Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

MIN-Fakultät
Fachbereich Informatik
Arbeitsbereich SAV/BV (KOGS)

Nützliche Bildverarbeitungs-Verfahren

BV-Praktikum im Sommersemester 2017

Leonie Dreschler-Fischer, David Mosteller
und Benjamin Seppke

Agenda

- Wiederholung: Bilder in Racket
- Schwellenwertverfahren/Binarisierung
  - Manuell
  - Histogramm-basiert
- Kantenfinder
  - Morphologisch
  - Canny
- Festlegen des freizustellenden Bereichs
- Koordinatensysteme und Projektionen auf unterschiedliche Spielfeldgrößen

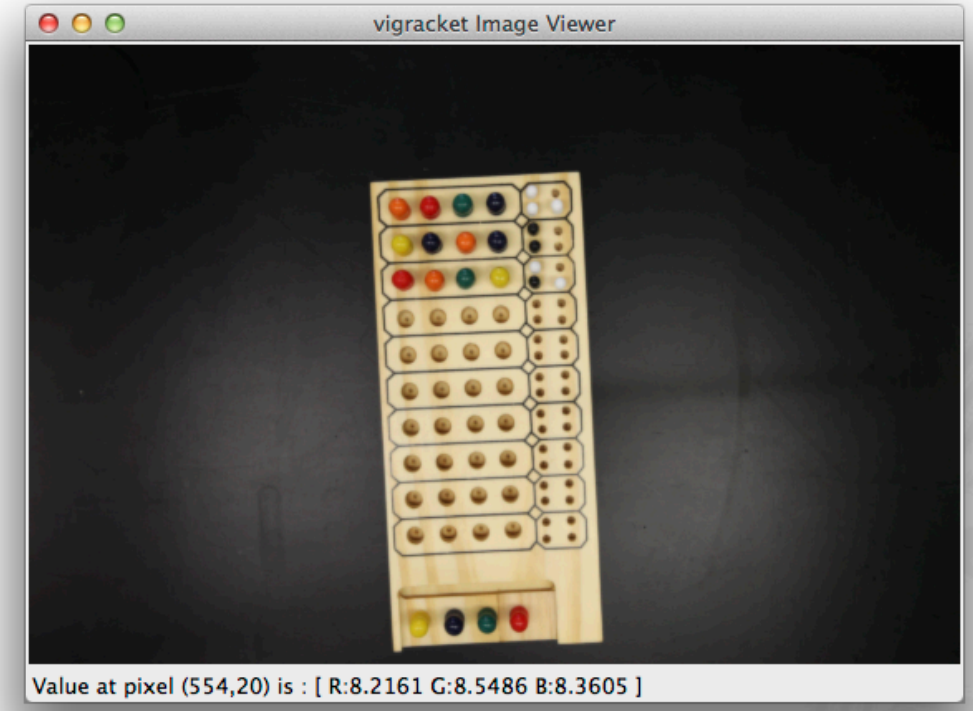
Wiederholung: Bilder in Racket

- Bilder werden in Racket als Listen von „carrays“ dargestellt:

  ‘(<carray>) für Grauwertbilder

  ‘(<carray> <carray> <carray>) für RGB-Farbbilder
- Bedeutung nur implizit, zum Beispiel beim Viewer vorhanden.
- Die meisten Algorithmen können auf beliebig lange Listen von carrays angewendet werden.
- Trennung zwischen Bild-Datensatz (oben) und Bild-Bitmap (mit image->racket-image)
  - VigRACKET-Operationen nur mit obiger Repräsentation kompatibel!
  - 2htdp/image-Operationen nur nach Konvertierung möglich!

Das Beispiel für heute:



- Vorverarbeitung: Verkleinern des Ausgangsbildes
- Aufgabe 1: Isolieren (Freistellen) des Spielfeldes aus dem Bild
  - Schwellenwertbasierte Verfahren
  - Kantenbasierte Verfahren
  - Segmentierung und Labelling
- Aufgabe 2: „Abtasten“ des freigestellten Spielfeldes in Bezug auf den Spielzustand
  - Koordinatentransformationen
  - Gitter festlegen

```
(define image_dir "/Users/seppke/...“)
(define img (load-image (build-path image_dir "IMG_8553.JPG")))
(define resized_img (resizeimage-percent img 10))
```

Binarisierung

Bei vielen Anwendungen ist es nützlich, nur zwischen zwei diskreten Grauwerten zu unterscheiden, "schwarz“ und "weiß", oder "1" und "0", "Objekt" und "Hintergrund".

