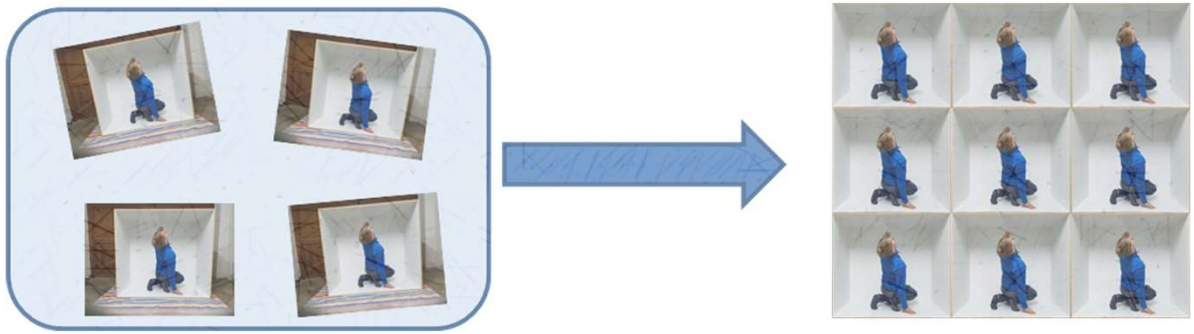
Digitale Bildverarbeitung 1

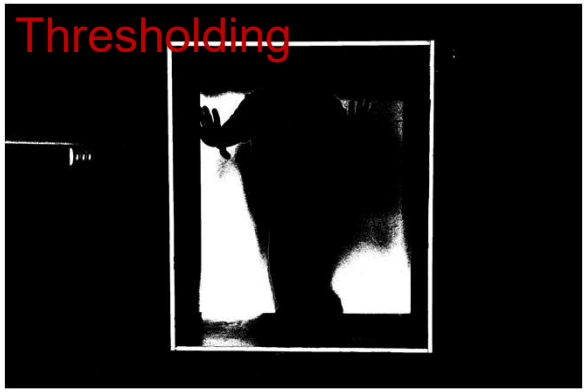
Einführung in die digitale Bilderverarbeitung
für Informatikstudierende im Bachelor

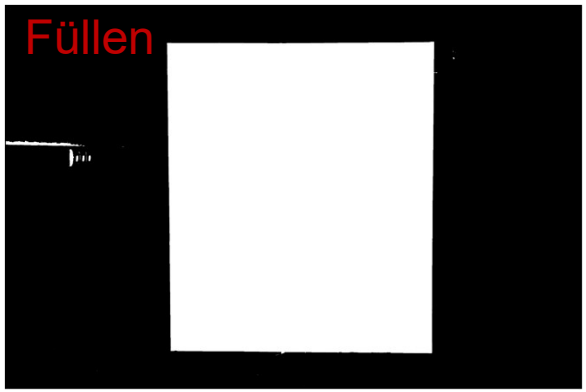
Vorlesung: Michael Möller – michael.moeller@uni-siegen.de
Übungen: Hannah Dröge – hannah.droege@uni-siegen.de

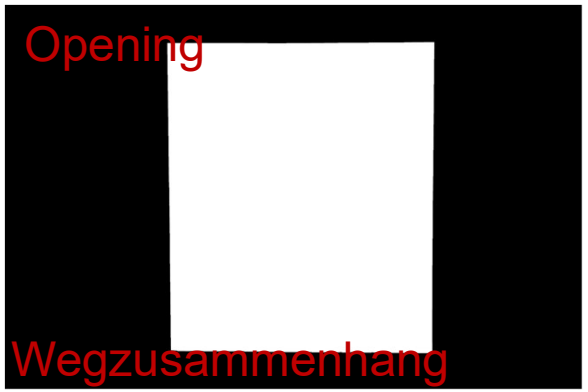


Bildsegmentierung

Vor langer, langer Zeit ...
… haben wir über Segmentierung und morphologische Operationen gesprochen.







Eckpunkte der Kiste

Wie kommen wir nun an die Koordinaten der Ecken?

Mögliche Idee:

Weißer Pixel mit kleinstem Abstand zum Punkt (0,0) gibt linke obere Ecke!

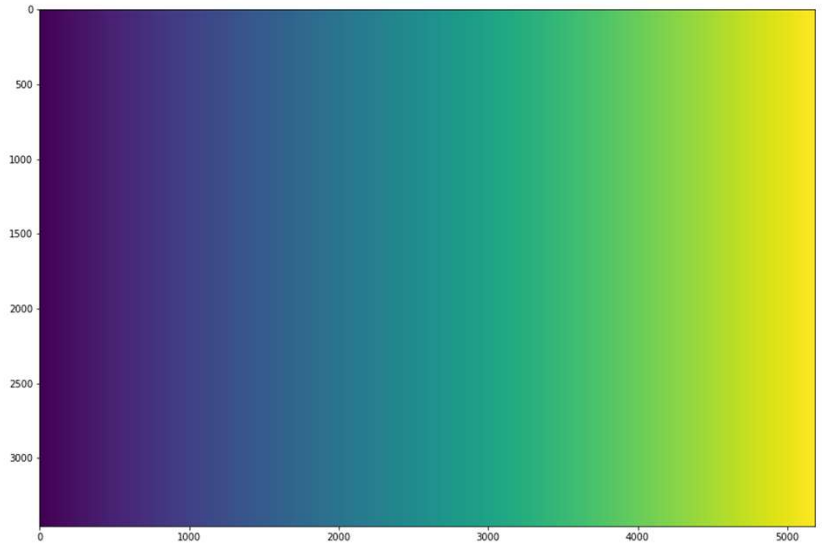
(alle anderen Ecken analog mit (n_y -1,0), (0,n_x-1), (n_x-1,n_y-1))

