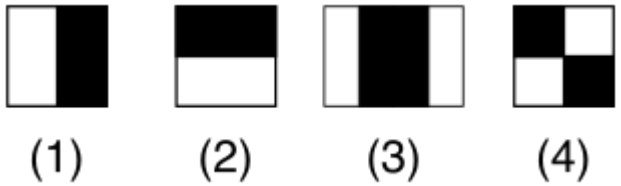
Face Detection vs. Face Recognition/Verification



Face detection > Vergleich gängiger Ansätze

Haar-Feature Classifier

- manuell entwickelte Filter als Features



- benötigt nur wenig Daten zum Training
- Implementierung in OpenCV
  https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html
- **schnelle Gesichtserkennung** (recheneffizient)
- funktioniert nur eingeschränkt bei geneigtem oder teilweise verdecktem Gesicht

[Viola 2004]

Multi-Task CNN (MTCNN)

- Feature Maps durch Training bestimmt
- sehr hoher Trainingsaufwand aber pre-trained Nets verfügbar (`pip install mtcnn`)
- Feedforward ist rechenaufwändiger als bei Haar-Feature Classifier
- **robuste Gesichtserkennung** auch bei geneigtem oder teilweise verdecktem Gesicht

⇒ Deep Learning

Zhang, Kaipeng; Zhang, Zhanpeng; Li, Zhifeng, Qiao, Yu: Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. 2016
https://arxiv.org/ftp/arxiv/papers/1604/1604.02878.pdf

Haar-Feature Classifier > Kernideen

Übergeordnetes Ziel:

Entwicklung einer sehr schnellen Gesichtserkennung, um diese auch bei zeitkritischen Anwendungen mit begrenzter Rechenperformanz einsetzen zu können

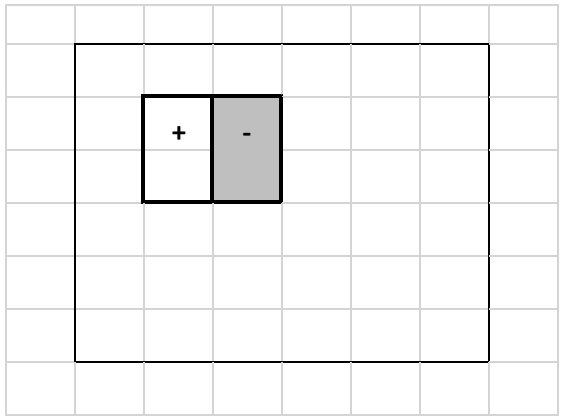
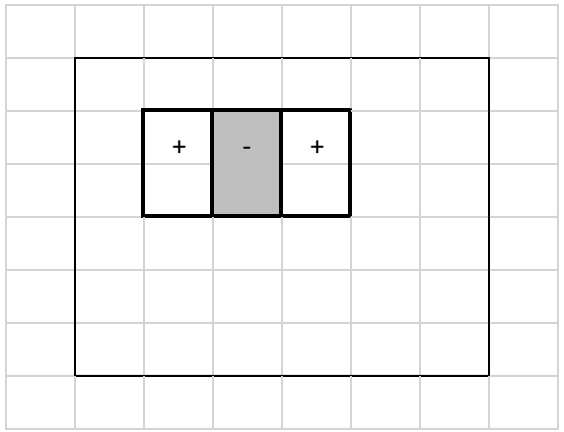
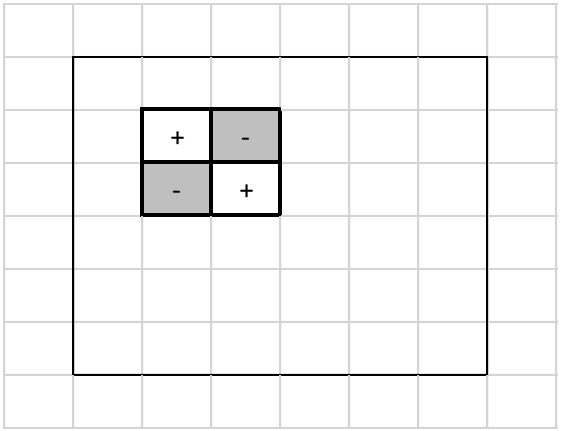
Drei Kernideen, die eine effiziente Erkennung von Gesichtern ermöglichen:

a) Integral Image:
   Bildrepräsentation, die eine schnelle Merkmalsberechnung auf Basis von einkanaligen Grauwertbildern unterstützt
b) Klassifizierungsalgorithmus:
   AdaBoost
c) Kaskade von Klassifizierern:
   Unterdrückung des Hintergrundes

