Schnapskönig

EDBV WS 2017/2018: AG A2

Jan Michael Laranjo (01425799)
Andreas Brunner (01429369)
Miran Jank (01526438)
Thorsten Korpitsch (01529243)
Aleksandar Marinkovic (01634028)

4. Januar 2018

1 Gewählte Problemstellung

1.1 Ziel

Beim Schnapsen treten 2 Spieler gegeneinander an. Zu Beginn bekommt jeder 5 Karten vom Stapel und eine Karte wird aufgedeckt, welche das Adut darstellt, danach wird jede Runde von beiden Spielern eine Karte gezogen. Jede Runde wird jeweils 1 Karte von jedem Spieler auf den Tisch gelegt wobei beide Spieler versuchen einen Stich zu machen. Stechen bedeutet, dass die eigene Karte entweder ein Adut ist und die Karte des Gegenübers keines ist, beziehungsweise, dass die eigene Karte einen höheren Wert hat als die des Gegenübers. Es wird solange gespielt bis der Stapel leer ist und beide Spieler keine Karten mehr auf der Hand haben, beziehungsweise einer der Spieler Karten im Wert von 66 Punkten gestochen hat. Sind alle Karten ausgespielt worden, wird der Gewinner ermittelt indem die gewonnenen Punkte beider Spieler gezählt werden und danach der mit der höheren Punktezahl gewinnt.

1.2 Eingabe

Der Benutzer muss nur in einem Ordner alle Spielzüge als Farbbild in einem gängigen Format(.PNG/.JPG/.JPEG) speichern, wobei die Bilder aufsteigend nach Spielzügen sortiert sein sollten. Das Programm interpretiert schließlich jedes Bild als einzelne Spielzüge.

1.3 Ausgabe

In der Konsole wird ein Zwischenstand nach jeder Runde ausgegeben, am Ende wird der Gewinner ausgegeben und der Endstand.

1.4 Voraussetzungen und Bedingungen

Ein Farbbild der Karten. Der Hintergrund sollte möglichst Einfarbig sein (nicht weiß, texturarm). Die Kamera soll sich in einem Winkel von 45 bis 135 Grad befinden. Die Karten müssen mit einem dünnen schwarzen Rand präpariert sein. Die obere Karte darf maximal 45% der anderen Karte überdecken.

1.5 Methodik

1. Threshold nach Otsu: Um in Kombination mit Zusammenhangskomponenten die Karten zu trennen

2. Geometrische Transformation: Um das Eingabebild vorzubereiten, wird aus dem, bis zu 45 Grad schrägen Bild, ein Bild aus der Vogelperspektive (90 Grad) transformiert, die hochkant steht

3. Template-Matching: Um die Spielkarte zu identifizieren

1.6 Evaluierung

- Werden beide Karten erkannt?
- Wird das Bild richtig transformiert, oder werden Buchstaben/Symbole verzerrt?
- Wird die Karte richtig identifiziert?

1.7 Datenbeispiel

Figure 1 - Karo-Ass sticht Kreuz-König (Farbzwang), 16 Punkte gewonnen.

Figure 2 - Karo-König sticht Karo-Dame, 7 Punkte gewonnen.

1.8 Zeitplan

Meilenstein	abgeschlossen am		Arbeitsaufwand in h	
	geplant	tatsächlich	geplant	tatsächlich
Vorarbeit Projekt	20.10.2017	21.10.2017	20	23
Prototyp erstellen	10.11.2017	01.12.2017	40	80
Geometrische Transformation	22.12.2017	27.12.2017	105	92
Threshold nach Otsu	18.12.2017	25.12.2017	67	60
Pattern-Matching	18.12.2017	30.12.2017	43	30
GT-Test	31.12.2017	31.12.2017	10	12
TH-Test	27.12.2017	29.12.2017	5	5
PM-Test	27.12.2017	29.12.2017	5	3
Unit-Test	05.01.2017	04.01.2017	5	2

2 Arbeitsteilung

Name	Tätigkeiten
Jan Michael Laranjo	Matlab: Prototyp, main.m, correctPerspective.m, decideCard.m, correctCorner.m, Bericht: Methodik (Template Matching), Gewählte Problemstellung
Andreas Brunner	Matlab: Prototyp, tmc.m Bericht: Arbeitsteilung, Evaluierung, Implementierung (Template Matching)
Miran Jank	Matlab: Prototyp, geom_tranfs_lowercard.m Bericht: Methodik (Geometrische Transformation), Implementierung (Geometrische Transformation)
Thorsten Korpitsch	Matlab: Prototyp, splitCards.m, thresholdOtsu.m, card2string.m Bericht: Evaluierung, Methodik/Implementierung (Vorverarbeitung, Trennen der Karten), Schlusswort
Aleksandar Marinkovic	Matlab: Prototyp, getTransformationMatrix.m, geotransform.m, geom_tranfs_uppercard.m Bericht: Evaluierung, Implementierung (Geometrische Transformation)

3 Methodik

3.1 Threshold nach Otsu

Die Aufgabe für die erste verwendete Methode ist das Trennen der Karten vom Hintergrund. Hier haben wir uns für ein Schwellenwertverfahren entschieden. Bei einem Schwellenwertvefahren wird ein Graustufenbild in ein Binärbild umgewandelt. Wir haben uns für den Threshold nach Otsu[1] entschieden, da es in der Kombination mit der Zusammenhangskomponente besonders dazu eignet um Objekte vom Hintergrund zu trennen.