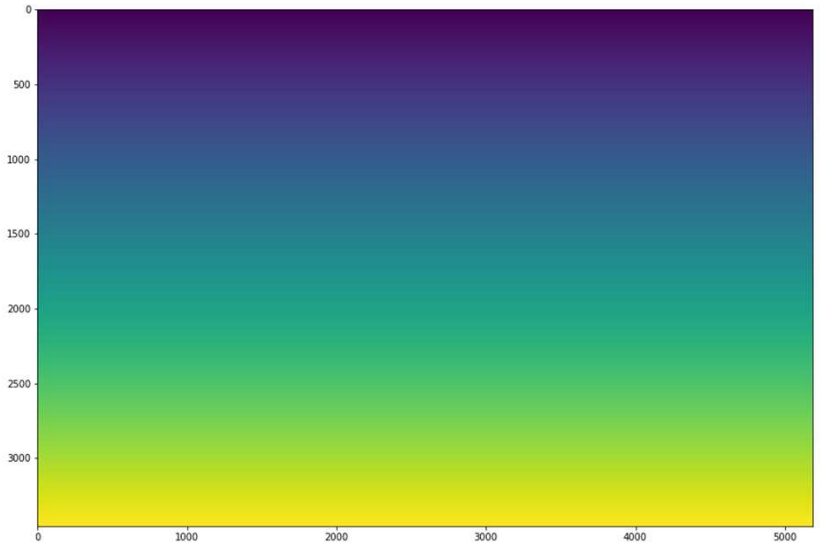
Eckpunkte der Kiste

Wie setzt man dies im Code um?

```
ny,nx = finalBinary.shape

# create a meshgrid that contains the coordinates of all points
xv, yv = np.meshgrid(np.arange(0,nx), np.arange(0,ny))
```

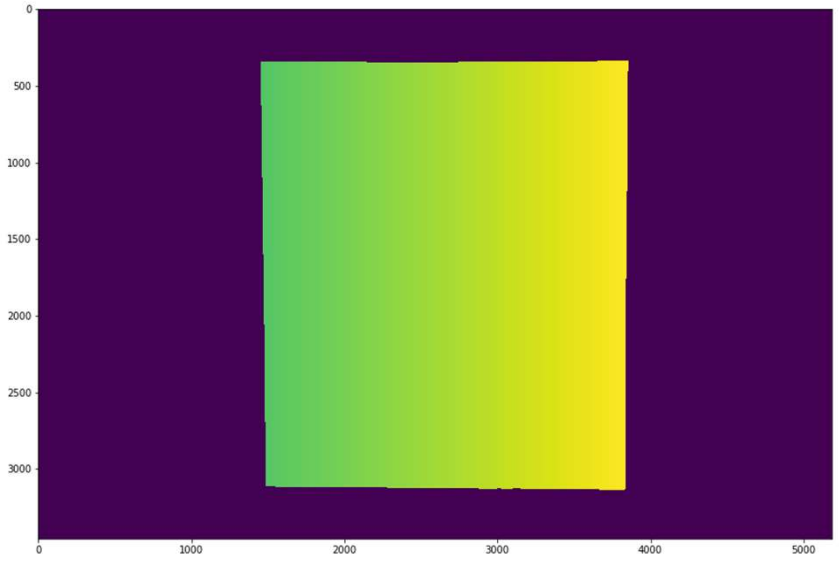


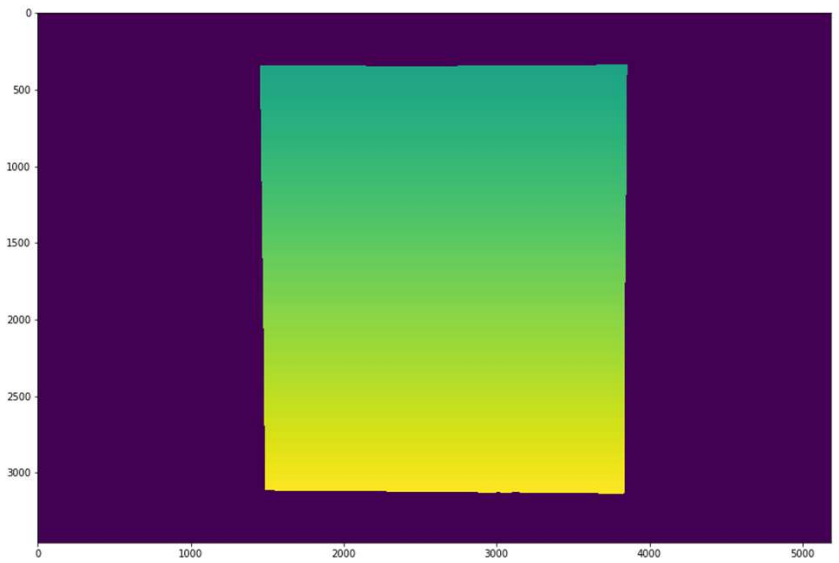


Eckpunkte der Kiste

Wir setzen die Koordinaten der Punkte an denen die binäre Maske nicht 1 ist auf „weit weg“ – hier z.B. auf (-ny, -nx)

```
# make sure that points where the finalBinary mask is 0 are never picked
xv[~finalBinary] = -nx
yv[~finalBinary] = -ny
```





