Computer Graphics and Virtual Reality

Bildgestützter Assistent für Dominospiele

9. Februar 2019

Autoren E. D., M. K., Christian Zöller

Zeitraum 2018/2019

Studiengang MSc. Informatik, Komplexe Software Systeme

Inhaltsverzeichnis

2 Einführung

Im Rahmen des Moduls „Computational Geometry and Virtual Reality“ soll ein Projekt umgesetzt werden, in dem Bildverarbeitung oder Bilddarstellung realisiert wird. Gegenstand dieser Projektarbeit soll eine Art Assistent sein, der einen realen Spieler beim Spielen von Domino[1] unterstützt.

Der Assistent unterstützt den (einen, nicht alle) Spieler bei der Auswahl des nächsten zu legenden Steins, also *welcher* seiner Steine *wo* (an welchen Stein auf dem Spielfeld) anzulegen ist.

2.1 Allgemeine Gegebenheiten

Folgend sind die Gegebenheiten erläutert, die durch das Spielprinzip bedingt sind. Anschließend sind Annahmen und zusätzliche Bedingungen beschrieben, die als nötig erachtet werden, um das Projektziel dem Projektumfang (Zeit, Personen, erzielbares Wissen) entsprechend erreichbar zu gestalten.

2.1.1 Aufbau des Spiels

- Das Dominospiel umfasst 28 Dominosteine.
- Das *Spielfeld* besteht aus einer planen Fläche, auf der die Spieler Steine anlegen.
- Die Steine der Spieler (*Spielersteine*) befinden sich in einem separaten Bereich.
- Ein *Ziehstapel* enthält verdeckt Steine, die noch nicht gespielt werden.
- Die *Spielfläche* besteht aus dem Spielfeld, Flächen für die Spielersteine, und einer Fläche für den Ziehstapel, der in unserem System unberücksichtigt bleibt.

2.1.2 Allgemeiner Ablauf des Spiels

Zu Beginn des Spiels ist das Spielfeld leer. Die Spieler nehmen 5 Steine auf bzw. legen sie in ihrem separaten Bereich ab. Die Spieler legen nun abwechselnd passende Steine auf das Spielfeld. Dabei kann der Spieler bei einem Zug alle seiner passenden[2] Steine anlegen und ggf. nachziehen. Zwei Steine sind passend, wenn die zusammenzulegenden Steinhälften die

[1] https://de.wikipedia.org/wiki/Domino
[2] genaue Beschreibung ausgespart, s. Wikipedia Artikel

gleiche Augenzahl aufweisen. Beim Anlegen wird der Stein an die passende Kante gelegt, sodass dazwischen praktisch keine Lücke entsteht.

2.2 Zusätzliche Festlegungen

Folgende zusätzliche Annahmen/Vorbedingungen werden zur Begrenzung des Projektumfangs definiert:

2.2.1 Beleuchtungsbedingungen

- Die Spielfläche soll „gleichmäßig“ ausgeleuchtet sein, sodass der Schattenwurf minimiert wird.

2.2.2 Zusätzliche Anforderungen an den Aufbau

- Die Dominosteine sind schwarz mit weißen Punkten und sind 45x27x7mm groß.
- Die Punkte auf den Steinen haben einen Durchmesser von 4mm.
- Die Fläche des Spielfeldes auf dem Tisch beträgt ca. 50cm x 35cm.
- Die Fläche für die Spielersteine beträgt ca. 10cm x 35cm.
- Die Spielfläche befindet sich auf einem hellen Tisch. Als Referenz gelten die Tische der Veranstaltungsräume des ZIMT.
- Die Steine des Spielers, der den Assistenten benutzt, müssen offen (plan) auf dem Tisch abgelegt werden.
- die Spielersteine sind an definierten Ablagestellen mit deutlich erkennbaren Lücken angeordnet.

3 Zeitplanung

KW	Arbeitspaket	Aufgaben
44	Einarbeiten und Vorbereiten	Dokumentationsdokument anlegen
		Recherche + Einarbeitung in OpenCV
		Vortrag vorbereiten
45	Vortrag Rechercheergebnisse	Arbeitspläne erstellen
		Rechercheergebnisse zusammenfassen
		Vortrag vorbereiten
46	Präsentation	
47	Stativ Konstruieren + bestellen	
	Bildverarbeitung I	Spielstein erkennen
		Augen auf Spielstein zählen
	Bildverarbeitung II	Spielstein in zwei Hälften teilen
		Augen auf den Spielsteinhälften zählen
48	Stativ bauen	
	Bildaufnahmen Bildverarbeitung III	Fotos von Dominopartiezügen erstellen
		Bildverarbeitung mit Fotos durchführen
		Probleme erkennen und Lösungen finden
50	Präsentation Zwischenstand	Zwischenstand zusammenfassen und Präsentation vorbereiten
	Bildaufnahmen	Fotos von Dominopartiezügen erstellen
	Logik für Zugvorschlag	Entwickeln
		Implementieren
51	Bilderstellung	Zugvorschlag graphisch anzeigen
	Bildaufnahmen	Fotos von Dominopartiezügen erstellen
	Testen	
02	Restpunkte	Beseitigen
	Dokumentation	Überarbeiten und finalisieren
	Abschlusspräsentation	Vorbereiten

Tabelle 3.1: Arbeitspakete

4 Theoretische Grundlagen

In diesem Kapitel werden die verwendeten Algorithmen vorgestellt und erklärt.

4.1 Canny-Algorithmus

Der Canny Algorithmus (engl.: canny edge detector) ist ein robuster Algorithmus zur Kantendetektion. Als Ergebnis gibt der Canny Algoritmus ein Bild zurück, welches nur noch die Kanten des Ausgangsbildes enthält. Folgend werden die Schritte des Canny-Algorithmus' erläutert.

4.1.1 Einfache Kantendetektion mit Sobel Operator

Der Sobel-Operator berechnet die Ableitungen der Bildpunkt-Helligkeitswerte. Gleichzeitig wird orthogonal zur Ableitungsrichtung geglättet [vgl. Wik18c].

Betonung der Horizontalen und Vertikalen Kanten

Erstellen eines Gradientenbildes des Quellbildes um hohe Frequenzen als Grauwerte darzustellen. Hohe Frequenzen sind große Helligkeitsänderungen im Quellbildes. Diese Helligkeitsänderungen zeigen sind mögliche Kanten im Quellbild.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

[Ope18a] [Wik18a]

Richtung der Kanten

$$\theta = arctan(\frac{G_y}{G_x})$$

Die Winkel werden auf die vier Winkel 0°, 45°, 90° und 135° durch Runden reduziert, da ein Pixel nur acht Nachbarpixel hat, die symmetrisch angeordnet sind. [Ope18a] [Wik18a]

4.1.2 Absolute Kantenstärke

Die absolute Kantenstärke berechnet sich aus dem euklidischen Betrag der beiden partiellen Ableitungen.

$$G(x, y) = \sqrt{(G_x)^2 + (G_y)^2}$$

[Ope18a] [Wik18a]

4.1.3 Non-maximum surpression

Bei der NMS wird sichergestellt, dass eine Kante nicht mehr als ein Pixel breit ist. Dabei wird für jedes Pixel die Werte G(x,y) mit den Nachbarpixeln vergleichen. Ist der Wert eines Nachbars größer, so wird der Grauwert des Pixels auf 0 gesetzt. So wird die Dicke einer Linie auf 1 reduziert. [Ope18a] [Wik18a]

4.1.4 Hysterese

Bei der Hysterese wird schlussendlich gesteuert, bei welcher Kantenstärke eine Kante als Kante gezählt wird. Dafür werden zwei Schwellwerte angegeben.

- Oberer Schwellwert
- Unterer Schwellwert

Nur Gradientenwerte, die zwischen dem oberen und unteren Schwellwert liegen werden als Kante genommen. [Ope18a] [Wik18a]

4.1.5 Anwendung in OpenCV

C++:

```
void Canny(InputArray image, OutputArray edges, double threshold1, double
↪  threshold2, int apertureSize=3, bool L2gradient=false)
```

[Ope18a]

4.2 FindCountours (Konturen Finden)

Zur Konturenerkennung wird die Funktion `cv::findContours` mit den Parametern `mode=CV_RETR_EXTERNAL` und `method=CHAIN_APPROX_SIMPLE` verwendet. Somit werden zunächst nur die äußersten Konturen erkannt, eingeschlossene werden zunächst ignoriert.

Der Method Parameter bezieht sich auf die Speicherung der Punkte. Hiermit werden nicht alle Punkte einer gefundenen Kante gespeichert, sondern Linien auf ihre Endpunkte reduziert.

C++:

```
void cv::findContours( InputOutputArray image, OutputArrayOfArrays contours,
↪  OutputArray hierarchy, int mode, int method, Point offset = Point() )
```

[Ope18b]

4.3 Floodfill (Flutfüllung)

Floodfill ist ein einfacher Algorithmus um gleichfarbige Flächen in einem Bild zu erkennen und mit einer anderen Farbe zu füllen [vgl. Wik18b]. Von dem angegebenen Startpunkt werden alle benachbarten Pixel geprüft, ob diese den gleiche Farb- bzw. Helligkeitswert haben und dann sofort mit dem neuen Farbwert überschrieben. Eine maximale Abweichung nach unten und oben kann mit berücksichtigt werden. Der Algorithmus kann rekursiv und iterativ durchgeführt werden. Das rekursive Vorgehen hat den Nachteil, dass dieser schnell zum Stacküberlauf führen kann und der Stackingaufwand durch rekursion im Vergleich zum Operationsaufwand des Algorithmus viel Zeit in Anspruch nimmt. [Wik18b] Bei OpenCv wird das iterative Vorgehen genutzt. [Ope18c]

4.3.1 Anwendung in OpenCV

C++:

```
int floodFill(InputOutputArray image, Point seedPoint, Scalar newVal, Rect*
↪  rect=0, Scalar loDiff=Scalar(), Scalar upDiff=Scalar(), int flags=4 )
```

4.4 Blur (Weichzeichner)

Mittels eines Weichzeichners wird der Kontrast in einem Bild verringert. Herausstechende Pixel werden damit unterdrückt. OpenCV bietet zwei Hauptweichzeichner mit Blur and Gaussian Blur.

