**Card Detector**

**EDBV WS 2015/2016: AG\_B3**

**Matrikelnummern**

Christopher Dick (0946375)  
Timon Höbert (1427936)  
Julian Lemmel (1427667)  
Thomas Anderl (1427841)  
Markus Klein (1426483)

08.01.2016

Inhalt

[1. Gewählte Problemstellung 3](#_Toc439867763)

[1.1 Ziel 3](#_Toc439867764)

[1.2 Eingabe 3](#_Toc439867765)

[1.3 Ausgabe 3](#_Toc439867766)

[1.4 Voraussetzungen 3](#_Toc439867767)

[1.5 Methodik 4](#_Toc439867768)

[1.6 Evaluierungsfragen 4](#_Toc439867769)

[1.7 Zeitplan 5](#_Toc439867770)

[2. Arbeitsaufteilung 6](#_Toc439867771)

[3. Methodik 7](#_Toc439867772)

[3.1 Vorbereitungen 7](#_Toc439867773)

[3.2 Segmentierung 7](#_Toc439867774)

[3.3 Kartenbestimmung 7](#_Toc439867775)

[3.4 Pattern-Matching 8](#_Toc439867776)

[4. Implementierung 9](#_Toc439867777)

[5. Evaluierung 10](#_Toc439867778)

[5.1 Korrekte Ergebnisse 10](#_Toc439867779)

[5.2 Teilweise korrekte Ergebnisse 10](#_Toc439867780)

[5.3 Fehler 11](#_Toc439867781)

[5.4 Gesamtevaluierung 11](#_Toc439867782)

[6. Schlusswort 13](#_Toc439867783)

[7. Literatur 14](#_Toc439867784)

# 1. Gewählte Problemstellung

## 1.1 Ziel

Der CardDetector soll anhand eines Eingabebildes Spielkarten möglichst richtig identifizieren können, d.h. er muss den Wert und die Farbe der Karte erkennen können.

## 1.2 Eingabe

Als Eingabe benötigt der CardDetector nur eine Bilddatei in einem der folgenden Formate: .png, .tiff, .jpg. Hierbei kann es sich um eine Fotografie, aber auch um ein computergeneriertes Bild handeln. Danach arbeitet das Programm von alleine und benötigt keine weitere Benutzerinteraktion.

## 1.3 Ausgabe

Ausgegeben wird vom Programm wieder das Eingabebild, wobei entdeckte Karten umrahmt und der erkannte Wert bzw. die erkannte Farbe einer Karte auf dem Ausgabebild zusätzlich dabeistehen.

## 1.4 Voraussetzungen

Um sicherzustellen, dass alle Karten richtig identifiziert werden, werden folgende Bedingungen an ein Eingabebild gestellt:

* Der Hintergrund muss einfarbig sein und muss in Kontrast zu den hellen Karten stehen.
* Die Karten dürfen sich nicht berühren, geschweige denn einander überlappen.
* Die Karten müssen möglichst parallel zum Bildrand liegen.
* Die Karten dürfen sich nicht mit dem Bildrand schneiden.
* Das Bild muss von oben aufgenommen werden, perspektivische Verzerrung ist nicht erlaubt.
* Es gibt ein vorgegebenes Kartendeck.
* Das Bild muss gleichmäßig beleuchtet sein und darf keine Glanzpunkte enthalten.