Eckpunkte der Kiste

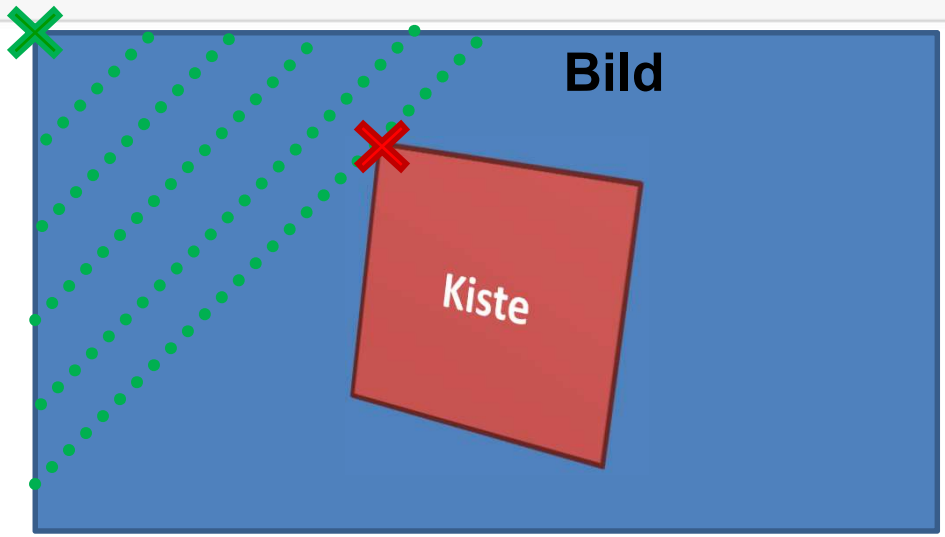
- For-Schleife über alle 4 Eckpunkte des Bildes.
- Für jeden Eckpunkt wird der Abstand zwischen allen Koordinaten und dem jeweiligen Eckpunkt im L1-Sinne (Manhattan Distanz) berechnet.
- Man merkt sich die Koordinaten des zugehörigen Punktes.

```
corners = np.array([[0,0],[0,nx-1],[ny-1,0],[ny-1,nx-1]])
ind = -np.ones([4,2])
for i in range(0,4):
    temp = np.abs(np.array(xv) - corners[i,1]) + np.abs(np.array(yv) - corners[i,0])
    ind[i,:] = np.unravel_index(np.argmin(temp, axis=None), temp.shape)

print(ind)
```

```
[[ 346. 1456.]
 [ 338. 3853.]
 [3111. 1488.]
 [3135. 3833.]]
```

Geometrische Vorstellung:



Eckpunkte der Kiste

```
fig = plt.figure(figsize=(20,10))
plt.imshow(np.stack((finalBinary,)*3, axis=-1)*testimg, 'gray')
for i in range(0,4):
    plt.plot(ind[i,1], ind[i,0], 'gx', markersize=22, markeredgewidth=4)
plt.show()
```

Finaler Setzkasten

For Schleife über alle Bilder im Ordner. Für jedes Bild

- Segmentiere die Kiste und bestimme die Eckpunkte wie gerade besprochen
- Berechne die Homographie, die die Kiste zu einem perfekten Rechteck macht
- Verwende ein resizing, welches jedes Rechteck auf die gleiche Größe interpoliert
- Schneide das Rechteck aus und speichere es

Erstelle ein großes (schwarzes) Bild welches

(yNr * Höhe eines Rechtecks) x (xNr * Breite eines Rechtecks) x 3

groß ist.

For Schleife über alle gespeicherten Rechtecke

- Schreibe das aktuelle Rechteck an die nächste „freie“ (schwarze) Stelle im großen Bild

Finaler Setzkasten

Datenschutz-Setzkasten

Hier war in der Vorlesung ein Setzkasten mit den Datenschutz-Bildern. Diesen Veröffentliche ich vorsichtshalber nicht online.

Segmentierung – mehr als Kisten

Bislang haben wir Kisten segmentiert, indem wir die Annahme gemacht haben, Kisten sind heller als der Hintergrund.

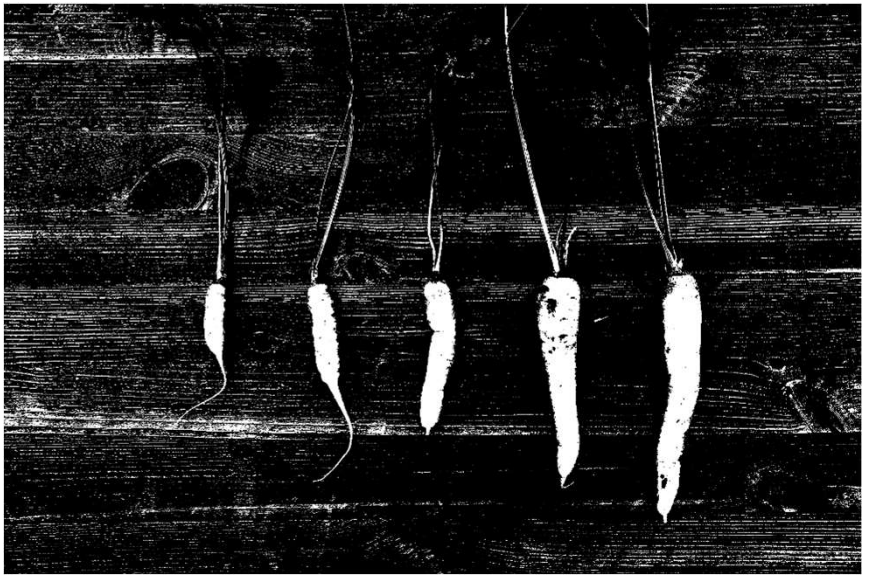
Für viele Aufgaben spielt die Farbe eine Rolle, z.B. hier Möhren auf einem Fließband vermessen, exemplarisch etwa