Haar Classifier > Merkmale

- In Anlehnung an Haar-Basisfunktionen (siehe [Papgeorgiou 1998])

- Unterscheidung von 3 Arten von Merkmalen

Two-rectangle feature	Three-rectangle feature	Four-rectangle feature
		

- Aufteilung des Bildes in 24 x 24 derartige Felder

Haar-Classifier > Integral Image

Idee: schnelle Berechnung der Differenzen über kumulierter Pixelwerte

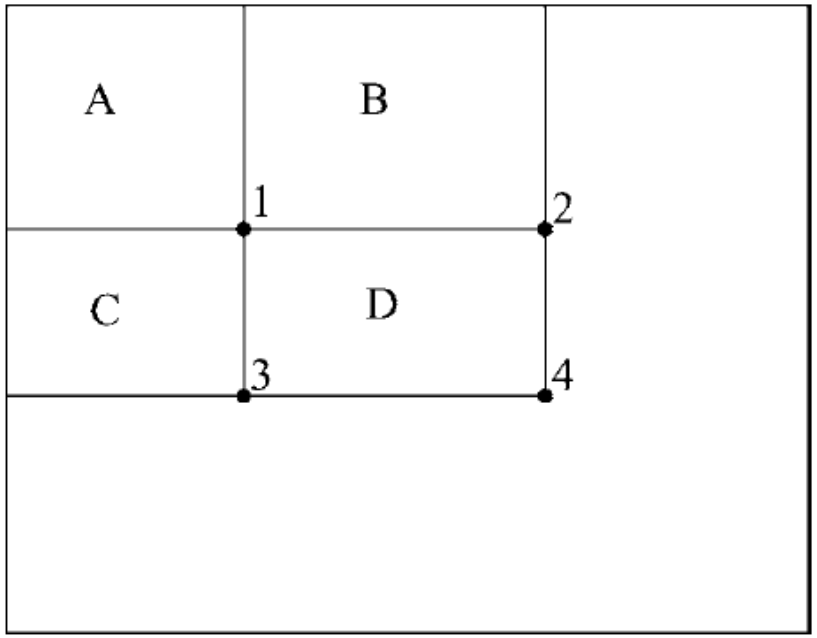
Berechnung:

- „Integral Image“ = Summe aller Grauwerte links und oberhalb einer Position (x,y)
- Einmalige Berechnung des Integral Image ii(x, y) aus dem Grauwertbild i(x, y)
  $ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y')$

Auswertung:

- Aufgabe: Berechnen Sie die Pixelsumme D aus den ii an den Positionen 1 … 4

- Summe D = ii(4) – ii(2) – ii(3) + ii(1)



[Viola 2004]

Haar-Classifier > Klassifizierung

- Klassifizierung basiert auf Haar-Basisfunktionen
- Klassifizierer wählt eine geringe Anzahl aus der Vielzahl der möglichen Haar-Filter aus, jeder Filter stellt eine Repräsentation des Bildes dar und kostet daher Rechenzeit (vgl. Feature Maps bei CNN)
- AdaBoost wählt im Training die relevanten Haar-Filter aus, indem einzelne Classifier kombiniert werden, von denen jeder nur ein Merkmal nutzen darf

Haar-Classifier > Kaskadierung von Klassifizierern

Ziel

- Geschwindigkeitssteigerung, indem nur vielversprechende Bildregionen im Detail analysiert werden

Lösungsansatz

- Attentional filter = Auswahl der Regions of Interest
  (vgl. Selective Search bei Object detection)
  Attentional filter wird etwa 50% des Bildes aussortieren, bei der 99% der Bilder behalten werden (auf Recall optimiert, weniger auf Precision)
- Kaskadierung von 38 Merkmalen
  zuerst schnell zu berechnende Merkmalsextraktion, wenn diese nicht ansprechen, wird das Bildsegment verworfen, so dass der nächstkomplexere nur noch auf der reduzierten Teilmenge arbeitet (Degenerated Decision Tree)

