jugend forscht

2019

# Tims Trio Tracker 2.0 – Ein Löser zum Triospiel mit (künstlicher) Intelligenz

Tim Palm

Klasse 11

LGH Schwäbisch Gmünd

Betreuer: Dr. Olga Lomonosova / Alexander Schönborn

Kategorie: Mathematik/Informatik

# Kurzfassung

In der Arbeit wurde eine Lösungshilfe für das Mathematik-Logikspiel Trio in Form einer Handy-App entwickelt. Das eigentliche Auffinden der Lösungskombination ist bei Kenntnis der Spielfeldanordnung recht einfach, da das Spielfeld lediglich systematisch zu durchsuchen ist. Vielmehr bestand die Schwierigkeit darin, die Spielfeldanordnung aus einem Handyfoto, welches nur aus einzelnen Bildpunkten besteht, sicher zu erkennen. Dazu wurde ein Bilderkennungsverfahren auf Basis künstlicher Intelligenz entwickelt, wobei Mechanismen eingebaut wurden, die mögliche Verdrehungen des Feldes, perspektivische Verzerrungen, Störelemente und inhomogene Hintergründe behandeln. Mit einer Erkennungsrate von 99.65% zeigt das Verfahren gute Ergebnisse. Neben der Anwendung zur Erkennung eines Triospielfeldes könnte das Verfahren bspw. auch bei der Erkennung von Verkehrsschildern beim autonomen Fahren zum Einsatz kommen.

# Inhalt

# 1 Motivation und Zielsetzung

Beim Mathematikspiel Trio geht es darum, eine zufällig gezogene Zahl, in einer 7x7-Matrix aus Ziffern zwischen 1 und 9 mithilfe einer Dreierkombination darzustellen, wobei davon zwei beliebige Zahlen multipliziert und die Verbleibende addiert oder subtrahiert wird. In meinem Matheunterricht steht dieses Spiel häufig auf der Tagesordnung, oft auch mit der zusätzlichen Herausforderung, mehrere Kombinationen zu einer Suchzahl zu finden. Um zu prüfen, ob überhaupt noch eine Kombination für eine gegebene Suchzahl existiert, habe ich im Jahr 2016 im Rahmen meines damaligen Jugend-Forscht-Projektes ein Programm geschrieben, welches ein Triospielfeld analysiert und sofort alle möglichen Dreierkombinationen anzeigt, die zu einer eingegebenen Suchzahl passen. Der praktische Nutzen war jedoch insofern begrenzt, als man die 7x7-Matrix per Hand an einem Computer eingeben musste, was einer spontanen Nutzung des Programms im Wege stand. Aus diesem Umstand heraus entstand die Idee, den damals entwickelten Suchalgorithmus in eine Handy-App zu implementieren. Man würde mit dieser dann ein Foto des aktuellen Spielfeldes machen und die App würde sofort alle relevanten Trios einfärben. Dabei stellte sich nun als größtes Problem heraus, die 49 Ziffern im Foto zu lokalisieren und zu benennen, bevor der eigentliche Suchalgorithmus angewendet werden konnte. Mustererkennung stellt für Maschinen immer noch eine große Herausforderung dar. Im konkreten Fall lag die Hauptaufgabe darin, einen Erkennungsprozess für Triospielfelder zu entwickeln, welcher für beide Spielmotive (s. Abb.1) gilt und auch Abweichungen im Hintergrund, in Anordnung der Kärtchen und in der Beleuchtung zulässt.

Abb. 1: Spielmotiv 1 (oben) und 2 (unten)

# 2 Eingesetzte Methoden zur Bilderkennung (Programmablauf)

Die entwickelte und in der Programmiersprache C# umgesetzte Methode zur Bilderkennung für das Trio-Spielfeld lässt sich grob in drei Abschnitte einteilen.


Abb. 2: Schematische Darstellung der Bilderkennung

Im ersten Schritt wird das Bild durch Anwendung von Filtern und Transformationen auf den späteren Erkennungsschritt vorbereitet. Im Verlauf der Segmentierung werden verschiedene Methoden angewendet, um die Nutzsignale (die Ziffern) vom Hintergrund zu trennen und im Endeffekt 49 Pixelgruppen zu identifizieren, bei denen es sich um die relevanten Ziffern handelt. Diese Regionen werden dann im abschließenden Schritt auf spezifische Eigenschaften hin untersucht, um sie einer der neun Klassen (Ziffern 1 – 9) zuzuordnen. Das Schaubild in Abb.3 zeigt eine Übersicht der implementierten Arbeitsschritte bis hin zur Erkennung der Ziffern, bevor der eigentliche Suchalgorithmus für die Trio-Kombinationen ansetzt. Die einzelnen Schritte werden nachfolgend näher erläutert.

## 2.1 Vorbearbeitung

In Vorbereitung der Segmentierung und Erkennung werden die im Folgenden beschriebenen Methoden eingesetzt, die zum einen weitere Bilddarstellungen erzeugen, und zum anderen daraus Informationen zur späteren Nutzung entnehmen.

### 2.1.1 Skalierung des Bildes

Da moderne Handys Fotos mit einer hohen Auflösung von vielen Megapixeln aufnehmen, wird die Datenmenge zunächst dadurch verringert, dass die längere Bildseite unter Beibehaltung des Seitenverhältnisses auf 1000 Pixel reduziert wird. Dabei wird das Originalbild in einer Schleife zeilenweise in Schritten gemäß


**Abb. 3: Ablaufplan der Bilderkennung**

($Breite_{alt}/Breite_{neu}$) bzw. in Spaltenrichtung gemäß ($Höhe_{alt}/Höhe_{neu}$) durchlaufen und die entsprechenden RGB-Farbwerte in eine neu skalierte Matrix übertragen.

### 2.1.2 Umwandlung in Graustufen und Farbhistogramme

Einen weiteren Beitrag zur Komplexitätsreduktion liefert die Umwandlung des Farbbildes in ein Graustufenbild. Dieser Schritt vereinfacht die Suche nach Nutzregionen, da nun nur noch einmalig ein Farbraum von 0 bis 255 vorliegt. Dazu wird jedes Farbpixel entsprechend folgender Formel in seine entsprechende Graustufe umgerechnet [1].

$$G = 0.299R + 0.587G + 0.114B \quad (1)$$

Diese Formel berücksichtigt, dass das Auge unterschiedliche Farben gleicher Helligkeit in unterschiedlicher Intensität wahrnimmt. Gleichzeitig wird für jeden Farb- bzw. Grauwert von 0 bis 255 zur späteren Bildung eines

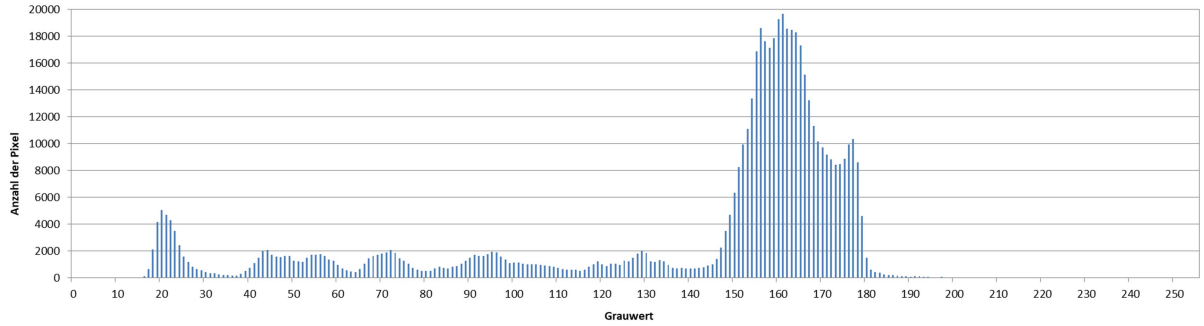


Abb. 4: Beispiel für ein Grauwerte-Histogramm

Histogramms ein entsprechender Zähler inkrementiert. Ein Beispiel-Histogramm ist in Abb. 4 zu sehen. Dieses Histogramm wird später zur Bestimmung eines Schwellenwertes der Schwarz-Weiß-Umwandlung eingesetzt.

### 2.1.3 Bestimmung der Bildkanten mit Sobelfilter

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

x-Filter

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

y-Filter

Die nächste Methode zur Informationsgewinnung besteht darin, alle Kanten im Bild zu identifizieren. Eine Kante ist durch einen sprunghaften Wechsel des Grauwertbereichs gekennzeichnet. Zur Bestimmung dieser Regionen kommt ein sogenannter Sobel-Operator [2] zum Einsatz, der auf das Graustufenbild angewendet wird. Dazu werden die links dargestellten 3x3-Filter auf das Bild wie folgt appliziert: Der Filter wird über jedes Pixel des Bildes gelegt und die dadurch überdeckten Grauwerte werden mit den entsprechenden Werten des Filters multipliziert. Sämtliche Produkte werden dann addiert und repräsentieren nun die Kantenintensität in x- bzw. y-Richtung. Die resultierende Kantenintensität wird gemäß

$$|G| = \sqrt{G_X^2 + G_Y^2} \tag{2}$$

bestimmt und die Summe dem Kantenbild als Grauwert übergeben. Um hier einen Höchstwert von 255 zu gewährleisten, werden die Filterwerte noch mit ihrem Maximalwert normiert. Abb.5 zeigt beispielhaft zwei Kantenbilder der beiden Spielmotive. Weiterhin wird aus den beiden kartesischen Kantenintensitäten für jedes Pixel auch ein entsprechender Kantenwinkel gemäß Gl. 3 zugeordnet.

$$\alpha = arctan\left(\frac{G_Y}{G_X}\right) \tag{3}$$

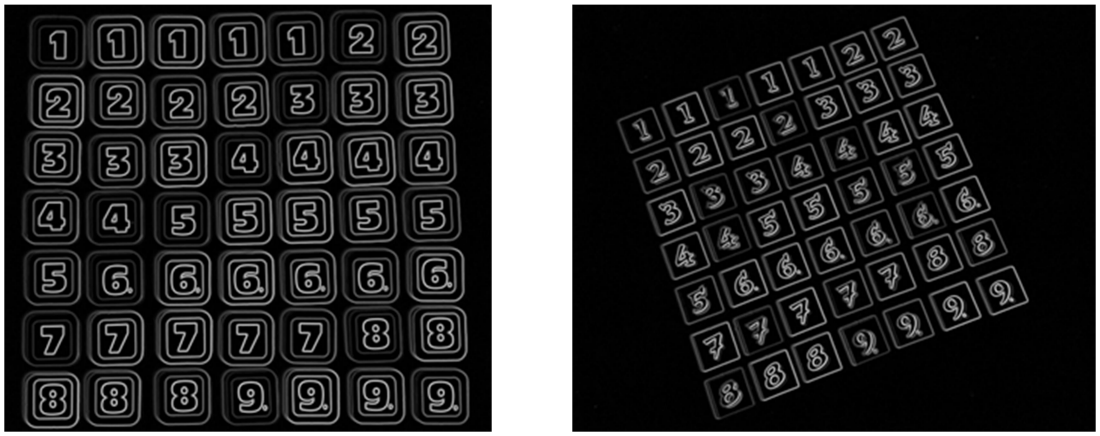

Abb. 5: Kantenbilder zweier Spielfeldfotos

### 2.1.4 Rotation des Bildes

Für den Fall, dass das Spielfeld etwas verdreht aufgenommen wurde, ist eine Drehkorrektur implementiert worden. Dazu bildet man aus den zuvor bestimmten Kantenwinkeln zunächst ein Histogramm der Winkel. Abb.6 zeigt die entsprechende Darstellung für die Fotos in Abb.5. Dadurch, dass die Kärtchen in der Regel parallel zueinander liegen, bilden sich bei Bild 1 (Abb.5 links) Maxima bei 90°, 180°, 270° und 360° aus. Die Verdrehung von Bild 2 in Höhe von ca. 25° spiegelt das Histogramm anschaulich wieder.
Der Drehwinkel wird nun dadurch bestimmt, dass im Winkelintervall von 45° bis 135° das absolute Maximum gesucht wird und dieses von der Idealstellung 90° abgezogen wird. Die Drehkorrektur wurde zunächst gemäß [3], [4] mit folgenden Gleichungen durchgeführt.