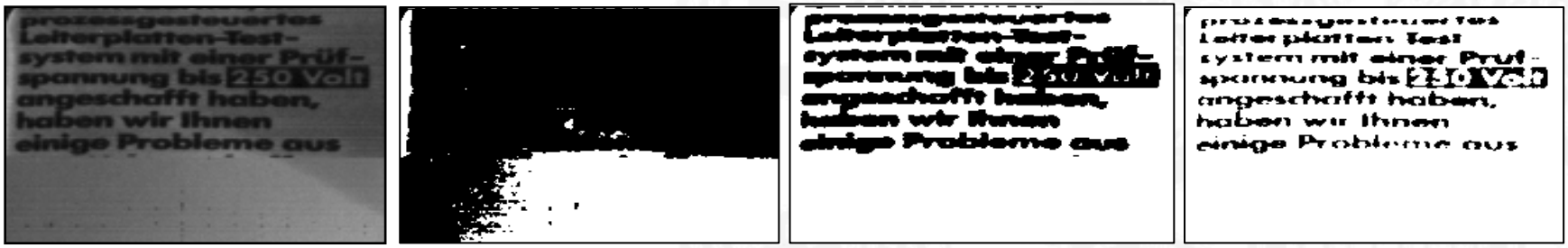
Beispiel: Objekt von Hintergrund trennen ("segmentieren")

Binarisieren = Bild in Binärbild transformieren

Binarisieren kann bei kontinuierlichem oder höher aufgelöstem Grautonbild erfolgen, auch bei verarbeiteten Bildern, z.B. Gradientenbetragsbildern.

Schwellenwertvergleich: $$g(x, y) = \begin{cases} 0 & \text{für } g(x, y) < T \\ 1 & \text{für } g(x, y) \geq T \end{cases}$$ T ist Schwellenwert

Grauwerte 0 – 255 | Schwellenwert 64 | Schwellenwert 32 | Schwellenwert 16



Schwellenwertbestimmung durch Probieren

Ein Schwellenwert, der Objekt und Hintergrund perfekt trennt, existiert nicht immer.

Auswahl durch Probieren:
Wähle Schwellenwert, bis eine Bildeigenschaft erfüllt ist, z.B.

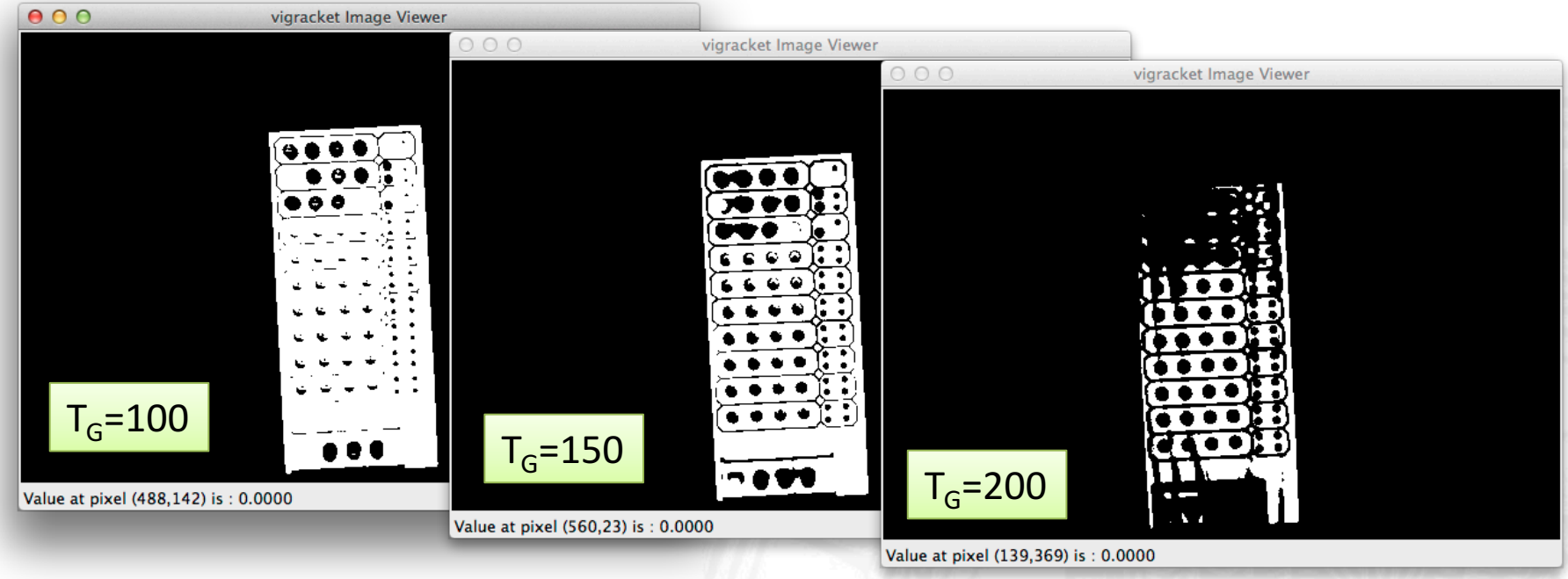
- Verhältnis von schwarzen zu weißen Pixeln $q = \frac{\#\ \text{weiße Pixel}}{\#\ \text{schwarze Pixel}} \Rightarrow q_0$
- Linienbreite $Linienbreite \Rightarrow \mathrm{d}_0$
- Zahl der zusammenhängenden Komponenten $Komponentenzahl \Rightarrow \mathrm{n}_0$