## 1.5 Methodik

Der grobe Ablauf des Algorithmus ist wie folgt:

* Einlesen und Prüfen des Bildes
* Glätten mittels Gauß-Filter
* Binärbild nach Threshold von Otsu ermitteln
* Segmentierung mittels Connected Component Labeling: Nur Labels, die der Größe der ersten entdeckten Karte entsprechen, werden weiterverwendet.
* Erkennen des Wertes: Bei Zahlenkarten werden die Symbole in der Mitte gezählt, bei Bildkarten wird der Buchstabe mittels Template Matching (Templates für Bube(B), Dame(D), König(K)) erkannt.
* Erkennen der Farbe: Template Matching für die Spielfarben Herz, Pik, Karo und Kreuz.
* Ausgeben von Wert und Farbe direkt auf dem Eingabebild

## 1.6 Evaluierungsfragen

Zur Evaluierung des CardDetectors können folgende Frage herangezogen werden:

* Wie viel Prozent der Testdatensätze liefern ein korrektes Ergebnis?
* Wird die richtige Anzahl an Karten auf einem Bild erkannt?
* Wie viele Karten eines Bildes werden korrekt bestimmt?
* Werden die Farben korrekt unterschieden?
* Werden die Kartenpositionen richtig erkannt?
* Werden die Bilder korrekt eingelesen und verarbeitet?

Fragen zu Grenzfällen:

* Wie weit können sich Karten überlappen ohne das Ergebnis zu beeinflussen?
* Bis zu welchem Neigungsgrad werden Karten trotzdem korrekt ausgewertet?
* Werden Karten, die über den Bildrand hinausgehen, dennoch richtig erkannt?
* Können Karten, die durch die Perspektive verzerrt sind, trotzdem korrekt bestimmt werden?
* Werden Karten eines anderen Kartendecks auch richtig identifiziert?
* Beeinträchtigen Glanzpunkte am Bild die Korrektheit des Ergebnisses?

## 1.7 Zeitplan

|  |  |  |  |
| --- | --- | --- | --- |
| Meilensteine | beendet am | beteiligte Personen | Aufwand in h |
| Regelmäßige Gruppentreffen, Organisation |  | alle | 5x15 |
| fertiger Otsu-Threshold | 03.11.2015 | Thomas | 9 |
| fertiger Gauß-Filter | 07.11.2015 | Markus | 7 |
| MATLAB-Prototyp (Programmablauf ohne eigene Funktionen) | 10.11.2015 | Timon, Christopher | 15 + 15 |
| fertiges Pattern-Matching | 20.11.2015 | Julian | 17 |
| funktionsfähiger Prototyp | 24.11.2015 | Timon, Christopher | 15 + 15 |
| Vorbereitung Zwischenpräsentation | 28.11.2015 | alle | 5x3 |
| Zwischenpräsentation | 02.12.2015 | alle | 5x4 |
| fertiges CCL | 20.12.2015 | Christopher | 5 |
| fertiger CardDetector (debugging, etc.) | 21.12.2015 | alle | 5x3 |
| Testdatensatz zusammenstellen | 24.12.2015 | Markus, Thomas | 10 + 2 |
| Evaluierung der Testsätze | 26.12.2015 | Markus, Thomas | 8 + 5 |
| fertiger Bericht, Abgabefertig | 08.01.2016 | alle | 3+2+3+1+1 |

# 2. Arbeitsaufteilung

|  |  |
| --- | --- |
| Name | Tätigkeit |
| Christopher Dick | MATLAB-Prototyp und Pipeline implementiert, CCL, Bericht (Punkt 4), readMe |
| Timon Höbert | MATLAB-Prototyp implementieren, Bericht (Punkt 3.2, 3.3) |
| Julian Lemmel | Pattern Matching, TestAll, Bericht (Punkt 4), Pattern Templates |
| Thomas Anderl | Datensatz, Kommentare, Otsu’s Threshold, Bericht (Punkt 3.1, Punkt 6) |
| Markus Klein | Datensatz, Gauß-Filter, Bericht (Punkt 1, Punkt 3.1, Punkt 5) |

# 3. Methodik

Der Ablauf des CardDetectors gliedert sich in drei große Teilbereiche: Das Vorbereiten des Eingabebildes für weitere Verarbeitung, die Segmentierung von Karten und das Bestimmen der einzelnen Karten.