$$x' = \left(x - \frac{b}{2}\right)\cos(\alpha) + \left(y - \frac{h}{2}\right)\sin(\alpha) + \frac{b}{2} \qquad y' = -\left(x - \frac{b}{2}\right)\sin(\alpha) + \left(y - \frac{h}{2}\right)\cos(\alpha) + \frac{h}{2} \tag{4}$$

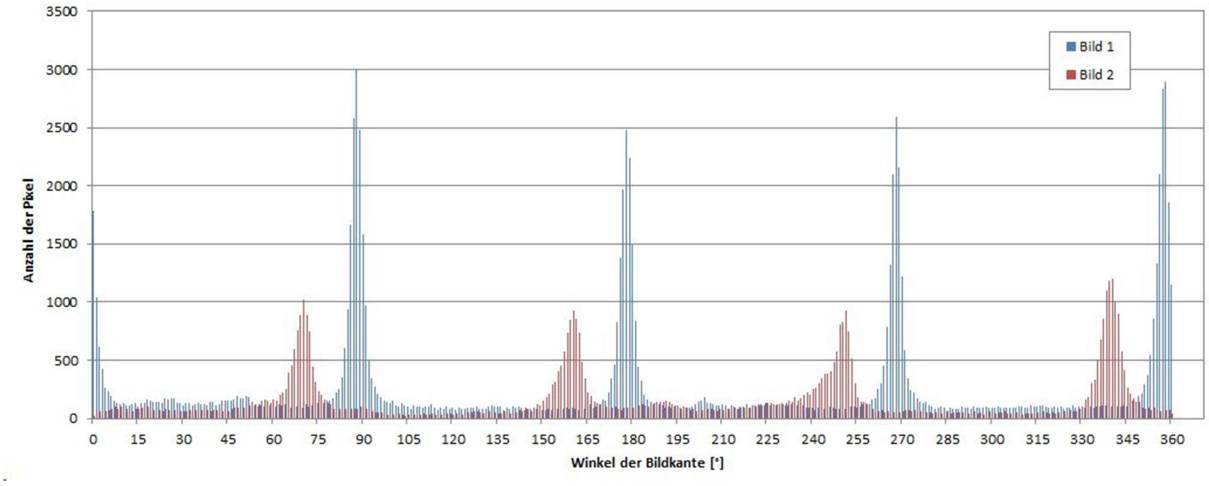


Abb. 6: Das Winkelhistogramm für die Bilder in Abb. 5

Hierbei ist $\alpha$ der Drehwinkel, (x,y) der Quellpunkt und (x‘,y‘) der Zielpunkt. Es ist anzumerken, dass die Formeln gegenüber [3] für das verwendete Fotokoordinatensystem mit dem Ursprung in der oberen linken Bildecke und x nach rechts angepasst wurden. Weiterhin ist eine Drehung um die Bildmitte und nicht um den Ursprung erforderlich, so dass die Formeln noch eine Verschiebung des Drehpunktes in die Bildmitte enthalten. Abb.7 zeigt das Ergebnis einer Drehkorrektur, bei der das Quellbild zeilenweise durchlaufen wird und jeder Farbwert in den entsprechend verdrehten Punkt im Zielbild übertragen wird. Es ist sichtbar, dass sich bei diesem Vorgehen ein Raster von schwarzen Punkten gebildet hat. Aufgrund von Rundungseffekten werden manchen Zielpunkten zwei Quellpunkte und anderen Zielpunkten gar keine Quellpunkte zugeordnet. Im letztgenannten Fall bleibt das Pixel auf seinem Initialwert von Null und somit schwarz.

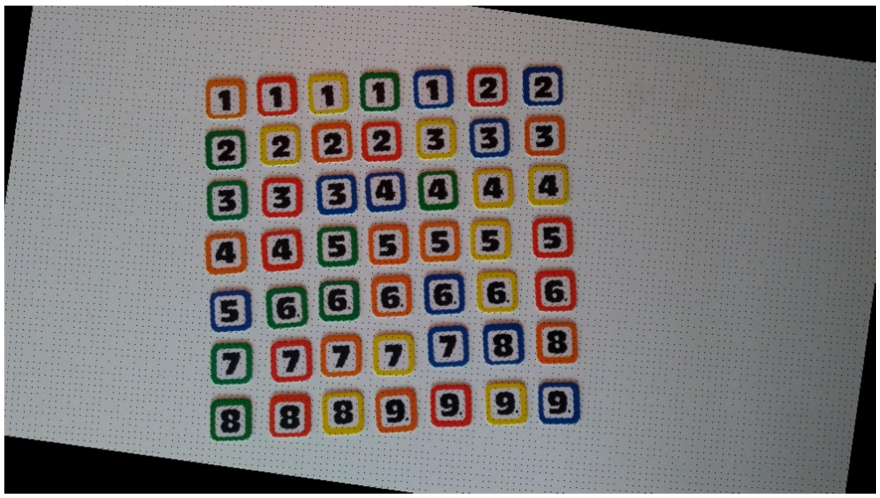

Abb. 7: Gedrehtes Bild mit scharzen Fehlstellen

Aus diesem Grund musste ein alternativer Ansatz für die Drehung gefunden werden, der jedem Zielpixel eine Farbzuordnung garantiert. Anstelle einer Schleife über alle Quellpixel wurde nun einfach über alle Zielpixel gelaufen und der entsprechende Quellpixel über die inverse Drehung bestimmt. Dazu war es erforderlich Gl.4 nach den Quellkoordinaten x und y umzustellen. Mit der Website Wolfram Alpha [15] ergaben sich folgende Gleichungen.

$$x = -\frac{-x'\cos(\alpha) + \frac{b}{2}\cos(\alpha) - \frac{b}{2}\cos(\alpha)^2 + y'\sin(\alpha) - \frac{h}{2}\sin(\alpha) - \frac{b}{2}\sin(\alpha)^2}{\cos(\alpha)^2 + \sin(\alpha)^2}$$

$$y = -\frac{-y'\cos(\alpha) + \frac{h}{2}\cos(\alpha) - \frac{h}{2}\cos(\alpha)^2 - x'\sin(\alpha) + \frac{b}{2}\sin(\alpha) - \frac{h}{2}\sin(\alpha)^2}{\cos(\alpha)^2 + \sin(\alpha)^2} \quad (5)$$

Die Quellkoordinaten x,y werden in der Regel nicht ganzzahlig sein, so dass kein Farbwert direkt ausgelesen werden kann. Der erste Ansatz, hier auf das nächste Pixel zu runden, wurde wegen Klötzchenbildung im Zielbild verworfen. Eine bessere Qualität wurde erzielt, wenn der Farbwert an den ermittelten Quellkoordinaten mittels bilinearer Interpolation bestimmt wurde.

## 2.2 Segmentierung

### 2.2.1 Bestimmung der Spalten- und Zeilenstruktur durch Fourier-Analyse

Für die spätere Segmentierung der Ziffern erwies sich als vorteilhaft, eine Abschätzung über die Lage der einzelnen Kärtchen vorzunehmen. Dies beugt dem Risiko vor, durch einen inhomogenen Hintergrund Phantomziffern zu segmentieren. Es wurde versucht, die Tatsache zu nutzen, dass die Kärtchen laut Spielprinzip immer ein regelmäßiges 7x7-Raster bilden. Es wurde folgender Ansatz entwickelt, diese Regelmäßigkeit zu detaillieren. Dazu wurden zunächst die X-Projektionen im Sobelbild (s. Abb. 5) bestimmt. Hierzu wurde zeilenweise die Summe aller Grauwerte mit einem Schwellenwert über 60 gebildet. Dadurch, dass die Ziffern sowie die Kärtchenränder selbst intensive Kanten bilden, war zu erwarten, dass die Projektionen über eine Kärtchen-Zeile hinweg eine hohe Projektionssumme ergeben. Abb. 8 zeigt die Darstellung der Projektionssummen über die Bildhöhe. Die sieben Maxima für die sieben Spielfeld-Zeilen sind gut zu erkennen. Im Folgenden galt es, die Lage dieser Maxima und somit der Zeilen zu bestimmen. Dazu wurden die ersten k Fourier-Koeffizienten von der Kurve gebildet [5].

$$a_k = \frac{2}{m}\sum_{i=1}^{m} x_i \cos\left(k\, i \frac{2\pi}{m}\right) \qquad b_k = \frac{2}{m}\sum_{i=1}^{m} x_i \sin\left(k\, i \frac{2\pi}{m}\right) \tag{6}$$

*m = Anzahl der Kurvenpunkte; $x_i$ = i-ter Kurvenpunkt*

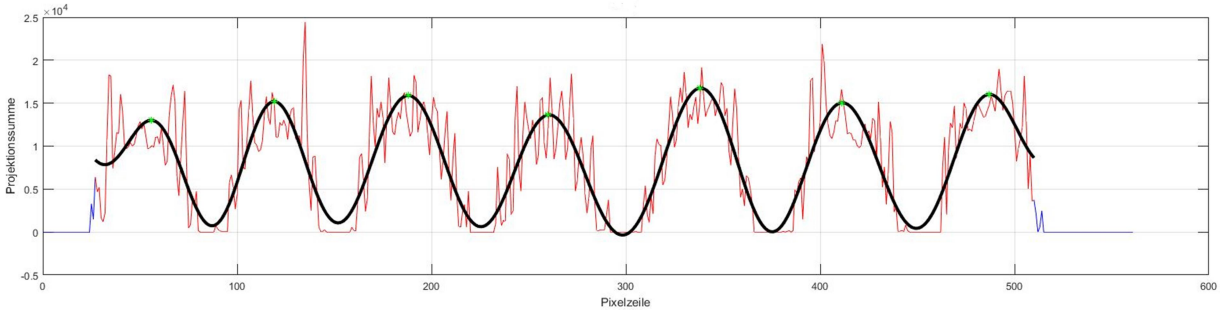


**Abb. 8: Darstellung der X-Projektionen (rot und blau) sowie eine Annäherung durch eine Fourier-Reihe (schwarz)**

Dabei macht man sich zu Nutze, dass jede beliebige Funktion als Summe der Kreisfunktionen Sinus und Kosinus mit aufsteigender Frequenz dargestellt werden kann. Mit dem Fourier-Koeffizienten nach Gl.6 geschieht dies gemäß:

$$x(i) = \frac{1}{2}a_0 + \sum_{k=1}^{\infty} a_k \cos\left(k\, i\, \frac{2\pi}{m}\right) + b_k \sin\left(k\, i\, \frac{2\pi}{m}\right) \tag{7}$$