Segmentierung – mehr als Kisten

Erste Idee: Möhren haben einen stärkeren Rotton als der Hintergrund.

Thresholding des Rotkanals?

```
plt.figure(figsize=(10,15))
plt.imshow(testimg[:,:,0]>0.7)
plt.show()
```

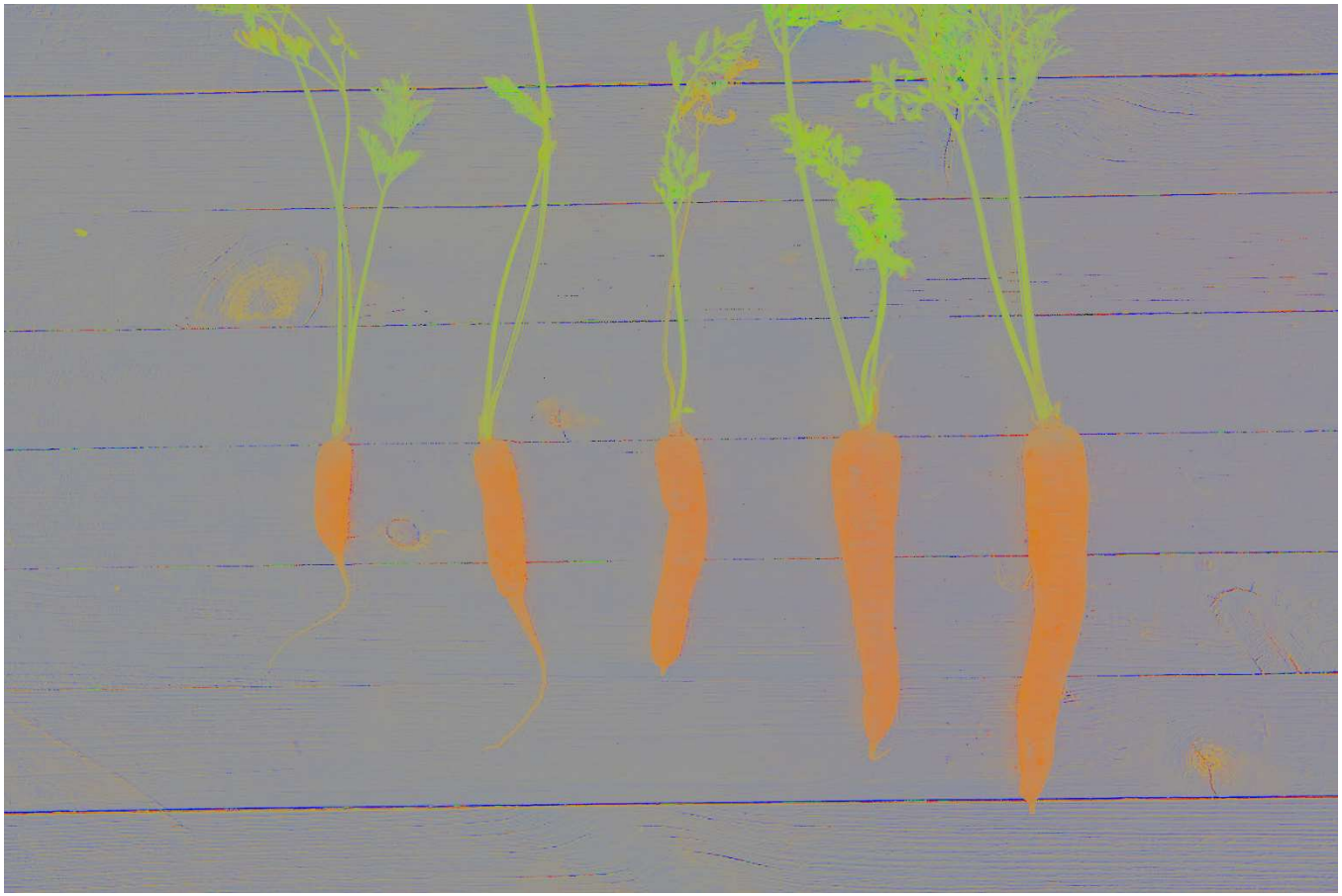


Nicht sehr gut… Grund: Alle weißen/hellen Pixel haben auch hohe Werte im Rotkanal!

Segmentierung – mehr als Kisten

Besser: Herausdividieren der Helligkeit,
z.B.