Beim Threshold nach Otsu wird versucht, die beiden Segmente (Vordergrund/Hintergrund) so kompakt wie möglich zu machen und die Überschneidung gering zu halten.

„Otsu'smethod selects the threshold by minimizing the within-class variance of thetwogroups of pixels separated by the thresholding operator." [2]

Um das Trennen der Karten zu vereinfachen, schränken wir die Eingabe dahingehend ein, dass beim Bild der Karten der Hintergrund nicht weiß sein darf und möglichst texturarm sein muss. Diese Einschränkungen machen unseren Threshold nach Otsu stabiler und sehr erfolgssicher beim Trennen der Karten vom Hintergrund.

3.2 Geometrische Transformation

Um die Karten im finalen Schritt wirklich erkennen zu können, müssen die Bilder vom Blickwinkel der Perspektive in die Orthogonale gebracht werden. Für diese Aufgabe haben wir die Geometrische Transformation gewählt, da wir sowohl perspektivisch verzerren, als auch normale Translationsmatrizen verwenden müssen. Zur Korrektur der Perspektive wird der DLT-Algorithmus (Direct Linear Transformation) [3] verwendet, um die Transformationsmatrix zu bestimmen.

Als Einschränkung wurde hier zwei wesentliche Aspekte getroffen: Das Bild muss in einem Winkel von 65° bis 135° aufgenommen werden, und es darf maximal die Hälfte einer Karte verdeckt werden. Die erste Einschränkung dient der Qualität und Stabilität des Verfahrens, denn wenn eine zu starke Verzerrung durch die Perspektive hervorgerufen wird, so kann sich das orthogonale Bild nur geringer Qualität erfreuen. Dies lässt sich dadurch erklären, dass bei der Transformation der Bildinhalte jeder einzelne Pixel gelesen wird, und an einen neuen Ort übersetzt wird – durch eine große Verzerrung hätten wir somit fehlende Information. Die zweite Einschränkung gilt der unteren Karte und dem Template Matching. Wäre mehr wie die Hälfte bedeckt, so könnte nicht mehr zuverlässig nach übereinstimmenden Bildbereichen gesucht werden, und auch für die Transformation der unteren Karte würden wesentliche Anhaltspunkte fehlen. Somit wurde beschlossen, dass zumindest eine horizontale Hälfte der Spielkarte immer ersichtlich sein muss.

3.3 Template-Matching

Um die Karte schlussendlich zu identifizieren haben wir uns für das Template Matching entschieden. Das Template-Matching bietet genau die Lösung zu unserer letzten Problemstellung. Mithilfe des Template-Matching versuchen wir herauszufinden ob es sich bei der Karte um Herz, Pik, Kreuz oder Karo handelt und ob es sich um Ass, König Dame, Bube oder Zehn handelt. Aus der Kombination dieser beiden Symbolen können wir die Karte eindeutig identifizieren.

Beim Template-Matching wird versucht ein Bild beziehungsweise einen Bildausschnitt in einem anderen Bild wiederzufinden. User Template-Matching wird mittels einer Korrelation-Matrix realisiert, es wird durch alle Möglichkeiten durchiteriert und das passendste wird der Karte zugewiesen.

4 Implementierung

4.1 Start und Vorverarbeitung

Eine Spielsimulation wird mit einem Aufruf von „main.m“ gestartet. Als Parameter wird hier der Ordner übergeben, in denen sich die Datensätze befinden. Diese müssen so angeordnet sein, dass sie aufsteigend jeweils eine Spielrunde repräsentieren.
Zuerst wird der Datensatz geladen und die einzelnen Bilder werden in ein Graustufenbild umgewandelt.

4.2 Karten trennen

Der Threshold ist die erste Methodik in der Methodik-Pipeline, in Kombination mit der Zusammenhangskomponente. Der Funktion splitCards.m wird das Eingabebild übergeben. Als erstes wird dieses in ein Graustufenbild umgewandelt und an die Funktion thresholdotsu.m übergeben.

Dort werden zuerst die aufkommenden Grauwerte gezählt mittles der histcounts-Funktion. Danach wird die gewichtete Summe aller Pixel berechnet, indem man die Anzahl der Pixel mit ihrem jeweiligen Grauwert multipliziert. Danach wird in einer for-Schleife durch alle Grauwerte durchiteriert. In jedem Schleifendurchgang werden die Pixel, die den aktuellen Grauwert besitzen, zum Hintergrund dazugerechnet. Der Vordergrund wird neu berechnet indem von der Summe der Pixel, die Pixel vom Hintergrund abzieht. Danach wird die Gewichtete Summe des Hintergrunds berechnet, analog dazu, wie am Beginn die gewichtete Summe aller Pixel berechnet wurde. Jetzt können die Durchschnittswerte des Hinter- und Vordergrunds berechnet werden. Als vorletzten Schritt berechnen wir die Between Class Variance.