4.4.1 (Box) Blur (Weichzeichner)

Erstellt ein weichgezeichnetes Bild mittels normalisiertem Box Filter. Dabei werden alle Pixel im Kernel gleich gewichtet.

$$K = \frac{1}{ksize.width * ksize.hight} \begin{bmatrix} 1 & 1 & ... & 1 \\ 1 & 1 & ... & 1 \\ ... & ... & ... & ... \\ 1 & 1 & ... & 1 \end{bmatrix}$$

- K ist der Kernel, also der Filteroperator.
- ksize.width ist die Anzahl der Spalten des Kernels.
- ksize.hight ist die Anzahl der Reiehen des Kernels

4.4.2 Anwendung in OpenCV

C++:

```
void blur(InputArray src, OutputArray dst, Size ksize, Point
↪  anchor=Point(-1,-1), int borderType=BORDER_DEFAULT )
```

4.4.3 Gaussian Blur (Gaußscher Weichzeichner)

Beim gaußschen Weichzeichner wird die gaußsche Normalverteilung im Kernel als Gewichtung verwendet. Eindimensionale gaußsche Funktion:

$$G(x) = \frac{1}{\sqrt{2\pi\omega^2}} e^{-\frac{x^2}{2\omega^2}}$$

In zwei Dimensionen ist es das Produkt aus der gaußschen Funktion beider Dimensionen:

$$G(x,y) = \frac{1}{\sqrt{2\pi\omega^2}} e^{-\frac{x^2+y^2}{2\omega^2}}$$

4.4.4 Anwendung in OpenCV

C++:

```
void GaussianBlur(InputArray src, OutputArray dst, Size ksize, double sigmaX,
↪  double sigmaY=0, int borderType=BORDER_DEFAULT )
```

4.5 Blob Detection (Punkt Detektion)

Die Blob Detektion dient der Findung von gleichartigen Regionen im Bild. Der Algoritmus der Blob Detection geht in mehreren Schritten vor:

- Wandelt das Quellbild in ein Binärbild mittels der *Threshold-Funktion*
- Extrahiert alle verbundenen Komponenten des Binärbildes mittels *findContours* und berechnet deren Zentren
- Gruppiert die Zentren, die nicht weitere als der Mindestabstand zwischen Blobs auseinander liegen.
- Aus den Gruppen werden die letztendlichen Zentren und deren Radius berechnet und als Position und Größe zurückgegeben

5 Verwandte Arbeiten

Das Problem der Augenzahlerkennung auf den Dominosteinen ist prinzipiell gleich mit dem Erkennen der Augenzahlen auf Würfeln. Hierzu gibt es bereits einige Projekte, die sich mit der Erkennung von Augenzahlen befassen.

Als Ankerpunkt wurde bei diesem Projekt das Projekt „Recognizing dices“ von Glenn De Backer verwendet [Bac18]. Das Vorgehen von Glenn De Backer war für den Dominostein-Assistenten hinreichend als Einstieg, aber bei weitem nicht ausreichend um zuverlässige Ergebnisse zu erhalten.

Auch Gideon Vos hat in seinem Projekt „Dice Detection using OpenCV“ die Augenzahlen von farbigen Würfeln bearbeitet. Hierbei liegt der Fokus jedoch auf der Farbunterscheidung der Würfel, die für den Domino-Assistenten irrelevant sind [Vos18].

6 Implementierung

Dieses Kapitel bietet einen Überblick über die vorgenommenen Implementierungsschritte. Der Domino-Assistent lässt sich in unterschiedliche Kernkomponenten aufteilen. Teile der Software werden durch Datenflussdiagramme nach Gane und Sarson dargestellt. Die verwendeten Symbole haben folgende Bedeutung:

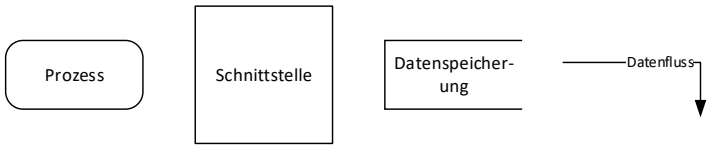


Abbildung 6.1: Legende für Datenflussdiagramme

6.1 Überblick

Hier sind die Kernkomponenten des Domino-Assistenten in ihrer Abfolge aufgeführt. Im ersten Schritt werden nacheinander die, von den Spielenden, gelegten Steine in der Bildaufnahme als Foto aufgenommen und als Bild gespeichert. In der Bildanalyse wird das Bild als Vorbereitung in verschiedene Kontextbereiche wie Spielfeld, Ziehfeld aufgeteilt. In den Kontextbereichen werden dann jeweils die Positionen der Dominosteinhälften und anschließend deren Augenzahl ermittelt. Diese Informationen werden zusammengeführt und an die Datenverarbeitung übergeben. Die Datenverarbeitung speichert die Informationen in geeigneter Weise, um darauf die Spiellogik anzuwenden. Nach dem die Datenverarbeitung den nächsten Zug berechnet hat, werden diese Zuginformationen von der Bildausgabe dem Spielenden angezeigt, sodass dieser seinen nächsten Zug durchführen kann.

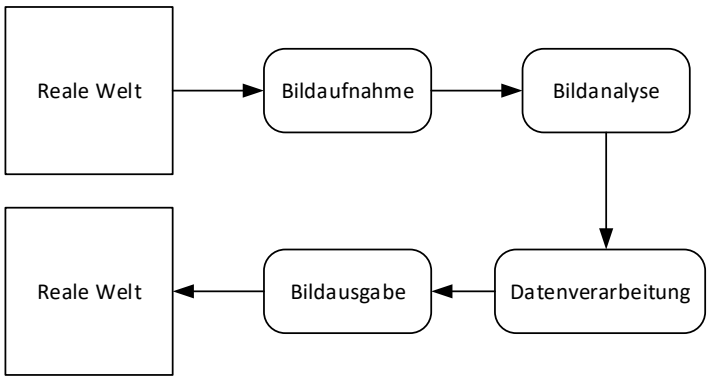


Abbildung 6.2: Überblick über die Kernkomponenten der Software

7 Bildaufnahme

Für den Dominoassistenten ist eine fehlerarme Aufnahme des Spielfeldes und der gelegten Dominosteinen für die Bildverarbeitung von elementarer Wichtigkeit. Der Aufbau soll gewährleisten, dass immer der gleiche Bildausschnitt von exakt der gleichen Position aufgenommen wird. Es ist für eine gleichmäßige Beleuchtung zu sorgen.

7.1 Aufbau/Setting

Für die Erfüllung der Voraussetzung einer immer gleichen Position mit gleichbleibendem Bildausschnitt eignet sich ein Stativ, das entsprechend positioniert wird. Als Kamera eignet sich unter anderem ein Smartphone. Der Zugriff erfolgt via Wireless Lan. Smartphones eignen sich besonders, da diese sehr verbreitet Verwendung finden, kabellos als IP-Cam betrieben werden können und für eine genügend große Zeit keine externe Stromversorgung benötigen.

7.2 Ablauf

Der Ablauf gliedert sich in zwei Phasen, die Initialisierungsphase und die Betriebsphase.

7.2.1 Initialisierungsphase

In der Initialisierungsphase wird einmalig beim Aufbau durchgeführt wobei das Smartphone und die Software konfiguriert wird. Mit dem Smartphone wird ein Hotspot eingerichtet, mit dem sich der PC mit der Domino-Assistent-Software verbinden kann. Anschließend wird die IP-Cam auf dem Smartphone gestartet und die IP-Adresse der IP-Cam notiert. Das Smartphone wird in der Halterung am Stativ befestigt und der Software wird die IP-Adresse übermittelt.

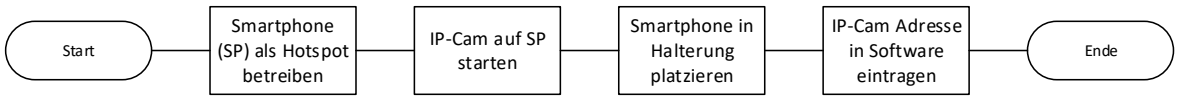


Abbildung 7.1: Ablauf der Initialisierungsphase

7.2.2 Betriebsphase

In der Betriebsphase fragt die Software die Bilder der IP-Cam zur Verarbeitung ab. Vor jedem Anlegen eines neuen Dominosteins wird ein Photo von dem Spielfeld aufgenommen. Zum Aufnehmen eines Bildes ruft die Software die Bildquelle der IP-Cam auf und speichert das Bild.

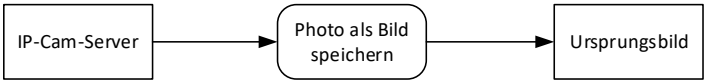


Abbildung 7.2: Ablauf eines Durchlaufs in der Betriebsphase

8 Bildanalyse

Bei der Bildanalyse wird sich sukzessiv dem Zielergebnis, in dem die Informationen über die Positionen und Augenzahlen der Hälften an die Datenverarbeitung weitergeben wird, angenähert. Dabei werden Teilbereiche mit einer Bildverarbeitung aufgearbeitet und anschließend analysiert. Die Zwischenanalyseergebnisse werden wieder zur weiteren Bildverarbeitung und Analyse herangezogen. Im ersten Schritt wird aus der Bildaufnahme das Ursprungsbild erstellt und mittels Bildvorverarbeitung auf ein Differenzbild reduziert. Das Differenzbild dient als Eingangsbild für die weitere Verarbeitung zur Positionsfindung eines Dominosteins, Erkennung der Dominohälften und abschließend das Herausfinden der Anzahl der Augenzahlen einer Hälfte.

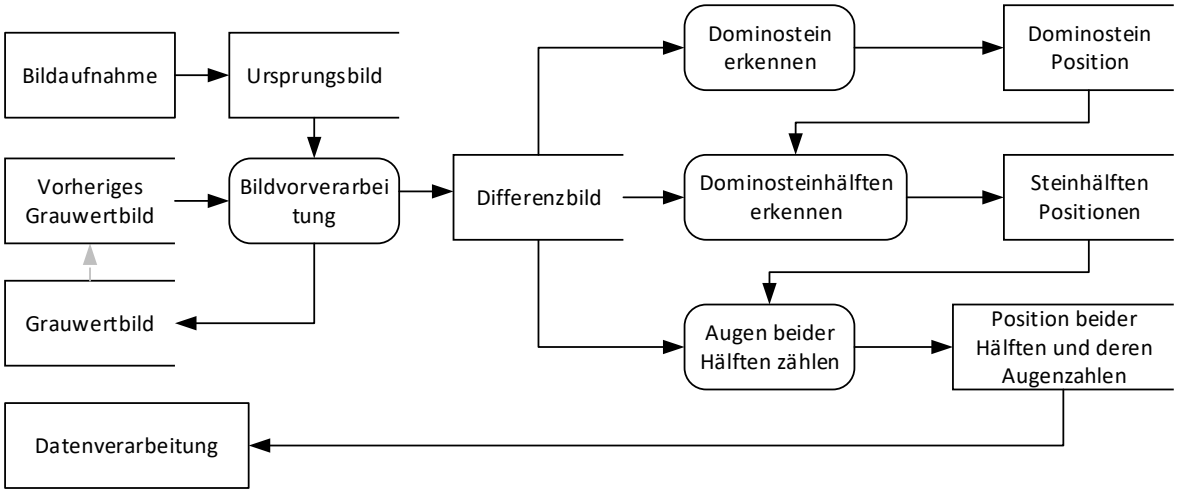


Abbildung 8.1: Ablauf der Bildanalyse

8.1 Bildvorverarbeitung

In der Bildvorverarbeitung wird das Ursprungsbild in die verschiedenen Kontextbereiche Spielfläche und Spielerfläche mit sieben einzelnen Flächen für die Steine des Spielers separiert. Aus diesen Bildern wird nach der Grauwertbildberechnung ein Differenzbild zur Weiterverarbeitung gebildet.