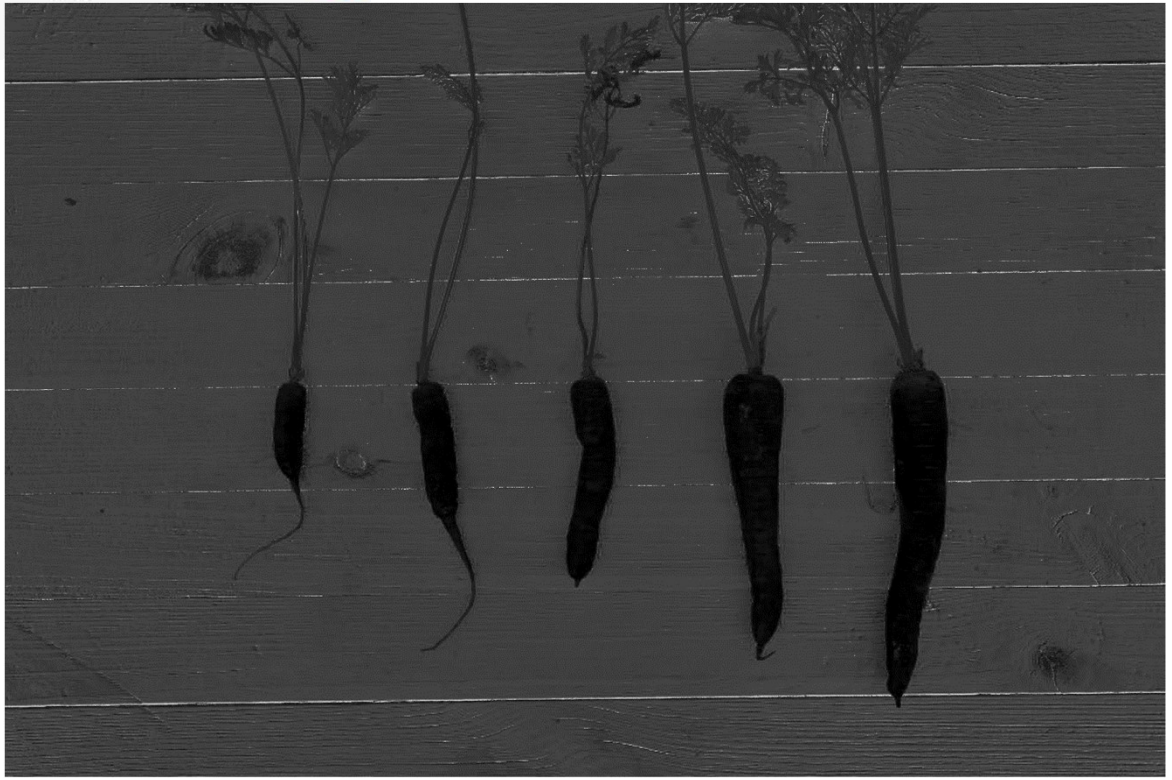
```
greyImg = 0.001+ np.sqrt(testimg[:,:,0]**2 + testimg[:,:,1]**2 + testimg[:,:,2]**2)
colorsOnly = testimg/np.repeat(greyImg[:, :, np.newaxis], 3, axis=2)
```



Segmentierung – mehr als Kisten

Dann, z.B. Winkel zwischen einer typischen Möhrenfarbe und jedem Pixel berechnen

```
color = np.ones(testimg.shape)
color[:,:,0] = 0.828
color[:,:,1] = 0.509
color[:,:,2] = 0.232
test = np.sum(color*colorsOnly, axis = 2)
test = np.arccos(test)
```



Segmentierung – mehr als Kisten

Dann: Übliche Schritte (Threshold, Closing, Hole filling, Opening, Objekte mit Mindestgröße Auswählen)

Segmentierung – mehr als Kisten

Außer Farben könnten natürlich auch andere Merkmale (Features) eine Rolle spielen.

Es gibt viel Literatur zur Extraktion von Features, die für bestimmte Aufgaben gut geeignet sind.

Abstrakt gesehen ist das Ziel immer eine Rechenoperation zu finden, welche das gewünschte Objekt stark von den anderen abhebt.

Wir haben uns hier nur Pixel-für-Pixel Operationen angeschaut, welche nur bei den einfachsten Segmentierungsproblemen erfolgreich sind.

Bei komplexen Aufgaben wird die Rechenoperation, die gute Feature extrahiert, heutzutage fast immer durch maschinelle Lernverfahren bestimmt.

Falls noch Zeit ist, werde ich am Ende der Vorlesung einen kurzen Einblick hierzu geben.

Clustering

Unser nächstes Thema erstmal: Analyse von Bildern ohne genau zu wissen, welche Objekte enthalten sind, bzw. wie diese aussehen.

Abstrakt: Teile ein gegebenes Bild in 2 (später k) viele Regionen auf.

Clustering

Idee: Wüssten wir die Farbe vom Flamingo, könnten wir unsere übliche Segmentierung verwenden.

Wohl berühmtestes Clusteringverfahren: **k-means Algorithmus**. Suche gleichzeitig nach 2 (später k) vielen Farben (Features) und einer Zuordnung jedes Pixels zu einer dieser Farben.

Schritt 1: Angenommen wir hätten die Farben

$$c^{vorder} \in \mathbb{R}^3 \qquad c^{hinter} \in \mathbb{R}^3$$

von Vorder- und Hintergrund.

