Object Tracking Abakus

Stefan Schäfers (2175460), Julian Roosch (2203745)

8. Januar 2016

1 Die Idee

Zu beginn des Projektes wurde ein Algorithmus zur Filterung eines maskierten Ausgangsbildes entworfen und implementiert. Dabei wurden folgende Schritte durchlaufen:

1. So lange unbesuchte Pixel im Bild existieren, das Bild Zeile für Zeile von links nach rechts durchsuchen
2. Wenn ein Weißes Pixel gefunden wird, schwarz färben
3. Zur eventuellen Schwerpunktsberechnung x- und y-Wert des Pixels auf Variablen addieren, Anzahl der gefundenen Pixel im Cluster um eins erhöhen
4. So lange unbesuchte Pixel im Cluster existieren, finde alle Nachbarpixel und wiederhole Schritt 2 und 3
5. Wenn die Anzahl der Pixel im Cluster eine festgelegte Mindestgröße überschreitet, wurde ein Objekt erkannt
6. Ermittlung des Schwerpunktes des Clusters anhand der definierten Variablen
7. Verwertung der Schwerpunkte durch Sounderzeugung

Schnell wurde klar, dass der Algorithmus keine gute Laufzeit hat und somit unnutzbar für unser Projekt ist. Er war jedoch unser Ausgangspunkt des Bilderverarbeitungsalgorithmus der Schritt für Schritt durch in OpenCV implementierte Methoden verbessert wurde.

2 Maskierung

Der erste Schritt unserer Bildverarbeitung ist die Übertragung des Ausgangsbildes in ein Binärbild. Hier wird festgelegt, welche Farben Maskiert werden sollen. Dazu wird das Bild vom RGB-Farbraum in den HSV-Farbraum übertragen.

Maskiert wird das Bild über Thresholds, also Grenzen, die ein Pixel in seiner Wertigkeit einhalten muss um maskiert oder nicht maskiert zu werden. Die Nutzung des RGB-Farbraumes ist dabei nicht sehr praktikabel. Möchte man beispielsweise wie in unserem Projekt die Farbe Grün maskieren, so werden Ober- und Untergrenzen im Farbraum definiert. Der RGB-Farbraum bietet also bezüglich der Einhaltung von Grenzen drei Werte zur Verfügung: der R-, G- und B-Wert. Problematisch an dieser Aufteilung ist, dass ein Grün, je nach Beleuchtung und Qualität der Kamera, auf einem Bild heller und dunkler abgebildet werden kann als die originale Farbe. Im Extremfall (dunkelstes Grün nahezu schwarz, hellstes nahezu weiß) sind die Thresholds also Beispielsweise: (0,10,0) - (50,255,0) - (245,255,245). Diese Thresholds umfassen jedoch auch viele Farben, die nicht maskierten werden sollen. Um die Begrenzung des Grüntons und einige dazugehörige Schattierungen zu vereinfachen, wird das bild in den HSV-Farbraum übertragen. Dort kann über eine starke Begrenzung im H-Wert der Farbton bestimmt werden, und über schwächere Begrenzungen der S- und V-Werte die nötigige Varianz zur Maskierung der Schattierungen ermöglicht werden.

```
1 cvtColor(input, input, CV_BGR2HSV);
2 medianBlur(input, input, 11);
3 Mat output(input.rows, input.cols, CV_8UC1);
4 inRange(input, Scalar(minH, minS, minV), Scalar(maxH
     , maxS, maxV), output);
```

Wie aus dem Codeausschnitt hervorgeht, werden hier erste Maßnahmen zur Fehlerbehebung vorgenommen. Dabei wird bereits vor der Maskierung das Ausgangsbild einem medianBlur unterzogen, um das Rauschen aus dem Originalbild zu verringern.

3 Filterung

Abhängig von der Qualität der Kamera, mit der die Bilder aufgezeichnet werden, muss das maskierte Bild weiter verarbeitet werden.

1. Das maskierte Bild wird noch einmal einem medianBlur unterzogen. Auch nach dem medianBlur *vor* der Maskierung verbleiben einige Fehler im maskierten Ausgangsbild. Durch das erneute Anwenden des medianBlur werden weitere Fehler (meist weiße Fehlerpixel) behoben.

2. Das nun relativ fehlerfreie Bild wird durch zwei Iterationen der erode-Funktion bearbeitet

3. Um die Formen des maskierten Bildes komplett zu schließen, wird das Bild anschließend durch zwei Iterationen der dillate-Funktion bearbeitet.

Die Punkte 2. und 3. sind dabei sehr wichtig für die weitere Verwendung des Bildes, sowohl in der Wertigkeit, als auch in ihrer Reihenfolge. Für die nächsten Schritte ist es sehr wichtig, dass die Kontur eines Pixelclusters klar abgegrenzt ist. Die Anwendung der erode-Funktion gefolgt von der dillate-Funktion sorgt dafür, dass Formen nach Innen geschlossen werden und weitere weiße Pixelfehler behoben werden. Wendet man die beiden Funktionen andersherum an, so würden die Formen nach außen geschlossen und schwarze Pixelfehler behoben werden.

4 Auswertung

Zum Auswerten der Positionen der Kugeln, müssen nun die Zentren der Pixelcluster ermittelt werden. Dafür werden zunächst die Konturen der weißen Areale definiert:

```
1 cv::findContours(contourOutput, contours,
     CV_RETR_LIST, CV_CHAIN_APPROX_NONE);
```

Die bereits in OpenCV implementierte Funktion findContours liefert dabei alle Konturpixel eines abgeschlossenen weißen Pixelclusters.

Hier wird nun klar, warum eine gute Filterung sehr wichtig ist: Der Algorithmus erkennt eine Kontur erst als eine eigene Kontur an, wenn diese eine gewissen Anzahl an Pixeln überschreitet. Würde sich beispielsweise ein weißes Fehlerpixel noch im Bild befinden, so würde dieses Pixel einer Kontur zugewiesen werden und somit die Berechnung des Zentrums stark beeinträchtigen.

Die Berechnung der Zentren erfolgt nun anhand der *Moments* einer *Kontur*.

```
1 Moments mu = moments(contours[i], false);
2 Point mc(mu.m10/mu.m00, mu.m01/mu.m00);
```

Dabei werden die x- und y-Werte der Moments einer Kontur gemittelt und der daraus resultierende Punkt als Mittelpunkt der Kontur definiert.

5 Verwendung

Bevor alle genannten Schritte durchgeführt werden, müssen immer gewissen Bedingungen erfüllt sein. Wichtige Kriterien sind dabei:

1. Es müssen exakt so viele Objekte (Pixelcluster) erkannt werden, wie erwartet wurden
2. Jeder Pixelcluster muss dabei eine Mindestgröße überschreiten
3. Das Zentrum einer Kontur darf nicht zu weit am Rand des Bildes sein
4. Das Zentrum einer Kugel darf im Laufe mehrerer Bilder in vertikale Richtung einen definierten Bereich nicht verlassen

Das Einhalten dieser Regeln hilft dabei weitere Fehler, die auf Grund der relativ schlechten Bildqualität der Kamera entstehen können, zu beheben. Bilder, in denen eine oder mehrere Regeln nicht eingehalten wird/werden, werden im Programmablauf ignoriert und nicht weiter verarbeitet.

Läuft hingegen der Programmcode für ein Bild komplett durch, so werden die Daten (Zentren der Konturen) an die Sounderzeugung skaliert weitergeleitet - die Bedienbarkeit ist somit hergestellt.