## 3.1 Vorbereitungen

Hier wird zuerst überprüft, ob ein gültiges Bild eingegeben wurde. Dazu wird zuerst getestet, ob es die angegebene Datei überhaupt gibt. Falls das Bild nicht in double type ist, wird es in diesen konvertiert. Falls das Bild kein Graustufenbild ist, wird es in diesem Schritt auch in ein solches umgewandelt.  
   
Danach wird das Bild mittels Gauß-Filter geglättet. [3] Der Sinn dieses Schrittes ist es, das Rauschen im Eingabebild zu mindern und sich in weiterer Folge um weniger winzige Labels in der nachfolgenden Segmentierung kümmern zu müssen.

Der letzte Schritt der Vorbereitung ist es, das geglättete Bild in ein Binärbild umzuwandeln. Benötigt wird es für die darauffolgende Segmentierung. Das Binärbild wird nach dem Threshold von Otsu gefunden. Dieser sucht nach dem Wert, bei dem die Varianz zwischen dem Vorder- und dem Hintergrund maximal bzw. die Varianz innerhalb dieser zwei Bereiche minimiert wird. [1][2][[3]

## 3.2 Segmentierung

Das geglättete Binärbild wird im Zuge der Bildsegmentierung mittels Connected Component Labeling (CCL) in einzelne Bereiche (Labels) gegliedert. Auf Basis der Labels arbeiten wir mit der größten horizontalen bzw. vertikalen Ausdehnung dieser, den Boundingboxen.  
  
Zur Filterung der Bereiche werden ausschließlich Boundingboxen weiterverarbeitet, welche mindestens 90% der Fläche der größten Boundingbox ausmacht. Dazu werden die Flächen aller Boundingboxen berechnet und aufsteigend sortiert. Mithilfe dieses Verfahrens werden alle kleinen Störsegmente herausgefiltert und nur die weißen Vierecke der Karten bestimmt. Das Label des Hintergrundes wird dabei außen vorgelassen.

## 3.3 Kartenbestimmung

Nachdem alle Karten segmentiert wurden, wird der Wert jeder Karte einzeln bestimmt. Dazu wird zuerst die Symbolik der Wertigkeit (z.B.: 6, 10, B, A) und unterhalb deren Symbol (Kreuz, Herz) in der oberen linken Ecke detektiert, um diese mittels Pattern-Matching zu unterscheiden. Hierfür wird die Boundingbox des Labels mit der kürzesten euklidischen Distanz zwischen dessen linken oberen Ecke und der linken oberen Ecke der Karte gesucht. Mit demselben Verfahren wird das Symbol unterhalb der Wertigkeit gesucht, bloß der linken unteren Ecke des Symbols als Referenzpunkt.   
  
Im Zuge dieses Prozesses wird außerdem geprüft, ob die Karte eine Bildkarte ist. Dies ist der Fall, sobald ein Label mit einer Boundingboxfläche größer 10% der Karte existiert, das Bild in der Mitte.  
  
Das Symbol wird via Pattern-Matching erkannt, der Wert abhängig von der Karte. Bildkarten werden auch via Pattern-Matching des Symbols oben links detektiert, alle anderen werden gezählt. Beim Zählen wird die Anzahl der Symbole in der Kartenmitte gezählt, welche dem Wert entspricht. Dazu wird der Mittelteil der Karte herausgeschnitten, also die Symbolik links und rechts weggeschnitten. Anschließend werden störende Randsegmente gelöscht, welche von den Symbolen überbleiben können. Danach werden einfach wieder alle Labels mit 90% der Fläche der Boundingbox des größten Labels gezählt.

## 3.4 Pattern-Matching

Für das Pattern-Matching wurden im Vorfeld die Pattern Templates erstellt.

Für die Erkennung des Kartenwerts (K, D, B) ist der übergebene Ausschnitt meist größer als der Buchstabe der erkannt werden soll. Deswegen wird der Ausschnitt noch einmal mit CCL aufgeteilt und auf die größte resultierende Bounding-Box beschnitten.

