

Billiard-AI

Ein intelligenter Billardtisch

Projekt 2

Studienrichtung:

Informatik - Computer Perception and Virtual Reality

Autor:

Lukas Seglias, Luca Ritz

Dozent:

Markus Hudritsch

Experte:

Datum:

26. Mai 2021

Inhaltsverzeichnis

1	Zusammenfassung	1
2	Einführung	3
3	Ziele	5
3.0.1	Planung	6
4	Billiard-AI	7
4.1	Aufbau	7
4.1.1	Kamera	7
4.1.2	Projektor	8
4.2	Messung	8
4.3	Architektur	9
4.4	Modellkoordinatensystem	10
4.4.1	Pixelkoordinaten zu Kamerakoordinaten	11
4.4.2	Kamerakoordinaten zu Weltkoordinaten	12
4.4.3	Weltkoordinaten zu Bandenkoordinaten	14
4.4.4	Bandenkoordinaten zu Modellkoordinaten	15
4.4.5	Modellkoordinaten zu Pixelkoordinaten	16
4.4.6	Kugelradius in Pixel	16
4.5	Detektion von Snooker-Kugeln	17
4.6	Visualisierung in Unity	20
5	Resultate	21
5.1	Detektionsgenauigkeit	21
5.2	Übersetzungsgenauigkeit der erkannten Kugeln	21
6	Weitere Arbeiten	23
7	Fazit	25
8	Anhang	37
8.1	Messgeräte	37
8.1.1	Messschieber	37
8.1.2	Lasermessgerät	37
8.2	Fehlerradius	37
8.3	Fehler - Grundwahrheit	40
8.4	Testbilder	40
8.4.1	Detektionstestbilder	40
8.4.2	Übersetzungsgenauigkeit	41

1 Zusammenfassung

2 Einführung

Wie vieles andere ist auch das Erlernen des Billardspiels eine schwierige Sache. Es stellen sich Fragen wie „Welche Kugel soll man anspielen?“, „Wie soll man die Kugel anspielen“ oder „Wie hält man den Queue richtig?“. Darauffolgend gibt es noch diverse weitere Überlegungen, welche den Profi vom Anfänger unterscheiden. Wie in anderen Spielen auch, ist hier Weitsicht gefragt. Es geht also nicht nur darum, eine Kugel zu versenken, sondern auch den Spielstand so zu verändern, dass optimal weitergespielt werden kann. Das Stichwort ist im Billard vor allem die Platzierung der weissen Kugel.

Diese Arbeit hat nun nicht den Anspruch, alle aufgeführten Fragestellungen beantworten zu können. In dieser ersten Iteration geht es vielmehr darum, die Basis zu legen, auf welcher in der Bachelor-Thesis aufgebaut wird. Um überhaupt daran zu denken, dem Spieler in irgendeiner Form eine Hilfestellung zu geben, muss zuerst der aktuelle Spielstand möglichst präzise erkannt und ebenso auf dem Tisch visualisiert werden. Ebendieses Vorgehen wird in den nachfolgenden Kapiteln erläutert.

3 Ziele

Ins Billard-Spiel einzusteigen ist nicht ganz einfach. Zu Beginn lässt es sich schlecht abschätzen, welchen Weg eine Kugel nehmen wird, wenn man sie anschlägt und den optimalen Stoss über mehrere Züge hinaus zu planen, erst recht. Denn bei fortgeschrittenen Spielen ist es oft wichtig, dass die weisse Kugel optimal für den nächsten Stoss platziert wird.

Es soll ein System entstehen, das dem Spieler den optimalen Stoss vorschlägt, basierend auf Kriterien und einer festgelegten Tiefe. Dazu soll eine Kamera den Spielstand auf dem Billiardtisch erkennen, verarbeiten und dem Spieler Hilfestellungen mittels eines Projektors anzeigen.

In dieser Arbeit soll das Fundament für die Umsetzung dieser Idee gelegt werden. Diese Basissetzen sich aus den folgenden Teilstücken zusammen:

Aufsetzen des Projekts Dazu gehören nebst Überlegungen zur Architektur und der Implementation davon auch das Aufsetzen der Dokumentation, welche in Latex geschrieben wird.

Aufbau des Systems Dies beinhaltet die Montage und genaue Ausrichtung der Kamera wie auch des Projektors.

Kalibrierung der Kamera Bestimmen der intrinsischen Kameramatrix, um genaue Bildanalyse betreiben zu können.

Erkennung der Kugelpositionen Die Kugeln und deren Position sollen anhand eines Farbbilds erkannt werden können.

Übersetzung in internes Koordinatensystem Mittels Marker, welche am Tisch angebracht werden, sollen die auflösungsabhängigen Pixelkoordinaten in ein standardisiertes internes Koordinatensystem überführt werden.

Anzeige des aktuellen Spielstands Die detektierten Positionen aller Kugeln sollen mittels eines Projektors auf den Tisch projiziert werden.

Genauigkeitsanalyse Die Genauigkeit der Erkennung der Kugelpositionen und der Übersetzung in ein internes Koordinatensystem soll untersucht werden.

Weitere zu erarbeitende Lösungen, welche nicht in dieser Arbeit umgesetzt wurden:

Klassifikation der Kugeln Wurden die Kugeln erkannt, sollen sie entsprechend der Farbe klassifiziert werden.

Theoriearbeiten zur Suche eines Stosses Um einen optimalen Stoss zu finden, bedarf es zunächst einiger theoretischen Grundüberlegungen. Dies beinhaltet z.B. einen Algorithmus, um einen Stoss zu finden sowie auch physikalisch korrekt zu beschreiben.

Suche eines einfachen Stosses Sobald der theoretische Ansatz erarbeitet wurde, soll eine erste einfache Suche implementiert werden. Diese soll nur Stosse berücksichtigen, welche eine Kugel direkt in eine Loch spielen.

Bewertung von Stössen Jeder gefundene Stoss muss bewertet werden aufgrund verschiedener zu erarbeitenden Kriterien.

Anzeige von Stössen Gefundene Stosse sollen über den Projektor dem Spieler angezeigt werden, um diesen zu unterstützen.

Suche indirekter Stosse Indirekte Stosse über die Bande oder über andere Kugeln sollen gefunden, bewertet und angezeigt werden.

Es ist weiterhin anzumerken, dass es in erster Linie um Snooker-Billard geht. Dies hat mehrere Gründe. Einerseits soll in dieser Arbeit nicht die Klassifikation der Kugeln im Zentrum stehen, sondern die Suche nach einem optimalen Stoss. Es wird angenommen, dass dies mit Snooker-Kugeln einfacher ist als mit Pool-Billard-Kugeln. Andererseits wird das Projekt zusammen mit einem Unternehmen durchgeführt, welches eventuell auch einen kommerziellen Ansatz verfolgen will. Da grössere Turniere wie Weltmeisterschaften in Snooker ausgetragen werden, kam schnell der Wunsch auf, das Hauptaugenmerk darauf zu legen. Nichtsdestotrotz wird die Anwendung so abstrakt gehalten, dass sie mit wenig Aufwand auf Pool-Billard portiert werden könnte. Dies wird aber vorläufig weder in Projekt-2 noch in der darauffolgenden Bachelor-Thesis von Relevanz sein.

3.0.1 Planung

Die Planung beinhaltet eine Auflistung der Ziele nach Deadline sowie Bearbeiter.

Ziel	Datum	Bearbeiter
Evaluation Beamer & Kamera	08.03.2021	Lukas & Luca
Überlegungen zum Aufbau	11.03.2021	Lukas & Luca
Entscheid Beamer- Kameratyp	11.03.2021	Lukas & Luca
Theorie der Kamera-Kalibrierung erarbeiten	18.03.2021	Lukas
Beschreibung Such-Algorithmus	18.03.2021	Luca
Theorie der Beamer-Kalibrierung erarbeiten	25.03.2021	Lukas & Luca
Aufbau des Systems	01.04.2021	Lukas & Luca
Definitive Kalibrierung	08.04.2021	Lukas & Luca
Kugel erkennen & Klassifikation	15.04.2021	Luca
Fine-Tuning Kugel erkennen & Klassifikation	29.04.2021	Luca
Marker - Transformation in internes Koordinatensystem	29.04.2021	Lukas
Resultate festhalten Erkennung, Klassifikation & Transformation	06.05.2021	Lukas & Luca
Einfache Suche implementieren	03.06.2021	Lukas & Luca

Tabelle 3.1: Ziele

4 Billiard-AI

In diesem Kapitel werden der Versuchsaufbau, die gewählte Architektur und die umgesetzten Lösungen beschrieben.