Als letztes wird überprüft ob die Between Class Variance, die bisher größte ist. Ist das der Fall, wird der Threshold auf den aktuell iterierten Grauwert gesetzt und die maximale Between Class Variance auf die gerade berechnete gesetzt. Nach dem durchlaufen der Schleife, wird das Bild mittels des optimalen Thresholds in ein Binärbild umgewandelt.

Figure 3 - Binärbild mittels Threshold nach Otsu

Auf dieses Binärbild wird die Zusammenhangskomponente angewendet um die Karten zu trennen. Aus den 2 größten Zusammenhangskomponenten werden 2 Bilder erstellt, die die Karten repräsentieren. Bei diesen werden noch die Löcher gefüllt.

Dies passiert, da wir nicht nur die Karten trennen wollen, sondern auch gleichzeitig herausfinden wollen, welche Karte die Obere und welche die Untere ist. Das Ganze berechnen wir ganz einfach indem wir die Fläche beider Karten vergleichen und die Karte mit der größeren Fläche die sein muss, die oben liegt, da sie ja einen Teil der unteren Karte verdeckt. Das Füllen der Löcher funktioniert vor allem bei Bildkarten unterschiedlich gut, wie man im nachfolgenden Bild erkennen kann, das kann zu Problemen führen bei der Erkennung welche Karte die Obere und welche die Untere ist.

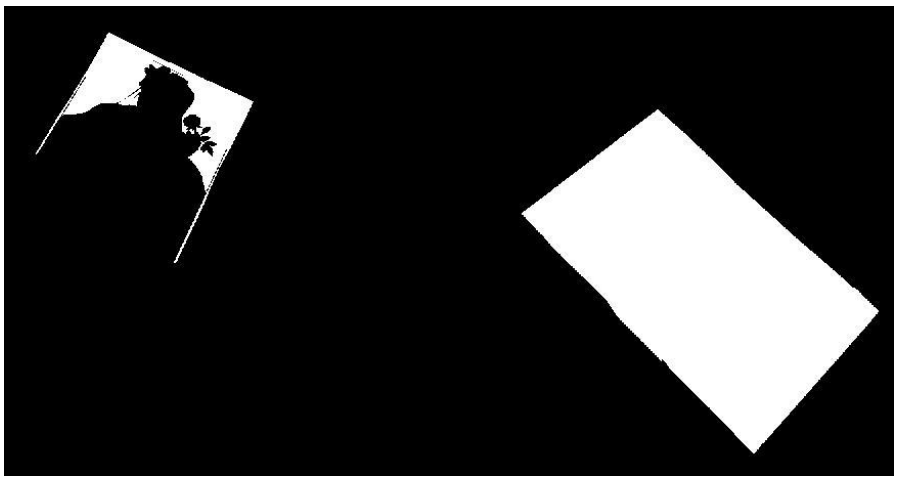

Figure 4 - Binärbilder der getrennten Karten als Montage. Dame(links), König(rechts)

Danach werden beide Karten an die 2. Methodik weitergegeben, die geometrische Transformation.

4.3 Geometrische Transformation

Bei der geometrischen Transformation müssen die Karten separat voneinander transformiert werden. Bei der oberen Karte werden zuerst die Ecken mithilfe einer Boundingbox bestimmt. Ein Algorithmus geht die Seiten der Boundingbox entlang und bestimmt die Koordinaten der Ecken.

Zusätzlich zu den Eckpunkten werden vier weitere Zielkoordinaten benötigt, eine Spielkarte hat ein Verhältnis von 5:8 daraus bilden wir die Basis für die Zielkoordinaten. Bei der Bestimmung der Zielkoordinaten wird zwischen zwei Fällen unterschieden und zwar ob die obere Ecke in die linke oder rechte Hälfte hin transformiert werden muss.

Figure 5 – rechte Ecke oben

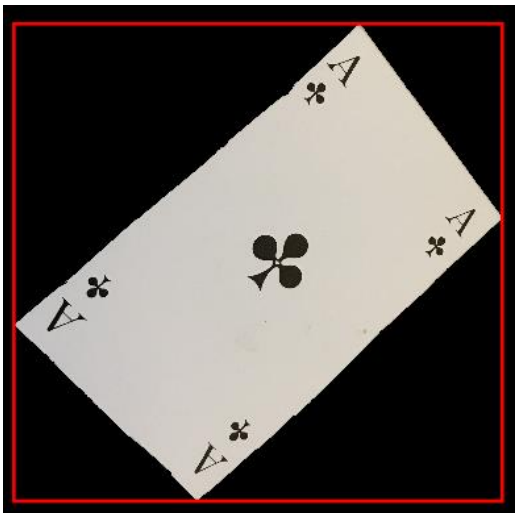

Figure 6 – linke Ecke oben

Um die richtigen Zielkoordinaten zu bestimmen, wird als nächstes die Distanz zwischen der oberen und der linken bzw. der rechten Ecke bestimmt. Da die Karten ein Seitenverhältnis von 5:8 besitzen, lässt sich damit die Orientierung der Karte bestimmen. Wenn die Distanz zwischen der oberen und linken Ecke kürzer als die Distanz zwischen der oberen und rechten Ecke ist (Figure 5) muss die linke Ecke an der Position (0, 0) transformiert werden. Wenn die Distanz länger ist (Figure 6) muss die obere Ecke an die Position
(0, 0) transformiert werden.

