Elektronische Bildverarbeitung
Dozent/Tutor: Udo Frese

SoSe17

Lukas Bertram
Sebastian Bliefert
Johannes Hochbein
Pascal Pieper

Übungsblatt 1

Lösungsvorschlag
Abgabe: 30.05.2017

Aufgabe 1

1 Gesamtkonzept

Um die Würfel im Versuchsaufbau zu zählen, wird das Eingabebild zunächst geeignet vorverarbeitet und anhand der Farben aufgeteilt. Die resultierenden Einzelbilder für jede Farbe werden binarisiert und anschließend einzeln der Würfelerkennung zugeführt. Diese ermittelt annähernd die geometrischen Mittelpunkte der segmentierten (Würfel-)flächen und nutzt diese Informationen zur Trennung der einzelnen Würfel mithilfe des „watershed"-Algorithmus. Anschließend werden die innerhalb der separierten Würfelflächen vorhandenen Augen mithilfe eines „blob detectors"gezählt und in der Statistik der Augenverteilung erfasst.

2 Teilalgorithmen

2.1 void preprocessColors(Mat&img, double highestValue))

- **Mat& img:** Referenz auf das Eingabebild
- **double highestValue:** Maximalwert, den die Farben annehmen können

Diese Funktion macht eine Farbwertspreizung, sodass der gesamte Farbbereich abgedeckt wird um die Segmentierung anhand von Farbwerten zu vereinfachen.

2.2 void seperateDiceColors(Mat& image, Mat& display, vector <Dice>& dices)

- **Mat& image:** Referenz auf das Eingabebild
- **Mat& display**: Referenz auf das zur Anzeige verwendete Bild
- **vector<Dice>& dices**: Verweis auf den mit den gefundenen Würfeln zu füllenden Vektor

Die Funktion *seperateDiceColors(...)* extrahiert die für die einzelnen Würfelfarben relevanten Pixel aus dem Eingabebild und ruft *segmentAndRecognizeFromBinImage(...)* für jedes der Ergebnisbilder auf. Für die Extraktion wird das Eingabebild zunächst in den HSV-Farbraum konvertiert und anschließend mithilfe der openCV Funktion „inRange()"auf das für die einzelnen Würfelfarben relevante Spektrum begrenzt und binarisiert.

2.3 void segmentAndRecognizeFromBinImage(Mat& image, vector<Dice>& dices)

- **Mat& image**: Referenz auf das binäre Eingabebild
- **vector<Dice>& dices**: Referenz auf den, mit den gefundenen Würfeln zu füllenden, Vektor

`segmentAndRecognizeFromBinImage()` segmentiert einzelne Würfel aus dem Eingabebild. Dazu werden auf einer Arbeitskopie des Bildes zunächst die Würfelflächen vollständig mit Weiß aufgefüllt (Hierfür Hintergrund per `floodfill()`) auffüllen, Ergebnis invertieren und auf das Eingabebild addieren), sodass die schwarzen Augen von den Würfelflächen verschwinden (siehe Abb. 1a).