Es hat sich gezeigt, dass mit den ersten zehn Fourier-Koeffizienten (k = 10) die wesentliche Charakteristik der Projektionskurve gut wiedergegeben werden kann (s. Abb. 8). Die hochfrequenten Schwankungen sind verschwunden, so dass man nun die Lage der Maxima und somit die Lage der Kärtchen-Zeilen bestimmen kann. Wegen des nun glatten Kurvenverlaufs kann die Lage der sieben Maxima nun durch die Bedingung bestimmt werden, dass der jeweilige Punkt ein lokales Maximum darstellt, dessen beiden unmittelbaren Nachbarpunkte einen geringeren Wert aufweisen.
Für die Y-Richtung (Bestimmung der Lage der Spalten) war das Vorgehen analog. Allerdings wurden die Y-Projektionen nun für jede Zeile separat durchgeführt. Somit können auch etwaige perspektivische Verzerrungen (s.Abb.9 rechts) behandelt werden, da die Spaltenlage nun für jede Zeile separat bestimmt wird.
Mit den Werten der Spalten- und Zeilenkoordinaten ist die Lage der Kärtchen nun bekannt. Diese Information wird im weiteren Verlauf noch an verschiedenen Stellen verwendet. Abb. 9 zeigt das Ergebnis dieser Prozessstufe. Die Quadrate um den Mittelpunkt der Kärtchen haben eine Kantenlänge von $0.6 * $ (Abstand der Peaks aus der Fourier-Analyse).

### 2.2.2 Bestimmung der vorliegenden Spielvariante

Der Hauptunterschied zwischen den beiden Motiven (s. Abb. 1) liegt darin, dass die Kärtchen in der zweiten Variante zum einen keinen Rand besitzen, sondern komplett eingefärbt sind, und dass zum anderen die Ziffern weiß sind. Der zweite Unterschied wird genutzt, um das vorliegende Spielmotiv zu bestimmen. Dazu wird nun die relative Anzahl der dunklen Pixel in jedem geschätzten Kärtchen aus Kapitel 2.2.1 (s. Abb. 9) ermittelt.

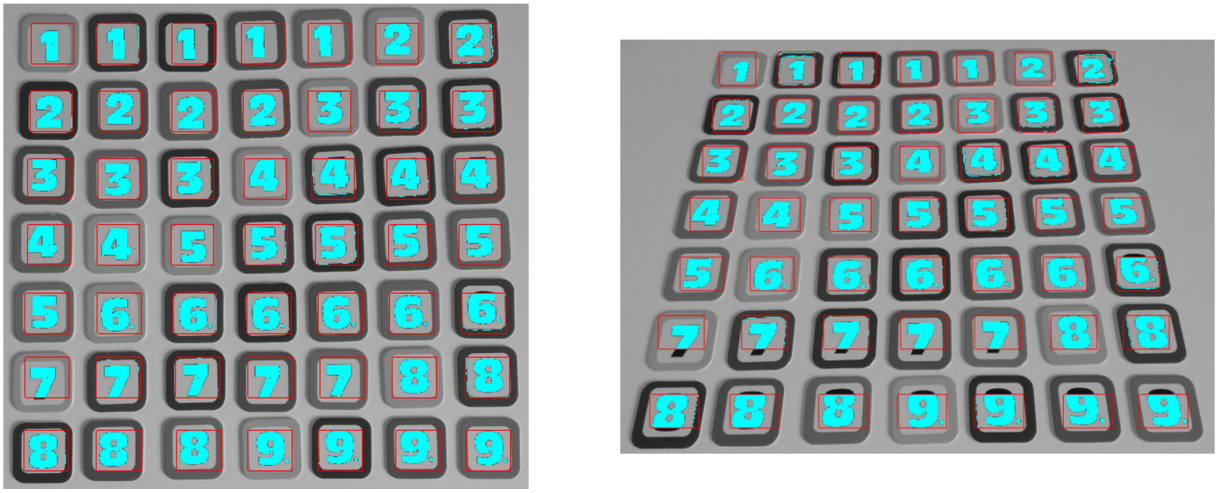

Abb. 9: Vermutete Lage der Kärtchen. (rechts bei einem perspektivisch-verzerrtem Bild)

Wenn der Mittelwert der relativen Anzahlen der größten 30 Elemente oberhalb eines Schwellenwertes liegt, wird angenommen, dass es sich um Spiel 1 handelt, da in Spiel 2 kaum dunkle Pixel auf den Kärtchen vorkommen. Die Bedingungen dafür, dass ein Pixel gezählt wird, sind die Folgenden:

- Grauwert < 60, (was alle Ziffern einschließt)
- RGB-Wert(e) jeweils < 120, (was dunkelfarbige Kärtchenhintergründe aus Spiel 2 ausschließt)
- Standardabweichung der RGB-Werte < 30, (bei dunklen Ziffern sind die einzelnen RGB-Werte ähnlich)

### 2.2.3 Invertierung des Bildes

Um in der Segmentierung und Klassifizierung gleiche Methoden anwenden zu können, werden im Fall von Spielmotiv 2 das Farb- und das Graustufenbild invertiert sowie deren Farbhistogramme aktualisiert. Zur Invertierung jedes Pixels wird der aktuelle Farbwert von 255 abgezogen und das Ergebnis als neuer Pixelwert gesetzt. So sind auch bei Spiel 2 dunkle Ziffern gewährleistet.

### 2.2.4 Umwandeln des Bildes in ein Binärbild

Zur weiteren Vereinfachung der Bilderkennung soll das Graustufenbild in ein Schwarz-Weiß-Bild (Binärbild) umgewandelt werden. Der Hintergrund soll hierbei weiß und der Vordergrund schwarz sein. Um eine automatische Umwandlung gewährleisten zu können, muss ein Schwellenwert, also eine Graustufe gefunden werden, der den Vorder- vom Hintergrund trennt. Zur Bestimmung des Schwellenwertes wurde das jeweilige Farbwertehistogramm aus Kapitel 2.1.2 zu Hilfe genommen. Zum einen wurde der am häufigsten vorkommende Grauwert ausgelesen und zum anderen der Schwerpunkt des Histogramms bestimmt. Dazu wurden die mit den jeweiligen Vorkommnissen gewichteten Grauwerte addiert und anschließend durch die Gesamtpixelzahl dividiert. Folgende Beziehungen aus nebenstehender Tabelle wurden auf Basis einer Vielzahl von Testfotos abgeleitet. Wegen der Kärtchenstruktur von Spielmotiv 2 musste ein alternativer Ansatz gefunden werden. Dazu wurden Fotos unterschiedlicher Helligkeit und somit unterschiedlicher Schwerpunktlagen des Farbwertehistogramms ausgewählt. Für diese Fotos wurde jeweils der kleinste und größte Schwellenwert bestimmt, bei dem noch eine zufriedenstellende Trennung von Vorder- und Hintergrund möglich war (Abb. 10, Punkte). Man kann eine Art Korridor erkennen, in den eine quadratische Funktion gelegt wurde. Anhand dieser wird der Schwellenwert für das Spielmotiv 2 in Abhängigkeit des Schwerpunktes bestimmt. Nun wird eine Schleife über alle Pixel des Graubildes gelegt. Jeder Grauwert, der unterhalb des Schwellenwertes liegt, wird zu einer 0 (schwarz) und die Grauwerte, die oberhalb liegen, werden zu einer 1 (weiß).

| **Wenn…** | **… dann** |
|---|---|
| SP > 110 & Max < 190 | SW = Max - 60 |
| Max > 190 | SW = SP - 60 |
| anderenfalls | SW = SP |
| SP = Schwerpunkt / SW = Schwellenwert | |

### 2.2.5 Entfernen von Fehlregionen gemäß Farbcode

Zur weiteren Verarbeitung werden alle Pixelwerte im Binärbild von +1 durch -1 ersetzt, so dass die Pixel bei der späteren Gruppierung auch die Labelnummer 1 annehmen können. Weiterhin werden Pixel, deren entsprechender RGB-Wert im Farbbild über 120 (bei Spiel 2 über 150) liegt, mit einer 0 überschrieben, da diese Pixel zum Kärtchenrand gehören. Ebenso werden alle Pixel zu Hintergrundpixeln, wenn sie eine Standardabwiechung ihres RGB-Wertes von über 30 (bei Spiel 2 von über 60) aufweisen, was ebenfalls auf farbige Ränder hinweist. Das Merkmal von dunklen Ziffernflächen ist, dass ihre RGB-Werte zum einen klein sind, und dass sie zum anderen alle gleichverteilt sind. Mit diesen Filtern können nun eine Vielzahl von Fehlpixeln entfernt werden.

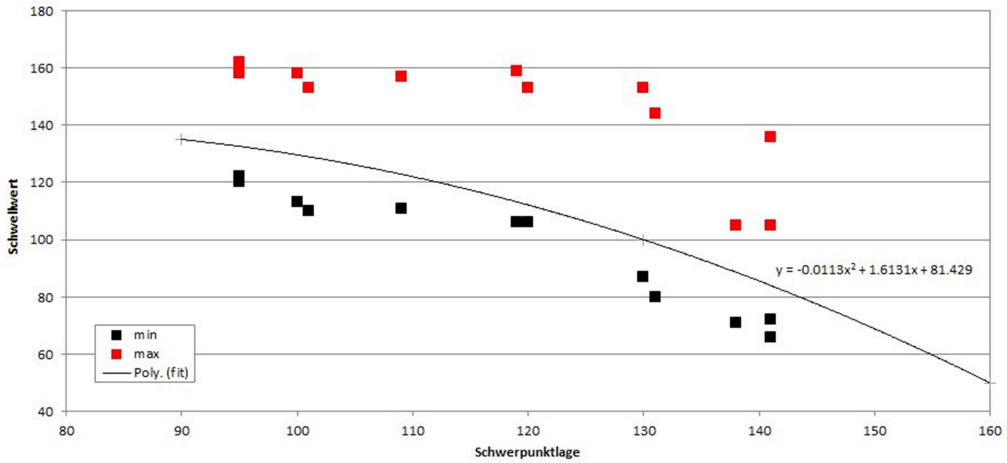


Abb. 10: Schwellenwert abhängig von jeder Schwerpunktlage (rechts wurde der Funktionsterm angegeben)

### 2.2.6 Zusammenhängende Bildregionen erkennen

Nachdem nun ein Teil aller Fehlregionen entfernt worden war, folgte nun die Bestimmung der zusammenhängenden Bildregionen. Dies wurde mit Hilfe des Connected-Components-Algorithmus [6] erreicht. Eine einfache und kompakte Variante ist ein rekursiver Ansatz. Dabei wird das Bild zeilenweise durchlaufen. Für jedes Pixel im Vordergrund (beim ersten Durchlauf ist dies das erste Pixel mit einer -1) werden die Nachbarn bestimmt, welche ebenfalls im Vordergrund sind. Für jeden gefundenen Nachbarn wird dieses Vorgehen rekursiv wiederholt bis jedem Pixel der Komponente die aktuelle Labelnummer zugewiesen ist. Anschließend wird die aktuelle Labelnummer um 1 erhöht. Dieser Algorithmus kann jedoch nur bei Komponenten mit weniger als ca. 20000 Pixeln durchgeführt werden, da infolge der Verschachtelungstiefe ansonsten ein Stapelüberlauf-Fehler auftritt. Alternativ existiert eine sequenzielle zweiphasige Variante des Algorithmus. Auch hier wird das Bild zeilenweise abgefahren. Bei jedem Vordergrundpixel werden dann seine drei obigen und der eine linke Nachbar geprüft (Abb. 11). Der kleinste Labelwert der vier Nachbarn wird für das aktuelle Pixel übernommen. Gleichzeitig wird in einer Baumstruktur gekennzeichnet, welche Labelnummern eine Nachbarschaftsbeziehung aufweisen (höhere Labelwerte sind „Children“ des kleineren übernommenen Labelwertes). Falls keine gültigen Vordergrund-Nachbarn existieren, wird ein neuer Labelwert zugewiesen. In der zweiten Phase des Algorithmus wird das Bild erneut zeilenweise durchlaufen und gemäß der vorher abgelegten Baumstruktur der Labelbeziehungen jedes Label durch sein oberstes Parentlabel ersetzt. Um fortlaufende Parentlabelnummern zu erhalten, werden diese als Ergänzung noch komprimiert.