Als nächstes werden die Eckpunkte und Zielkoordinaten als Parameter an die Funktion getTransformationMatrix.m übergeben um die Transformationsmatrix zu bestimmen. Die Matrix wird mithilfe des DLT-Algorithmus (Direct Linear Transformation) ermittelt.

Bevor das Bild transformiert wird muss es in einem double Typ umgewandelt werden. Dies ist notwendig da später beim Interpolieren kein uint8 Typ akzeptiert wird.

Der Funktion geotransform.m werden als Parameter das Bild und die Transformationsmatrix übergeben. Falls es sich beim Bild um ein Graustufenbild handelt wird das Bild ohne Probleme transformiert. Da wir jedoch RGB-Bilder als Input haben müssen alle drei Kanäle einzeln transformiert werden.

In geotransform.m werden als erstes die x- und y-Werte festgelegt welche gleich der Anzahl der Spalten / Reihen ist. Diese Werte werden der meshgrid Funktion übergeben, um zwei Matrizen xi und yi zu erstellen. Die xi-Matrix ist eine Kopie der x-Werte welche y-mal wiederholt werden und yi sind die y-Werte x-Mal kopiert. Nun können die Daten mit der Transformationsmatrix multipliziert werden, danach werden die homogenen x- und y-Werte normalisiert. Diese werden als Query-Points für die Interpolation verwendet. Die Query-Points und das Bild werden der interp2-Funktion übergeben und diese führt die Interpolation aus.

Figure 7 - Transformation vom Kreuz König

In geom_transf_lowercard.m wird genau diese Transformations-Pipeline eingehalten, und somit die untere Karte von Perspektive auf Orthogonal transformiert – Die Boundingbox wird erstellt, Eckpunkte werden berechnet, diese der Transformationsmatrix übergeben, und somit wird ein korrigiertes Bild berechnet.

Ein großer Unterschied besteht allerdings in der Berechnung der Eckpunkte, sowie beim finalen Mapping. Dadurch, dass die untere Karte von der oberen überlappt wird, geht die Information für die vierte (und eventuell sogar dritte) Ecke verloren. Um eine neue Ecke zu erstellen, wird auf das natürliche Seitenverhältnis von Karten zugegriffen.

Da eine volle Karte in der Ratio 8:5 steht, und wir immer (durch unsere vorher getroffene Beschränkung) eine horizontale Hälfte der Karte sehen, erhalten wir somit effektiv sichtbare 4:5. Also können wir die gleiche Transformation anwenden, wenn wir die Zielmatrix mit dem originalen Punkten ([0,0];[0,8];[5,0];[5,8]) auf die neuen Verhältnisse anpassen ([0,0];[0,4];[5,0];[5,4]).

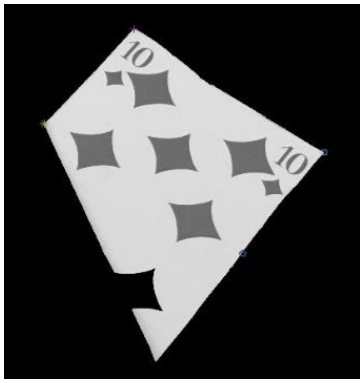

Figure 8 - untere Karte

Nun müssen also nur noch die zu transformierenden Ecken gefunden werden – Drei dieser Ecken sollten schon bestehen. Die vierte wurde abgeschnitten, und somit wird sie zum unteren Schnittpunkt der Karte. Somit verbleibt nur noch eine Fehlerhafte Ecke, der obere Schnittpunkt der Karte, angrenzend an die Längste gerade. Diese Gerade wird gekürzt, bis sie gleich lang wie die kürzeste ist. Somit entsteht an diesem Punkt die neue vierte Ecke, mit Hilfe welcher Transformiert wird, um ein orthogonales Bild zu erhalten.