4.1 Aufbau

Die Kamera wie auch der Projektor werden mittels eines Baugerüsts über dem Tisch platziert. Dieses ermöglicht die nötige Flexibilität, um die optischen Objekte unabhängig voneinander optimal auszurichten. Der Aufbau ist in Abbildung 4.1 ersichtlich.



Abbildung 4.1: Baugerüst

4.1.1 Kamera

Als Kamera wurde eine Intel RealSense Depth Camera D435 verwendet [Unk21b]. Die intrinsischen Kameraparameter des Farbsensors wurden mithilfe des RealSense SDK 2.43.0 [Unk21d] ausgelesen. Beim Farbsensor handelt es sich um

einen *OmniVision OV2740* [Unk21c]. Die Auflösung des Sensors beträgt 1920x1080 und die Pixelgröße beträgt $1.4\text{ }\mu\text{m}$ [Unk21f].

Nachfolgend ist die allgemeine intrinsische Kameramatrix in *column-major order* aufgeführt. Siehe [Unk21e] für die Erläuterung der Parameter in *row-major order*.

$$\begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (4.1)$$

Die konkreten Werte für die Brennweite (f_x, f_y) [Pixel], das optische Zentrum (c_x, c_y) [Pixel] und der Schiefekoeffizient s sind nachfolgend eingefügt.

$$\begin{pmatrix} 1375.68884 & 0 & 974.842407 \\ 0 & 1375.85425 & 539.362732 \\ 0 & 0 & 1 \end{pmatrix} \quad (4.2)$$

Die Parameter für die Modellierung der radialen und tangentialen Verzerrung k_1, k_2, k_3, p_1, p_2 sind gemäß RealSense SDK alle 0 und das verwendete Modell wird *Inverse Brown-Conrady* genannt.

4.1.2 Projektor

Der eingesetzte Projektor ist ein *BenQ MW820ST*, siehe [Unk21g]. Es handelt sich um einen Kurzdistanzbeamer, welcher etwa 1m über dem Billardtisch angebracht wurde.

4.2 Messung

Der Tisch ist an den Seiten leicht schräg, was ein genaues Messergebnis verunmöglicht. Weiterhin ist er an den Kanten nicht fest und kann leicht deformiert werden. Um diese Ungenauigkeitsfaktoren möglichst gering zu halten, wurden die gegenüberliegenden Tischkanten mit je zwei Holzstücken (Abbildung 4.2, 1), deren Breite individuell über einen Messschieber millimetergenau bestimmt wurde, ausgestattet. Nun kann die Distanz zwischen den Holzstücken gemessen und anschließend die Breite der Hölzer dazuaddiert werden. Die Messung selbst (Abbildung 4.2, 2 und 3) wird über ein Laser-Messgerät¹ durchgeführt. Die Messungenauigkeit wird mit $\pm 1.5\text{ mm}$ angegeben. Die Abbildung 4.2 zeigt die Messung in X-Richtung, diese Messung wird ebenfalls in gleicher Art und Weise in Y-Richtung durchgeführt.

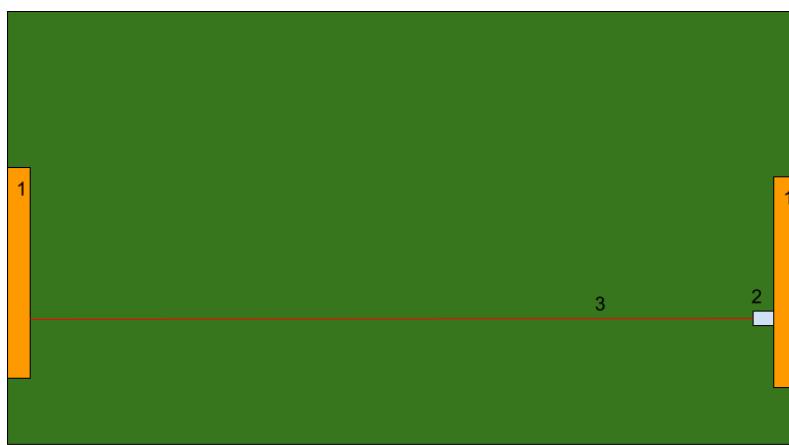


Abbildung 4.2: Messung - Tisch

Um die Messung der Kugeln zu erläutern, muss zuerst das Koordinatensystem eingeführt werden. Die Kugeln selbst werden über eine Kamera aufgenommen und deren Position daher auch in einem Pixelkoordinatensystem bestimmt. Dies ist aus

¹Die genauen Spezifikationen sind im Anhang 8.1 ersichtlich.

mehreren Gründen ungünstig. Einerseits soll die allgemeine Abhängigkeit zur Kamera vermieden werden, andererseits ist das Pixelkoordinatensystem für die Visualisierung nicht gut geeignet.

Die Kugeln werden also in ein Modellkoordinatensystem übersetzt². Bei diesem befindet sich der Ursprung in der Mitte des Tisches und die X- wie auch die Y-Achse bilden die Breite und Höhe in Millimeter ab.

Um nun die Position der Kugel zu bestimmen, wird zuerst deren Abstand zu den Banden gemessen (siehe Abbildung , 1 und 2). Dies geschieht wie beim Ausmessen des Tisches über ein Holzstück und das Lasermessgerät. Wurden beide Distanzen bestimmt, so kann der Mittelpunkt der Kugel berechnet werden, indem noch der Radius, welcher durch einen Messschieber bestimmt wurde, dazugeaddiert wird. Es gilt nun, diesen Punkt in das Modellkoordinatensystem zu übersetzen. Dazu wird jeweils die halbe Breite wie auch Höhe (beachte den Ursprung des Modellkoordinatensystems) von dem Mittelpunkt abgezogen. Abschliessend werden noch die korrekten Vorzeichen über den Quadranten bestimmt.

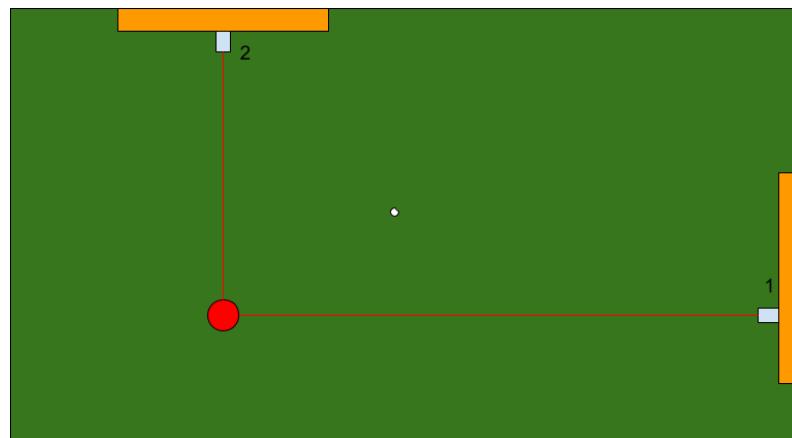


Abbildung 4.3: Messung - Kugel

4.3 Architektur

Die eigentliche Funktionalität der Bildanalyse und Suche wird in einer Core-Library untergebracht, welche nativ in C++ entwickelt wird. Die Anzeige der Hilfestellungen für den Spieler soll mithilfe von Unity erfolgen. Unity stellt die Schnittstelle zwischen dem Spieler und der ganzen Applikation dar und muss sich daher mit der Core-Library austauschen. Dafür wird eine C/C++-Bibliothek entwickelt, die ein C-Interface zur Core-Library bereitstellt. Unity lädt diese Bibliothek und wandelt erhaltene native Objekte in Objekte um, welche in Unity verwendet werden können. Eine Darstellung kann in Abbildung 4.4 entnommen werden.