Der Ausschnitt wird auf die Größe der Templates gebracht (40x40px) und schließlich wird die Differenz zu jedem der einzelnen Templates berechnet. Von diesen berechneten Differenzen wird die kleinste ausgewählt und somit auf das Symbol bzw. den Wert der Karte geschlossen.

# 4. Implementierung

**TODO**: Hier ein paar Worte zur Implementierung(wie wurden Methoden praktisch umgesetzt, Programmgerüst/Pipeline erklären) und wie der User das Programm verwendet.

# 5. Evaluierung

Der gesamte Testdatensatz umfasst 26 Bilder, die alle mit einer Nikon D3000 Spiegelreflexkamera aufgenommen wurden. Diese können in drei Unterkategorien eingeteilt werden: Bilder, wo alle Karten korrekt erkannt wurden, Bilder, die nur teilweise richtige Ergebnisse liefern, und Bilder mit fehlerhaften Ausgaben.

## 5.1 Korrekte Ergebnisse

Auf jedem Bild in Kategorie werden alle Karten richtig identifiziert. Grund dafür ist, dass die einzelnen Bilder die Voraussetzungen einhalten und keine Grenzfälle darstellen. Unter anderem werden hier verschiedene Auflösungen, verschiedene Positionen und Anzahl der Karten und diverse Hintergründe getestet.

Dateien dieser Gruppe sind:

* Aufloesung.jpg
* FullHouse.jpg
* Gelbstich.jpg
* Paar.jpg
* Super.jpg
* TooMuch.jpg
* ZickZack.jpg

## 5.2 Teilweise korrekte Ergebnisse

Bilder in dieser Kategorie liefern sowohl richtig erkannte Karten, aber auch falsch erkannte Karten zurück. Das kommt daher, dass jedes der Bilder eine Voraussetzung missachtet und daher nicht mehr gewährleistet ist, dass der CardDetector korrekte Ergebnisse zurückliefert. Unter anderem sind in dieser Kategorie Bilder mit ungleichmäßiger Beleuchtung, Bilder mit berührenden Karten, aber auch Bilder mit anderem Kartendeck enthalten.

Dateien dieser Gruppe sind:

* Aneinander.jpg
* Asse.jpg
* Drilling.jpg
* MixedSets.jpg
* Radioactive.jpg
* Random.jpg
* SizeMix.jpg

## 5.3 Fehler

Auch in dieser Kategorie sind Bilder, die nicht den Voraussetzungen entsprechen. Hier wird jedoch keine einzige Karte richtig erkannt. Diese fehlerbehafteten Bilder testen unter anderem Überlappung von Karten, über den Rand hinausragende Karten, perspektivische Verzerrung, mehrfarbige Hintergründe usw.

Dateien dieser Gruppe sind:

* 45Degrees.jpg
* Abgeschnitten.jpg
* BabyCard.jpg
* BunterHintergrund.jpg
* Drehung.jpg
* FalschesDeck.jpg
* LowQuality.jpg
* Perspektive.jpg
* SuperFlat.jpg
* Ueberlappend.jpg
* Umgefallen.jpg
* Verdreht.jpg

## 5.4 Gesamtevaluierung

Nur 26,92 % der Testdatensätze liefern ein korrektes Ergebnis. Das klingt zwar nach einem schlechtem Programm, rührt aber tatsächlich daher, dass sich nur 7 von 26 Daten zur Gänze an die Voraussetzungen für ein Eingabebild halten.   
  
Die beiden Bedingungen an ein Eingabebild, die meistens der Grund für falsche Ergebnisse sind, sind der kontrastreiche Hintergrund und die gleichmäßige Beleuchtung. Dieser Fehler tritt beim Umwandeln in ein Binärbild auf. Hauptsächlich werden diese Probleme wegen dem Unterschied vom menschlichen Sehen zur Bildverarbeitung hervorgerufen.   
Ein Hintergrund, der für das menschliche Auge genug Kontrast zu den Spielkarten hat, ist in der Bildverarbeitung oftmals zu ähnlich. Und oft sind es Schatten oder Glanzpunkte, die den Menschen selbst kaum beim Erkennen der Karten behindern, die aber der Grund für falsche Ergebnisse oder nicht erkannte Karten im CardDetector sind.