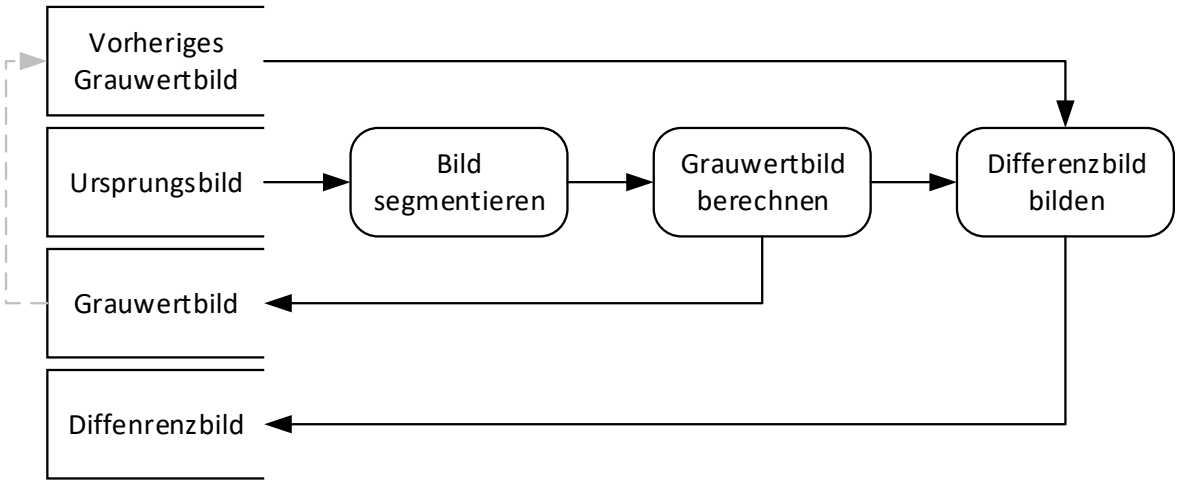


Abbildung 8.2: Bildvorverarbeitung

Ursprungsbild

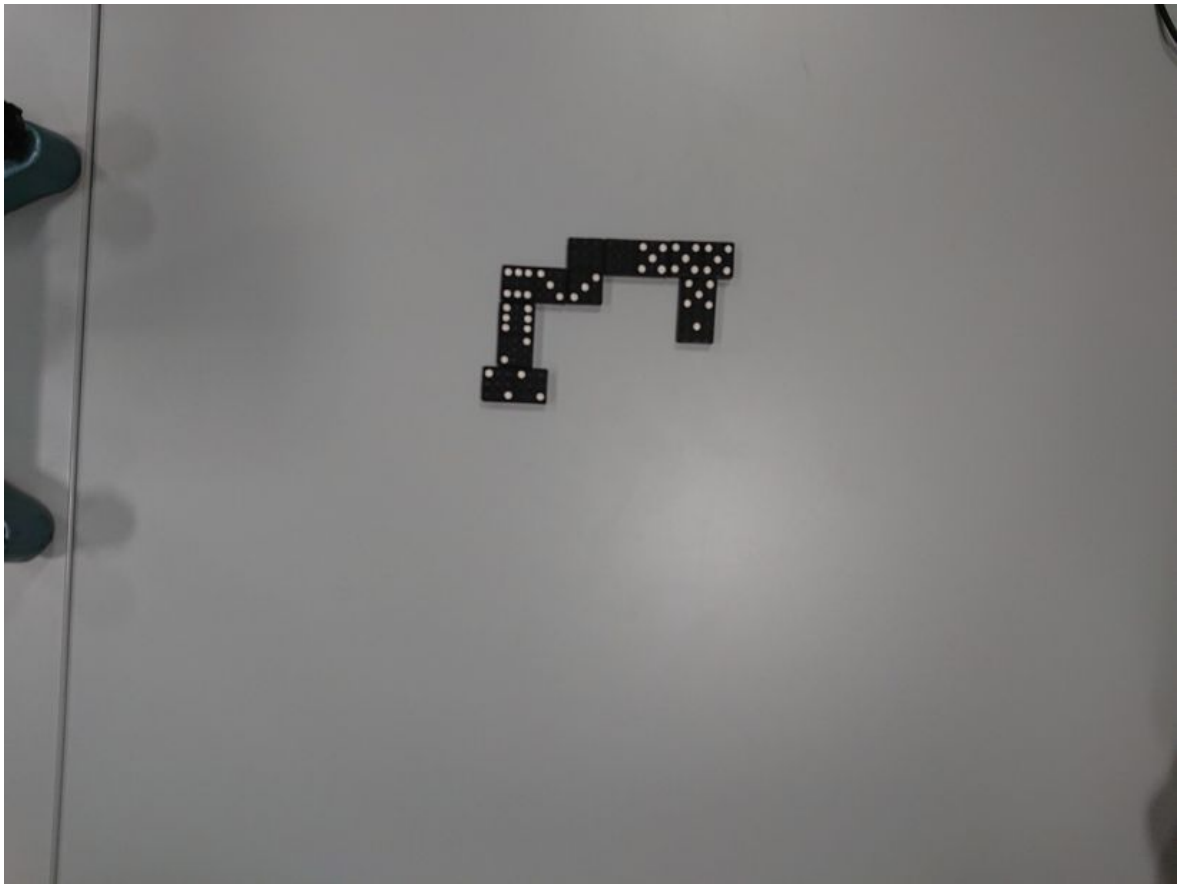


Abbildung 8.3: Ursprungsbild

Bildsegmentierung- Region of Interest

Das Dominospiel nimmt grundsätzlich zwei zu unterscheidende Bereiche auf. Zum einen das Spielfeld, auf dem die Steine beider Spieler erweiternd angelegt werden. Zum anderen die Spielersteinfläche, auf der die Dominosteine des Spielers liegen, der vom Domino-Assistenten unterstützt wird. Die Spielersteinfläche wird in sieben Bereiche unterteilt. In jedem dieser sieben Bereiche wird maximal ein Dominostein platziert. Alle Bereiche werden fortführend auf die gleiche Weise verarbeitet.

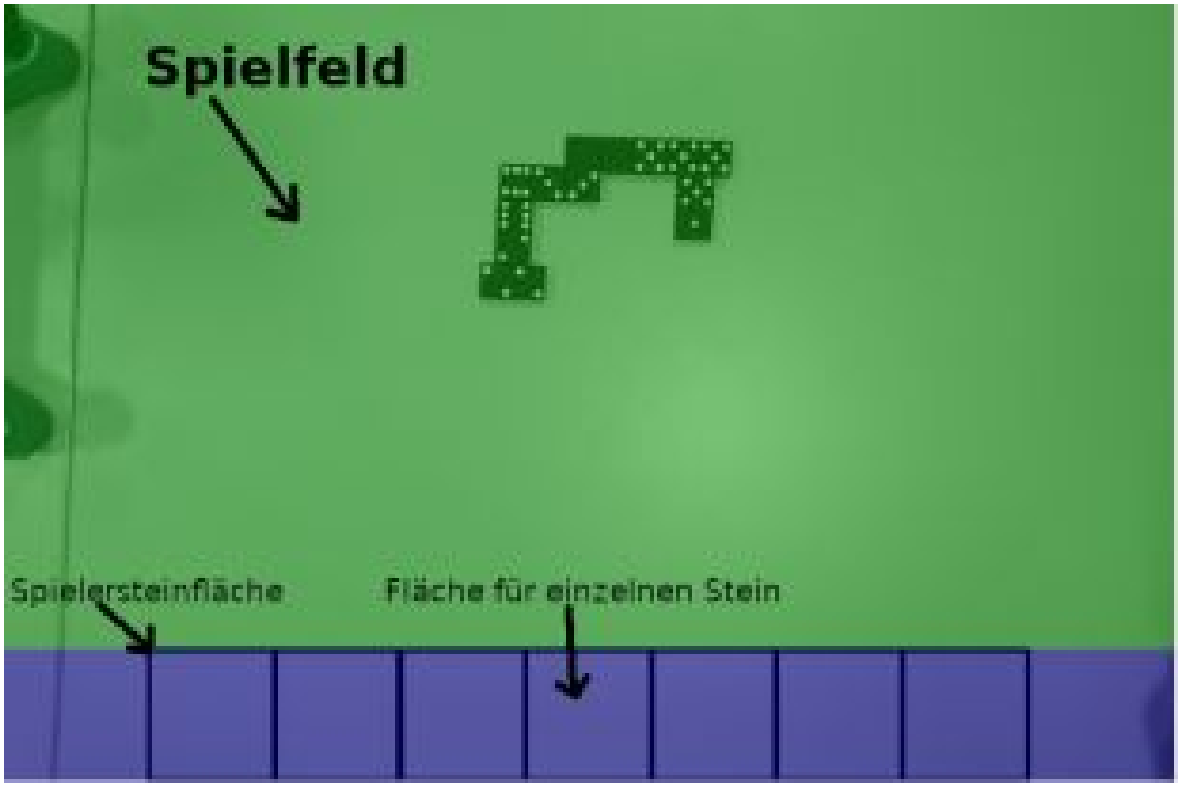


Abbildung 8.4: Bereichsauswahl

Grauwertbild

Für die Anwendung bringen die im aufgenommenen Bild enthaltenen Farben keinen Mehrwert an Informationen. Ein Grauwertbild enthält eine deutlich reduzierte Datenmenge und ist somit schneller zu verarbeiten.

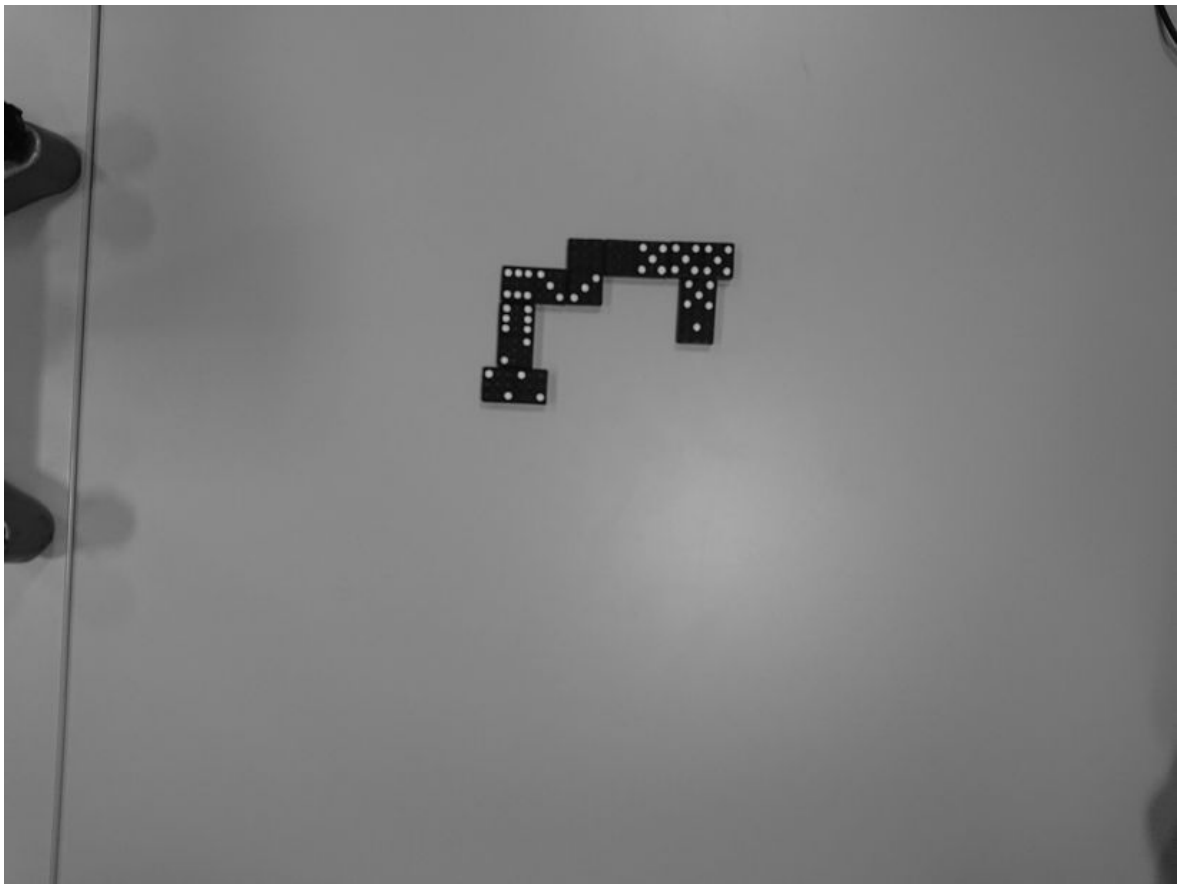
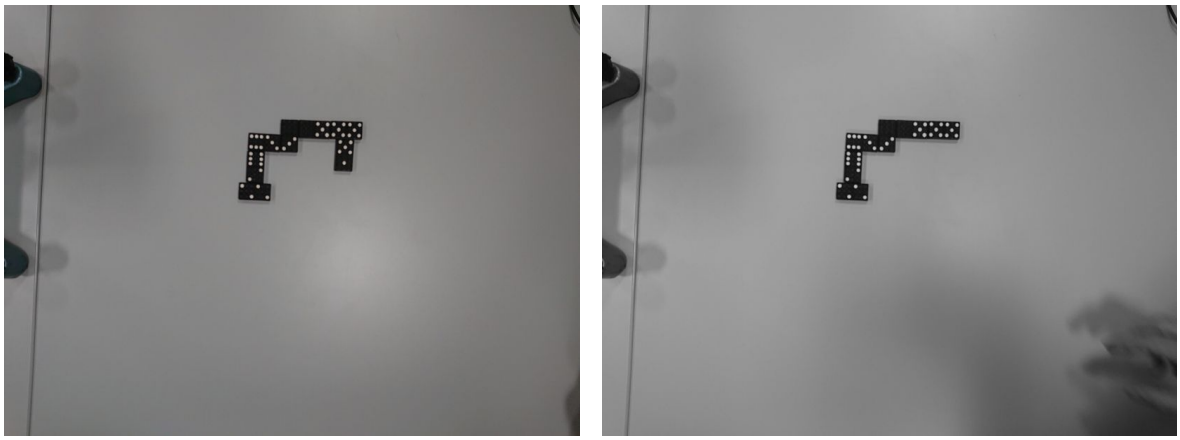