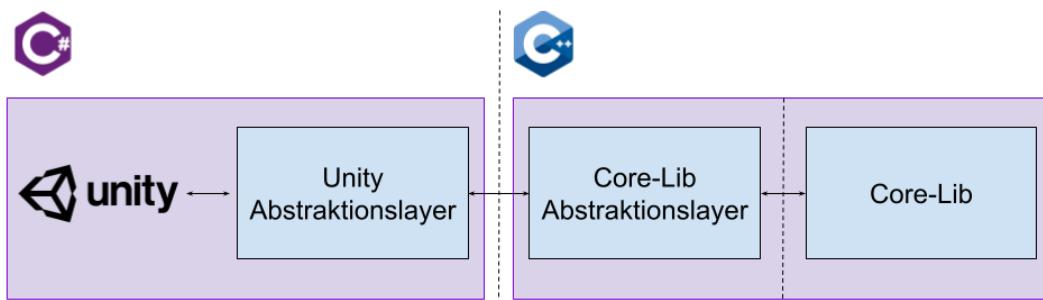


Abbildung 4.4: Grobarchitektur - Applikationsumgebung

Die Core-Library selbst besteht aus den folgenden Komponenten:

billiard_capture Dient als Schnittstelle zur Kamera und stellt die Bilder im OpenCV-Format bereit.

billiard_detection Erstellt aus dem Spielstand eine interne Repräsentation, welchen den Status beschreibt.

billiard_snooker Enthält Snooker-spezifische Funktionalität, u.a. die Erkennung der Snooker-Kugeln auf dem Billiardtisch.

²Siehe Kapitel 4.4

billiard_search Verwendet den aktuellen Status wie auch eine zusätzliche Such-Beschreibung, um einen optimalen Stoss zu berechnen.

billiard_physics Stellt Funktionalität bereit, um physikalische Berechnungen durchzuführen.

Für die Umsetzung der Core-Library wird OpenCV³ 5.4.2 verwendet.

4.4 Modellkoordinatensystem

Bei der Aufnahme des Spielstandes über eine Kamera können die Kugeln im Bild erkannt und deren Zentrum, was der Position entspricht, in Pixelkoordinaten im Bild ermittelt werden. Die Pixelkoordinaten sind für die weitere Verarbeitung, sei es Analyse und Darstellung des Spielstandes oder Simulation von Spielzügen, nicht geeignet. Es wird ein Koordinatensystem benötigt, welches möglichst unabhängig von der Auflösung, Position und Blickrichtung der Kamera ist.

Idealerweise werden die Pixelkoordinaten in ein zu definierendes Koordinatensystem transformiert, welches für die weitere Verarbeitung verwendet werden kann. Nachfolgend wird dieses Koordinatensystem, das Modellkoordinatensystem genannt wird, und wie folgt definiert ist:

1. Das Koordinatensystem ist zweidimensional.
2. Der Ursprung befindet sich in der Mitte des Billardtisches auf Höhe der Kugelmittelpunkte.
3. Die X-Achse ist parallel zu der längeren Seite und die Y-Achse ist parallel zu der kürzeren Seite des Tisches.
4. Die Länge eines Einheitsvektors entspricht 1mm.

Diese Definition ist insofern praktisch, als dass die Banden und Löcher des Billardtisches sehr einfach in diesem Koordinatensystem definiert werden können. In Abbildung 4.5 ist der Ursprung und die Achsen des Koordinatensystems eingezeichnet.

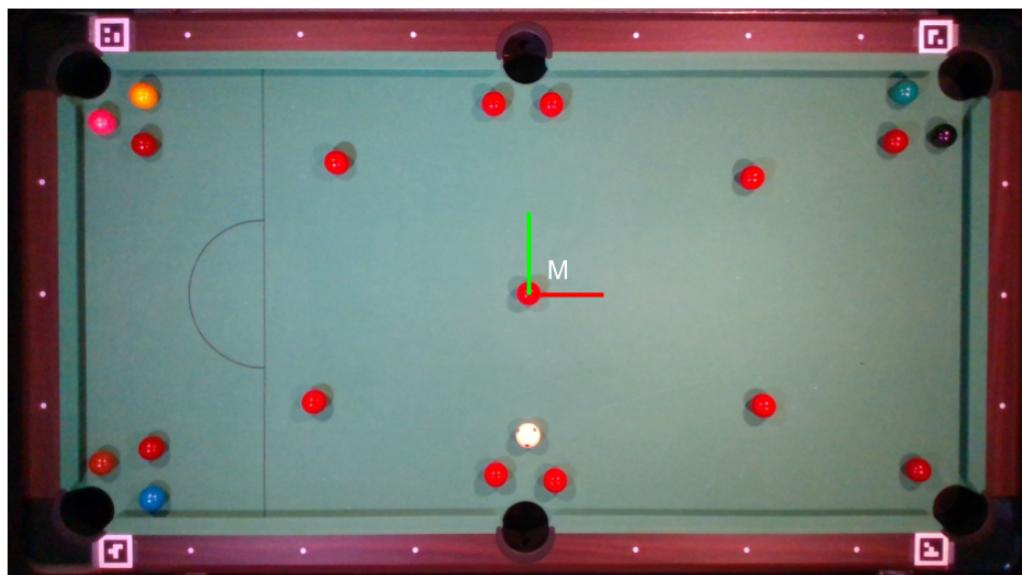


Abbildung 4.5: Modellkoordinatensystem

Die Transformation vom Pixelkoordinatensystem zum Modellkoordinatensystem erfolgt über mehrere Koordinatensystemtransformationen:

1. Pixelkoordinaten zu Kamerakoordinaten, siehe Abschnitt 4.4.1
2. Kamerakoordinaten zu Weltkoordinaten, siehe Abschnitt 4.4.2
3. Weltkoordinaten zu Bandenkoordinaten, siehe Abschnitt 4.4.3
4. Bandenkoordinaten zu Modellkoordinaten, siehe Abschnitt 4.4.4

³Siehe <https://opencv.org>

Alle Koordinatensysteme bis auf das Pixelkoordinatensystem sind in der Einheit Millimeter definiert, um nicht benötigte Skalierungen in den Transformationen zu vermeiden.

4.4.1 Pixelkoordinaten zu Kamerakoordinaten

Für die Transformation von Pixelkoordinaten zu Kamerakoordinaten wird die intrinsische Kameramatrix und die Grösse der Sensorpixel der verwendeten Kamera benötigt. Die intrinsische Kameramatrix K ist nachfolgend erneut aufgeführt, diese wurde zusammen mit der verwendeten Kamera in Abschnitt 4.1.1 beschrieben.

$$K = \begin{pmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \quad (4.3)$$

Nun soll ein 2D-Punkt im Pixelkoordinatensystem in einen 3D-Punkt im Kamerakoordinatensystem überführt werden. Sei der Pixelpunkt $p = (p_x, p_y)$, dann kann diesem der *principal point* $c = (c_x, c_y)$ abgezogen werden, um den Ursprung in die Mitte des Bildsensors zu verschieben. Anschliessend müssen die Koordinaten mit der Sensorgrösse $s = (s_x, s_y)$ [mm] skaliert werden, um die Koordinaten von der Einheit Pixel in die Einheit Millimeter umzuwandeln. Die Z-Komponente des resultierenden Punktes P_C im Kamerakoordinatensystem ist die Brennweite aus der intrinsischen Kameramatrix. Dabei gilt es zu beachten, dass die Parameter f_x und f_y der intrinsischen Kameramatrix in der Einheit Pixel sind, und daher ebenfalls mit der Sensorgrösse s skaliert werden müssen.

$$f = f_x \cdot s_x \quad (4.4)$$

$$P_{C,x} = (p_x - c_x) \cdot s_x \quad (4.5)$$

$$P_{C,y} = (p_y - c_y) \cdot s_y \quad (4.6)$$

$$P_{C,z} = f \quad (4.7)$$

$$P_C = \begin{pmatrix} P_{C,x} \\ P_{C,y} \\ P_{C,z} \end{pmatrix} \quad (4.8)$$

Abbildung 4.6 zeigt das Bildkoordinatensystem und stammt von [Sat20], wo eine ausführlichere Beschreibung der intrinsischen und extrinsischen Kameramatrix entnommen werden kann.

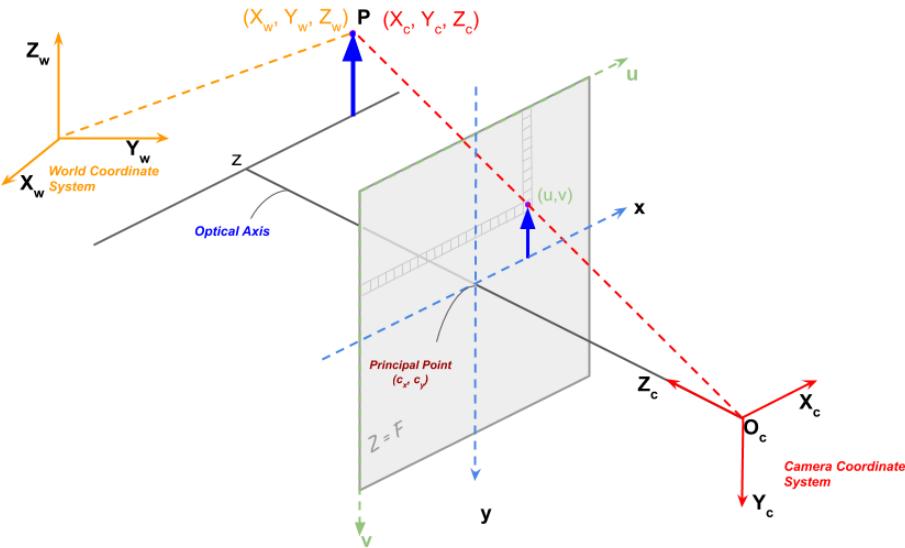


Abbildung 4.6: Bildkoordinatensystem. Bildquelle: [Sat20]

Damit ist diese Transformation abgeschlossen und der Punkt in Kamerakoordinaten kann in Weltkoordinaten übersetzt werden.