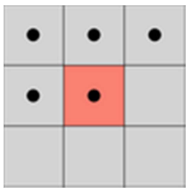

Abb. 11: In rot das aktuelle Pixel. Die Punkte zeigen die relevanten Nachbarn an. [6]

### 2.2.7 Entfernen von Fehlregionen gemäß geometrischer Eigenschaften

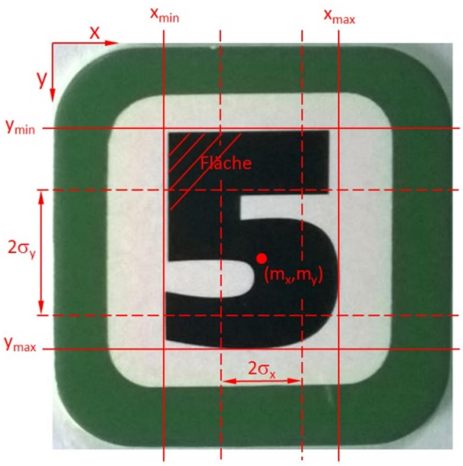


Abb. 12: geometrische Eigenschaften einer Region

Aufgrund von Hintergrundstörungen ist die Wahrscheinlichkeit hoch, dass mehr als 49 Komponenten erkannt werden. Als Maßnahme dagegen wurden mehrere Stufen von Filtern implementiert, die auf Basis von geometrischen Eigenschaften der Region diese als mögliche Ziffer erkennen. Die in Abb.12 gezeigten Eigenschaften wurden bei allen erkannten Regionen abgeleitet. Diese wurden mit Grenzwerten verglichen, die zuvor anhand von Probebildern bestimmt wurden. Abb. 13 veranschaulicht beispielhaft das Vorgehen für eine geometrische Größe. Die Datenpunkte spiegeln die Standardabweichung in x-Richtung für eine Auswahl von Ziffernregionen in Abhängigkeit der Anzahl der Pixel pro Region wieder. Die Punkte bilden einen linear ansteigenden Korridor, der durch zwei Geraden eingegrenzt werden kann. Diese Geraden bilden somit die Grenzwerte, mit denen beliebig erkannte Regionen als Ziffern klassifiziert werden können. Analog wurden Grenzwerte für alle anderen geometrischen Eigenschaften bestimmt und auf die Regionen angewendet. Je nach Beschaffenheit des Bildhintergrundes können Fehlregionen auftreten, die auch innerhalb der bestimmten Grenzwerte liegen. Die Kreise in Abb. 13 stellen beispielsweise solche Fehlregionen dar. Von daher ist nicht sichergestellt, dass durch diesen Filter lediglich Ziffern übrig bleiben. Daher wurden drei weitere Filterstufen entwickelt.

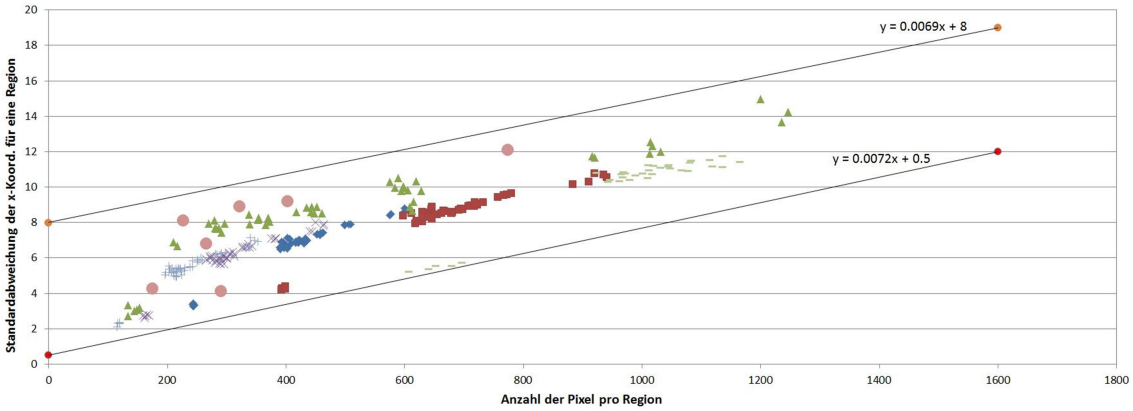


Abb. 13: Geometrische Eigenschaften der Stabw. in Abhängigkeit von der Pixelanzahl (die Kreise sind Fehlkomponenten)

Die zweite Filterstufe bestand darin, die Tatsache auszunutzen, dass Zahlen infolge ihres Hintergrunds von hohen Farbwerten im Sobelbild (Kante) umgeben sind. Zunächst wird eine Komponente nur dann als gültig angesehen, wenn ihre Erstreckung mehr als vier Pixel vom Bildrand entfernt ist. Weiterhin wird in jeder verbliebenen Region in einem um drei Pixel erweiterten Bereich jedes Pixel mit einem Farbwert größer als 80 im Sobelbild gezählt. Durch den Farbkontrast muss jede Ziffer von hohen Intensitäten im Sobelbild umrandet sein. Daher liefert eine große Anzahl dieser Pixel einen Hinweis auf eine Ziffer. Ist deren Anzahl größer als die dreifache Wurzel der Regionsfläche, so wird diese Region als mögliche Ziffer behalten. Anderenfalls werden sie verworfen.

Die dritte Filterstufe wird nur dann ausgeführt, wenn es nach Abschluss der zweiten Filterstufe immer noch mehr als 49 gültige Komponenten gibt. Hier werden die Informationen aus Abschnitt 2.2.1 genutzt und sämtliche Regionen aussortiert, deren Mittelpunkt nicht in einem der in Abb. 9 dargestellten Quadrate liegt.

Liegen auch danach immer noch mehr als 49 Komponenten vor, kommt eine vierte Filterstufe zum Einsatz. Diese nutzt die erwartete Gitterstruktur des Triofeldes aus. Aus Abschnitt 2.2.1 wurden die Zeilenhöhen des Spielfeldes abgeschätzt. Nun werden für jede Region die Nachbarregionen bestimmt, deren Mittelpunkte sich innerhalb der vertikalen Zeilenbegrenzung für die aktuelle Region befinden. Wurden für eine Region weniger als sechs Nachbarregionen gefunden, ist diese nicht Teil der Gitterstruktur und somit keine Ziffer.

### 2.2.8 Umwandeln in eine 64x64-Matrix

Die erkannten Komponenten liegen bislang noch ungeordnet innerhalb einer Spielfeldzeile vor, da der Connected-Components-Algorithmus wegen des zeilenweisen Durchlaufens die Komponenten nach ihrer Höhenlage sortiert. Um dem entgegenzuwirken, wurden die Komponenten innerhalb einer Zeile nach deren X-Position umgeordnet. Dies wurde mit Hilfe des Bubble-Sort-Algorithmus [7] realisiert. Dieser Algorithmus ist zwar relativ ineffizient, da aber lediglich sieben Komponenten pro Zeile sortiert werden, fällt diese Eigenschaft nicht ins Gewicht. Er funktioniert folgendermaßen: Über jedes Element wird ein Vergleich mit seinem Nachbarn vollzogen. Sollte der aktuelle Wert größer sein als der Nachbarwert, werden die Elemente getauscht. Dies wird solange durchgeführt, bis irgendwann kein Element mehr getauscht wurde (dann sind alle Elemente sortiert).

Für die spätere Erkennung ist es von Vorteil, wenn die Zahlen alle in einem einheitlichen Format vorliegen. Somit wurden alle Ziffern auf eine einheitliche Größe skaliert. Die Entscheidung fiel dabei auf die Verwendung einer 64x64-Matrix, da bei kleineren Auflösungen sonst später die geometrischen Merkmale beim Erkennen zu hohe Schwankungen aufweisen. Dazu wird das Verhältnis p aus der ursprünglichen und der gewünschten Auflösung gebildet. Im Anschluss wird jedes p-te Pixel in die neue Matrix übertragen. Ist die originale Zahl bspw. doppelt so groß (128x128) wie die gewünschte Auflösung, dann wird nur jedes 2. Pixel übertragen. Bei einem Hintergrundpixel wird 0 übertragen und bei einem Vordergrundpixel 1.

## 2.3 Erkennung der Ziffern mithilfe von geometrischen Merkmalen

Nach der Segmentierung wird davon ausgegangen, dass es sich bei den 49 unterschiedlichen Regionen um die gesuchten Ziffern handelt. Bei der nun folgenden Klassifizierung gilt es, jeder der Regionen eine Ziffer von 1 bis 9 zuzuweisen, um den eigentlichen Suchalgorithmus zur Ermittlung der Triolösungen anwenden zu können.

Dazu wurden die im Folgenden beschriebenen zwölf geometrischen Eigenschaften einer Region bestimmt. Bei deren Auswahl wurde sichergestellt, dass die Eigenschaften invariant gegenüber Rotation waren. Zwar wird eine globale Verdrehung des Spielfeldes zu Beginn erkannt und korrigiert (s. Kap. 2.1.4), jedoch können noch immer einzelne Kärtchen lokal verdreht sein.