Bei logarithmischer Suche kann die Zahl der Probeversuche klein gehalten werden.

Beispiel:

Um einen Schwellenwert $0 \leq T \leq 255$ auszusuchen, braucht man höchstens 8 Versuche zur Bestimmung der besten Annäherung an q_0.

Beispiel mit manuell ermittelten Schwellenwerten

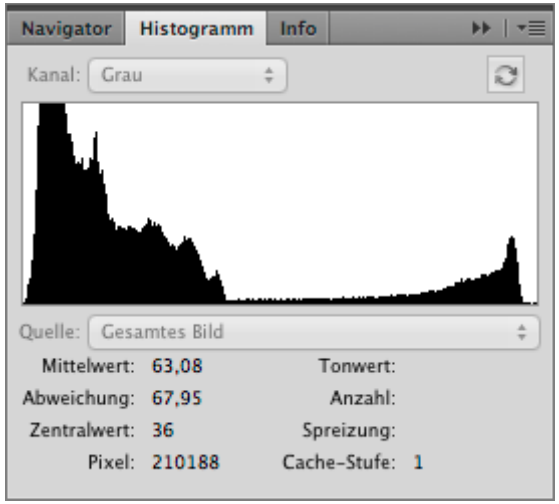


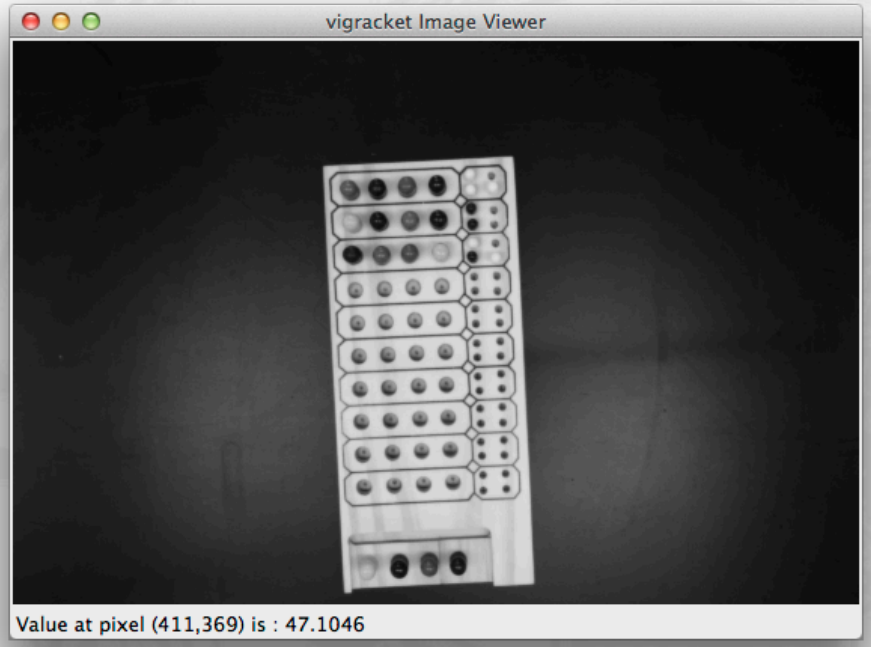

```
(define work_img (image->green resized_img))
(show-image (image-map (lambda (x) (if (> x 100.0) 255.0 0.0)) work_img))
(show-image (image-map (lambda (x) (if (> x 150.0) 255.0 0.0)) work_img))
(show-image (image-map (lambda (x) (if (> x 200.0) 255.0 0.0)) work_img))

(define threshold_img (image-map (lambda (x) (if (> x 150.0) 255.0 0.0)))
```