Dann könnten wir einen Pixel an der Stelle (i, j) dem Vordergrund zuordnen falls

$$\|f_{i,j,:} - c^{vorder}\| < \|f_{i,j,:} - c^{hinter}\|$$

Ansonsten gehört er zum Hintergrund.

Clustering

Formal:

Schritt 1: Gegeben $c^{vorder} \in \mathbb{R}^3$ und $c^{hinter} \in \mathbb{R}^3$ bestimme die Segmentierung $m \in \mathbb{R}^{n_y \times n_x}$ des Bildes $f \in \mathbb{R}^{n_y \times n_x \times 3}$ mittels

$$m_{i,j} = \begin{cases} 1 & \text{falls } \|f_{i,j,:} - c^{vorder}\| < \|f_{i,j,:} - c^{hinter}\|, \\ 0 & \text{sonst.} \end{cases}$$

Schritt 2: Gegeben eine Segmentierung $m \in \mathbb{R}^{n_y \times n_x}$ des Bildes $f \in \mathbb{R}^{n_y \times n_x \times 3}$

Bestimme geeignete Vorder- und Hintergrundfarben, indem die Durchschnittsfarben vom Vorder- und Hintergrund gebildet werden.

$$c^{vorder} = \frac{1}{\sum_{i=0}^{n_y-1} \sum_{j=0}^{n_x-1} m_{i,j}} \sum_{i=0}^{n_y-1} \sum_{j=0}^{n_x-1} f_{i,j,:}\ m_{i,j}$$

$$c^{hinter} = \frac{1}{\sum_{i=0}^{n_y-1} \sum_{j=0}^{n_x-1} (1 - m_{i,j})} \sum_{i=0}^{n_y-1} \sum_{j=0}^{n_x-1} f_{i,j,:}\ (1 - m_{i,j})$$