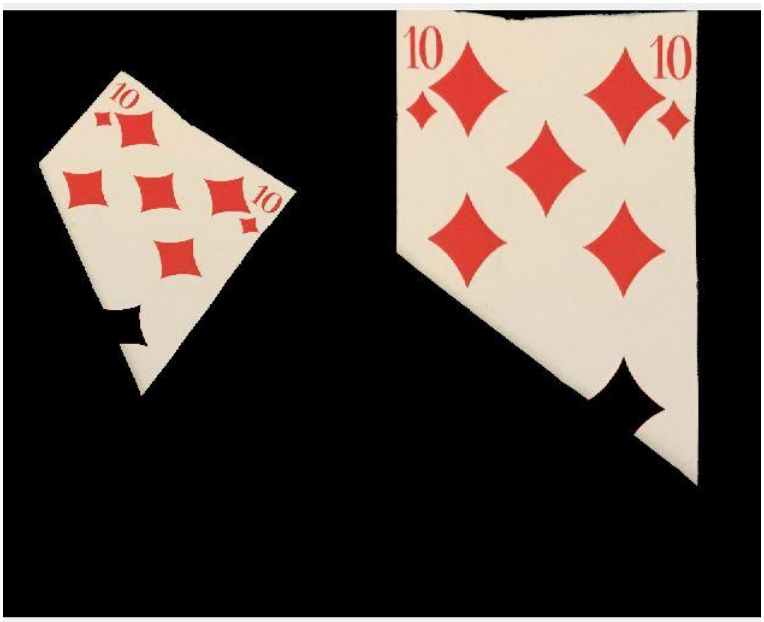

Figure 9 - Transformation der unteren Karte

4.4 Template Matching

Nach der geometrischen Transformation muss die Karte identifiziert werden. Es gibt im Spiel jeweils 4 Symbole (Herz, Pik, Karo, Kreuz) und 4 Buchstaben und eine Zahl (Bube, Dame, König, Zehn, Ass). Es gibt daher insgesamt 9 Template-Bilder.
Das Template Matching wird mit Hilfe einer Korrelationsmatrix umgesetzt. Der Funktion „templateMatcher.m“ werden drei Parameter übergeben. Der Parameter „target“ repräsentiert die Karte, die im Algorithmus untersucht wird.

Dann gibt es noch dem Parameter „template“, der das Template darstellt. Um doppelte Funktionen zu vermeiden, wurde der Parameter „isTOP“ eingeführt, der jeweils angibt, ob es sich um die obere oder untere transformierte Karte handelt.

Der Ablauf der Funktion schaut folgendermaßen aus. Im ersten Schritt wird der Parameter „isTOP“ überprüft. Je nach dem Wert von diesem Parameter wird der Faktor für die äußere for-Schleife gesetzt. Betrachtet man die Sichtbarkeit beider Karten, so kann festgestellt werden, dass die obere Karte immer vollständig sichtbar ist und die untere Karte zu einem gewissen Anteil verdeckt ist. Dies erklärt auch den Faktor, der am Anfang gesetzt wird, da natürlich bei der oberen Karte nicht das ganze Bild durchlaufen werden muss. Es reicht auch, wenn nur die Korrelationswerte der oberen Hälfte berechnet werden, da die untere Hälfte gespiegelt ist und das Template somit nicht immer passend ist.

Figure 10 - Beispiel für das Template Matching der oberen Karte

Danach werden die übergebenen Bilder mit Hilfe der Methodik von Otsu in Grauwertbilder konvertiert, damit in der Schleife die Korrelationswerte berechnet werden können. Würde man mit den Farbkanälen arbeiten, so würde das Ergebnis verfälscht werden.
In der Schleife wird anschließend in jedem Durchlauf ein Ausschnitt aus dem Target-Bild, welches die gleiche Größe wie das Template hat, genommen und mit dem Template verglichen. Die Korrelationswerte werden in eine Matrix gespeichert.
Je höher der Korrelationswert, desto höher ist die Wahrscheinlichkeit, dass das Symbol oder der Buchstabe/Zahl gefunden wurde.
Am Ende wird aus der Korrelationsmatrix der maximale Wert gesucht, dieser repräsentiert anschließend das Matching.

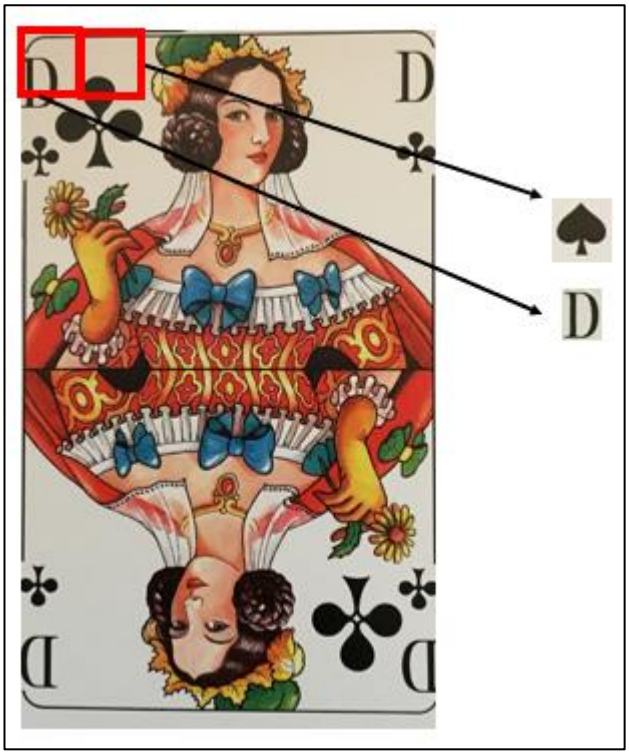

Figure 11 - Ablauf des Algorithmus

5 Evaluierung

Die Datensätze für unser Projekt lassen sich grundsätzlich in zwei Kategorien unterteilen.