Bildhistogramme

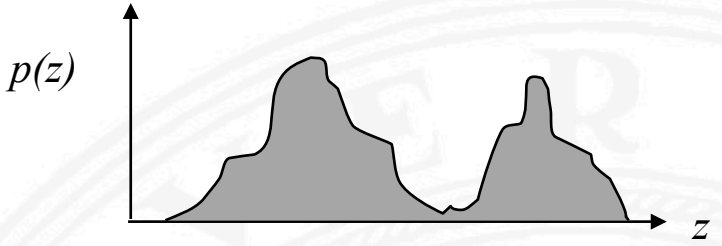
- Idee: Automatische Bestimmung des Schwellenwerts anhand der Bildstatistik
- Histogramm: Haufigkeitsdiagramm aller Helligkeitswerte (meist im Intervall [0..255]) des Bildes
- Beispiel:





Schwellenwertbestimmung aufgrund der Grauwertverteilung

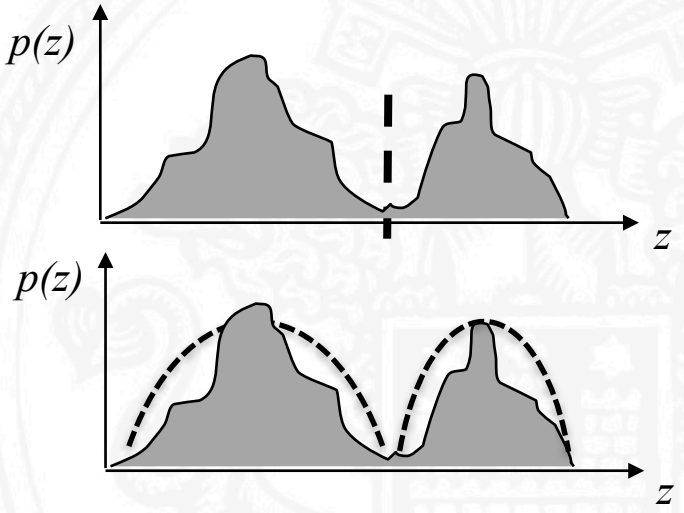
Eine Grauwertverteilung (Wahrscheinlichkeitsdichte oder Histogramm) kann bimodal sein:



Was ist ein plausibler Schwellenwert?

Zwei übliche Methoden zur Bestimmung eines plausiblen Schwellenwertes:

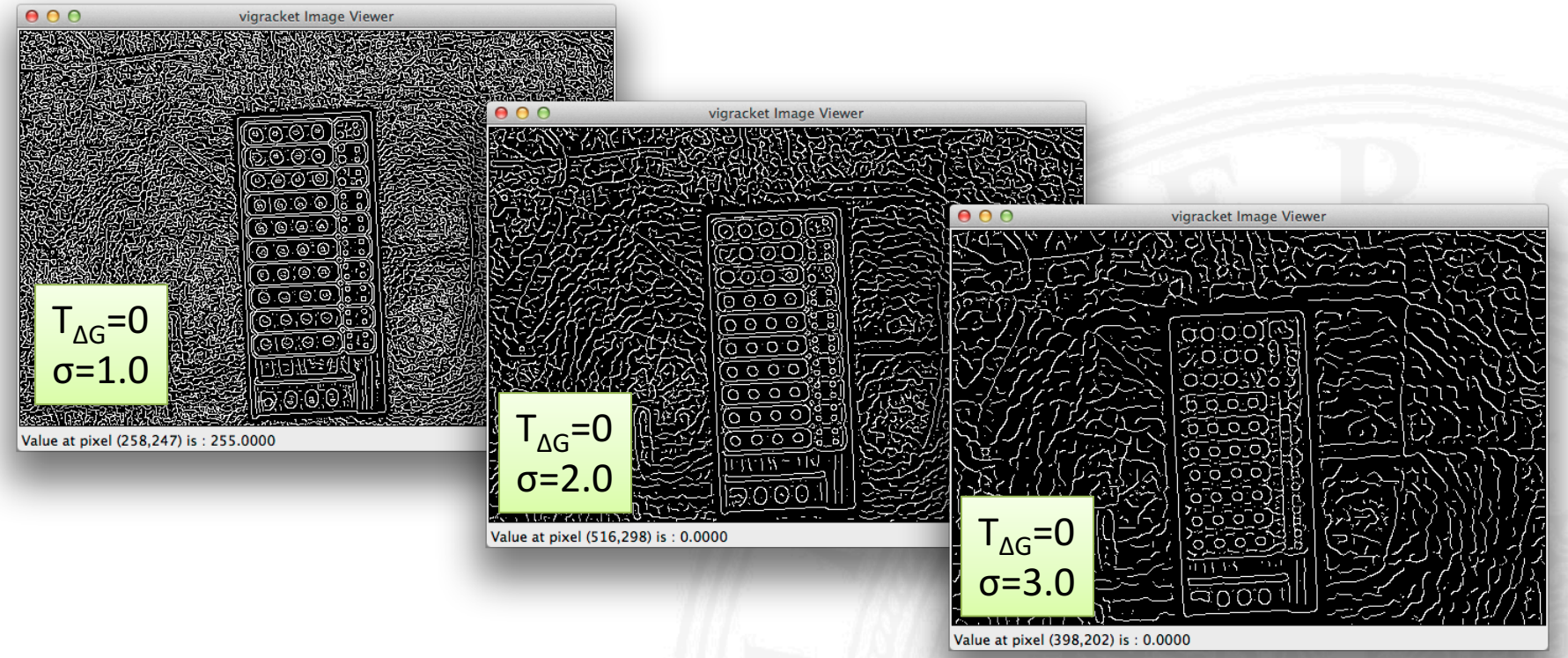
1. Suche das „Tal“ zwischen zwei „Hügeln“
2. Passe Hügelschablonen an und bestimme (deren) Schnittpunkt.