Clustering

Einige Ergebnisse: Auf RGB

Clustering

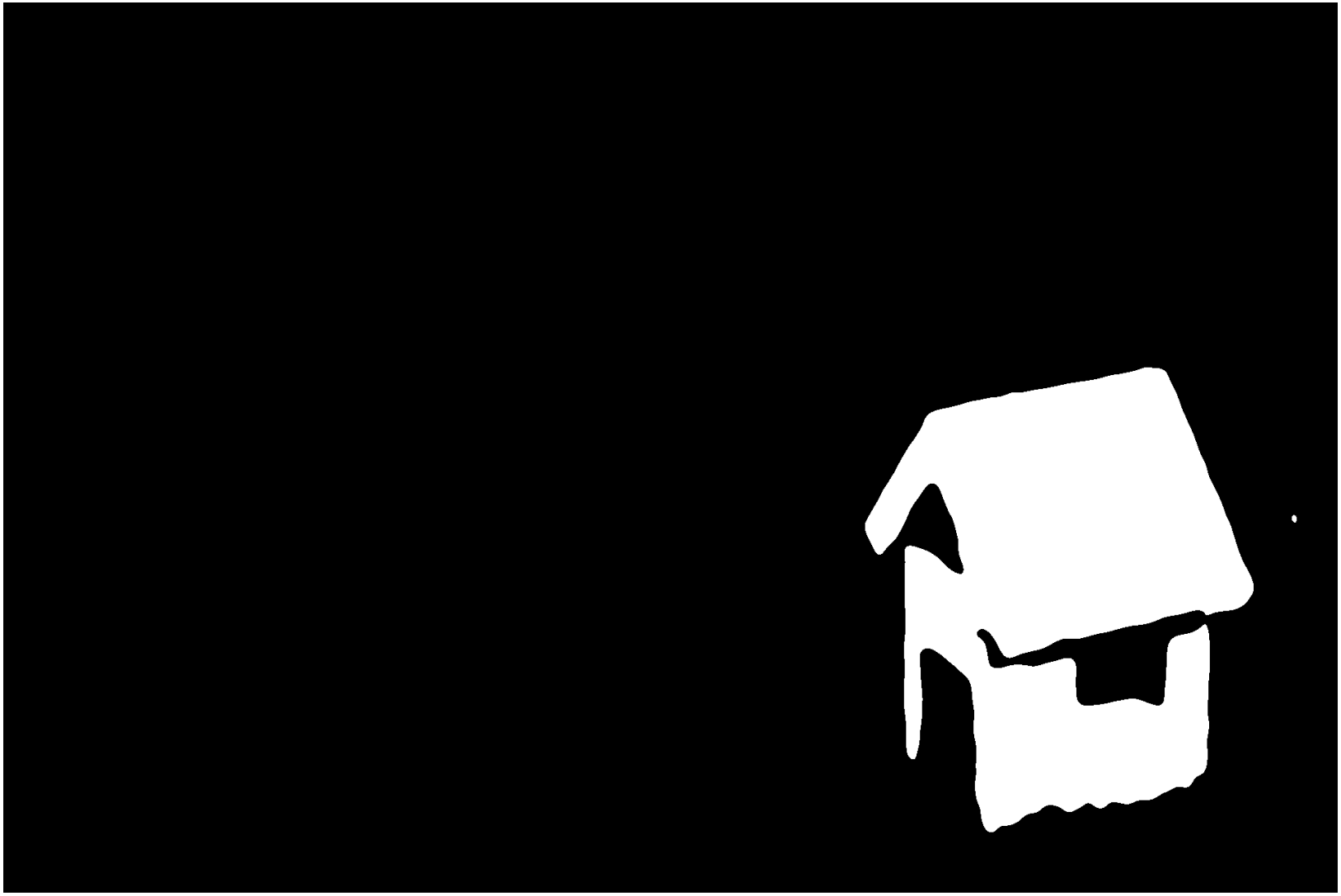
Einige Ergebnisse: Auf RGB

Clustering

Einige Ergebnisse: Gauß-gefiltert

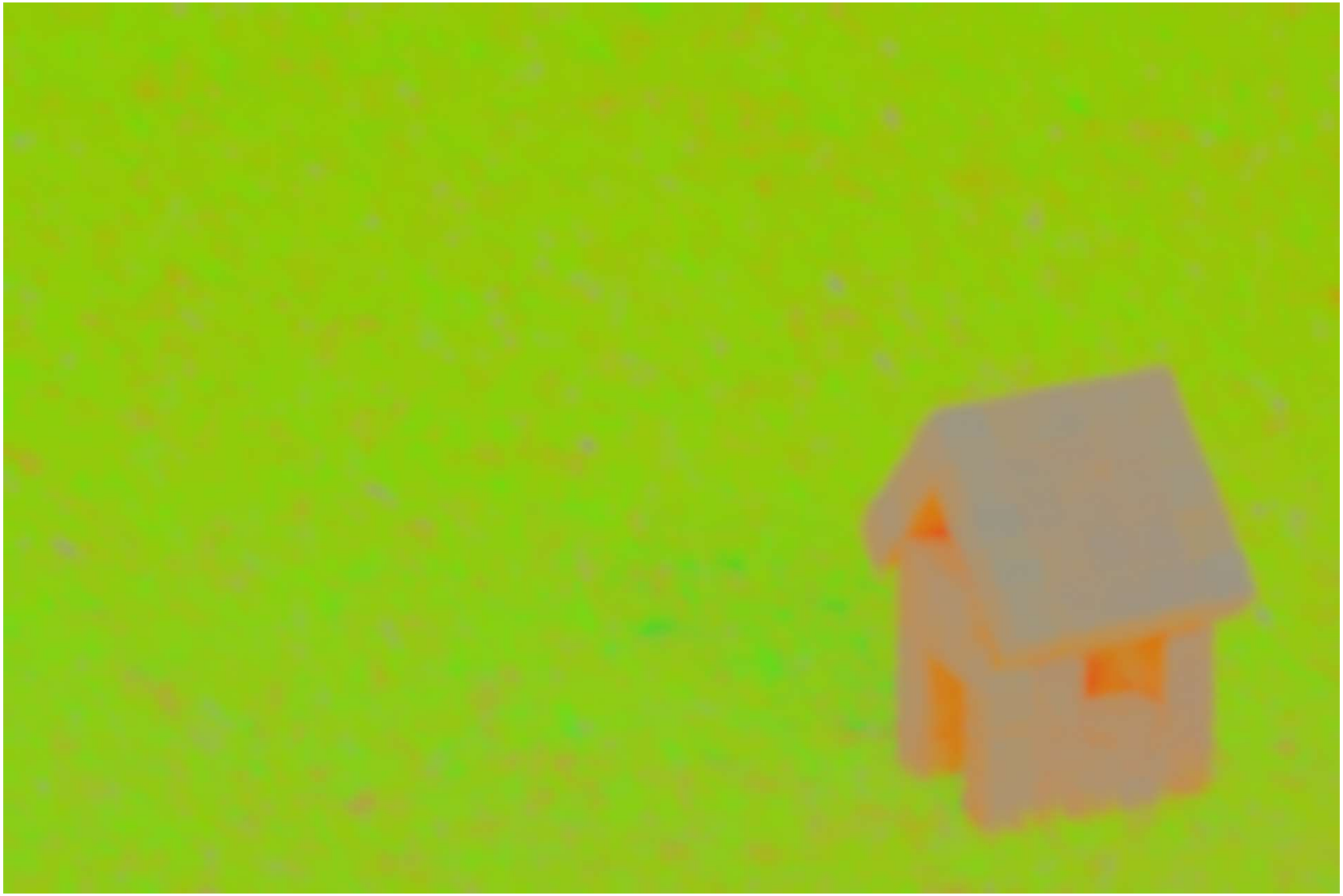
Clustering

Einige Ergebnisse: Gauß-gefiltert



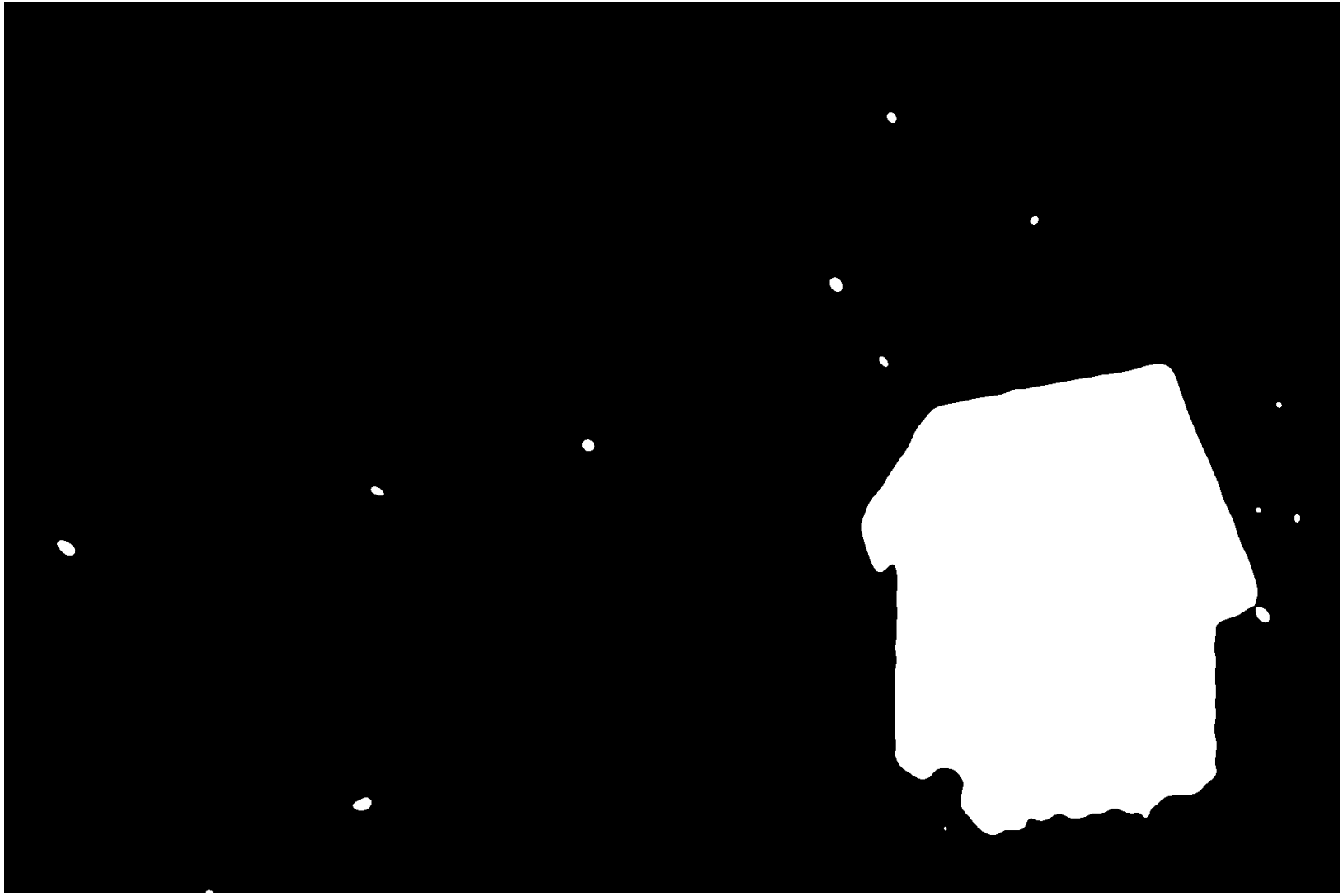
Clustering

Einige Ergebnisse: Gauß-gefiltert + Intensitätsnormalisiert



Clustering

Einige Ergebnisse: Gauß-gefiltert + Intensitätsnormalisiert



Clustering

Erweiterung des k-means Algorithmusses auf mehr als 2 Klassen

Schritt 1: Gegeben $c^l \in \mathbb{R}^3,\ l \in \{1, \ldots, k\}$ bestimme die Segmentierung $m \in \mathbb{R}^{n_y \times n_x \times k}$ des Bildes $f \in \mathbb{R}^{n_y \times n_x \times 3}$ mittels

$$m_{i,j,:} = e_l \quad \text{falls} \quad \|f_{i,j,:} - c^l\| \leq \|f_{i,j,:} - c^t\| \quad \forall t \in \{1, \ldots, k\}$$

e_l: l-te Einheitsvektor

Schritt 2: Gegeben eine Segmentierung $m \in \mathbb{R}^{n_y \times n_x \times k}$ des Bildes $f \in \mathbb{R}^{n_y \times n_x \times 3}$