4.4.2 Kamerakoordinaten zu Weltkoordinaten

Der Punkt P_C in Kamerakoordinaten muss nun in die reale Welt überführt werden. Da der Punkt P_C relativ zur Kamera positioniert ist, muss die Position der Kamera in der realen Welt bestimmt werden. Somit braucht es eine *pose estimation* der Kamera, d.h. die Position und Orientierung der Kamera in der realen Welt müssen bestimmt werden. Eine *pose estimation* benötigt eine kalibrierte Kamera und ein Objekt bekannter Grösse, welches im Bild erkannt werden kann.

Als Objekt bekannter Grösse und Position wurde ein ArUco-Board [ope21a] mit vier Markern verwendet. Die verwendeten Marker mit einer Seitenlänge von 50mm und einer Grösse von 3x3 bits sind in Abbildung 4.7 aufgeführt.

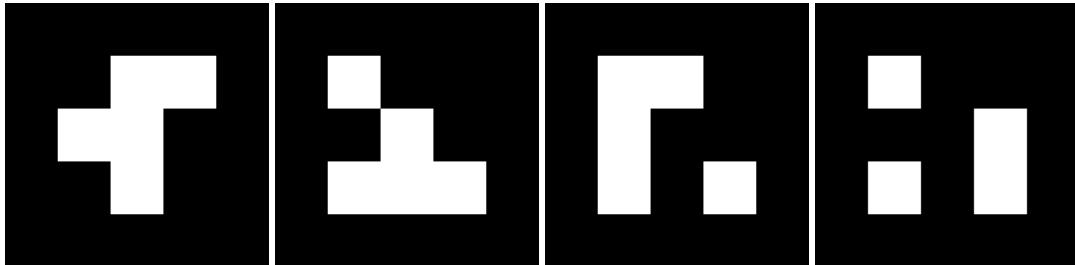


Abbildung 4.7: Verwendete ArUco markers

Diese Marker wurden auf dem Tisch wie in Abbildung 4.8 angebracht und deren relative Position zueinander ausgemessen.

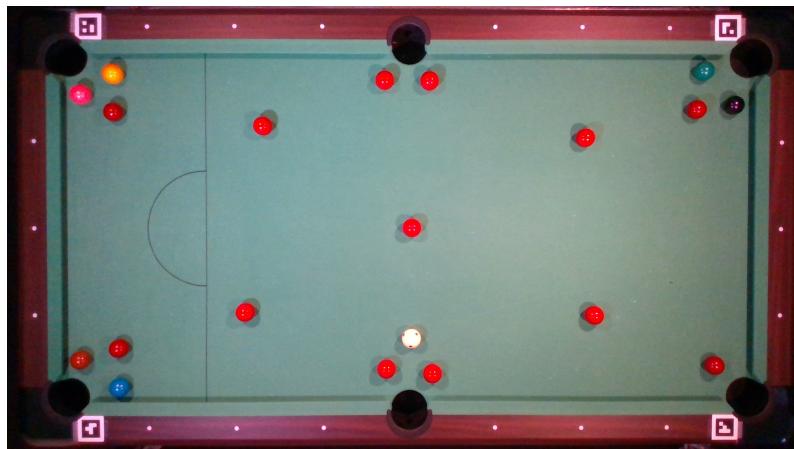


Abbildung 4.8: Billardtisch mit ArUco markers

Das ArUco board ist definiert, sobald alle Eckpunkte der einzelnen Marker relativ zu einem gewählten Weltkoordinatenursprung bekannt sind. Als Ursprung wurde der Mittelpunkt des Markers in Abbildung 4.9 unten links gewählt. Die X-Achse ist parallel zur langen Seite des Tisches und die Y-Achse ist parallel zur kurzen Seite des Tisches.

Mit der Definition des bekannten Objekts in der realen Welt kann nun die *pose estimation* durchgeführt werden, um daraus die extrinsische Kameramatrix zu bestimmen. Die extrinsische Kameramatrix E beschreibt die Transformation eines Punkts P_W in Weltkoordinaten zu einem Punkt P_C in Kamerakoordinaten und ist wie folgt definiert [Kyl12]:

$$E = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (4.9)$$

Die inverse Transformation E^{-1} transformiert einen Punkt P_C in Kamerakoordinaten in einen Punkt P_W in Weltkoordinaten. So kann die Position der Kamera in Weltkoordinaten C_W anhand der Position in Kamerakoordinaten C_C und der Matrix E^{-1}



Abbildung 4.9: Weltkoordinatensystem

bestimmt werden:

$$C_C = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \quad (4.10)$$

$$C_W = E^{-1} \cdot C_C \quad (4.11)$$

Ebenso kann die Position des Pixels in Weltkoordinaten P_W bestimmt werden, wobei P_C in Abschnitt 4.4.1 erarbeitet wurde:

$$P_C = \begin{pmatrix} P_{C,x} \\ P_{C,y} \\ P_{C,z} \end{pmatrix} \quad (4.12)$$

$$P_W = E^{-1} \cdot P_C \quad (4.13)$$

Es gilt zu beachten, dass die Position des Pixels in Weltkoordinaten P_W nicht der Position der Kugel, welche evtl. an dieser Pixelposition detektiert wurde, entspricht. Die projektive Abbildung der realen Welt auf den Bildsensor kann nicht rückgängig gemacht werden, weil die Tiefeninformation verloren gegangen ist.

Da bekannt ist, dass sich alle Mittelpunkte der Kugeln auf dem Tisch auf einer Ebene befinden, kann die Pixelkoordinate eines Kugelmittelpunkts trotzdem in eine Weltkoordinate transformiert werden. Es gilt die Weltkoordinate B_W der Kugel zu bestimmen. Dazu kann eine Linie L zwischen Kamera-Weltkoordinate C_W und Pixel-Weltkoordinate P_W gezogen werden und deren Schnittpunkt mit der Ebene Σ , welche auf Höhe der Kugelmittelpunkte in Weltkoordinaten H ist, bestimmt werden. Die Linie L ist in der Parameterform mit dem Stützvektor \vec{q} und dem Richtungsvektor \vec{v} und dem Skalierungsfaktor λ definiert. Sei B eine Ebene in der Normalenform mit dem Stützvektor \vec{a} und dem Normalenvektor \vec{n} , dann gilt:

$$C_W = E^{-1} \cdot C_C \quad (4.14)$$

$$P_W = E^{-1} \cdot P_C \quad (4.15)$$

$$\vec{q} = C_W \quad (4.16)$$

$$\vec{v} = P_W - C_W \quad (4.17)$$

$$L : B_W = \vec{q} + \lambda \cdot \vec{v} \quad (4.18)$$

$$\vec{a} = \begin{pmatrix} 0 \\ 0 \\ H \end{pmatrix} \quad (4.19)$$

$$\vec{n} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \quad (4.20)$$

$$\Sigma : (B_W - \vec{a}) \cdot \vec{n} = 0 \quad (4.21)$$

$$(4.22)$$

Durch Einsetzen der Liniengleichung in die Ebenengleichung kann der Schnittpunkt ermittelt werden.

$$(\vec{q} + \lambda \cdot \vec{v} - \vec{a}) \cdot \vec{n} = 0 \quad (4.23)$$

$$\vec{q} \cdot \vec{n} + \lambda \cdot \vec{v} \cdot \vec{n} - \vec{a} \cdot \vec{n} = 0 \quad (4.24)$$

$$\lambda \cdot \vec{v} \cdot \vec{n} = \vec{a} \cdot \vec{n} - \vec{q} \cdot \vec{n} \quad (4.25)$$

$$\lambda \cdot \vec{v} \cdot \vec{n} = (\vec{a} - \vec{q}) \cdot \vec{n} \quad (4.26)$$

$$\lambda = \frac{(\vec{a} - \vec{q}) \cdot \vec{n}}{\vec{v} \cdot \vec{n}} \quad (4.27)$$

Falls $\vec{v} \cdot \vec{n} = 0$ gilt, dann gibt es keinen Schnittpunkt. Die Kugel-Weltkoordinate kann durch Skalierung der Linie bestimmt werden:

$$B_W = \vec{q} + \lambda \cdot \vec{v} \quad (4.28)$$

$$B_W = C_W + \lambda \cdot (P_W - C_W) \quad (4.29)$$

$$B_W = C_W + \frac{(\vec{a} - \vec{q}) \cdot \vec{n}}{\vec{v} \cdot \vec{n}} \cdot (P_W - C_W) \quad (4.30)$$

Damit ist die Weltkoordinate des Kugelmittelpunkts B_W anhand dessen Pixel-Weltkoordinate P_W ermittelt.