Abbildung 8.5: Grauwertbild

Differenzbild

Das Spielfeld wird immer von derselben Position aufgenommen. Um einen neu angelegten Dominostein zu separieren, wird das Differenzbild von dem vorherigen Bild (ohne den neuen Dominostein) und dem aktuellen Bild (mit dem neuen Dominostein) gebildet. Als Ergebnis bleibt der neu angelegte Dominostein im Differenzbild. Das Differenzbild wird aus den Grauwertbildern gebildet. Das Differenzbild enthält aufgrund veränderter Lichtverhältnisse von einem Bild zum nächsten neben dem gewünschten Dominostein und zudem häufig auch noch weitere minimale Unterschiede, die durch die Umwandlung in ein Binärbild weiter reduziert werden können.



(a) Grauwertbild (Quellbild) (b) vorheriges Grauwertbild

Abbildung 8.6: Grauwertbilder

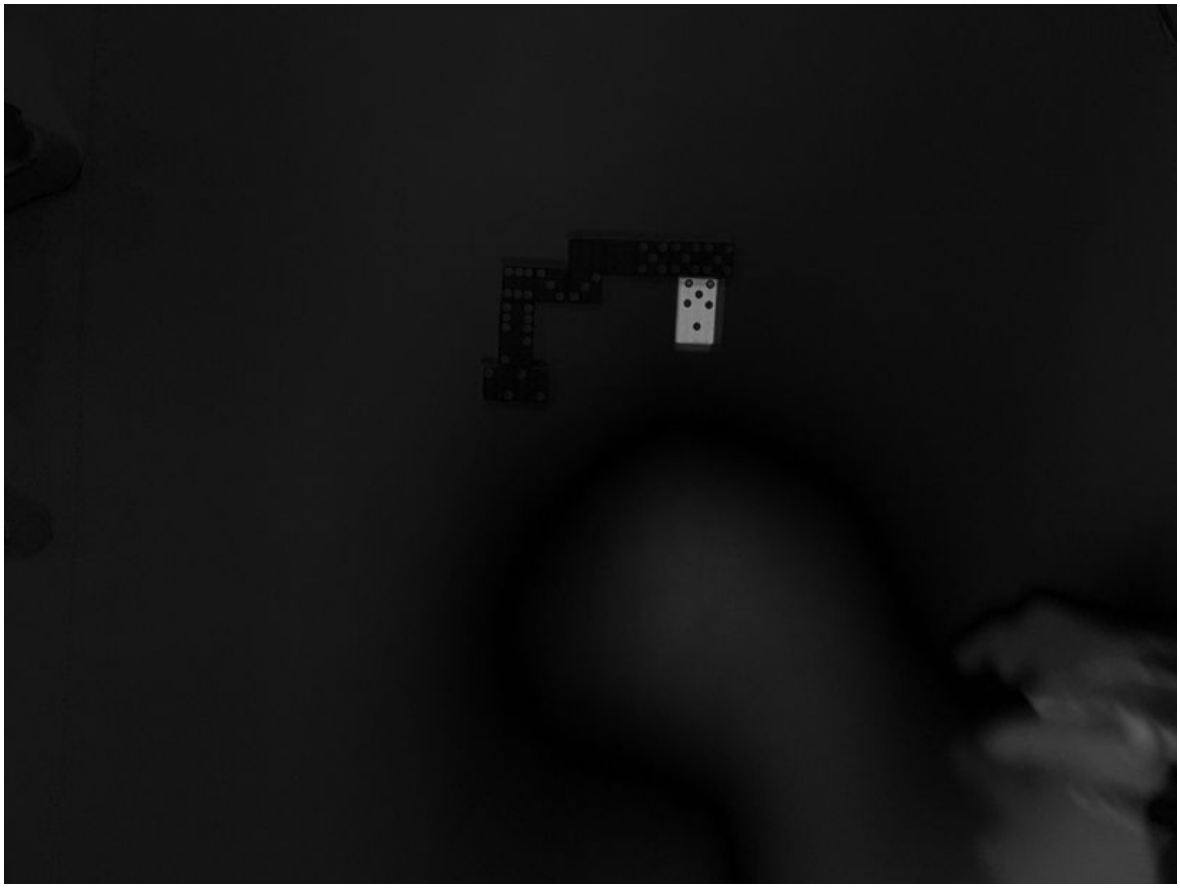


Abbildung 8.7: Differenzbild

8.2 Dominostein erkennen

In dem Differenzbild, in dem der zu erkennende Dominostein abgebildet ist, soll der Dominostein von der Software erkannt werden und zur Weiterverarbeitung aufbereitet werden. Zur Weiterverarbeitung ist das Bestimmen der Position des Dominosteins im Bild notwendig. Es wird eine Bildverarbeitung mit anschließender Bildanalyse durchgeführt.

8.2.1 Bildverarbeitung

In dem Differenzbild werden die Konturen gefunden und somit auch der gesuchte Dominostein. Zum Finden der Konturen ist es notwendig, ein Binärbild zu erstellen, um die Kanten besser erkennen zu können.

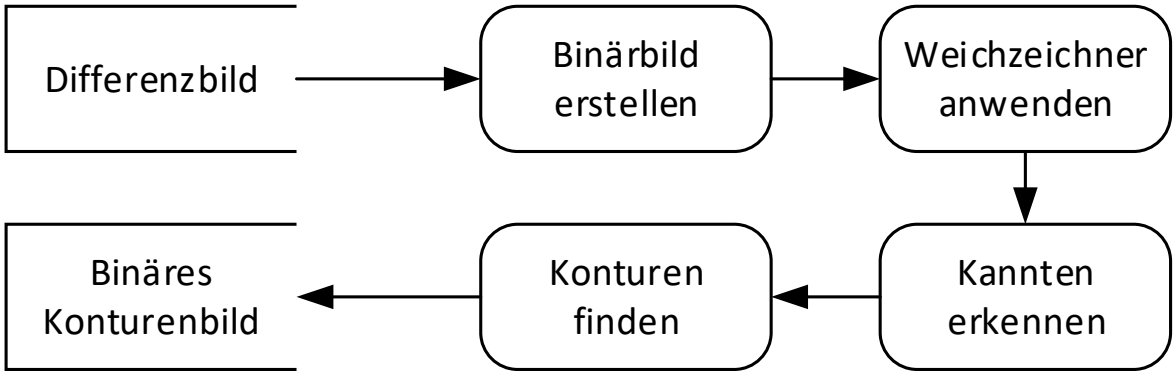


Abbildung 8.8: Bildverarbeitung - Ablauf der Dominosteinerkennung

Binärbild

Aus dem Differenzbild wird das Binärbild berechnet. Als Schwellwert zur Bestimmung der Binärwerte ist 42 gesetzt. In verschiedenen Testläufen hat sich ein Schwellwert zwischen 30 und 50 als sehr geeignet ergeben und mit dem Wert 42 konnten in dem meisten Fällen das gewünschte Ergebnis erzielt werden, im Gegensatz zu Ansätzen mit dynamischer Schwellwertberechnung.

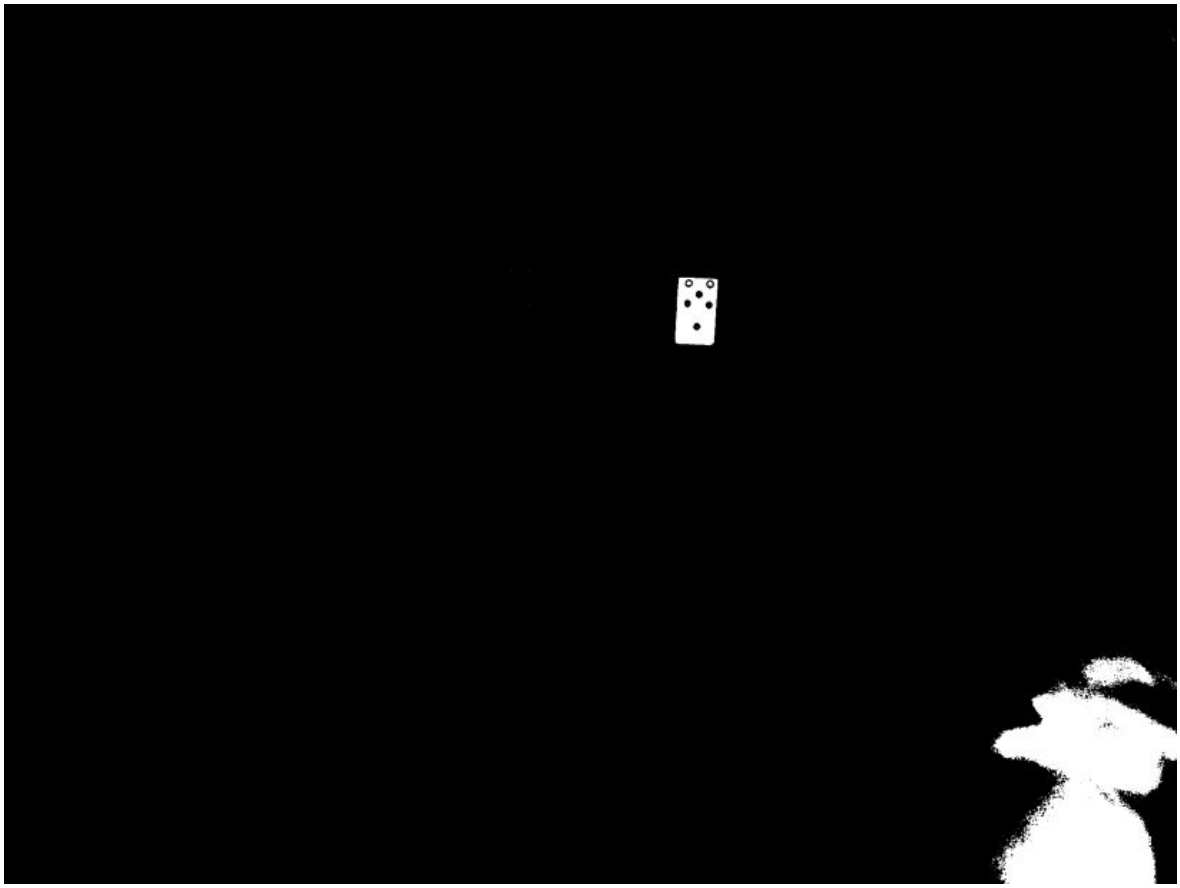


Abbildung 8.9: Binärbild

Weichzeichnen

Das Weichzeichnen reduziert das Rauschen im Binärbild und sorgt für durchgängige Kanten. Es wird ein Weichzeichner verwendet, der alle Pixel im Kernel gleich gewichtet berücksichtigt. Die Kernelgröße beträgt 3*3.