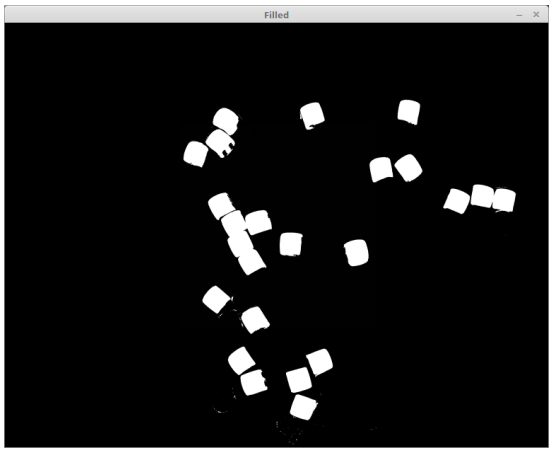
Mithilfe dieses Bildes werden die „Schwerpunkte“der Würfelflächen unter Verwendung des durch den distanceTransform-Algorithmus generierten Bilds (Abb. 1b) bestimmt: Das Ergebnisbild (8 Bit Graustufen) wird durch Anwendung eines Schwellwerts, Dilatation und anschließender Konturfindung überführt in einen Vektor aus Markern, die die „Schwerpunkte“der Würfel repräsentieren. Mit „Schwerpunkt“ist hier das Maximum der Entfernungen eines jeden Pixels innerhalb der Würfelfläche zum Rand der Fläche gemeint. Die so gewonnen Marker)Abb. werden als Startpunkte für den Watershed-Algorithmus gebraucht, welcher die eigentliche Segmentierung der einzelnen Würfel vornimmt. Dazu füllt der Algorithmus das Eingabebild anhand der Marker in alle Richtungen auf, bis sich die einzelnen Füllgebiete entweder treffen oder der Hintergrund den Füllvorgang begrenzt (genauere Informationen zu `watershed(...)` unter [1]) und stellt die Konturen der so gefundenen Gebiete zur Verfügung (siehe Abb. 1d). Aus jeder dieser Konturen wird mit `minAreaRect(...)` das für die Augenzählung relevante Gebiet bestimmt und zusammen mit weiteren Parametern an *countBlobs(...)* übergeben (siehe Abb. 1e). Zusätzlich wird jeder gefundene Würfel im Vektor *dices* für spätere Operationen abgelegt.

2.4 Dice countBlobs(SimpleBlobDetector& d, Mat& orig, RotatedRect& elem, vector<Point>& approx)

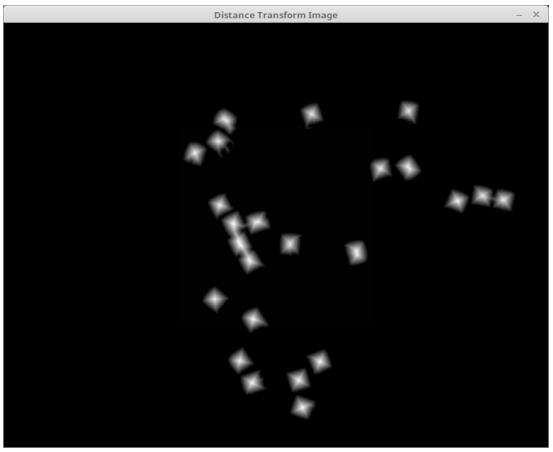
- **SimpleBlobDetector& d:** Referenz auf eine Instanz des openCV SimpleBlobDetectors
- **Mat& orig:** Referenz auf das Eingabebild (8 Bit Graustufen)
- **RotatedRect& elem:** Referenz auf ein RotatedRect, das die Rotation des zu prüfenden Würfels repräsentiert
- **vector<Point>& approx:** Referenz auf die Position und Dimension des zu prüfenden Würfels

In dieser Funktion werden die Augen auf einem Würfel gezählt. Dazu wird der Würfel erst in eine einheitliche Position gedreht und durch Dilatation und Erosion sowohl rauschen entfernt als auch die einzelnen Augen sauber voneinander getrennt und stärker abgerundet. Dann wird der SimpleBlobDetector aus openCV eingesetzt. Dieser geht wie folgt vor (laut Dokumentation von openCV):