Kantenfinder

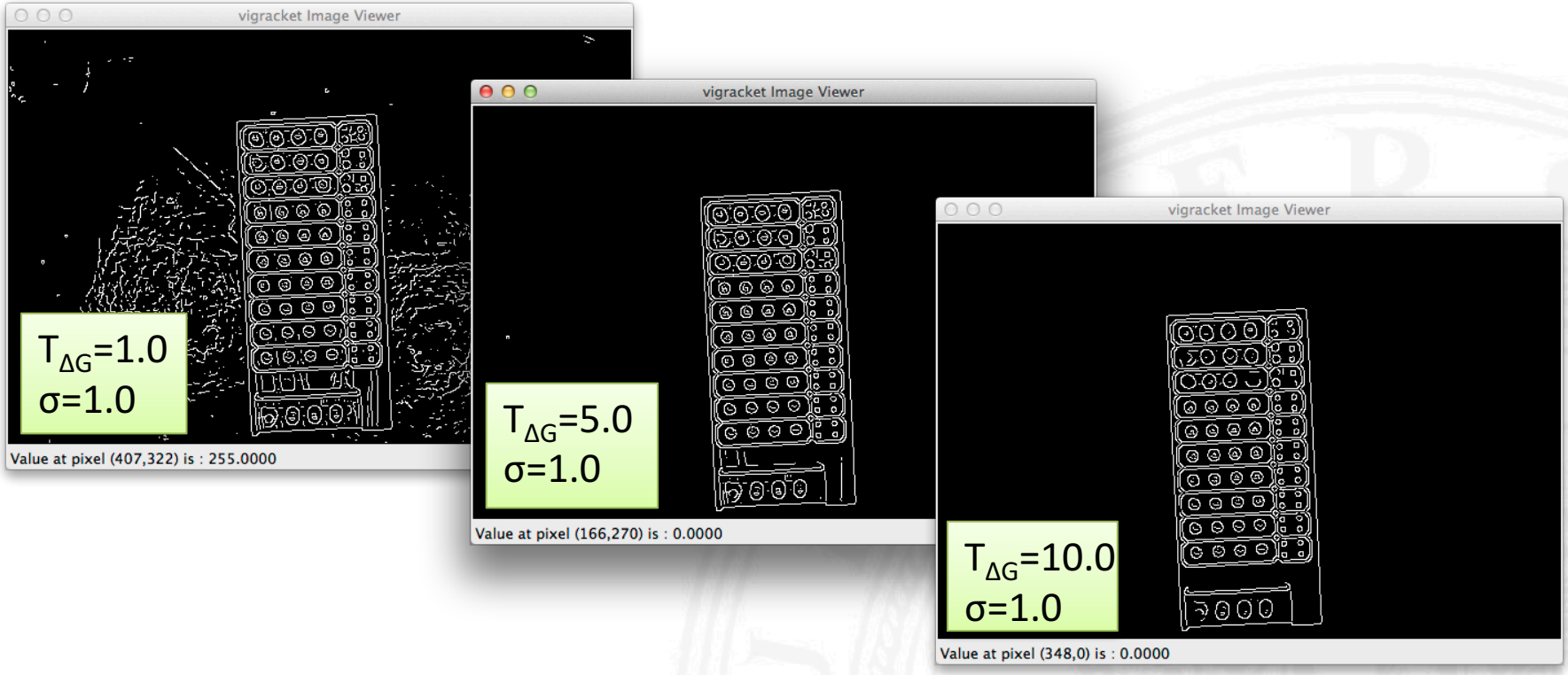
- Duale Sichtweise auf das Problem: Finde das Spielfeld anhand der begrenzenden Kanten
- Verfahren:
  - Maximum der ersten Ableitung der Bildfunktion
  - Nullstellen der zweiten Ableitung der Bildfunktion
- Beispiel: Canny-Kantendetektor
  - Parameter:
    - Unterer Schwellenwert der Kantenstärke
    - Skala auf der die Kanten ermittelt werden
  - Vorgehensweise:
    - Auffinden und Ablaufen von lokalen Maxima
    - Hysterese Verfahren