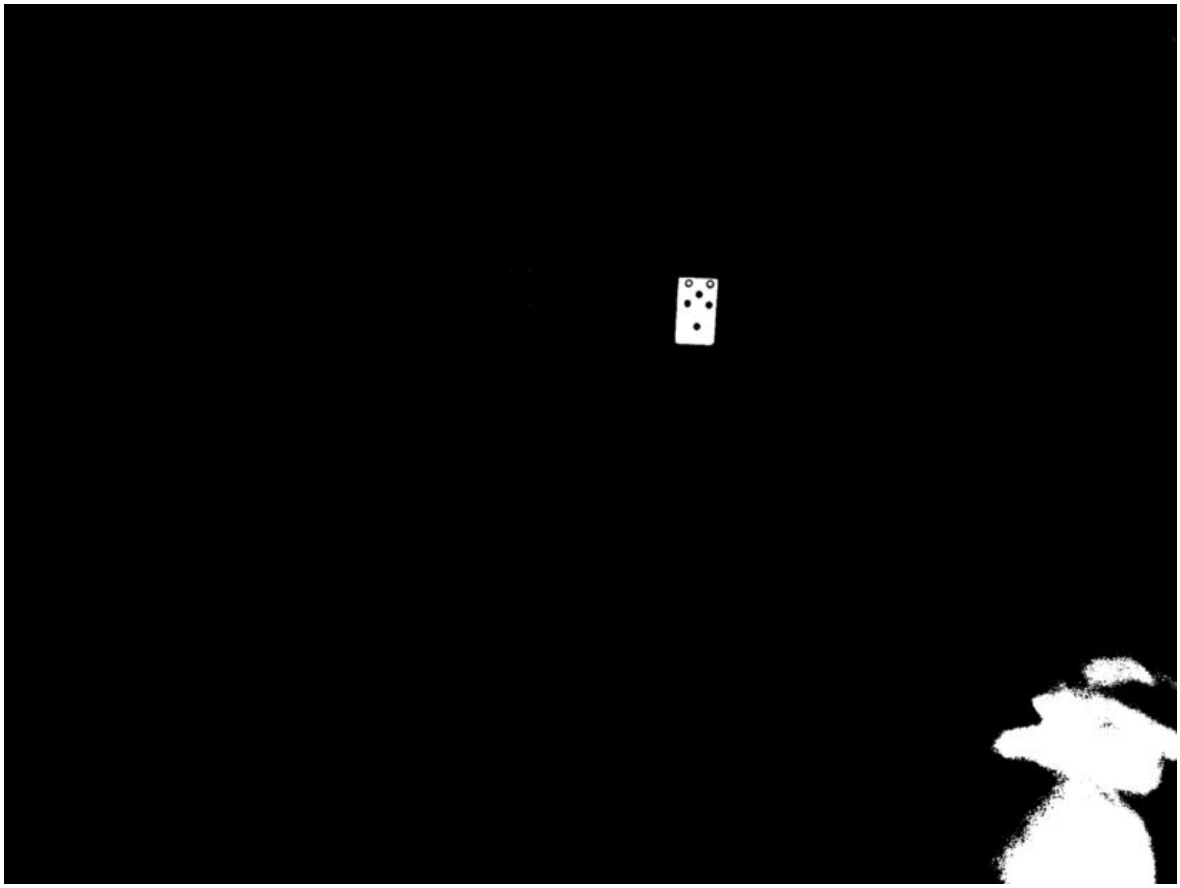


Abbildung 8.10: Weichzeichnen

Kanten erkennen

Die Kanten werden im Binärbild mittels Canny Edge Detection bestimmt. Da es sich um ein Binärbild handelt, werden die Schwellwerte auf 0 und 1 gesetzt.

Abbildung 8.11: Kanten

Konturen finden

Um ein zusammenhängendes Objekt im Bild zu identifizieren, eignen sich Konturen. Aus diesem Grund wird eine Kontursuche auf das Binärbild anschließend an die Kantenerkennung ausgeführt. Die Konturen werden an den gefunden Kanten entlang gefunden.

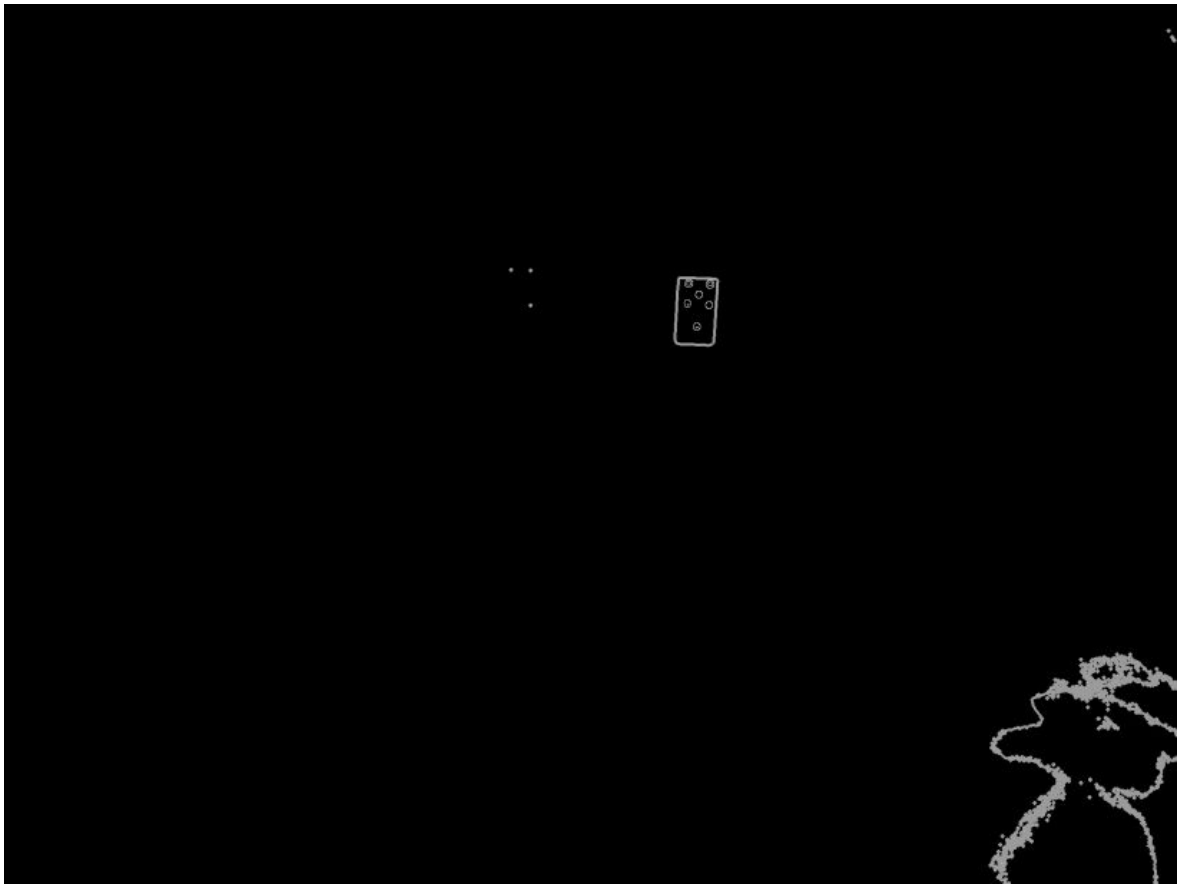


Abbildung 8.12: Binäres Konturenbild

8.3 Bildanalyse

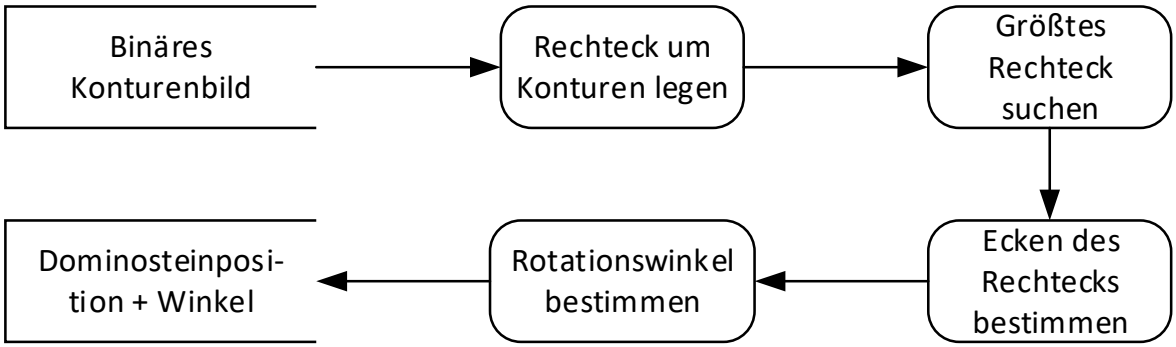


Abbildung 8.13: Domionstein Analysevorgehen

8.4 Dominosteinhälften erkennen

Für die Erkennung der Domniohälften wird der vollständige Dominostein ausgeschnitten, die Rotation des Dominosteins im Bild korrigiert, zugeschnitten und anschließend halbiert, wodurch beide Hälften separiert werden. Da die Bearbeitungsvorschrift für beide Hälften gleich ist, werden die Berechnungen für beide Hälften parallel in zwei Threads ausgeführt.

8.4.1 Bildverarbeitung

Die Bildverarbeitung für die Hälftenerkennung besteht nur aus der Separierung des Dominosteins im Differenzbild in ein neues Bild mit dem gesamten Dominostein.

Dominosteinseparierung

Um die berechnete Position des Dominosteins wird ein umgrenzendes Rechteck (Bounding Box) gelegt. Die Bounding Box wird als ROI auf das Differenzbild angewendet und somit ausgeschnitten. Das Ergebnisbild enthält den separierten Dominostein.