Bei Eingabebildern, die gegen die Voraussetzungen verstoßen, ist großteils die Segmentierung der Punkt, an der die Probleme auftreten. Diese weist den Karten ein Rechteck zu, dass parallel zum Bildrand ist.

Durch das Überlappen, das Berühren, das Hinausragen über den Bildrand, perspektivische Verzerrung oder verschiedene Neigungsgrade der Karten wird den einzelnen Karten ein Segment zugewiesen, dass nicht mit der tatsächlichen Größe der Karte übereinstimmt und wodurch die weitere Verarbeitung verfälscht wird.

Es ist möglich, Karten von verschiedenen Kartendecks zu erkennen, aber falls bei einem solchen anderen Deck zum Beispiel ein Bube das Buchstabenkürzel J statt B hat, wird dieser vom CardDetector nicht als Bube eingestuft und somit falsch erkannt.  
Ein ähnliches Problem tritt bei der Farberkennung auf: Unterscheiden sich die Symbole(Herz, Pik, usw.) zu sehr von den Templates im CardDetector, werden auch die Farben nicht richtig erkannt und es werden falsche Ergebnisse geliefert.

# 6. Schlusswort

Zusammenfassend ist festzustellen, dass der CardDetector Karten - wie von uns gehofft - mit hoher Wahrscheinlichkeit richtig erkennen kann.   
  
Etwas problematisch ist die Laufzeit der eigens implementierten Funktionen. Diese überschreiten jene der vorgefertigten MATLAB-Funktionen deutlich, weswegen ein optionaler Fast-Mode eingeführt wurde, welcher die performanteren Versionen verwendet. Dieser wird nützlich, wenn große Bilder bzw. Bilder mit vielen Karten vom CardDetector verarbeitet werden.   
  
Neben der Performance könnte man das Programm auch hinsichtlich der Anforderungen an die Bilder etwas lockern, die auf Grund der vorliegenden Methodik in dieser strikten Form vorhanden sind. So kann beispielsweise die Rotation von Karten mithilfe von CCL nur schwer erkannt und korrigiert werden (hierfür empfiehlt sich eher ein Ansatz mit Kantendetektion).

(**TODO**: Weitere Verbesserungen? Oder sollte zusammenfassend noch etwas erwähnt werden?)

# 7. Literatur

[1] Wikipedia. Otsu's method. online document, last visit: 02.01.2016, 2016. URL: [https://en.wikipedia.org/wiki/Otsu%27s\_method](https://en.wikipedia.org/wiki/Otsu's_method).

[2] unknown. Otsu Threshold. Online document, last visit: 02.01.2016, 2016. http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html.

[3] Wilhelm Burger; Mark J. Burge. *Principles of Digital Image Processing. Advanced Methods*. Springer, London, 2013.

[4] R. Walczyk; A. Armitage; T.D. Binnie. Comparative study on connected component labeling algorithms for embedded video processing systems. In L. Deligiannidis Hamid R. Arabnia, editor, *IPCV’10*, Las Vegas, USA, 2010.

[5] Wen-Yuan Chen; Chin-Ho Chung. Robust poker image recognition scheme in playing card machine using hotelling transform, dct and run-length techniques. *Digital Signal Processing*, (20):769–779, 2010.

[6] unknown. Poker vision: Playing cards and chips identification based on image processing (pattern recognition and image analysis). online document, last visit: 22/10/2015, 2011. URL: http://what-when-how.com/pattern-recognition-and-image-analysis/poker-vision-playing-cards-and-chips-identification-based-on-image-processing-pattern-recognition-and-image-analysis/.