Beispiele: Anwendung des Canny-Kantendetektors




```
(show-image (cannyedgeimage work_img 1.0 0.0 255.0))
(show-image (cannyedgeimage work_img 2.0 0.0 255.0))
(show-image (cannyedgeimage work_img 3.0 0.0 255.0))
```

Beispiele: Anwendung des Canny-Kantendetektors




```
(show-image (cannyedgeimage work_img 1.0  1.0 255.0))
(show-image (cannyedgeimage work_img 1.0  5.0 255.0))
(show-image (cannyedgeimage work_img 1.0 10.0 255.0))

(define canny_img (cannyedgeimage work_img 1.0 10.0 255.0))
```

Festlegen des freizustellenden Bereichs

- Idee: Suche nach dem begrenzenden Rechteck um das (gedrehte) Spielfeld herum
- Unterteilung des Bildes in Regionen durch Labelling:

```
(define canny_labels (labelimage  canny_img))
```

- Für jede Region können Statistiken/Features abgerufen werden:

```
(define canny_features (extractfeatures canny_img canny_labels))
```

- Wir suchen die größte Region, die nicht Hintergrund ist!

Verwendung von Regionsfeatures

- Beispiel:

```
(image->list canny_features)

'(((0.0 +inf.0 +inf.0 -inf.0 -inf.0 0.0 0.0 3.4028234663852886e+38 -3.4028234663
(203636.0 0.0 0.0 561.0 373.0 280.66784667968875 185.9539794921875 0.0 0.0 0.0 0.0
(822.0 206.0 76.0 346.0 366.0 275.6910095214844 221.77859497070312 255.0 255.0 25
(386.0 209.0 81.0 334.0 234.0 284.9844665527344 127.30052185058594 255.0 255.0 25
(83.0 297.0 84.0 326.0 104.0 310.4337463378906 93.67469787597656 255.0 255.0 255.
(201.0 212.0 85.0 295.0 108.0 254.05970764160156 96.3631820678711 255.0 255.0 255
(17.0 315.0 87.0 320.0 92.0 317.23529052734375 89.35294342041016 255.0 255.0 255.
(51.0 255.0 89.0 268.0 104.0 261.3529357910156 96.60784149169922 255.0 255.0 255.
(47.0 276.0 89.0 287.0 103.0 281.3404235839844 95.68085479736328 255.0 255.0 255.
(50.0 236.0 91.0 248.0 105.0 241.82000732421875 98.18000030517578 255.0 255.0 255
(50.0 217.0 92.0 230.0 105.0 223.24000549316406 98.72000122070312 255.0 255.0 255
(2.0 326.0 97.0 326.0 98.0 326.0 97.5 255.0 255.0 255.0 0.0)
(15.0 294.0 103.0 300.0 108.0 297.1333312988281 105.93333435058594 255.0 255.0 25
(26.0 309.0 106.0 326.0 114.0 319.3461608886719 107.61538696289062 255.0 255.0 25
(175.0 213.0 107.0 296.0 130.0 252.23428344726562 117.17142486572266 255.0 255.0
(23.0 300.0 107.0 307.0 115.0 303.9565124511719 111.26087188720703 255.0 255.0 25
...
```