### 2.3.1 Bestimmung von Momenten

Momente sind hilfreich, um einzelne Objekte in einem segmentierten Bild zu beschreiben. Sie sind ein Maß dafür, wie die Pixel um den Schwerpunkt der Region herum verteilt sind. Ein Moment der Ordnung pq wird nach folgender Formel definiert [8]:

$$m_{pq} = \sum_x \sum_y x^p y^q \, I(x,y) \tag{8}$$

*x, y sind die Koordinaten des aktuellen Pixels; I(x, y) ist der Grauwert am Pixel x, y; p, q sind die Ordnungsnummern des Moments.*

Demnach werden in einer Schleife über alle Pixel die Summen aus deren Grauwerten, multipliziert mit den jeweiligen Koordinaten zur Potenz p bzw. q, gebildet. Beim Moment $m_{00}$ werden beispielsweise nur alle Farbwerte addiert. Somit ergibt sich die Fläche der Vordergrundkomponente. Mithilfe von Momenten höherer Ordnung können weitere Eigenschaften bestimmt werden, die eine Ziffernregion beschreiben. Die Schwerpunktkoordinaten lassen sich bspw. folgendermaßen ausdrücken:

$$\bar{x} = \frac{m_{10}}{m_{00}} \qquad \bar{y} = \frac{m_{01}}{m_{00}} \tag{9}$$

*$\bar{x}, \bar{y}$ ist hierbei die Flächenschwerpunktlage in x- bzw. y-Richtung.*

Alle diese Momente sind jedoch nicht translationsinvariant. Um dies zu erreichen, wird die allgemeine Definition (8) um einen Term ergänzt:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q \, I(x,y) \tag{10}$$

*x, y sind die Koordinaten des aktuellen Pixels; I(x, y) der Grauwert am Pixel x, y; $\bar{x}, \bar{y}$ sind die Flächenschwerpunktlagen in x- bzw. y-Richtung.*

Dies sind die sogenannten zentralen Momente. Hier werden von den jeweiligen Koordinatenwerten die Mittelwerte sofort abgezogen. Somit ist es egal, ob ein Koordinatenwert bei gleichem Objekt groß oder klein ist, da sein jeweiliger Abstand zum Mittelwert immer derselbe ist. Diese Momente sind jedoch noch nicht skalierungsinvariant. Dies kann durch Normierung mit einer Potenz der Regionsfläche ($m_{00}$) erreicht werden [8]. Diese normierten zentralen Momente sind wie folgt definiert:

$$\eta_{pq} = \frac{\mu_{pq}}{m_{00}^a} \qquad a = \frac{p+q+2}{2}, \text{ gilt nur für } p+q > 2 \tag{11}$$

Rotationsinvariante Momente $\phi_i$ lassen sich nun durch Kombinationen aus den normierten zentralen Momenten (s. Gl.11) ableiten. Dies geschah zuerst durch den chinesischen Mathematiker Ming-Kuei Hu, weshalb diese Momente auch „Hu-Momente“ genannt werden.

$$\phi_1 = \eta_{20} + \eta_{02} \tag{12}$$

$$\phi_2 = (\eta_{20} - \eta_{02})^2 + 2\eta_{11}^2 \tag{13}$$

$$\phi_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{14}$$

$$\phi_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{15}$$

Mithilfe dieser Momente lassen sich die segmentieren Ziffern nun charakterisieren. Abb. 14 zeigt für das in Abb.7 gezeigte Spielfeld die ersten vier Hu-Momente, normiert mit ihrem jeweiligen Maximalwert. Es ist gut zu erkennen, dass sich die Wertebereiche der Hu-Momente für die unterschiedlichen Ziffern zum Teil drastisch unterscheiden, so dass die Werte als Erkennungsmerkmal herangezogen werden können. Interessant ist die Ähnlichkeit der Momente bei der 6 und der 9, was eine Bestätigung für die Rotationsinvarianz der Momente ist.

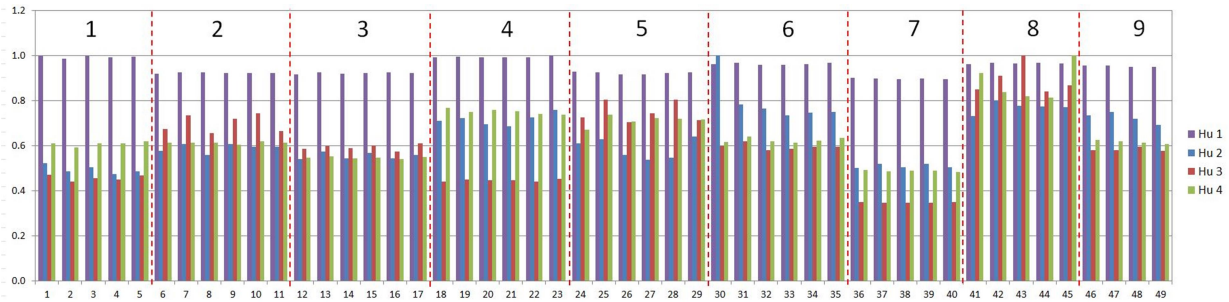


**Abb. 14: Hu-Momente für das Spielfeld in Abb.7**

### 2.3.2 Bestimmung der Rundheit einer Komponente

Als weiteres Unterscheidungsmerkmal zwischen den Ziffern wurde die Rundheit einer jeden Komponente bestimmt. Die Rundheit eines Objektes ist folgendermaßen definiert [9]:

$$R = 4\pi \frac{A}{U^2} \tag{16}$$

*U ist der Umfang der Komponente und A deren Fläche.*

Der Maximalwert, den die Rundheit annehmen kann, ist 1. Dieser Wert wird einem Kreis zugeordnet, da der Kreis bei gegebener Fläche grundsätzlich den kleinsten Umfang besitzt. Ein Quadrat besitzt gemäß Gl.16 eine

Rundheit von 0.785, ein gleichseitiges Dreieck einen Wert von 0.6 und ein Rechteck mit dem Seitenverhältnis 1:10 einen Wert von 0.26. Zur Berechnung der Rundheit einer erkannten Region wird allerdings ihr noch unbekannter Umfang benötigt. Dieser wurde mithilfe des Moore-Neighborhood-Algorithmus bestimmt. Bei diesem wird zuerst ein Startpixel ermittelt, indem eine Schleife die Ziffernmatrix spaltenweise durchläuft. Sobald man ein zur Komponente zugehöriges Pixel gefunden hat (dieses ist das erste Pixel des Umfangs), wird für diesen das erste Nachbarpixel gesucht, welches ebenfalls dem Vordergrund zugehörig ist. Dabei verläuft die Suche gegen den Uhrzeigersinn und beginnt bei demjenigen Nachbarpixel, welches als letztes vor Auffinden des ersten Umfangpixels durchlaufen wurde. Der erste gefundene Pixel ist dann der nächste Umfangspixel, für welches wiederrum das nächste Nachbarpixel entsprechend der Umlaufsvorschrift gesucht wird. Der Algorithmus stoppt, sobald das Startpixel wieder von der gleichen Richtung betreten wurde, wie es zu Beginn (bei der Startpixelsuche) geschehen war. Abb. 15 zeigt die berechnete Rundheit für das in Abb. 7 gezeigte Spielfeld. Hier kann man erkennen, dass die Eigenschaften in den einzelnen Zahlengruppen nahezu gleich sind, aber im Vergleich der Zifferngruppen untereinander Unterschiede aufweisen.

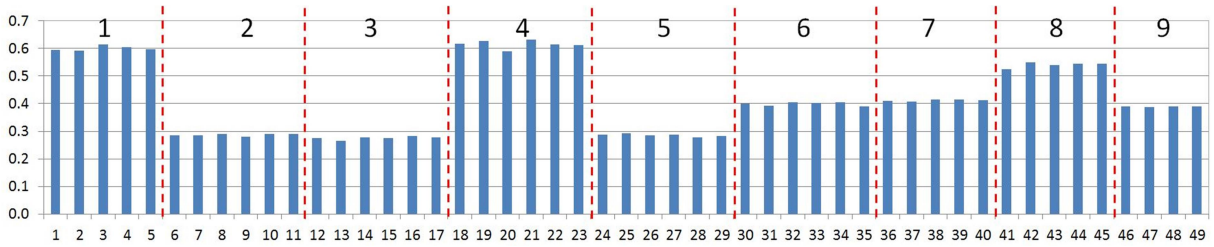


Abb. 15: Rundheit der verschiedenen Zahlen

### 2.3.3 Fläche der konvexen Hülle

Abb. 16: Konvexe Hülle um die Zahl 7.

Die nächste Eigenschaft ist die Fläche der konvexen Hülle. Eine konvexe Hülle schließt eine Punktemenge derart ein, dass man immer zwei beliebige Punkte miteinander verbinden kann, ohne dass diese Verbindung die Hülle schneidet. Abb. 16 zeigt die konvexe Hülle der Ziffer 7. Nun erhofft man sich, dass durch den Flächeninhalt der konvexen Hülle Rückschlüsse auf die Ziffer gezogen werden können. Dabei wird das Verhältnis zwischen der Fläche der konvexen Hülle und der Fläche der Komponente gebildet.

Die konvexe Hülle wird mithilfe des Giftwrapping-Algorithmus [11] bestimmt. Dieser wiederum baut auf der Orientierung [12] auf, welche drei Punkte zueinander besitzen können. Diese kann entweder gegen den Uhrzeigersinn oder im Urzeigersinn sein. Der Algorithmus geht dabei folgendermaßen vor: Es wird ein Startpunkt p gefunden, der am weitesten links liegt. Dies ist der erste Punkt der konvexen Hülle. Von diesem wird ein Dreieck mit einem zu prüfenden Punkt l sowie einem beliebigen weiteren Punkt gebildet. Der letzte Punkt des Dreiecks wird für alle Punkte der Region (außer p und l) durchgewechselt und die Orientierung des Dreiecks geprüft. Wenn alle Punkte durchgetauscht wurden und das Dreieck stets eine Orientierung im Uhrzeigersinn hatte, dann liegt der anfangs gewählte Punkt l auf der konvexen Hülle. Andernfalls wird ein beliebiger neuer Punkt für l gewählt und die Prüfung der Orientierung startet von neuem. Der neue Punkt der konvexen Hülle bildet nun den ersten Punkt des Dreiecks und mit der Prüfung eines neuen Punktes l gegen alle verbliebenen Punkte der Region werden die gleichen Schritte durchlaufen, um den nächsten Punkt der konvexen Hülle zu finden.

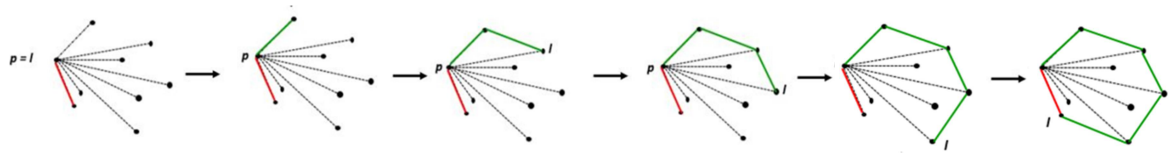


Abb. 17: Ablauf des Giftwrapping-Algorithmus

Sind alle Eckpunkte der konvexen Hülle gefunden, muss nun deren Fläche bestimmt werden. Dies geschieht nach [13] in einer Schleife über die Punkte der konvexen Hülle gemäß folgender Gleichung:

$$A = \frac{1}{2}\left(\sum (x_{alt} + x_{neu})(y_{alt} - y_{neu})\right) \tag{17}$$

*$x_{alt}$ bzw. $y_{alt}$ sind die Koordinaten des vorherigen Punktes auf der konvexen Hülle, während $x_{neu}$ bzw. $y_{neu}$ die Koordinaten des nächsten sind.*

Für diese Fläche wird nun das Verhältnis zur Fläche der jeweiligen Zahl gebildet. Die Ergebnisse für das Spielfeld in Abb. 7 sind in Abb. 18 aufgetragen.