4.4.3 Weltkoordinaten zu Bandenkoordinaten

Die vorletzte Transformation von Weltkoordinaten zu Bandenkoordinaten ist eine einfache Translation des Ursprungs, siehe dazu Abbildung 4.10. Der Ursprung wird nicht rotiert, lediglich verschoben. Diese Verschiebung kann auch in der Angabe der Weltkoordinaten des ArUco boards berücksichtigt werden, wodurch diese Translation obsolet wird. Sie wurde belassen, da dies eine kleine Operation darstellt und dadurch das Ausmessen des ArUco boards und dessen Verschiebung zum Bandenursprung klar getrennt ist.



Abbildung 4.10: Welt-Ursprung W und Banden-Ursprung R

Wenn die Verschiebung von Welt-Ursprung zu Banden-Ursprung $T_{wb} = (\Delta x, \Delta y, 0)$ ist, dann ist die Kugelposition in Bandenkoordinaten B_R :

$$B_R = B_W + T_{wb} \quad (4.31)$$

$$B_R = B_W + \begin{pmatrix} \Delta x \\ \Delta y \\ 0 \end{pmatrix} \quad (4.32)$$

Somit steht nur noch die letzte Transformation bevor.

4.4.4 Bandenkoordinaten zu Modellkoordinaten

Zuletzt wird der Ursprung erneut verschoben, vom Ursprung des Bandenkoordinatensystems zum Ursprung des Modellkoordinatensystems, siehe dazu Abbildung 4.11.

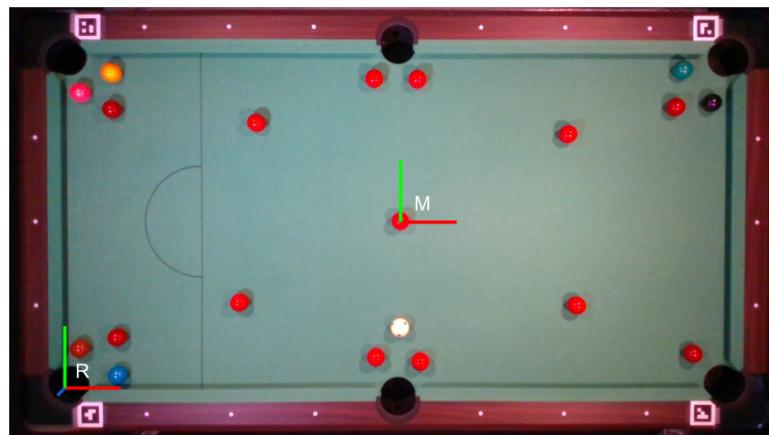


Abbildung 4.11: Banden-Ursprung B und Modell-Ursprung R

Für diese Verschiebung muss lediglich die Länge und Breite des Bereichs, welcher von den Banden begrenzt wird, gemessen werden, siehe Abbildung 4.12.

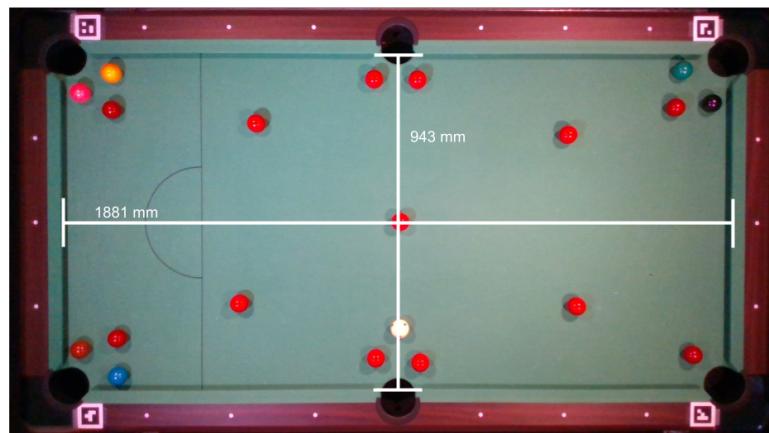


Abbildung 4.12: Länge und Breite des Spielbereits

Wenn der Spielbereich die Länge L und Breite B hat, dann ist die Kugelposition in Modellkoordinaten B_M :

$$\Delta x = -\frac{L}{2} \quad (4.33)$$

$$\Delta y = -\frac{B}{2} \quad (4.34)$$

$$T_{bm} = \begin{pmatrix} \Delta x \\ \Delta y \\ 0 \end{pmatrix} B_M = B_R + T_{bm} \quad (4.35)$$

Somit konnte die Kugelposition in Pixelkoordinaten in ein Modellkoordinatensystem umgewandelt werden, welches für die weitere Verarbeitung von Nutzen ist.

4.4.5 Modellkoordinaten zu Pixelkoordinaten

Im Rahmen der Erkennung der Kugeln auf einem Bild ist es von Nutzen, die Pixlekoordinate einer Modellkoordinate zu bestimmen. Beispielsweise kann der Radius der Kugeln in Pixel bestimmt werden, wenn die Länge des Tisches und der Radius der Kugeln in Millimeter bekannt sind. Dies ist nützlich, da somit die genaue Pixelgrösse der Kugeln nicht fix hinterlegt sein muss.

Um dies zu bewerkstelligen, müssen einige der zu Beginn des Kapitels aufgeführten Transformationen rückgängig gemacht werden. Die totwendigen Schritte sind die folgenden:

1. Modellkoordinaten zu Bandenkoordinaten
2. Bandenkoordinaten zu Weltkoordinaten
3. Weltkoordinaten zu Kamerakoordinaten
4. Weltkoordinaten zu Pixelkoordinaten

Die Umwandlung von Modell- zu Bandenkoordinaten und von Banden- zu Weltkoordinaten sind die inversen Translationen, die in den Abschnitten 4.4.4, resp. 4.4.3, beschrieben wurden.

Der Schritt von Welt- zu Pixelkoordinaten erfolgt direkt über die OpenCV-Funktion *projectPoints* [ope21b], welche die extrinsische Kameramatrix, die intrinsische Kameramatrix und die Verzerrungskoeffizienten erfordert. Im Prinzip wendet die Funktion die extrinsische, gefolgt von der intrinsischen Kameramatrix auf die Weltkoordinaten an, führt die projektive Abbildung (*perspective divide*) aus und wendet die Verzerrung an, um Pixelkoordinaten zu erhalten.

4.4.6 Kugelradius in Pixel

Mithilfe des in Abschnitt 4.4.5 beschriebenen Algorithmus kann eine Modellkoordinate in eine Pixelkoordinate überführt werden. Der Radius der Kugel in Millimeter und die Länge des Spielbereichs (Länge zwischen den zwei kürzeren Banden) in Millimeter können genutzt werden, um den Pixelradius der Kugel zu bestimmen. Dazu werden zwei Punkte des Modellkoordinatensystems gewählt, R_1 und R_2 , welche auf den gegenüberliegenden Banden im Abstand L liegen. Sei $f(P_M)$ die Funktion, welche eine Modellkoordinate in eine Pixelkoordinate übersetzt, dann gilt für den Kugelradius x in Pixel:

$$I = \frac{L}{2} \quad (4.36)$$

$$R_1 = \begin{pmatrix} -I \\ 0 \\ 0 \end{pmatrix} \quad (4.37)$$

$$R_2 = \begin{pmatrix} I \\ 0 \\ 0 \end{pmatrix} \quad (4.38)$$

$$P_1 = f(R_1) \quad (4.39)$$

$$P_2 = f(R_2) \quad (4.40)$$

$$\lambda = \frac{P_2 - P_1}{L} x = R \cdot \lambda \quad (4.41)$$

Dabei handelt es sich bei λ um den Faktor Pixel pro Millimeter.