Bestimme geeignete Farben für jedes Segment, indem die Durchschnittsfarben vom jeweiligen Segment gebildet wird.

$$c^l = \frac{1}{\sum_{i=0}^{n_y-1} \sum_{j=0}^{n_x-1} m_{i,j,l}} \sum_{i=0}^{n_y-1} \sum_{j=0}^{n_x-1} f_{i,j,:}\ m_{i,j,l}$$

Clustering


```
# This is not a good initialization: Sometimes we get empty clusters!
k=6
ny,nx,nc = testimg.shape
c = np.random.rand(nc,k)

maxIter = 20
for i in range(0,maxIter):
    # step 1: update mask
    diff = np.zeros((ny,nx,k))
    for l in range(0,k):
        cl_3d = c[:,l][np.newaxis, np.newaxis,:]
        cl_image = np.repeat(np.repeat(cl_3d, ny, axis=0), nx, axis=1)
        diff[:,:,l] = np.sum((cl_image- testimg)**2,axis = 2)

    indi = np.argmin(diff, axis=2)


    # stept 2: update colors
    for l in range(0,k):
        ml = indi==l
        count = np.max([np.sum(ml),1])
        ml_3d = np.repeat(ml[:,:,np.newaxis], nc, axis=2)
        c[:,l] = 1/count*(ml_3d*testimg).sum(axis=0).sum(axis=0)
```

Clustering

Eingabebild

Clustering

K-means – jeder Pixel hat die Farbe c^l des Clusters zu dem er gehört. (hier k=12)