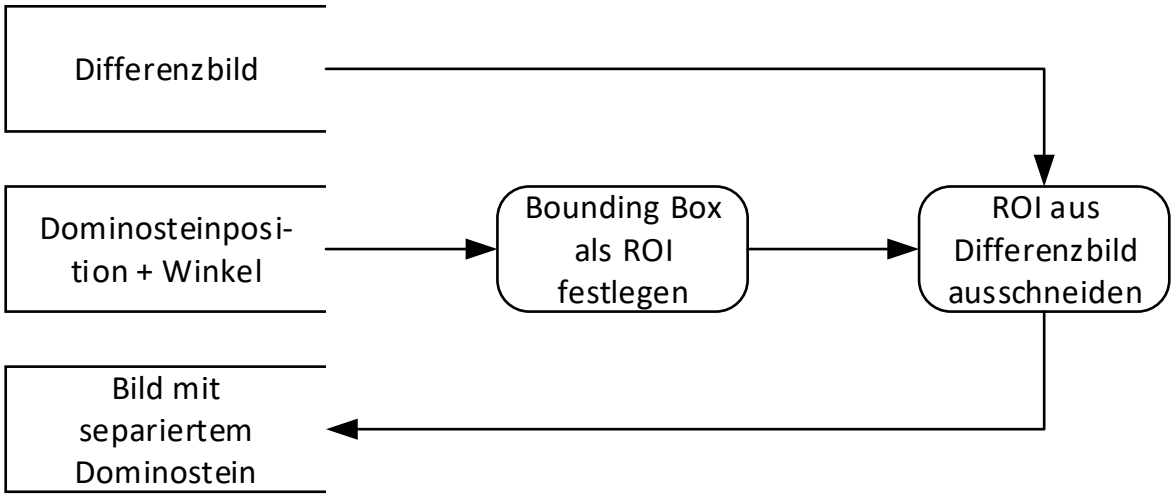


Abbildung 8.14: Bildverarbeitung Dominosteinhälftenerkennung

8.4.2 Bildanalyse

Um beide Hälften zu erhalten, muss der Dominostein halbiert werden.

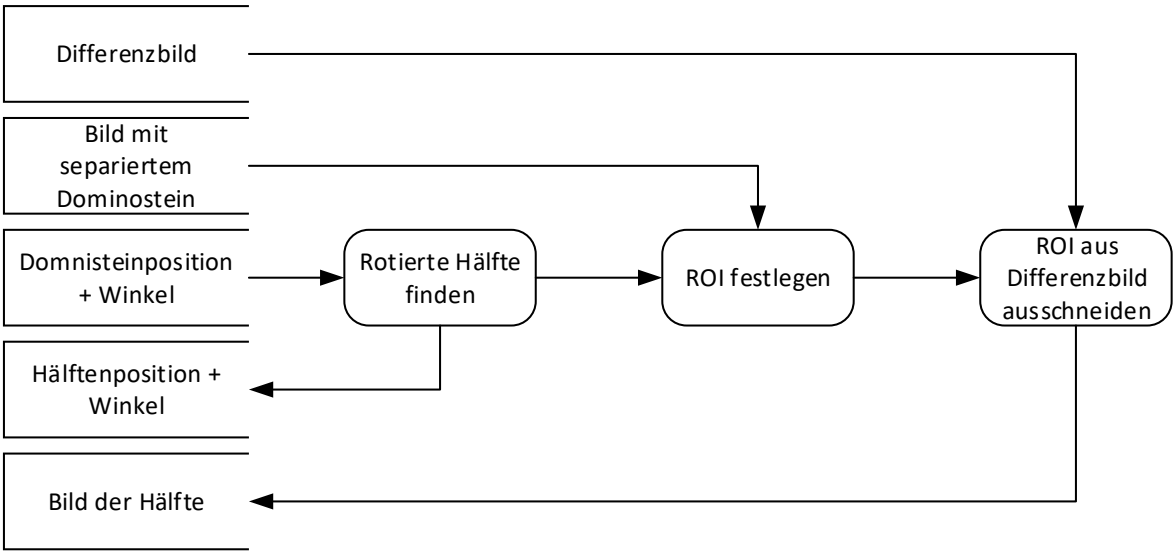


Abbildung 8.15: Bildanalyse Dominosteinhälftenerkennung

Rotierte Hälften finden

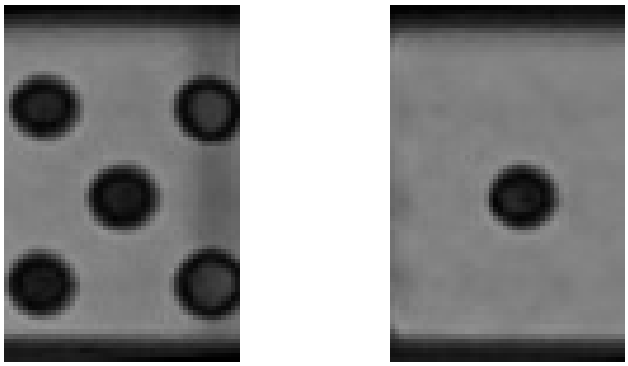
Für die Berechnung der Eckpunkte einer Hälfte werden ausschließlich die Eckpunkte des Dominosteins benötigt. Ein zufälliger Eckpunkt des Dominosteins wird zum Starteckpunkt einer Hälfte definiert. Für die andere Hälfe wird der diagonal gegenüberliegende Eckpunkt als Starteckpunkt definiert. Der am nächsten liegende Eckpunkt zum Starteckpunkt gehört ebenfalls zum jeweiligen Stein. Auf der Hälfte der Strecke zwischen dem zum Starteckpunkt nächstliegenden Punkt und dem Starteckpunkt der anderen Hälfe liegt der dritte Punkt einer jeden Hälfte. Mit drei Eckpunkten lässt sich eindeutig das Rechteck bilden. Die OpenCV Datenstruktur für rotierte Rechtecke berechnet den Winkel des Rechtecks aus den gegeben Punkten im Bild.

Region Of Interest festlegen

Für die Region Of Interest wird wieder um das rotierte Rechteck eine Bounding Box gelegt.

ROI aus Differenzbild ausschschneiden

Die ROI enthält bei rotierten Dominosteinen noch Teile von der anderen Hälfte. Die gesuchte Domniohälfte liegt exakt in der Mitte der ROI. Durch Transformation und Translation der rotierten Hälfte wird ein Rechteck passend über die gewünschte Dominosteinhälfte gelegt und nicht zur Hälfte gehörende Elemente weggeschnitten. Der Inhalt des Rechtecks wird als Bild der Hälfte gespeichert.



(a) Bildhälfte A (b) Bildhälfte B

Abbildung 8.16: Bild der Hälften

8.5 Augenzahl erkennen

Um die Augenzahlen auf den Dominosteinhälften zu erkennen ist eine weitere Bildverarbeitung nötig, dessen Verlauf in Abbildung 8.17 dargestellt ist. Die Prozesse zur Bildverarbeitung werden erklärt und beispielhaft für einen Spielstein visualisiert. Zur Darstellung in diesem Dokument wird ein Rahmen um die Bilder gesetzt, damit alle Verarbeitungsschritte deutlich werden können.

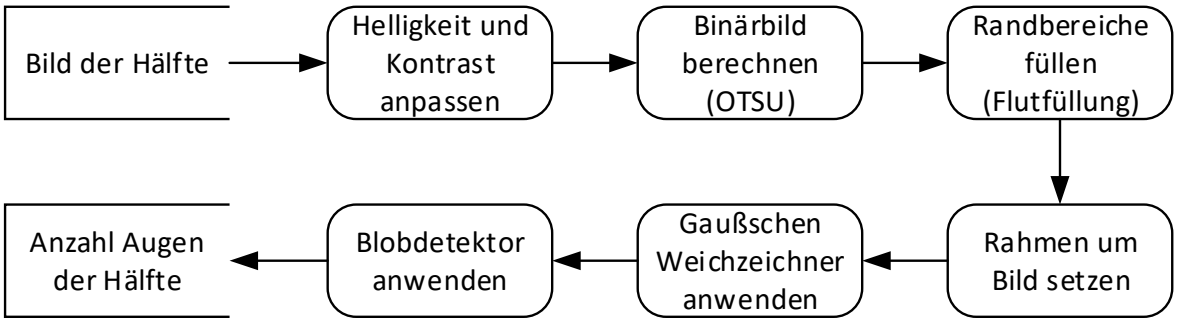
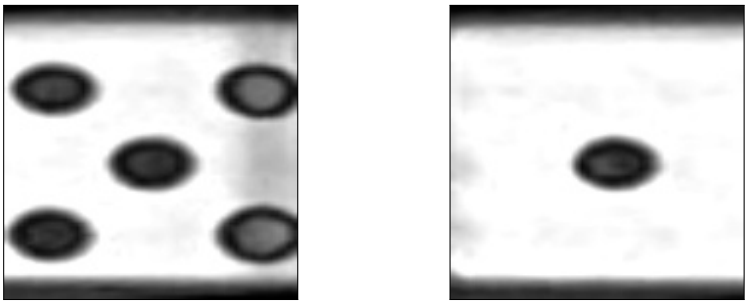


Abbildung 8.17: Bildverarbeitung Augen einer Hälfte zählen

8.5.1 Bildverarbeitung

Helligkeits- und Kontrastanpassung

Das Bild der Hälfte ist ein Grauwertbild. Die Grauwertbildqualität ist abhängig von der Beleuchtung bei der Aufnahme des Quellbildes. Da die Beleuchtung nicht immer gleich und nicht optimal ist, ist eine Helligkeits- und Kontrastanpassung notwendig. Auf diese Weise verschwimmen leichte Schatten und Störungen mit dem Hintergrund, während das Nutzsignal, die Augen des Steins, hervortreten.

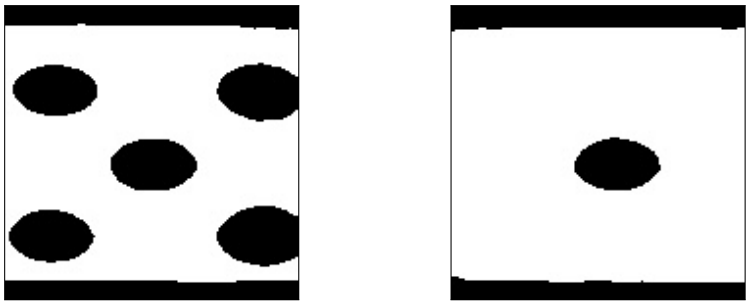


(a) Bildhälfte A (b) Bildhälfte B

Abbildung 8.18: Optimiert auf Helligkeit und Kontrast

Binärbildberechnung mit OTSU

An dieser Stelle wird das Binärbild mit dem Algorithmus von OTSU berechnet. OTSU hat einen dynamischen Schwellwert, der die Streuung der Intensitäten zur Schwellwertberechnung nutzt.

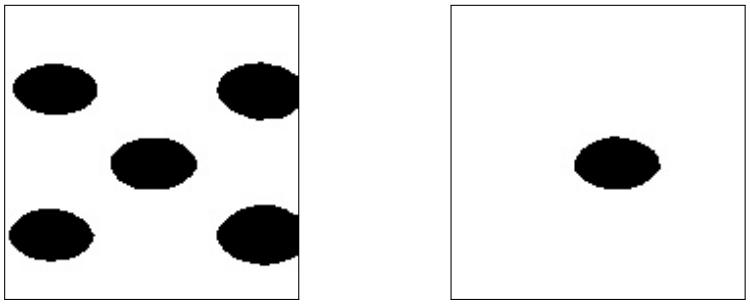


(a) Bildhälfte A (b) Bildhälfte B

Abbildung 8.19: Binärbild

Randbereichfüllung

Am Rand und den Ecken des Bilds der Dominohälfte sind in der Regel unerwünschte schwarze Bereiche vorhanden. Diese Bereiche werden mit Flutfüllung von den Ecken aus mit weiß aufgefüllt.