Vorbereitung

- Trainierten Classifier in Arbeitsverzeichnis laden
- Bereits mit OpenCV auf Rechner installiert, typischerweise hier:
  - `Ubuntu: /usr/share/opencv/haarcascades`
  - `Windows: C:\Users\XXX\anaconda3\envs\ST\Lib\site-packages\cv2\data`

- Ablauf

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) # nur fuer Grauwertbilder
faces = face_cascade.detectMultiScale(frame_gray, 1.3, 5)
(xpos, ypos, width, height) = faces[0]  # faces ist Liste von Bounding Boxes
```

scaleFactor:
um welchen Faktor wird das Bild je Iteration verkleinert

minNeighbours:
Anzahl Nachbarn,
damit Face behalten wird

Haar-Classifier > Implementierung


```python
import cv2

cap = cv2.VideoCapture(0)
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

while cv2.waitKey(1) != 27:

    ret, frame = cap.read()

    if ret:
        # face detection
        frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cascade.detectMultiScale(frame_gray, 1.3, 5)

        if len(faces) > 0:
            (xpos, ypos, width, height) = faces[0]
            cv2.rectangle(frame, (xpos, ypos), (xpos+width, ypos+height), (255, 0, 0), 2)

        cv2.imshow('Follow the face', frame)

cap.release ()
cv2.destroyAllWindows ()
```

Hinweis zu Parallelisierung per Threads

Threading

Das Senden der Befehle zu der Drohne muss parallel zu der Ausführung der Gesichtserkennung erfolgen. Hier bietet sich der Einsatz von Threads an.

```python
# nicht im jupyter notebook, sondern als Python-Datei ausfuehren

import threading
import time

mode = 0

def show_mode():
    while True:
        if mode == 1:
            print("\nMode: 1")
        elif mode == 2:
            print("\nMode: 2")
        time.sleep(0.5)


thread = threading.Thread(target=show_mode)
thread.daemon = True
thread.start()


while(mode < 3):
    mode = int(input("Set mode 0/1/2 or any higher number for exit:"))
```

Python

Was Sie mitnehmen sollen …

1. Sie können Bilder und Videos aus einer Webcam einlesen.
2. Sie kennen den Aufbau eines Bildes und können gezielt auf Bereiche und Farbkanäle zugreifen.
3. Sie können Rechenoperationen auf Bilder anwenden.
4. Sie haben die Problematik des Datentyps uint8 bezüglich eines möglichen Überlaufs verstanden.
5. Sie können eine Schwellwertsegmentierung durchführen.
6. Sie können eine Bounding Box eines Segments berechnen und visualisieren.
7. Sie können den Unterschied zwischen Face Detection, Face Recognition und Face Verification erklären.
8. Sie können die grundsätzliche Funktion eines Face Detection Systems erklären.
9. Sie können die Funktionsweise eines Haar-Classifiers für Face Detection erklären und dabei auf die Idee des Integral Image, des Klassifizierers und die Kaskadierung eingehen.
10. Sie können eine Software zu Face Detection implementieren.

Sie können einfache Bildverarbeitungsanwendungen in OpenCV erfolgreich umsetzen.

Offene Fragen

Bildaufnahme:

- Wie kann die Kamera der Drohne über den PC angesteuert und per OpenCV ausgelesen werden?
- Wie kann mit `grab/retrieve` statt `read` ein Delay durch Bilder in der Queue minimiert werden.

Multitasking:

- Wie kann gleichzeitig zur Bildverarbeitung mit der Bibliothek `threading` ein Steuerbefehl an die Drohne gegeben werden?

Face Detection:

- Welche Verfahren können mit vertretbarem Implementierungsaufwand unter Nutzung von OpenCV und Python alternativ realisiert werden?

Face Detection (advanced)

- Wie ist das Programm zu erweitern, damit die Foto-Drohne nur ein ganz bestimmtes Gesicht verfolgt?

Quellen

- [Viola, 2004] Paul Viola and Michael J. Jones. Robust real-time face detection. International Journal of Computer Vision, 57(2):137–154, 2004. https://link.springer.com/content/pdf/10.1023/b:visi.0000013088.49260.fb.pdf

- [Bolme, 2010] David S. Bolme, J. Ross Beveridge, Bruce A. Draper, and Man Lui Yui. Visual object tracking using adaptive correlation filters. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. https://www.cs.colostate.edu/~vision/publications/bolme_cvpr10.pdf