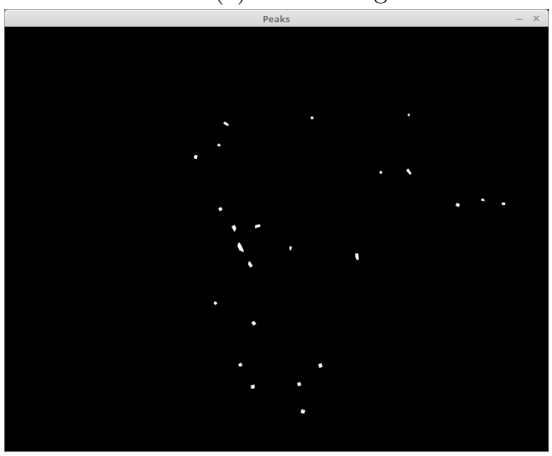
1. Das Graustufenbild wird in anhand mehrer Schwellwerte, in einem zu übergebenen bereich, in mehrere Binärbilder überführt.
2. Durch einen Konturfindungsalgorithmus, der nicht näher spezifiziert ist, werden zueinandergehörige Teile des Bildes ermittelt und deren Zentrum berechnet.



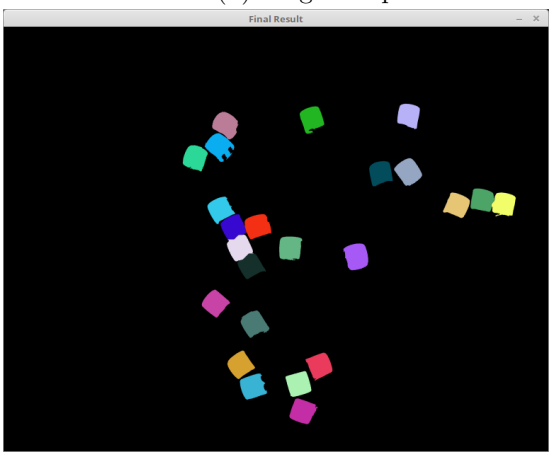
(a) Filled image



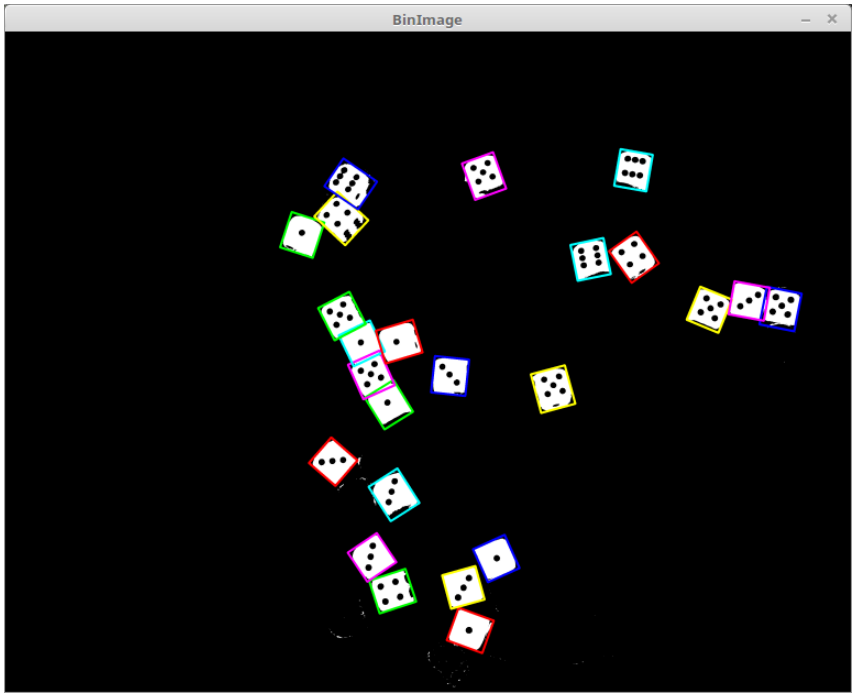
(b) Height map



(c) Peaks of height map



(d) Coloration of watershed output



(e) Found Elems marked in original segmented image

Abbildung 1: Workflow of algorithm

3. Die einzelnen Zentren werden zu Gruppen zusammengefasst, indem näher als `minDistBetweenBlobs` (ein zu übergebender Parameter) beieinander liegende Zentren zusammengefasst werden.
4. Aus den erzeugten Gruppen werden nun wiederum die tatsächlichen Mittelpunkte und Radien der Blobs errechnet und zurückgegeben.