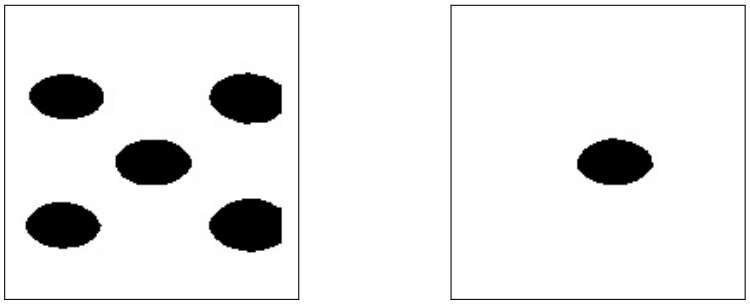


(a) Bildhälfte A (b) Bildhälfte B

Abbildung 8.20: Randbereichfüllung

Rahmensetzung

Ein weißer Rahmen um das Bild ist dann notwendig, wenn ein Auge des Steins an den Rand reicht. Berührt das Auge den Rand des Bildes, wird dieses nicht vom später eingesetzten Blobdetector erkannt. Der weiße Rahmen sorgt dafür, dass kein Auge an den Rand reicht.

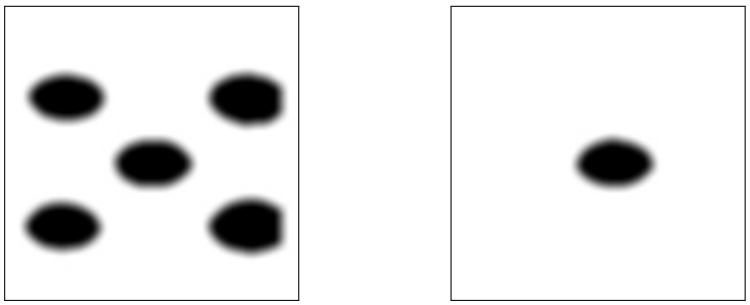


(a) Bildhälfte A (b) Bildhälfte B

Abbildung 8.21: Rahmensetzung

Gaußscher Weichzeichner

Der Gaußsche Weichzeichner sorgt für Rauschminderung und für gleichmäßigere Augen.



(a) Bildhälfte A (b) Bildhälfte B

Abbildung 8.22: Weichzeichner

8.5.2 Bildanalyse

Blobdetector

Der Blobdetector erkennt schwarze Punkte im Bild und zählt deren Anzahl. Die *Anzahl Augen der Hälfte* ist hier der Rückgabewert.

8.5.3 Ergebnis der Bildverarbeitung

Am Ende der Bildverarbeitung sind die Hälftenpositionen und dessen Winkel, sowie Augenzahl bekannt. Diese Werte werden genutzt und im folgenden Bild veranschaulicht. Das Bild enthält eine farbige Markierung der Spielsteinhälften, dessen Augenzahl im Zentrum und dessen Eckpunkte.

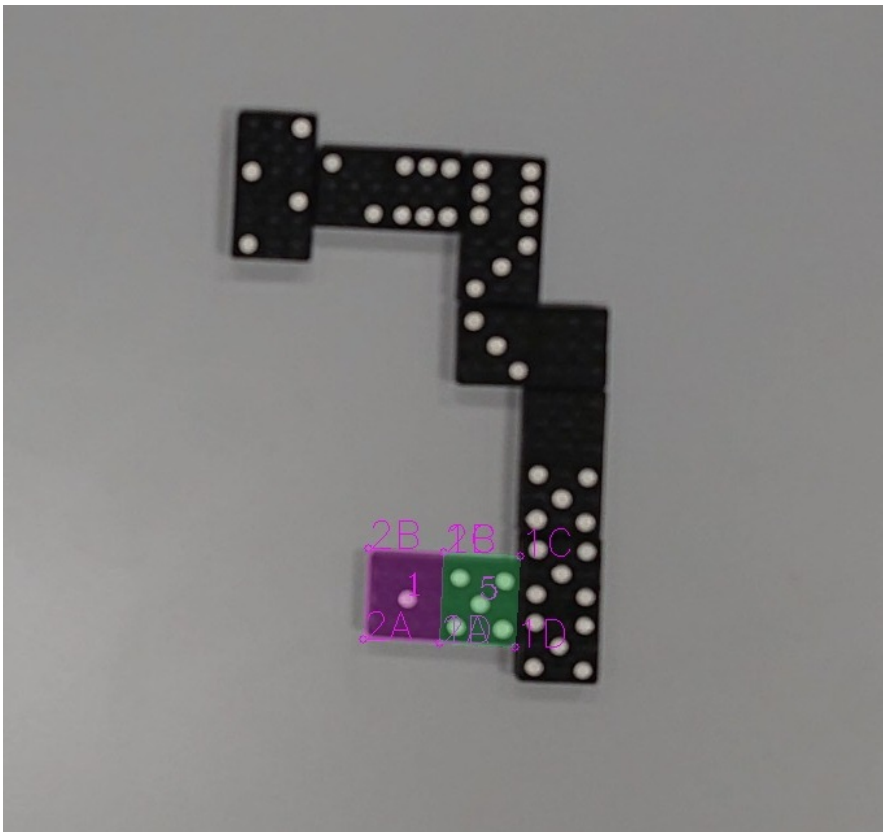


Abbildung 8.23: Ergebnis der Bildverarbeitung

9 Datenverarbeitung

Da die Dominosteine schrittweise erkannt werden, müssen die Zwischenergebnisse gespeichert werden, um letztlich dem Spieler den nächsten Zug vorzuschlagen. Die erkannten Dominosteine werden durch das Programm des Domino-Assistenten temporär im Arbeitsspeicher gehalten. Die Datenverarbeitung kann also in zwei Bereiche aufgeteilt werden:

- Speicherung der Dominosteine in einer Datenstruktur
- Vorschlagen des nächsten Zuges

9.1 Datenstruktur

Die erkannten Dominohälften liegen für die Datenverarbeitung als `cv::RotatedRect` und die erkannten dazugehörigen Ziffern als `unsigned int` vor. Diese Informationen werden mehrfach gekapselt, um schrittweise Informationen und Funktionalität hinzuzufügen.

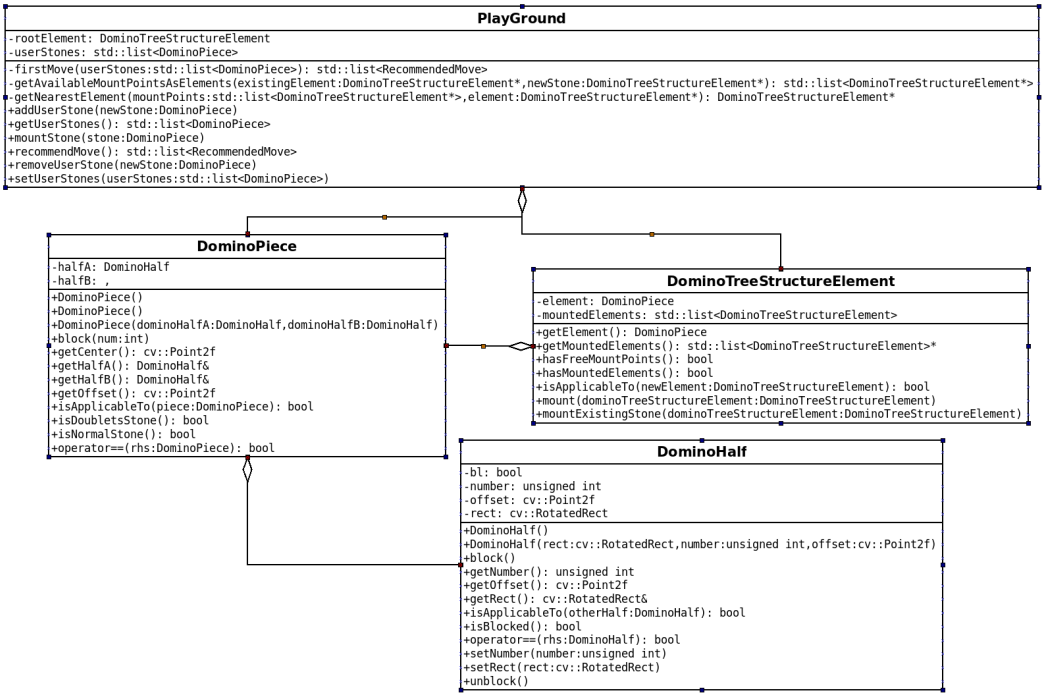


Abbildung 9.1: Klassendiagramm der Datenstruktur

Das `cv::RotatedRect` und die dazugehörige Ziffern werden in ein `DominoHalf` Objekt gekapselt. Das RotatedRect Objekt wird beibehalten, da hier auch die Informationen über die

genaue Lage gespeichert sind, inklusive des sich daraus ergebenen Zentrums der Dominohälfte, die für das Anlegen eines Steins benötigt wird. Es stellt eine Funktion zur Prüfung bereit, ob eine andere Dominosteinhälfte hier anlegbar ist. Eine solche Funktion wird auf jeder Stufe der Kapselung bereitgestellt. Die Funktionen unterscheiden sich durch zusätzliche Prüfungsbedingungen.

Ein `DominoPiece` enthält zwei `DominoHalf` Objekte sowie die sich daraus ergebende Information, ob es sich um einen Pasch-Stein oder einen normalen Stein hat. Dies wird anhand der Augenzahlen der beiden Dominohälften festgestellt. Daneben stellt es die Funktion zur Prüfung bereit, ob ein DominoPiece an dieses anlegbar ist.

Die Klasse `DominoTreeStructureElement` kapselt das DominoPiece und fügt Funktionalität zur Verkettung von DominoPieces in einer Baumstruktur hinzu. Realisiert wird dies durch eine Liste von Zeigern auf die angelegten `DominoTreeStructureElement` Objekte. Zudem stellt diese Struktur die Funktion zum Anlegen eines anderen DominoTreeStructureElements bereit, sowie intern dafür benötigte Hilfsfunktionen.

Die Klasse `PlayGround` enthält das erste auf das Spielfeld abgelegte Element, an das dann alle weiteren Elemente angelegt werden, sowie eine Liste der Steine des Spielers, der das System benutzt. Die Steine des Gegenspielers bleiben unbetrachtet. Diese Klasse stellt dann hauptsächlich die Funktion zum Anlegen eines Dominosteins und zum Zugvorschlag bereit.

9.2 Anlegen eines Steins

Die grundlegende Idee zur Speicherung der aneinandergelegten Steine ist die Nachbildung der Anlegestellen der Steine bzw. Steinhälften. Grundsätzlich können an einen Dominostein zwei weitere Steine angelegt werden, an jede Hälfte einer. Bei Pasch-Steinen (Steinen mit zwei gleichen Ziffern) sind es vier, bzw. zwei pro Hälfte.