4.5 Detektion von Snooker-Kugeln

Ziel ist es, die Position, d.h. den Kugelmittelpunkt, jeder Kugel in Pixelkoordinaten so genau wie möglich zu bestimmen. Diese Pixelkoordinaten können anschliessend wie in Kapitel 4.4 beschrieben in ein Modellkoordinatensystem umgewandelt werden, welches für die weitere Verarbeitung genutzt werden kann.

Die Anwendung des Circle Hough transform [Wik21a] für die Extraktion der Kugeln ist naheliegend, führt allerdings auf dem Graustufenbild dazu, dass auch Schatten als Kugeln detektiert werden. Diese gilt es zu vermeiden, weshalb zunächst eine Segmentation aufgrund anderer Eigenschaften erfolgen muss. In Abbildung 4.13 ist das Eingabebild mit Schatten abgebildet.

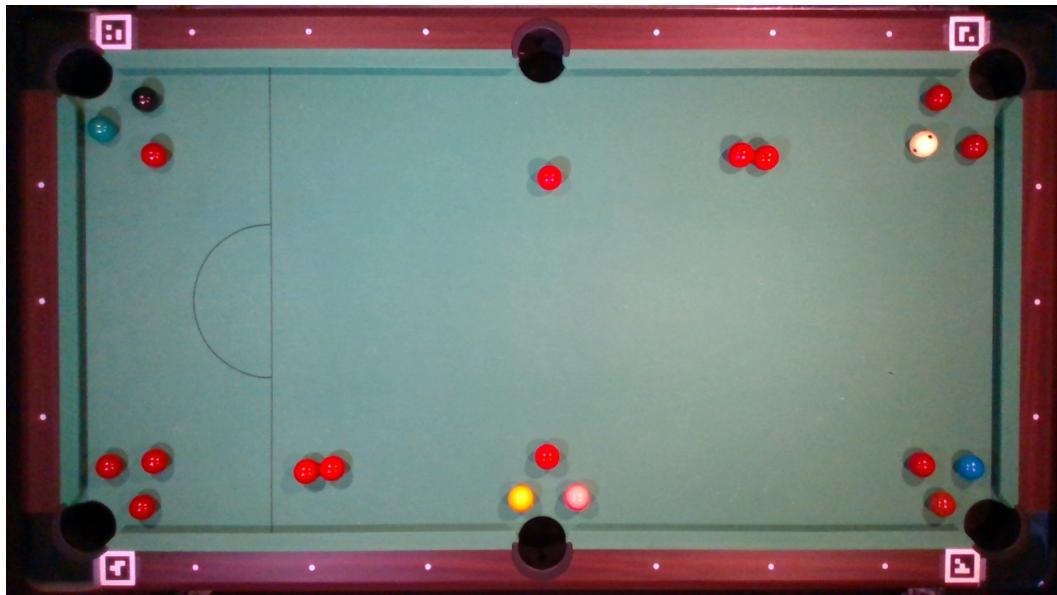


Abbildung 4.13: Eingabebild mit Schatten

Zunächst wird ein Gaussfilter mit der Kernelgrösse von 5×5 auf das Eingabebild angewendet, um Rauschen zu unterdrücken. Anschliessend wird das Bild vom RGB-Farbraum in den HSV-Farbraum [Wik21b] umgewandelt, um eine einfacher Segmentation zu ermöglichen, weil dadurch einzelne Eigenschaften wie Farbwert, Sättigung und Helligkeit gefiltert werden können.

Des weiteren sollen Kreise, welche ausserhalb des Spielfeld liegen, ausgeschlossen werden können. Dazu wird eine binäre Maske, ein einkanaliges Bild mit den Werten 255 (= wahr) und 0 (= falsch), erzeugt. In der Maske hat jedes Pixel, an dem ein Kugelmittelpunkt sein darf, den Wert 255, alle anderen den Wert 0. Das Modellkoordinatensystem wurde in Kapitel 4.4 beschrieben. Die Banden und Löcher des Tisches können in Modellkoordinaten definiert werden, sofern der Tisch ausgemessen wurde. In Kapitel 4.4.5 wurde erarbeitet, wie eine Modellkoordinate in eine Pixelkoordinate überführt werden kann. Wird diese Transformation für alle Punkte der Banden und Löcher im Modellkoordinatensystem gemacht, können die entstandenen Pixelpunkte in ein Bild eingezeichnet werden. Somit kann eine binäre Maske erzeugt werden, welche lediglich das Spielfeld umfasst und damit den Rand des Tisches und die Löcher aus einem Bild entfernen kann. In Abbildung 4.14 wurde diese Maske nach Anwendung auf das Eingabebild aus Abbildung 4.13 dargestellt.

In Snooker gibt es 22 Billiardkugeln, wovon 15 rot, und je eine weiss, schwarz, braun, gelb, grün, blau und pink sind.

Die Erkennung wurde wie folgt aufgeteilt:

- Farbige Kugeln (rot, braun, gelb, grün, blau, pink)
- Weisse und pinke Kugel
- Schwarze Kugel

Die Segmentation der farbigen Kugeln erfolgt auf dem Sättigungskanal und diejenige der weissen und schwarzen Kugeln auf dem Helligkeitskanal des Bildes. Eine Segmentation auf dem Farbwertskanal ist insbesondere für die grüne und blaue Kugel schwierig, weil der Billardfilz grün ist und dadurch die Grenze zwischen grünem Tisch und grüner Kugel zu wenig markant ist, siehe Abbildung 4.15.

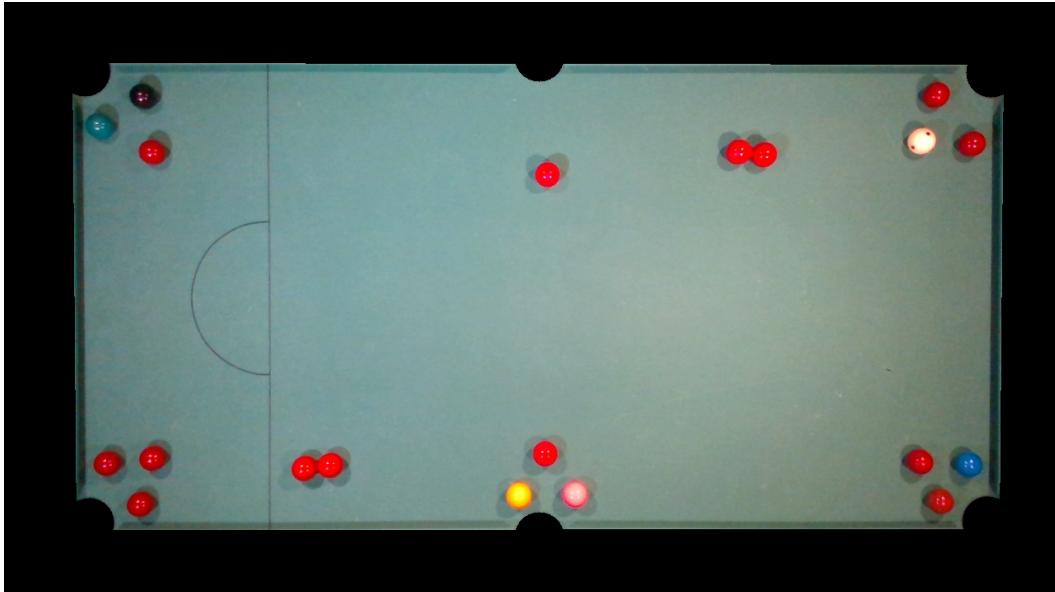


Abbildung 4.14: Spielfeld-Maske. Zur Visualisierung auf Abbildung 4.13 angewendet.