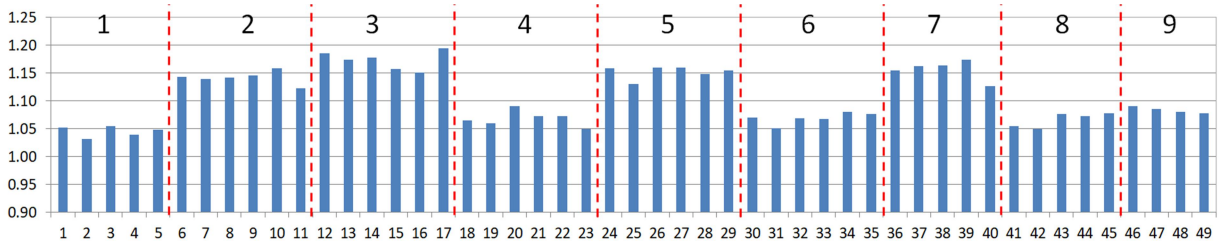


Abb. 18: Verhältnis der Fläche der konvexen Hülle zur Fläche der Zifferregionen für das Spielfeld aus Abb.7

### 2.3.4 Bestimmung von elliptischen Fourier-Deskriptoren

Dieses Merkmal macht sich die Unterschiede im Verlauf der Berandungen der einzelnen Ziffern zu Nutze. Abb. 19 veranschaulicht das Vorgehen dabei.

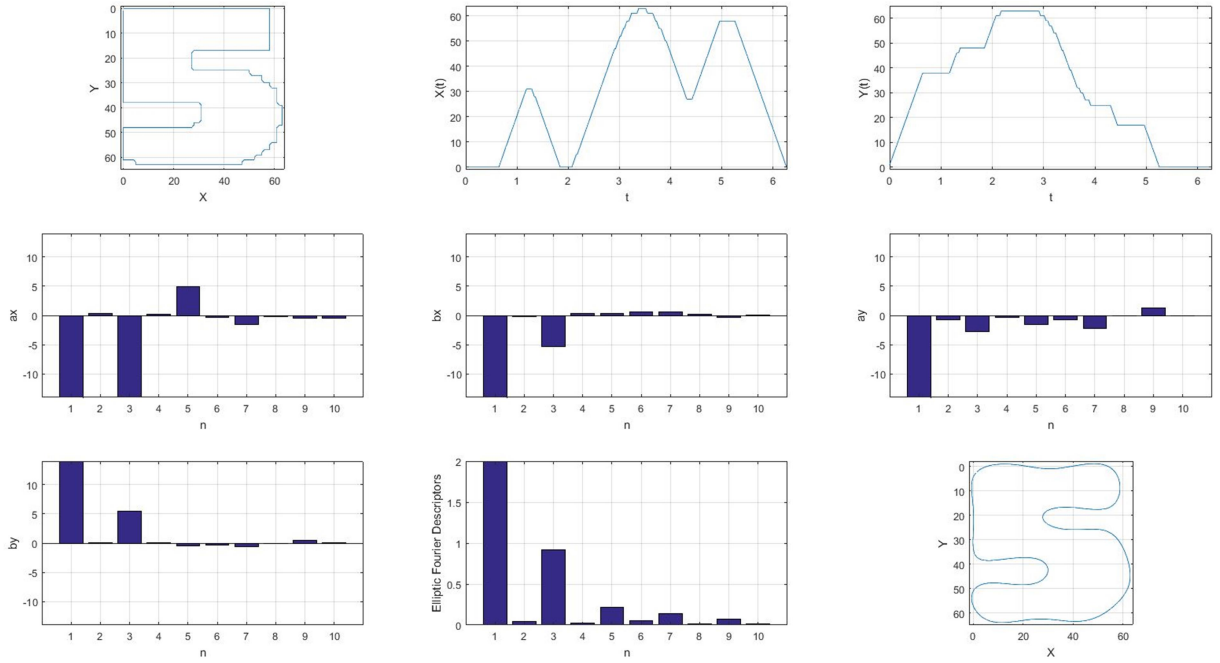


Abb. 19: Schritte zur Darstellung einer Ziffer durch Fourier-Deskriptoren

*Beschreibung von Abb.19 von oben links nach unten rechts:* Die Darstellung des Randes der Zahl 5 im ersten Diagramm wird mithilfe des Moore-Neighborhood-Algorithmus (s. Kap. 2.3.2) realisiert. Die beiden Diagramme daneben zeigen die Darstellung der x- bzw. y-Koordinate über der Bogenlänge, die bereits auf eine Länge von 2π normiert wurde. Die Kurven starten jeweils im Ursprung (linke obere Ecke) gegen den Uhrzeigersinn. Dann werden gemäß Gl. 6 die Fourier-Koeffizienten bis zur Ordnung 10 für die beiden Kurven oben rechts bestimmt. Diese Koeffizienten sind in den Diagrammen 4 bis 7 aufgetragen. Die Verteilung dieser Koeffizienten ist bereits charakteristisch für die unterschiedlichen Ziffern, aber noch nicht rotationsinvariant. Dazu errechnet man nach [5] die elliptischen Fourier-Deskriptoren (Gl. 18). Die Ergebnisse dazu sind im vorletzten Diagramm abgebildet.

$$EFD_k = \frac{\sqrt{a_{x_k}^2 + a_{y_k}^2}}{\sqrt{a_{x_1}^2 + a_{y_1}^2}} + \frac{\sqrt{b_{x_k}^2 + b_{y_k}^2}}{\sqrt{b_{x_1}^2 + b_{y_1}^2}} \tag{18}$$

*a und b aus Gl. (6)*

Das letzte Diagramm in Abb. 19 zeigt schließlich die Ziffer 5, reproduziert nach Gl. (7) aus den ersten sechs Fourier-Koeffizienten. Abb. 20 zeigt für das Spielfeld aus Abb. 7 die verwendeten fünf Fourier-Deskriptoren der Ordnung 2 bis 6. Die charakteristische Verteilung wird auch hier wiederum deutlich.

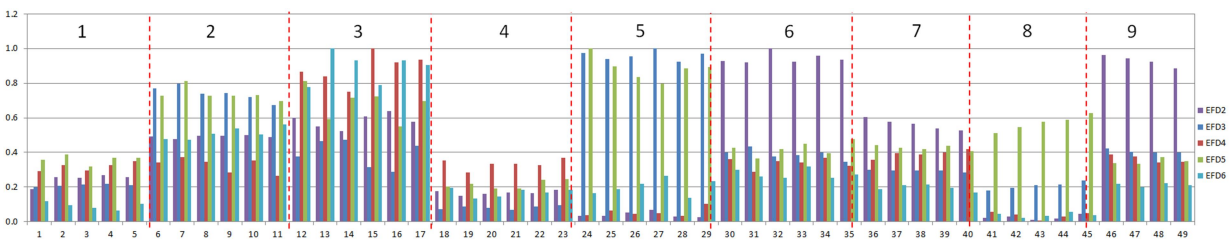


Abb. 20: elliptische Fourier-Deskriptoren für alle Ziffern

### 2.3.5 Bestimmung von Hintergrundeinschlüssen

Mit eventuellen Hintergrundeinschlüssen wird ein letztes Merkmal bestimmt. Zwar besitzen nur die Ziffern 4, 6, 8 und 9 Einschlüsse (6, 8 und 9 bei Spielmotiv 2), jedoch ist dieses Kriterium für die betroffenen Ziffern sehr eindeutig. Wegen der mit 64x64 Pixeln recht kleinen Datenmenge kann hierzu die in Kapitel 2.2.6 beschriebene rekursive Variante des Connected-Components-Algorithmus benutzt werden. Zusätzlich wird bei jedem Pixel geprüft, ob es am Rand der Matrix liegt, und dann entsprechend ein Flag gesetzt. Nur Regionen, die keine Randpixel aufweisen (und größer als 16 Pixel sind), sind innere Einschlüsse. Pro Ziffer werden die Anzahl der inneren Regionen (z. B. zwei bei der Ziffer 8, eins bei der Ziffer 6) sowie deren Labelnummer zur späteren Auswertung gespeichert.

### 2.3.6 Klassifizierung der einzelnen Ziffern

Die zwölf Merkmale pro Ziffer aus den Kapiteln 2.3.1-2.3.5 bilden einen Merkmalsvektor, anhand dessen die Ziffern erkannt werden sollen. Im Programm sind als Referenz Merkmalsvektoren von drei ausgewählten Spielfeldern für jedes der beiden Spielmotive hinterlegt. Für jeden der 49 (k) Merkmalsvektoren (f) wird ein Abstandsmaß d zu jedem der 147 (3x49) Referenzvektoren (r) gebildet. Formel (19) veranschaulicht dies:

$$d_{k,r} = \sum_{i=1}^{11} \left| \frac{r_i - f_i}{f_i} \right| + 20(f_{12} - r_{12})^2 \tag{19}$$

Demnach wird für die ersten 11 Merkmale die Summe der relativen Abweichungen der einzelnen Komponenten bestimmt. Wegen der möglichen Division durch Null wird für das letzte Merkmal der Hintergrundeinschlüsse nur eine einfache Differenzbildung vorgenommen. Aufgrund der Eindeutigkeit dieses Merkmals wird es noch mit dem Faktor 20 multipliziert, um es höher zu gewichten. Zur Bestimmung der Ziffer wird nun ein minimales Abstandsmaß $d_{k,r}$ zum Referenzvektor r gesucht. Zur Erhöhung der Sicherheit wird nicht einfach das absolute Minimum ausgewertet, sondern die fünf kleinsten der 147 Abstandsmaße pro Ziffer. Die Ziffer, die am häufigsten von den fünf gefundenen Referenzvektoren repräsentiert wird, wird als die endgültig erkannte Ziffer gesetzt.
Aufgrund der Rotationsinvarianz sämtlicher Merkmale sind die Merkmals- und Referenzvektoren bei der 6 bzw. 9 quasi identisch (bei Spielmotiv 2 unterscheiden sie sich geringfügig), so dass eine Unterscheidung mit den bisherigen Methoden nicht möglich ist. Deshalb wird im Falle einer erkannten 6 oder 9 die Ziffernmatrix nochmals zeilenweise durchlaufen und der Mittelwert der Koordinaten der in Kap. 2.3.5 bestimmten Hintergrundeinschlüsse bestimmt. Liegt der Mittelwert der Höhenkoordinate in der oberen Hälfte der Matrix (m < 32), handelt es sich um eine 9, anderenfalls (m > 32) um eine 6.
Somit sind nun alle 49 Ziffern segmentiert und klassifiziert. Zum Abschluss wird nun noch der Suchalgorithmus gestartet, der alle möglichen 3er-Kombinationen der 7x7-Spielfeldmatrix nach der gesuchten Zahl scannt. Alle gefundenen Trios werden dem Nutzer dann auf dem Spielfeld angezeigt.