2.5 void addToStatistics (vector<int>& statistics, const vector<Dice>& dices)

- **vector<int>& statistics:** Vector in dem die Würfel nach Augenzahl aufsummiert werden
- **const vector<Dice>& dices:** Vector mit allen erkannten Würfeln

void addToStatistics (vector<int>& statistics, const vector<Dice>& dices) itteriert alle gefundenen Würfel durch und zählt wie häufig die möglichen Augenzahlen vorgekommen sind.

Das Ergebnis wird in den Vektor statistics gespeichert.

2.6 bool passed (const vector<int>& statistics)

- **const vector<int>& statistics:** Vector mit den nach Augenzahl aufsummierten Würfeln

bool passed (const vector<int>& statistics) prüft ob die in den Vector aufsummierten Würfel normalverteilt sind. Hierbei Wird der Chi-Quadrat-Test mit einem Testniveau von 90% zugrunde gelegt.

In unserem Fall erwarten wir, dass alle Würfel aus der Losgröße n gleichhäufig vorkommen:
Erwartete häufigkeit je Augenzahl = ne = n/6.

x empirisch(xemp) ist nun die quadratische Abweichung der soll und ist Werte geteilt durch die erwarteten Werte. Hierdurch wirken sich Fehler ungeachtet des Vorzeichens gleich stark auf das Ergebnis aus.

Die Summe aller somit gebildeten Werte muss bei einem Testniveau von 90% und 5 Freiheitsgraden unter 1,61 bleiben damit dieser Test bestanden wird und die Methode true als Rückgabe liefert.

Aufgabe 2

tbd.

[CODE]

Aufgabe 3

3 Konzept:

Der Keratograf sendet ein definiertes Muster aus konzentrischen, abwechselnd schwarzen und weißen Kreisen in Richtung Auge. Je nach Position des Auges vor dem Keratografen ergibt sich

eine entsprechende Spiegelung der Ringe auf der Oberfläche der Hornhaut. Die Kamera ist im Zentrum des Keratografen platziert und nimmt das Bild der Spiegelung auf. Nach Bestimmung des gemeinsamen Mittelpunkts der Kreise wird der Abstand der Übergänge von Schwarz zu Weiß und von Weiß zu Schwarz entlang eines radialen Schnitts vermessen. Dazu wird eine Halbgerade festgelegt, die zunächst horizontal durch den Mittelpunkt verläuft und in diesem beginnt. Mit definierter Schrittweite wird der Schnittpunkt der Geraden mit einem fiktiven, zum Mittelpunkt konzentrischen Kreis auf diesem Kreis umlaufend verschoben. In jedem Zyklus werden nach einem "Weiterdrehenälle Kanten in zur Halbgeraden senkrechter Richtung zunächst gefunden und dann in ihrem Abstand zum Mittelpunkt (Beginn der Halbgeraden und damit Urpsrung des zur Vermessung genutzten Koordinatensystems) vermessen.

4 Bewertung der Messwerte:

Der theoretisch perfekte Abstand zwischen den Kanten ist bekannt für eine perfekt kugelfömige Honrhaut. Die Differenzen zwischen den theoretischen und den tatsächlich gemessenen Abständen gibt Aufschluss über die Form der Hornhaut: - Ist der gemessene Abstand kleiner als der theoretisch erwartete Wert, so ist die Hornhaut an dieser Stelle konkaver als die perfekte Kugel. - Ist größer als der theoretisch erwartete Wert, so ist die Hornhaut an dieser Stelle konvexer als die perfekte Kugel.

Literatur

[1] OpenCV Documentation, *Image Segmentation with Watershed Algorithm*, `http://docs.opencv.org/3.1.0/d3/db4/tutorial_py_watershed.html`, abgerufen am 30.05.17, 12:00Uhr.