Daher verfügt die `DominoHalf` Klasse über eine `block` Funktion, die zum Anlegen benötigt wird. Das Blocken wird in der Klasse `DominoPiece` durchgeführt, da hier die dazu benötigte Information über die Anzahl der Anlegestellen gespeichert ist. Bei Anlegen eines anderen Steins wird die betreffende Ziffer geblockt, sobald die maximale Anzahl von an diese Hälfte anlegbarer Steine erreicht ist. Im `DominoTreeStructureElement` wird beim Anlegen anhand der Position der Steine entschieden, an welchen Stein er wohl angelegt wurde. Hierzu werden die Zentren der Dominosteinhälften herangezogen. So wird zum einen der Dominostein ermittelt, an den der aktuelle Dominostein angelegt wurde, zum anderen auch, welche Hälfte des Steins jeweils betroffen ist und die `block` Funktion entsprechend aufgerufen.

9.3 Zugvorschlag

Da der Schwerpunkt dieses Projekts auf dem Bildverarbeitungsteil liegt, wurde hier auf einen komplexeren Mechanismus zur Zugvorhersage verzichtet und stattdessen eine einfache Basismethode implementiert.

In einem nächsten Zug soll der Nutzer einen seiner Steine an die auf dem Spielfeld liegenden anlegen können. D.h. in Frage kommen jene Kombinationen aus Spielerstein und liegenden Steine, wo eine Dominosteinhälfte der Spielersteine mit einer der freien Dominosteinhälften zusammenpasst. Die Funktion `recommendMove()` liefert also eine Liste dieser Kombinationen zurück, bzw. eine leere Liste, wenn kein nächster Zug möglich ist. Diese Liste wird erstellt durch Iterieren über die Spielersteine und die liegenden Steine, wobei die Anlegbarkeitsprüfungen nach unten in der Kapselung durchgereicht werden. Hierbei wird geprüft, ob es freie Anlegestellen gibt und eine Kombination der Ziffern übereinstimmt.

Aus dieser List wird aus o.g. Grund lediglich der ggf. vorhandene erste Eintrag als Zugvoschlag herangezogen. Da die Liste alle gültigen Züge enthält, kann hierauf eine weitere strategische Analyse aufgebaut werden.

10 Bildausgabe

Die Datenverarbeitung liefert den nächsten Zug, den der Benutzer ausführen soll. Dieser Vorschlag muss dem Benutzer geeignet vermittelt werden. Für die Darstellung wird eine Überlagerung über das zuletzt erstelle Bild erzeugt. Die Überlagerung enthält einen Pfeil vom Spielerstein zum Zielstein im Spielfeld.

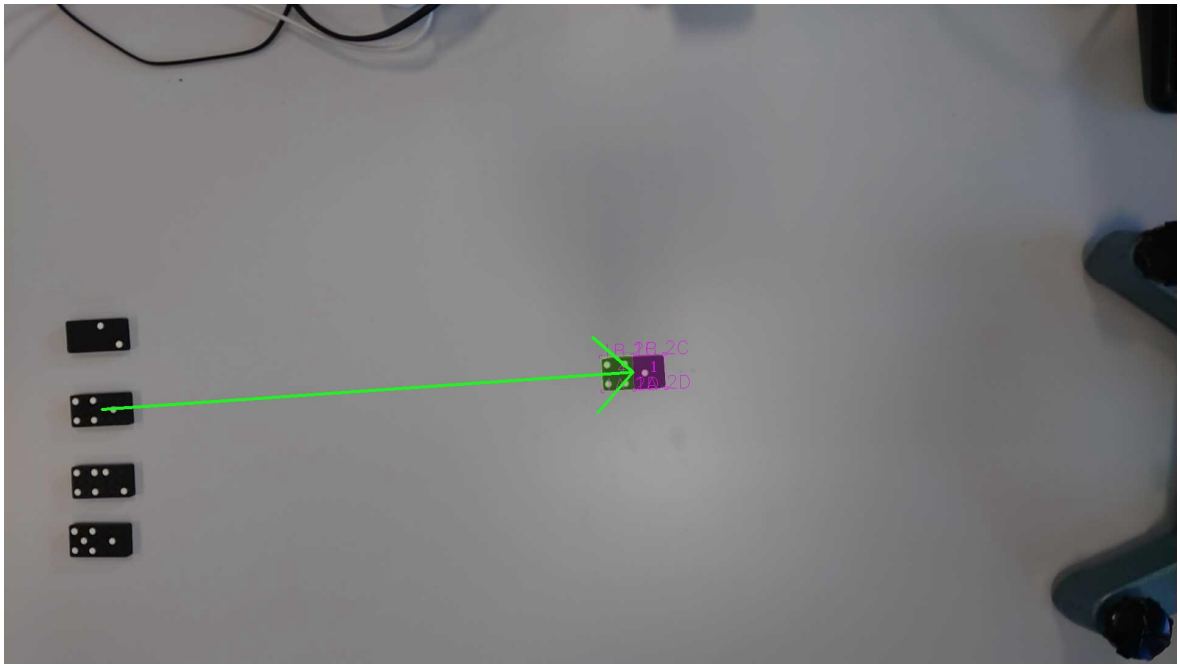


Abbildung 10.1: Bildausgabe mit Vorschlag

Falls der Spieler keinen passender Stein zum Legen hat und somit kein geeigneter Zug vorgeschlagen werden kann, wird dies durch einen Schriftzug im aktuellen Bild kenntlich gemacht (Abbildung 10.2).

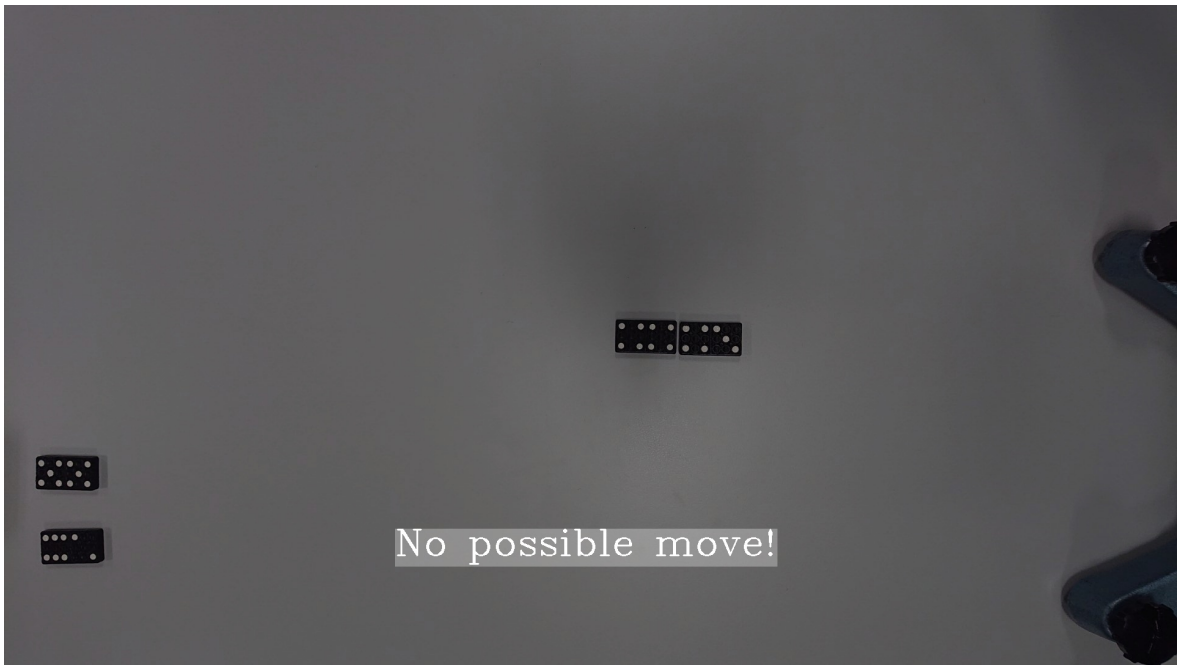


Abbildung 10.2: Bildausgabe bei keinem Vorschlag

11 Fazit

In diesem Projekt wurde ein Domino-Assistent entwickelt, der den jeweils nächsten Zug vorschlägt, sofern ein Zug möglich ist. Zur Erkennung der Steine und der Augenzahlen wurden eine Vielzahl von Bildverabreitungsschritten durchgeführt, um verschiedene Aufgaben und Probleme zu lösen. Durch ein sukzessives Vorgehen konnten die Ergebnisse immer weiter verbessert werden. Größtes Hindernis bei der Bildverarbeitung war die Beleuchtung, die je nach Tageszeit zu sehr guten, oder auch zu unbrauchbaren Ergebnissen führte. Aufgrund der Tatsache, dass bei diesem Projekt nur ein Hardware-Setting verfügbar war, waren die Testmöglichkeiten sehr eingeschränkt, sodass die Bildverarbeitung nur anhand von wenigen Testbildern, ca. 70 Bilder, die zu ähnlicher Tageszeit und damit ähnlicher Beleuchtung gewonnen wurden, verbessert werden konnte.

12 Ausblick

12.1 Beleuchtung

Aufgrund der Tatsache, dass die Bildverarbeitung sehr belichtungsabhängig ist, ist eine weitere externe Beleuchtung für dieses Projekt essentiell. So sollte in der nächsten Phase ein Beleuchtungskonzept entwickelt werden.

12.2 Erweiterte Seperation

Derzeit wird nach jedem Legen eines Steins das komplette Spielfeld untersucht und geprüft, ob dort ein neuer Stein liegt. Da die Positionen der bereits gelegten Steine und auch der Steine, an denen angelegt werden kann, in der Datenstruktur gespeichert und somit bekannt sind, können entsprechend kleinere Ausschnitte im Bild untersucht werden. Diese kleineren Ausschnitte können parallel verarbeitet werden. Auf diese Weise können drei Verbesserungen erreicht werden:

- Vermeidung von Reflektionsunterschieden, die durch schon vorher gelegte Steine in den betrachteten Bildausschnitten auftauchen und zu Irritationen in der Auswertung führen.
- Kleinere Ausschnitte lassen sich schneller verarbeiten.
- Mehrere Ausschnitte lassen sich parallel verarbeiten.

Literatur

[Bac18] Glenn De Backer. *recognizing-dices*. 2018. URL: https://www.simplicity.be/article/recognizing-dices/. (accessed: 09.11.2016) (siehe S. 12).

[Ope18a] OpenCV. *Canny Edge Detector*. 2018. URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html. (accessed: 29.11.2018) (siehe S. 7 f.).

[Ope18b] OpenCV. *Finding contours in your image*. 2018. URL: https://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html. (accessed: 03.12.2018) (siehe S. 9).

[Ope18c] OpenCV. *Floodfill*. 2018. URL: https://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#floodfill. (accessed: 21.12.2018) (siehe S. 9).

[Vos18] Gideon Vos. *Dice Detection using OpenCV*. 2018. URL: https://gideonvos.wordpress.com/2018/09/17/dice-detection-using-opencv/. (accessed: 14.11.2018) (siehe S. 12).

[Wik18a] Wikipedia. *Canny Algorithmus*. 2018. URL: https://de.wikipedia.org/wiki/Canny-Algorithmus. (accessed: 17.11.2018) (siehe S. 7 f.).

[Wik18b] Wikipedia. *Floodfill*. 2018. URL: https://de.wikipedia.org/wiki/Floodfill. (accessed: 21.12.2018) (siehe S. 9).

[Wik18c] Wikipedia. *Sobel-Operator*. 2018. URL: https://de.wikipedia.org/wiki/Sobel-Operator. (accessed: 13.12.2018) (siehe S. 7).