## 2.4 Erkennung der Ziffern anhand eines neuronalen Netzes

Als alternativer Ansatz zu den in Kap 2.3 beschriebenen Methoden wurde ein neuronales Netz für die Klassifikation implementiert. Das Prinzip von neuronalen Netzen besteht darin, auf Basis einer großen Anzahl von Trainingsdaten relevante Ziffernmerkmale eindeutig zu extrahieren und diese dann im späteren Erkennungsprozess anzuwenden. Ein neuronales Netz besteht aus einer Eingabe- und einer Ausgabeschicht, sowie aus versteckten Schichten zur Ausbildung der Erkennungsmerkmale [14]. Die einzelnen Schichten besitzen eine problemabhängige Anzahl von Knotenpunkten (Neuronen). Sämtliche Knotenpunkte benachbarter Schichten sind über sogenannte Gewichte miteinander verknüpft, sodass nach bestimmten Rechenvorschriften Informationen im neuronalen Netz weiter transportiert werden. Der Aufbau ist in Abb. 21 nochmals veranschaulicht.

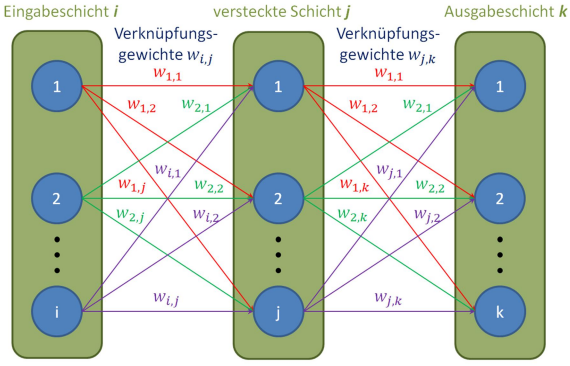


Abb. 21: Aufbau eines dreischichtigen neuronalen Netzes

### 2.4.1 Initialisierung des neuronalen Netzes

Gewählt wurde ein dreischichtiges neuronales Netz mit 64 x 64 = 4096 Eingabeknoten und neun Ausgabeknoten. Wie in Kapitel 3 erläutert wird, wurde die Knotenzahl der versteckten Schicht auf 400 festgelegt. Sämtliche Gewichte werden anfangs zufällig gesetzt. Der Wertebereich wird durch den Kehrwert der Wurzel aus der Knotenzahl der vorherigen Schicht bestimmt $(1/\sqrt{n})$. Die Lernrate $lr$ wird auf 0.1 und die Lernepochen $ep$ werden auf 10 gesetzt. Als Aktivierungsfunktion kommt die Sigmoid-Funktion zum Einsatz:

$$y = \frac{1}{1 + e^{-x}} \tag{20}$$

*Sigmoidfunktion: x ist der Eingangswert in eine Schicht, y der entsprechende Ausgabewert, e ist die euklidische Zahl.*

### 2.4.2 Abfragen des neuronalen Netzes

Die 4096 Pixel der segmentierten Ziffern werden den Eingabeknoten übergeben. Der Eingabewert $E_j$ an jedem Knoten der versteckten Schicht wird durch die Summe der Eingabewerte, multipliziert mit dem jeweiligen Gewichten der Verbindungslinien, gebildet (siehe Gl. 21). Die Ausgabewerte der versteckten Schicht erhält man durch Anwendung der Aktivierungsfunktion auf die entsprechenden Eingangswerte:

$$E_j = \sum_i w_{ij} \cdot A_i \tag{21}$$

*$E_j$ ist der Eingangswert am Knoten j; $w_{ij}$ ist das Gewicht zwischen Knoten i und j; $A_i$ ist der Ausgabewert des Knotens i der vorherigen Schicht.*

Der Ausgabewert der aktuellen Schicht lautet dann (in der Eingabeschicht gilt $A_i = E_i$):

$$A_j = \frac{1}{1 + e^{-E_j}} \tag{22}$$

*$E_j$ ist der Eingangswert am Knoten j; e ist die euklidische Zahl; $A_j$ ist der Ausgabewert des Knotens j*

Diese Schritte wiederholen sich zur Ausgabeschicht hin. Bei korrekter Gewichtsverteilung würde derjenige Ausgabeknoten einen vergleichsweise hohen Wert aufweisen, dessen Index der eingespeisten Ziffer in der Eingabeschicht entspricht. Um diese optimale Gewichtsverteilung zu erreichen, muss das Netz zuvor jedoch trainiert werden.

### 2.4.3 Trainieren des neuronalen Netzes

Der Trainingsprozess geschieht im Endeffekt dadurch, dass die Gewichte des neuronalen Netzes entsprechend der Abweichungen von Soll- und Ist- Werten (Fehler F) an den Ausgabeknoten angepasst werden. Die Fehler in der Ausgabeschicht werden bei jedem Trainingsbild zunächst per Backpropagation entsprechend der Gewichtsanteile auf die versteckte Schicht zurückgeführt. Jedes der Verknüpfungsgewichte wird dann proportional zur Ableitung des Fehlers nach diesem Gewicht korrigiert:

$$\Delta w_{jk} = -lr \frac{\partial F_k}{\partial w_{jk}} \tag{23}$$

*lr ist die Lernrate; $F_k$ ist der Fehler am Knoten k; $w_{jk}$ ist das Gewicht zwischen d. Knoten j und k; $\frac{\partial F_k}{\partial w_{jk}}$ ist die partielle Ableitung von $F_k$ nach $w_{jk}$*

Die Lernrate soll zu große Änderungen der Gewichte vermeiden. Das negative Vorzeichen stellt sicher, dass die Korrektur der Gewichte in Richtung des Minimums zeigt. Die Ableitung in Gl. 23 lautet gemäß [14] wie folgt:

$$\frac{\partial F_k}{\partial w_{jk}} = -2A_j(T_k - A_k)\frac{1}{1 + e^{-\sum_j w_{jk} \cdot A_j}}\left(1 - \frac{1}{1 + e^{-\sum_j w_{jk} \cdot A_j}}\right) \tag{24}$$

*$A_j$ ist der Ausgabewert am Knoten j der vorherigen Schicht; $T_k$ ist der Sollwert am Knoten k; $A_k$ ist der Ausgabewert am Knoten k; e ist die euklidische Zahl; w sind die jeweiligen Gewichte.*

Der Fehlerterm $(T_k - A_k)$ kann so nur in der letzten Schicht bestimmt werden. Zur Korrektur der Gewichte in allen anderen Schichten wird er durch die rückgeführten Fehler aus der Backpropagation ersetzt.
Für den Trainingsprozess wurden für jedes der beiden Spielmotive 39 Fotos herangezogen. Dabei wurden drei verschiedene Hintergründe gewählt (weißer Hintergrund, Parkett sowie Motivteppich). Für diese drei Gruppen wurden die Spielfelder jeweils durch Drehungen und Verschiebungen modifiziert, um möglichst viele Spielfeldlagen in das Training einzubinden. Abb. 22 zeigt die 39 Spielfelder für Spielmotiv 1 an. Es wurden zunächst alle einzelnen Ziffern mit den Methoden aus Kapitel 2.2 segmentiert, wobei gerade bei den Bildern auf dem Teppich alle Filter aus Kap. 2.2.7 zum Einsatz kommen mussten, um die vielen Hintergrundregionen zu entfernen. Dem neuronale Netz wurden somit 39 x 49 = 1911 Ziffern bestehend aus je 4096 Pixeln zum Training übergeben. Die oben beschriebene Trainingsmethode wurde dann zehn Mal (Epochen) für alle Ziffern durchlaufen und die entstandenen Gewichtsmatrizen dem Erkennungsprogramm für die Abfrage des Netzes zur Verfügung gestellt.

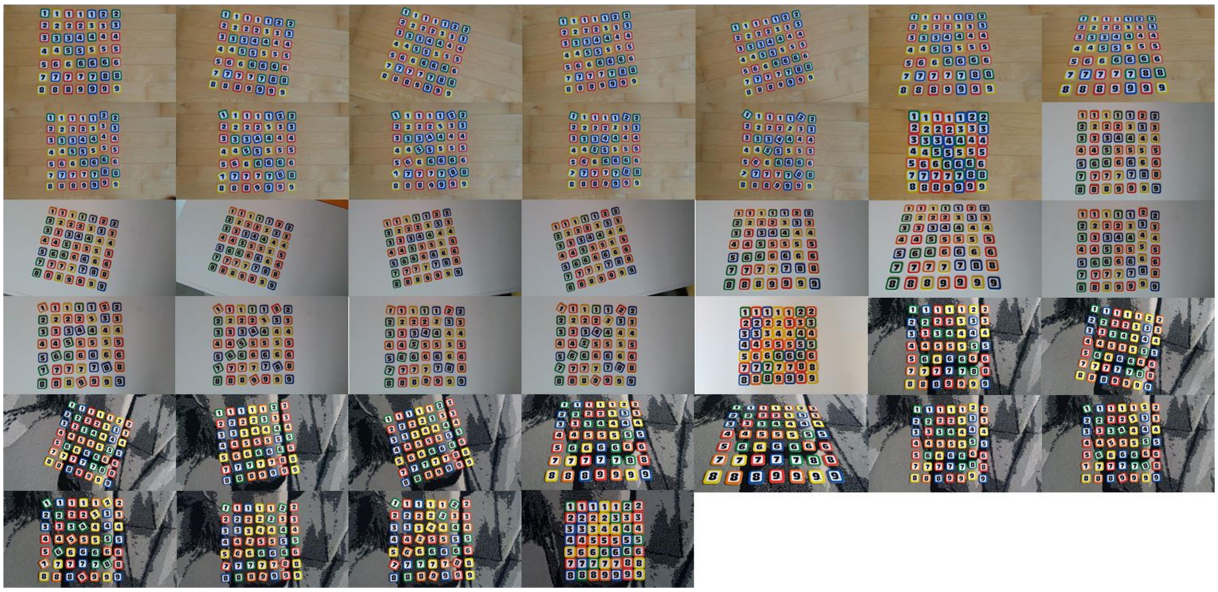

Abb. 22: Trainingsdaten für das neuronale Netz (Spielmotiv 1)

# 3 Bewertung der Erkennungsalgorithmen

In den Kapiteln 2.3. und 2.4 wurden zwei unterschiedliche Erkennungsstrategien entwickelt. Zum einen wurden geometrische Eigenschaften mit mathematischen Ansätzen für jede segmentierte Region bestimmt. Diese wurden mit Referenzvektoren verglichen und so die jeweilige Ziffer bestimmt (machine learning). Zum anderen wird die Merkmalsextraktion und Erkennung automatisiert in einem neuronalen Netz erledigt (deep learning).
Zur Bestimmung der Erkennungsraten der beiden Ansätze wurden für das erste Spielmotiv 47 und für das zweite Spielmotiv 58 neue Spielfelder mit verschiedenen Kameraeinstellungen aufgenommen. Somit standen $47 * 49 = 2303$ bzw. $58 * 49 = 2842$ verschiedene Ziffern zur Erkennung bereit. Die Fotos wurden wiederum über verschiedene Hintergründe verteilt und mit einer anderen Kamera aufgenommen, um Unterschiede zu den Trainingsbildern zu erhalten. Für den Ansatz über geometrische Merkmale ergeben sich für das erste Spielmotiv 15 Fehlerkennungen bzw. 43 für Spielmotiv 2. Dies entspricht einer Erkennungsrate von 99.34% bzw. 98.49%. Der Grund für Fehlerkennungen war in erster Linie eine ungünstige Wahl der Schwellenwerte für die Binärisierung des Fotos bei Aufnahmen mit schlechten Lichtverhältnissen, so dass als Folge Teile der Ziffern in den Hintergrund verschwanden.
Zur Bewertung des neuronalen Netzes wurden zunächst die optimalen Werte für die Anzahl der versteckten Knoten, die Größe der Lernrate und die Anzahl der Lernepochen bestimmt. Abb. 23 und 24 zeigen diese Abhängigkeiten.