Die erste Art der Datensätze sind die Templates, die wir für das Template-Matching einsetzen. Sie umfassen genau 4 Symbole und 5 Alphanumerische Zeichen, welche die Kartentypen Pik, Herz, Karo, Kreuz und den Kartenwert Ass, König, Dame, Bub, 10 beschreiben. Es wäre hier möglich für ein anderes Kartenset, welches sich in der Symbolik unterscheidet, neue Templates zu erstellen um unser Programm auch bei diesem Kartenset einzusetzen.

Die zweite Kategorie der Datensätze bilden dann die Spielverläufe selbst. Grundsätzlich wird nach jedem gespielten Zug ein Foto gemacht, welches dann als Input für das Programm dient. Zu dem verwendeten Aufnahmegerät und den Eigenschaften der Bilder kommen wir später. Der Datensatz der Spielverläufe hat keine feste Anzahl an Bildern und kann somit beliebig groß werden.

Wir haben für das Testen unseres Programms insgesamt drei Datensätze zur Verfügung gestellt. Jeder Datensatz unterscheidet sich vom anderen durch die Anzahl der gespielten Züge und auch in den gespielten Karten beider Spieler, um möglichst viele verschiedene Karten-Kombinationen abzudecken. Der erste Datensatz umfasst 7 Bilder, der zweite 8 Bilder und der dritte Datensatz umfasst 10 Bilder.

Figure 12 - Beispiel eines gespielten Zugs - König sticht Dame

Für die Bilder wird kein spezielles Equipment benötigt. Alle Bilder unserer Datensätze wurden mit einer herkömmlichen Smartphone-Kamera aufgenommen. Die Bilder werden anschließend noch skaliert, um die Laufzeit des Programms etwas zu verbessern.

Die Bilder haben eine Auflösung von 2016x1512 und wurden genau um die Hälfte skaliert, um die Bildgröße unter 1MB zu halten. Es werden je Durchlauf des Programms relativ viele Bildoperationen angewendet, die durchaus rechenintensiv

sind. Mit dieser Bildgröße kann die Laufzeit des Programms relativ gut in einen annehmbaren Bereich gebracht werden.

Der Hintergrund muss einfarbig und so dunkel wie möglich sein, damit man mit unseren angewendeten Algorithmen die Karten verarbeiten kann. In unseren Datensätzen wird ein dunkler grüner Hintergrund verwendet, dass ermöglicht eine fehlerfreie Trennung der beiden Karten vom Hintergrund. Hier sei angemerkt, dass der dunkle Hintergrund eine Voraussetzung ist.

5.1 Evaluierungsfragen

5.1.1 Werden beide Karten erkannt?

Nach zahlreichen Testdurchläufen werden beide Karten - also obere und untere Karte - in allen Fällen erkannt und voneinander getrennt. Die beiden Karten werden nach dem Trennen als separate Bilder gespeichert. Dies kann beispielsweise wie folgt aussehen.