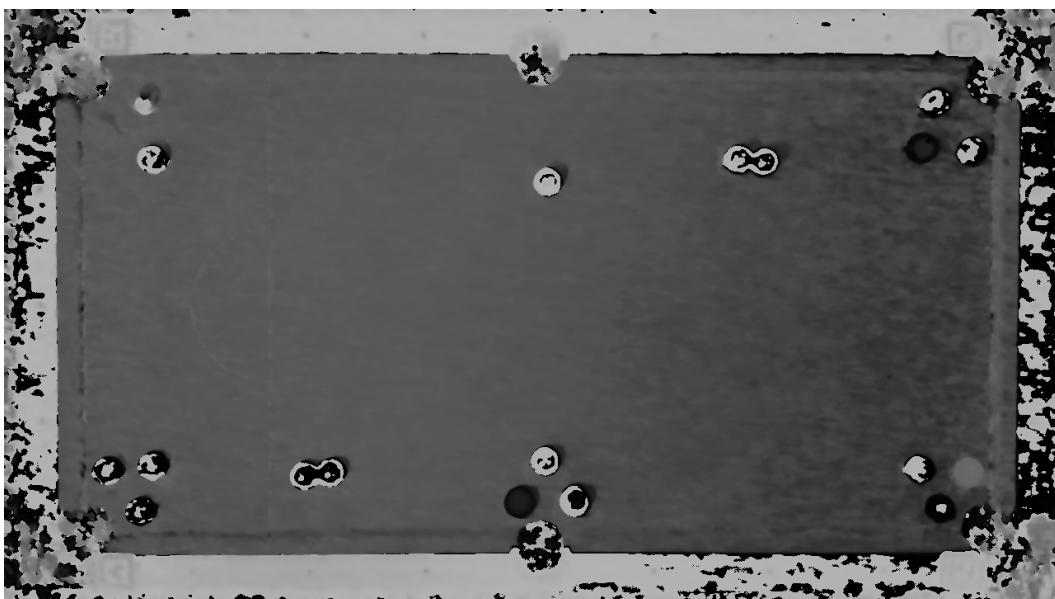


Abbildung 4.15: Der Farbwertskanal. Sehen Sie die grüne Kugel?

Die pinke Kugel ist insofern ein Spezialfall, als dass sie je nach Ausleuchtung eine stark gesättigte Farbe oder eine grosse Helligkeit, ähnlich der weissen Kugel, aufweist. Das bedeutet, dass sie potentiell sowohl aufgrund der Sättigung als auch der Helligkeit segmentiert wird und dadurch doppelt detektiert wird. Dies gilt es zu verhindern.

Für die Klassifikation der Kugeln, also die Bestimmung der Farbe der Kugel, ist der Farbwert trotzdem sehr sinnvoll. Die Klassifikation wurde allerdings im Rahmen dieser Arbeit nicht vorgenommen.

Für alle nachfolgend beschriebenen morphologischen Operationen wird ein rechteckiges structuring element der Grösse 3x3 Pixel verwendet. Ausserdem ist der gesuchte Kugelradius in Pixel, welcher an den Hough transform übergeben wird, anhand des Kugelradius in Millimeter berechnet, siehe dazu Kapitel 4.4.6.

Für die farbigen Kugeln mit hoher Sättigung wird ein Filter auf dem Sättigungskanal des HSV-Bilds angewendet, um eine binäre Maske zu erhalten.

1. Aufbau einer binären Maske mittels Filterung auf dem Sättigungskanal des HSV-Bilds.
2. Morphologisches Opening der binären Maske, um kleine Störpixel zu entfernen.

3. Morphologisches Closing der binären Maske, um kleine Löcher zu füllen.
4. Anwendung der binären Maske auf das Graustufenbild, um alle unerwünschten Kugeln auszuschliessen.
5. Canny edge detection [Can86] auf dem maskierten Graustufenbild, um ein Kantenbild zu erhalten.
6. Hough transform auf dem Kantenbild, um Kreise zu erhalten.
7. Verwerfen der Kreise mit Mittelpunkten ausserhalb der Spielfeld-Maske.
8. Verwerfen der Kreise mit Mittelpunkten ausserhalb der binären Maske.
9. Verwerfen der Kreise mit Mittelpunkten innerhalb der binären Maske der schwarzen Kugel.

Die schwarze Kugel weisst eine geringe Sättigung und kann nicht aufgrund des Farbwerts segmentiert werden. Das zuverlässigste ist demnach eine Filterung auf dem Helligkeitswert. Die einzelnen Schritte der Detektion der schwarzen Kugel sind nachfolgend beschrieben:

1. Aufbau einer binären Maske mittels Filterung auf dem Helligkeitskanal des HSV-Bilds.
2. UND-Verknüpfung der binären Maske mit der binären Spielfeld-Maske, um die Löcher und den Rest des Tisches auszuschliessen.
3. Morphologisches Closing der binären Maske, um kleine Löcher zu füllen.
4. Morphologisches Opening der binären Maske, um kleine Störpixel zu entfernen.
5. Anwendung der binären Maske auf das Graustufenbild, um alle unerwünschten Kugeln auszuschliessen.
6. Canny edge detection auf dem maskierten Graustufenbild, um ein Kantenbild zu erhalten.
7. Hough transform auf dem Kantenbild, um Kreise zu erhalten.
8. Verwerfen der Kreise mit Mittelpunkten ausserhalb der Spielfeld-Maske.
9. Verwerfen der Kreise mit Mittelpunkten ausserhalb der binären Maske.

Die weisse Kugel weisst einen hohe Helligkeitswert auf, welcher zuverlässiger als der Sättigungswert gefiltert werden kann. Nachfolgend ist der Ablauf der Detektion der weissen Kugel beschrieben.

1. Aufbau einer binären Maske mittels Filterung auf dem Helligkeitskanal des HSV-Bilds.
2. Morphologisches Opening der binären Maske, um kleine Störpixel zu entfernen.
3. Morphologisches Closing der binären Maske, um kleine Löcher zu füllen.
4. Anwendung der binären Maske auf das Graustufenbild, um alle unerwünschten Kugeln auszuschliessen.
5. Morphologisches Closing des maskierten Graustufenbilds, da dadurch die roten Punkte, welche auf der weissen Kugel aufgemalt sind, schwächer werden.
6. Canny edge detection auf dem maskierten Graustufenbild, um ein Kantenbild zu erhalten.
7. Hough transform auf dem Kantenbild, um Kreise zu erhalten.
8. Verwerfen der Kreise mit Mittelpunkten ausserhalb der Spielfeld-Maske.
9. Verwerfen der Kreise mit Mittelpunkten ausserhalb der binären Maske.
10. Verwerfen der Kreise mit Mittelpunkten innerhalb der binären Maske der farbigen Kugeln, um doppelt detektierte Kugeln, wie etwa die Pinke, zu vermeiden.

Mithilfe der Maske für die weisse Kugel wird teilweise auch die pinke Kugel detektiert, sofern diese eine geringe Sättigung und hohe Helligkeit aufweist.

Von den übriggebliebenen Kreisen werden die Mittelpunkte extrahiert und dienen als Resultat der Detektion. Die detektierten Kreisradien sind irrelevant, da der Durchmesser der Kugeln bekannt ist.

Das Resultat der Zusammenführung der detektierten Kreise ist in Abbildung 4.16 dargestellt.

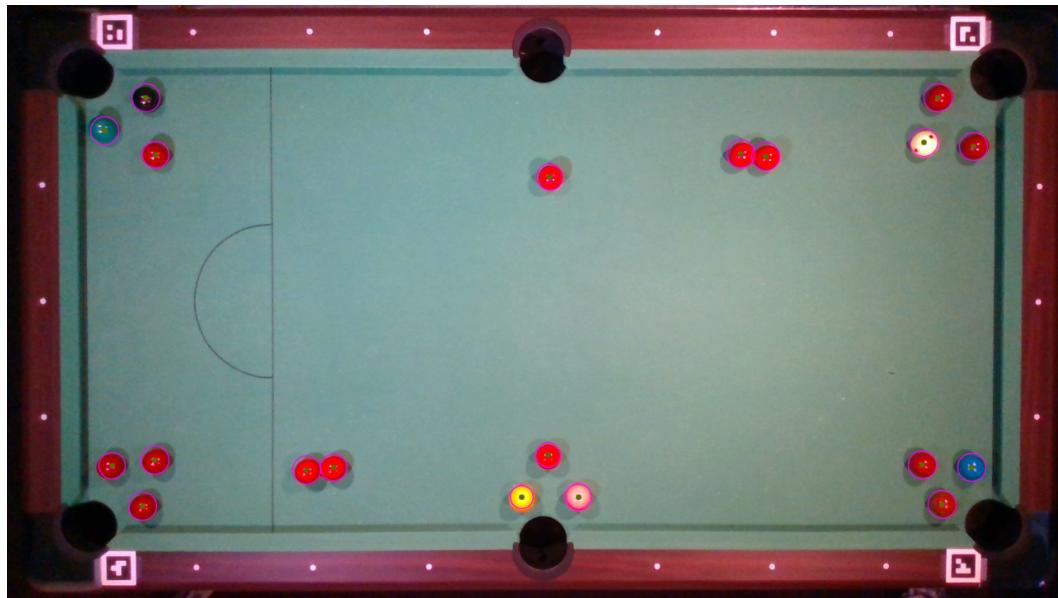


Abbildung 4.16: Resultat der Detektion. Die Kreismittelpunkte und -umkreise sind eingezeichnet.