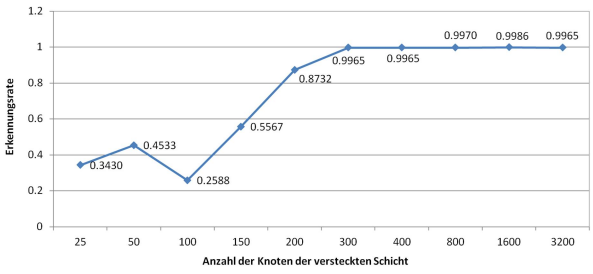


Abb. 23: Optimale Anzahl der versteckten Knoten (bei Lernrate 0.1 und bei einer Epochenzahl von 10);

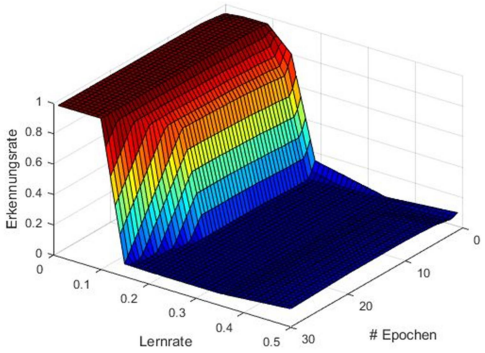


Abb. 24: Optimale Parameter für Lernrate und Anzahl der Epochen (bei Anzahl der versteckten Knoten = 400)

Mit 400 Knoten in der versteckten Schicht ergibt sich ein guter Kompromiss aus Erkennungsrate und Rechenzeit. Für die Lernrate wurden gemäß Abb. 24 ein Wert von 0.1 sowie die Anzahl der Epochen von 10 gewählt. Damit ergibt sich eine Erkennungsrate von 99.65% für Spiel 1 sowie von 99.90% für Spiel 2. Beide Ansätze weisen somit eine zufriedenstellende Erkennungsrate auf, zumal einige Testbilder mit absichtlicher Unschärfe und Verwackelungen sowie inhomogener Beleuchtung erzeugt wurden. Das neuronale Netz ist hierbei noch etwas besser. Der Grund kann u.a. mit der nebenstehend dargestellten „6“ erläutert werden. Aufgrund ungünstiger Lichtverhältnisse kann es vorkommen, dass eine 6 in Spiel 2 nicht komplett geschlossen ist, und somit gemäß Kapitel 2.3.5 keine Hintergrund-

einschlüsse aufweist. Zudem sinkt die Rundheit ab, da der Umfang der Ziffer durch den Defekt signifikant gewachsen ist. Beim Vergleich des Merkmalsvektors mit den Referenzvektoren kann dann die Wahl auf eine Ziffer ohne Hintergrundeinschluss und/oder größerer Rundheit fallen. Das neuronale Netz hingegen ist für derartige Details nicht anfällig und erkennt diese Ziffer trotzdem korrekt. Die folgende Tabelle fasst die Erkennungsraten der beiden Ansätze über geometrische Merkmale (CV) und neuronale Netze (KNN) nochmals zusammen:

| **Erkennungsrate** | CV | KNN |
|---|---|---|
| Spielmotiv 1 | 99.34% | 99.65% |
| Spielmotiv 2 | 98.49% | 99.90% |

# 4 Erstellung einer Handy-App

Um den Trio-Löser in einer realen Spielsituation praktisch einsetzen zu können, wurden die Algorithmen in einer Handy-App umgesetzt. Diese wurde in Xamarin.Forms programmiert. Diese Sprache ist ein Mix aus XAML und C#. Weiterhin bietet Xamarin.Forms die Möglichkeit, Apps plattformübergreifend zu erstellen (Android, iOS und Windows Phone)[1].

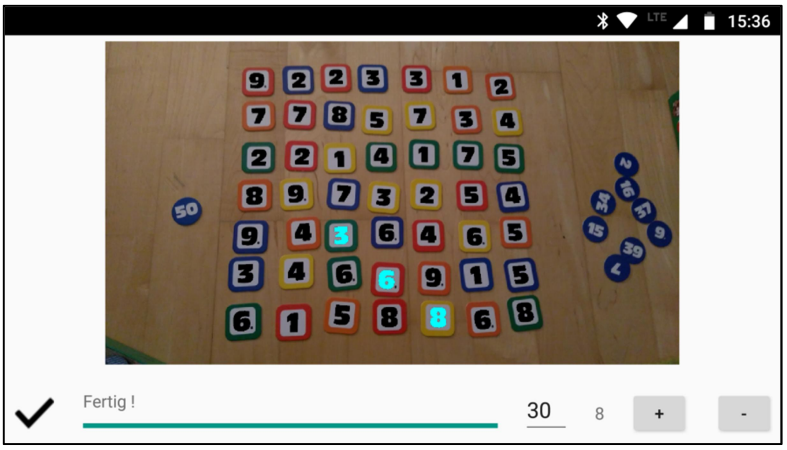


Abb. 25: Screenshot der Handy- App

Beim Starten der App wird der Benutzer gefragt, ob er jetzt ein Bild aufnehmen oder ein bereits existierendes aus der Galerie zur Analyse verwenden möchte. Weiterhin ist es dem Benutzter möglich, ein paar Einstellungen vorzunehmen, wie bspw. den Klassifizierungsansatz gemäß Kap 2.3 oder 2.4 zu wählen oder den Schwellenwert für die Schwarz-Weiß-Umwandlung manuell einzustellen. Sobald alle Einstellungen getätigt wurden, beginnen die Algorithmen, das Bild zu analysieren. Der Fortschritt wird in einem Info-Fenster veranschaulicht. Im Anschluss ist es dem Benutzer möglich, die gewünschte Suchzahl einzustellen (dort, wo in Abb. 25 die Zahl 30 steht). Das Durchschalten aller gefundenen Lösungen wird mit den zwei Knöpfen ganz rechts durchgeführt. Die fertige App „TrioSolver“ steht seit Januar 2019 im Google Playstore zur Installation bereit [16][App]. Unter den in der App integrierten Einstellungen (Options/Settings) lässt sich zur Demonstration der Funktionsweise ein Testbild laden.

# 5 Zusammenfassung und Ausblick

In der vorliegenden Projektarbeit wurde eine Lösungshilfe für das Mathematik-Logik Spiel Trio in Form einer Handy-App entwickelt. Das eigentliche Auffinden der Lösungskombination ist bei Kenntnis der Spielfeldanordnung recht einfach, da das Spielfeld lediglich systematisch zu durchsuchen ist. Vielmehr bestand die Schwierigkeit darin, die Spielfeldanordnung aus einem Handyfoto, welches nur aus einzelnen Bildpunkten besteht, sicher zu erkennen. Dazu wurde ein Bilderkennungsverfahren auf Basis künstlicher Intelligenz entwickelt und dabei Mechanismen eingebaut, die mögliche Verdrehungen des Feldes, perspektivische Verzerrungen, Störelemente und inhomogene Hintergründe behandeln. Mit einer Erkennungsrate von 99.65% zeigt das Verfahren gute Ergebnisse. Neben der Anwendung zur Erkennung eines Triospielfeldes könnte das Verfahren bspw. auch bei der Erkennung von Verkehrsschildern beim autonomen Fahren zum Einsatz kommen. Eine weitere Verbesserung des Verfahrens könnte dadurch erreicht werden, dass der Schwellenwert für die Umwandlung in ein Binärbild nicht mehr global, sondern für jeden Bildabschnitt lokal bestimmt wird. Durch ungünstige Beleuchtungsverhältnisse kann ein globaler Schwellenwert relevante Informationen in Teilen des Bildes entfernen. Durch Bestimmung eines lokalen Schwellenwertes würden die lokalen Gegebenheiten analysiert und die Qualität der segmentierten Regionen und damit auch die Erkennungsraten verbessert werden.

[1] Die TTT-App unterstützt momentan nur Android-basierte Betriebssysteme.

# 6 Literaturverzeichnis


[1] https://de.wikipedia.org/wiki/Grauwert (aufgerufen am 27.07.2018)

[2] https://de.wikipedia.org/wiki/Sobel-Operator (aufgerufen am 27.07.2018)

[3] https://de.wikipedia.org/wiki/Drehmatrix (aufgerufen am 27.07.2018)

[4] http://www.hinterseher.de/Diplomarbeit/Transformation.html (aufgerufen am 31.07.2018)

[5] Mark S. Nixon& Alberto S. Aguado: "Feature Extraction & Image Processing for Computer Vision", 3.Auflage, Academic Press, 2013

[6] https://en.wikipedia.org/wiki/Connected-component_labeling (aufgerufen am 13.08.2018)

[7] https://de.wikipedia.org/wiki/Bubblesort (aufgerufen am 17.08.2018)

[8] https://de.wikipedia.org/wiki/Moment_(Bildverarbeitung) (aufgerufen am 18.08.2018)

[9] https://en.wikipedia.org/wiki/Shape_factor_(image_analysis_and_microscopy) (19.08.2018)

[10] https://en.wikipedia.org/wiki/Moore_neighborhood (aufgerufen am 20.08.2018)

[11] https://de.wikipedia.org/wiki/Gift-Wrapping-Algorithmus (aufgerufen am 21.08.2018)

[12] https://www.geeksforgeeks.org/orientation-3-ordered-points (aufgerufen am 21.08.2018)

[13] https://www.mathopenref.com/coordpolygonarea2.html (aufgerufen am 21.08.2018)

[14] Tariq Rashid: „Neuronale Netze", 1. Auflage, O'Reilly Verlag, 2017

[15] https://www.wolframalpha.com/ (aufgerufen am 06.08.2018)

[16] https://play.google.com/store/apps/details?id=com.triosolver (aufgerufen am 06.01.2019)



# 7 Danksagung


Ich möchte mich hier bei meinem Vater bedanken, der mich immer unterstützt hat und mich jedes Mal motiviert hat, wenn ich ins Stocken geraten bin. Weiterhin möchte ich mich bei meinen beiden Betreuern, Herr Schönborn und Frau Lomonosova, bedanken, die mir immer mit Rat und Tat zur Seite standen und sehr viel von ihrer Freizeit für mein Projekt geopfert haben.


# 8 Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Ausarbeitung in eigenständiger Arbeit verfasst habe. Mehr als die angegebenen Quellen habe ich nicht verwendet.