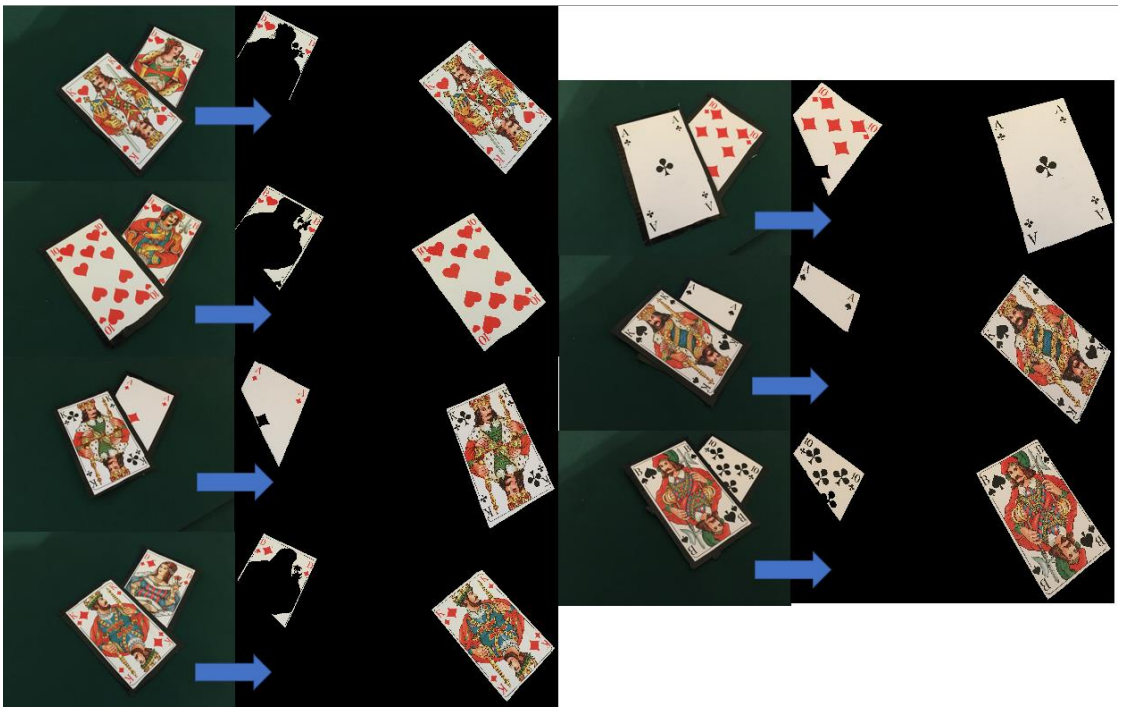

Figure 13 - Kartentrennung - Datensatz 1

In Figure 13 kommt es zu einem Vertauschen der oberen und unteren Karte. Grund dafür ist, dass nach der Umwandlung in ein Binärbild, das Füllen der Löcher nicht so gut funktioniert (siehe auch die ersten zwei Trennungen in Figure 14) wie erwartet und dadurch das Bild des Königs als Hintergrund „erkannt“ wird. Daher ist die Vordergrundfläche der Königs-Karte kleiner als die der Ass-Karte und wird als untere Karte erkannt.

Figure 14 - Oben und Unten vertauscht

In Figure 14 ist der Output nach der Kartentrennung eines kompletten Spiels zu sehen. Auf dem Output-Bild befindet sich auf der rechten Seite immer die vordere Karte und auf der linken Seite die untere Karte. Einige Probleme gab es mit den Karten "10" und "Ass". Diese enthalten keine Texturen, daher überwiegt der Weißanteil und wird auch dann als vordere Karte erkannt, wenn sie unten liegt. Da die Karten nach dem Weißanteil getrennt werden (obere Karte hat im Normalfall immer eine größere weiße Fläche als die untere), mussten hier einige Änderungen vorgenommen werden. Wenn eine der beiden Karten "10" oder "Ass" unten liegen, so muss mehr als 45 Prozent der Karte verdeckt sein, damit sie als untere erkannt wird. Ein gescheitertes Beispiel schaut folgendermaßen aus.

5.1.2 Wird das Bild richtig transformiert, oder werden Buchstaben/Symbole verzerrt?

Die Transformation von der oberen Karte ist in allen Fällen erfolgreich. Da eine Karte horinzentral symetrish ist müssen nur zwischen zwei Fällen unterschieden werden. Nämlich ob die Karte schräg links oder schräg rechts liegt.

Figure 15 - Fallunterscheidung

Bei der Transformation der oberen Karte sind die Bilder minimal bis gar nicht verzerrt. Es gibt nur ein paar Unebenheiten (siehe Ass Figure 15) die praktisch keine Auswirkungen auf das Template-Matching haben. Ansonsten sind die Symbole und Karten gut erkennbar.

Bei der unteren Karte wird das Output-Bild auch zum größten Teil richtig transformiert, jedoch gibt es hier einige Schwierigkeiten.

Figure 16 - Korrekte Transformation

Ein Problem was ab und zu vorkommt ist, dass die Karte zwar richtig transformiert wird, jedoch leicht verzerrt ist. Das Bild wird vertikal in die Länge gezogen, der Grund weshalb es gestreckt wird ist unbekannt. Auch wenn die Karte leicht verzerrt ist, sollte es auf das Template-Matching keine größeren Auswirkungen haben.

Figure 17 - Problem: Streckung der Karte

Ein weiterer Fehler, der immer wieder auftaucht ist, wenn ein Teil der oberen Hälfte der unteren Karte verdeckt ist. Hier wird die Transformation nicht korrekt ausgeführt. Statt die Karte aufrecht zu stellen ist die Transformation um 90° versetzt.