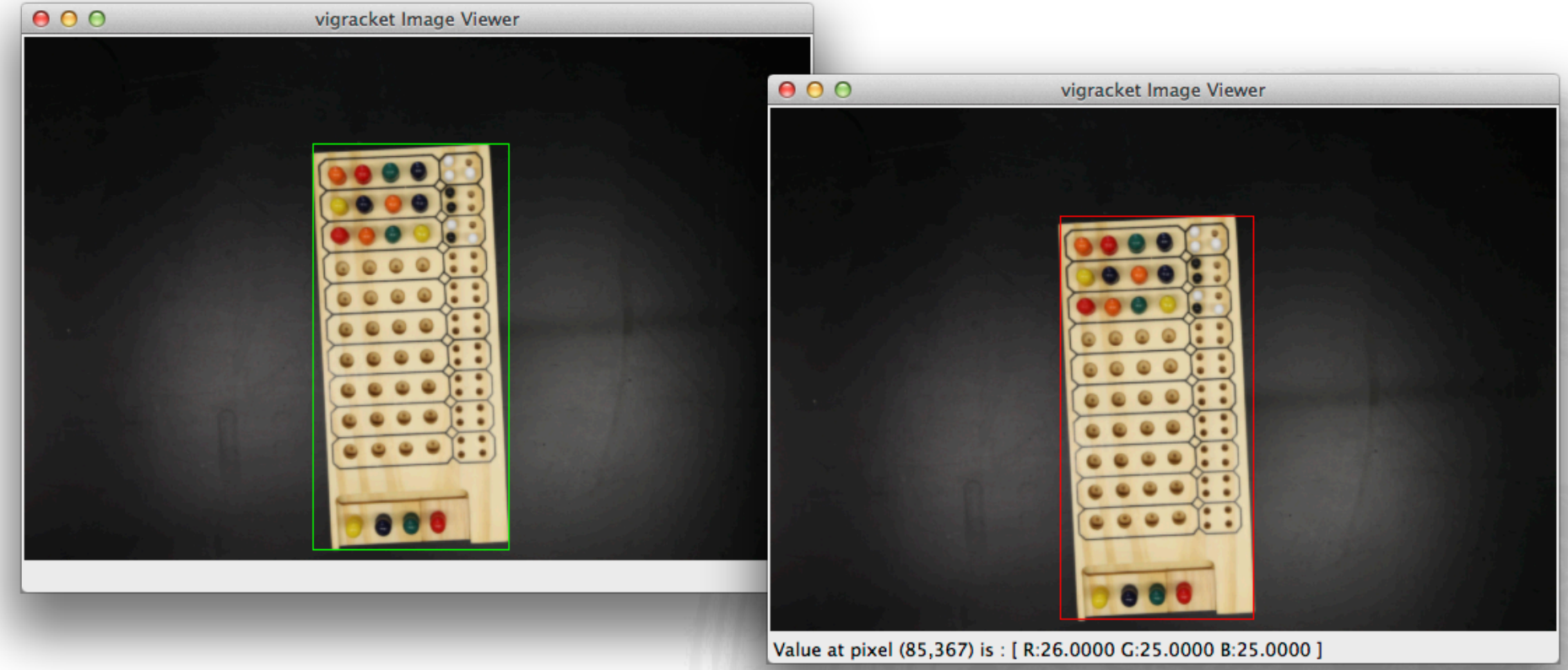
Hintergrund

Bestimmung des umschließenden Rechtecks

```
;;Find the region with the most pixels assigned to
(define (largestRegionID features [idx 0] [max_idx 0])
  (if (= idx (image-height features))
      max_idx
      (largestRegionID features
                       (+ idx 1)
                       (if (> (image-ref features 0 idx 0)
                              (image-ref features 0 max_idx 0))
                           idx
                           max_idx))))

;;Uses the above function to find the bounding box of the largest region
(define (largestRegionBBox image)
  (let* ((labels (labelimage image))
         (features (extractfeatures image labels))
         (max_region_id (largestRegionID features 2 2)))
    (vector (inexact->exact (image-ref features 1 max_region_id 0)) ;left
            (inexact->exact (image-ref features 2 max_region_id 0)) ;upper
            (inexact->exact (image-ref features 3 max_region_id 0)) ;right
            (inexact->exact (image-ref features 4 max_region_id 0)) ;lower
           )))
```

Beispiel: Erkennung des begrenzenden Rechtecks




```
(show-image (racket-image->image (overlay-bboxes resized_img (list bbox1) '(green))))
(show-image (racket-image->image (overlay-bboxes resized_img (list bbox2) '(red))))
```

Ausschneiden des Bildes anhand des begrenzenden Rechtecks

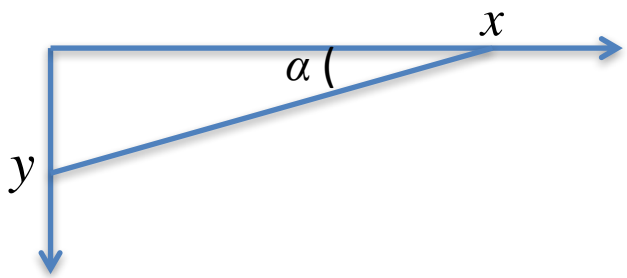
- Funktion aus subimage verwenden!
- Anwenden auf das (mit Canny) erzielte Ergebnis:

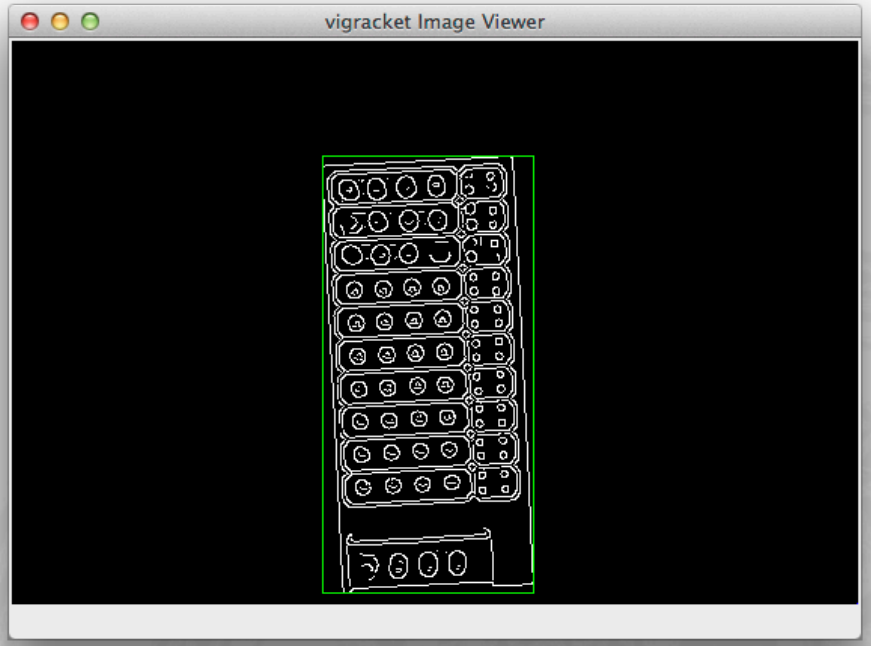
```
(define crop1_img (subimage   resized_img
                              (vector-ref bbox1 0)
                              (vector-ref bbox1 1)
                              (vector-ref bbox1 2)
                              (vector-ref bbox1 3)))
```

- **Achtung:** Leichte Drehung nach wie vor vorhanden..

Erkennung der Rotation als Nachverarbeitung

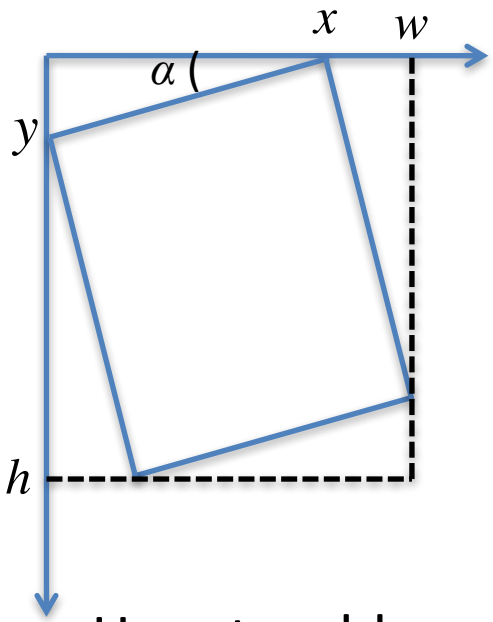
- **Idee**: Betrachten der Schnittpunkte zwischen begrenzendem Rechteck und Maske/Canny Ergebnisbild
- Geometrische Approx. der Drehung durch Schnittpunktposition





$$\alpha = \arctan\left(\frac{y}{x}\right)$$

Alternative Rotationserkennung



$$\alpha = \arctan\left(\frac{y}{x}\right)$$

$$= \arctan\left(\frac{w-x}{x}\right)$$

$$= \arctan\left(\frac{y}{h-y}\right)$$

$$\ldots$$

- Hauptprobleme: x,y unscharf definiert
- Außerdem möglich:
  Begrenzende Rechtecke durch Polygone ersetzen
  Vorteile:
  - Informationen zum Ausschneiden direkt vorhanden
  - Transformation (inkl. Rotation) direkt aus Eckpunkten ablesbar

Ende der Präsentation

Vielen Dank für die Aufmerksamkeit!

Die hier vorgestellten Folien sowie der Quelltext zum Ausprobieren sind ab sofort auch der Veranstaltungs-Seite zu finden!

Viel Spaß beim Bilder Aufnehmen und Auswerten!