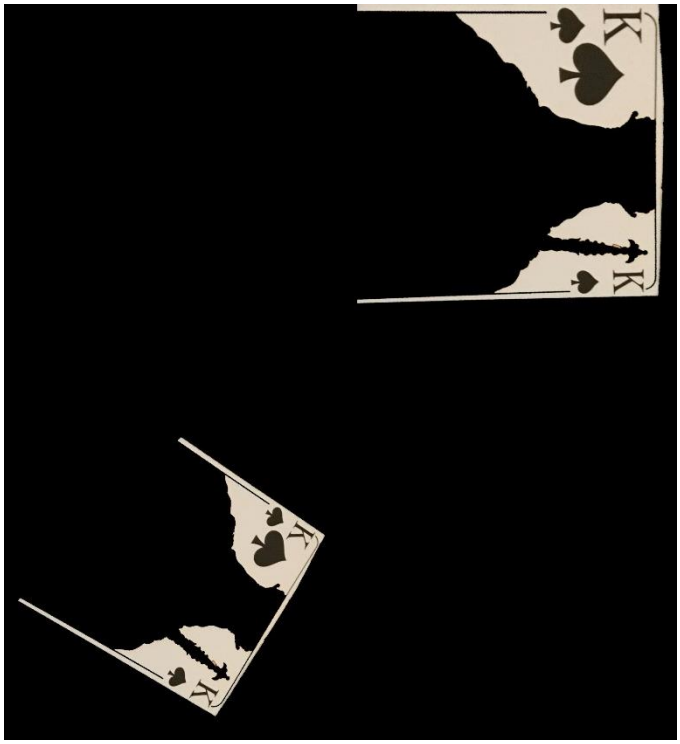

Figure 18 - Problem: 90° versetzt

5.1.3 Wird die Karte richtig identifiziert?

Mit Hilfe einer Korrelationsberechnung werden circa 50% aller Karten identifiziert. Der Template-Datensatz sieht wie folgt aus.

Figure 19 - Templates für die Identifizierung der Karten

In Figure 19 sind alle Templates zu sehen, die für die Identifizierung der Karte benötigt werden. Die Identifizierung wird in zwei Schritte aufgesplittet. Im ersten Schritt wird das Symbol erkannt (untere Zeile Figure 19) und im zweiten Schritt wird anschließend der Buchstabe beziehungsweise die Ziffer (obere Zeile Figure 19) erkannt.

Die identifizierten Eigenschaften der Karten werden anschließend auf der Konsole ausgegeben. Ein Beispiel-Output für den ersten Datensatz sieht wie folgt aus.

Figure 20 - Erfolgreiche Identifizierung der Karten

Anfangs gab es große Schwierigkeiten bei der Unterscheidung zwischen Bube und Dame beziehungsweise Herz und Pik, da beide Templates jeweils sehr ident sind. Dieses Problem wurde mit einer Skalierung und mit verbesserten Templates, die direkt aus dem Output der geometrischen Transformation extrahiert wurde, gelöst.

Wie bereits erwähnt, werden etwas mehr als 50% der Karten erkennt. Die Schwierigkeit hier in der Kombination der Symbole und Buchstaben, aber auch an der fehlerfreien Verarbeitung aller Bilder der Datensätze. Die Karten wurden zusätzlich als Vorbedingung an den Rändern abgeklebt, hier entsteht eine kleine Ungenauigkeit, die sich anschließend bei der geometrischen Transformation bemerkbar macht. Diese Abweichungen sind ausschlaggebend für das Template-Matching.

Figure 21 - Ähnliche Templates (Bube, Dame / Herz/Pik)

6 Schlusswort

Unsere Schlussfolgerung ist, dass der Anfang des Projekts, also das Finden geeigneter Methoden um unsere Problem zu lösen, beziehungsweise einen ersten Prototypen zu entwickeln der schwierigste Schritt für uns waren. Insbesondere das Finden der Karten auf dem Bild, bzw. die Findung der Kanten und Ecken um eine Geometrische Transformation im nächsten Schritt zu ermöglichen. Sobald das erledigt war und wir beim Prototyping eine Methode (Canny-Edge-Detection) gegen eine andere ausgetauscht hatten (Threshold nach Otsu) ging es sehr zügig voran und wir konnten die Zeit die wir aufholen, welche wir beim Prototyping verloren hatten.

Zurzeit gibt es noch Probleme, wenn der Hintergrund zu hell ist, werden die Karten teilweise nicht getrennt, da unser Programm die Karten mittels Threshold nach Otsu und der Zusammenhangskomponente trennt. Wird dann der Hintergrund als Vordergrund erkannt und ist eine größere Zusammenhangskomponente als die kleinere Karte, wird diese nicht mehr erkannt und stattdessen wird der Hintergrund als 2. Karte erkannt.

Ein weiteres Problem bildet das Template-Matching bei der unteren Karte. Bei der oberen Karte, machen wir uns die Eigenschaften einer Spielkarte zu Nutze, da sie Achsensymmetrisch ist und beschränken uns beim Matching auf das erste ¼ der Karte. Dies hält die Laufzeit sehr niedrig, schwächt aber nicht die Erfolgsrate. Bei der unteren Karte haben wir es nicht geschafft sicherzustellen, dass eine „nicht-abgeschnittene“ Ecke immer an derselben Stelle ist, und müssen daher beim Matching die ¾ der Karte überprüfen, dass die Laufzeit erhöht.

Eine große Verbesserung unserer Lösung wäre, wenn die Karten nicht mit einem schwarzen Rand präpariert werden müssten. Beziehungsweise wenn man einen Schritt weitergeht, dass nicht ein bestimmtes Kartendeck verwendet werden muss, sondern ein beliebiges (Kartedecks unterscheiden sich oft in der Symbolik, was bei uns dazu führt, dass die Karten nicht mehr richtig identifiziert werden).

Literatur

[1] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/threshold.pdf, 12.11.2017

[2] https://www.cse.unr.edu/~bebis/CS791E/Notes/Thresholding.pdf, 25.12.2017

[3] https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf, 20.12.2017