4.6 Visualisierung in Unity

Dem Benutzer soll über eine geeignete Möglichkeit der aktuelle Spielstand wie auch die Suchresultate visualisiert werden. Dazu wird wie bereits im Kapitel 4.3 erwähnt, eine Applikation mithilfe der Engine Unity erstellt. Die Applikation erfüllt mehrere Ziele:

Darstellung des aktuellen Spielstands Der aktuelle Spielstand wird dem Benutzer präsentiert. Es besteht die Möglichkeit, Kugeln auszuwählen und eine Suche über diese zu starten.

Darstellung des Suchergebnisses Sobald eine Suche durchgeführt wurde und Ergebnisse vorliegen, können diese über eine Animation angezeigt werden.

Erstellen neuer Spielstände Als Zusatz wird eine Möglichkeit geschaffen, um Spielstände manuell zu generieren. Dies wird ausschliesslich während der Entwicklung verwendet.

Die Visualisierung des aktuellen Spielstands kann in der Abbildung 4.17 entnommen werden. Die Kugeln können ein- wie auch ausgeblendet werden. Dies ist ebenso mit dem Tisch möglich, der standardmässig nicht angezeigt wird. In der Abbildung ist weiterhin ersichtlich, dass die Positionen der Kugeln angezeigt werden können. Auch diese Information ist nur für die Entwicklung relevant.



Abbildung 4.17: Unity - Spielstand

5 Resultate

Das Kapitel beinhaltet die Beschreibung der Genauigkeiten der Messresultate wie auch der Kugeldetektion.

5.1 Detektionsgenauigkeit

Ein Mass für die Genauigkeit der Detektion bildet die absolute Differenz zwischen der erwarteten wie auch der effektiv detektierten Anzahl der Kugeln. Hierbei werden die Positionen nicht berücksichtigt, die Erkennungsgenauigkeit dieser wird im Kapitel 5.2 beschrieben und millimetergenau bestimmt. Dazu wurden diverse Testbilder¹ aufgenommen.

Bild	Anzahl erwartete Kugeln	Anzahl detektierte Kugeln	Absoluter Fehler
8.4	10	9	10 - 9 = 1

Tabelle 5.1: Detektionsgenauigkeit

Der durchschnittliche Fehler beträgt: ? [Kugeln]

5.2 Übersetzungsgenauigkeit der erkannten Kugeln

Es gilt zwei Aspekte zu beachten. Einerseits soll die Genauigkeit der Übersetzung von Pixel- zu Modellkoordinatensystem, andererseits auch die Genauigkeit der Detektion eines Kugelmittelpunkts bestimmt werden. Um Letzteres zu erreichen, muss Ersteres bekannt sein. Ist bekannt, welcher Fehler bei der Übersetzung geschieht, kann erkannt werden, wie gross der Fehlerbeitrag der Detektion des Kugelmittelpunkts ist. Dies ist wichtig, um bei grösseren Abweichungen von der erwarteten Fehlertoleranz am richtigen Ort anzusetzen. Die Gesamttoleranz der beiden möglichen Fehlerquellen liegt bei 3mm^2 . Diese Fehlertoleranz $F(x, y)$ beschreibt einen Radius um den erwarteten Kugelmittelpunkt. Der Radius setzt sich aus dem Fehler in X- wie auch in Y-Richtung zusammen. Er wird also wie in Gleichung 5.1 beschrieben.

$$F(x, y) = \sqrt{x^2 + y^2}|F(x, y) \leq 2 \quad (5.1)$$

Zusätzlich muss noch der Fehler des Messgeräts berücksichtigt werden, welcher mit $\pm 1.5\text{mm}$ angegeben wird. Da dieser Fehler sowohl in X- wie auch Y-Richtung vorkommen kann, liegt der maximale Messfehler bei $\sqrt{1.5^2 + 1.5^2} = 2.12\text{mm}$. Somit liegt der Gesamtfehlerradius bei 5.12mm , wobei allfällige Messfehler von Hand nicht berücksichtigt werden. Wichtig zu beachten ist ebenfalls, dass der Kugelmittelpunkt von der Core-Library subpixelgenau bestimmt wird. Die Grundwahrheit wiederum wird in ganzen Pixel angegeben. So kommt eine weitere Abweichung dazu, welche maximal 0.983 mm beträgt³, sollte das erkannte Subpixel zum Grundwahrheitspixel gehören. Diese Abweichung ist nun aber kein Fehler der Detektion, sondern der Grundwahrheit an sich. Der Fehler könnte also um diesen Maximalwert reduziert werden. Darauf wird aber im Folgenden verzichtet, da lediglich der Worst-Case betrachtet wird. Es wird also angenommen, dass die Grundwahrheit genauer ist, als die Detektion.

Um die Genauigkeit zu messen, wurden mehrere Testbilder⁴ erstellt. Auf diesen ist jeweils eine Kugel ersichtlich, von der wie in Kapitel 4.2 beschrieben die Modellkoordinate bestimmt wird. Diese Messung dient als Wahrheit zur Verifikation der Berechnungen.

In einem ersten Schritt muss also die Genauigkeit der Umrechnung der Koordinatensysteme bestimmt werden. Dazu wird manuell der erwartete Kugelmittelpunkt in Pixelkoordinaten pro Testbild bestimmt. Dieser Punkt wird anschliessend umgerechnet und es wird der absolute Fehlerwert in Millimeter angegeben.

¹ Alle Testbilder sind im Anhang 8.4.1 ersichtlich

² Für die Auswirkung des Fehlerradius siehe Anhang 8.2

³ Für Herleitung siehe Anhang 8.3

⁴ Alle Testbilder sind im Anhang 8.4.2 ersichtlich

Bild	Pixelkoordinaten	Erwartete Modellkoordinaten	Detektierte Modellkoordinaten	Absoluter Fehler
8.5 8	[629, 743]	[-374.85, -216.15]	[-373.308, -214.712]	[-1.542, -1.438], 2.108mm
8.6 9	[814, 287]	[-180.85, 283.15]	[-178.844, 281.705]	[-2.006, 1.445], 2.472mm
8.6 10	[1273, 413]	[337.85, 142.15]	[336.345, 140.874]	[1.505, 1.276], 1.973mm
8.6 11	[1561, 621]	[664.15, -91.85]	[661.979, -93.1876]	[2.171, 1.3376], 2.549mm
8.6 12	[475, 639]	[-560.85, -112.85]	[-561.435, -112.964]	[0.585, 0.114], 0.596mm
8.7 13	[980, 537]	[5.85, 0.15]	[6.50545, 1.31031]	[-0.655, -1.160], 1.332mm
8.7 14	[1640, 225]	[751.15, 350.15]	[750.001, 352.548]	[1.149, -2.398], 2.659mm

Tabelle 5.2: Messresultate Koordinatensystem

Der durchschnittliche Messfehler beträgt: 1.956 [mm]

In einem zweiten Schritt wird nun noch die Detektion der Kugeln miteinbezogen. Es wird also nicht mehr manuell der Kugelmittelpunkt bestimmt, dies übernimmt nun ebenfalls die Core-Library. Der Fehler wird nun nochmals gleich berechnet wie bei Tabelle 5.2 unter Verwendung derselben Testbilder (siehe Tabelle 5.3, Spalte „Gesamter absoluter Fehler“). Abschliessend kann der Fehler der Detektion in Millimeter über die absolute Differenz des Fehlerwerts ohne Detektion und des Fehlerwerts mit Detektion bestimmt werden (siehe Tabelle 5.3, Spalte „Absoluter Fehler Detektion“).

Bild	Erwartete Modellkoordinaten	Detektierte Modellkoordinaten	Gesamter absoluter Fehler	Absoluter Fehler Detektion
8.5	[13, 13]	[9, 9]	[4, 4], 5.65mm	5.65mm - 4.24mm = 1.41mm

Tabelle 5.3: Messresultate Detektion

Der durchschnittliche Fehler der Detektion beträgt: ? [mm]

Der durchschnittliche Gesamtfehler beträgt: ? [mm]

6 Weitere Arbeiten

7 Fazit

Abbildungsverzeichnis

4.1 Baugerüst	7
4.2 Messung - Tisch	8
4.3 Messung - Kugel	9
4.4 Grobarchitektur - Applikationsumgebung	9
4.5 Modellkoordinatensystem	10
4.6 Bildkoordinatensystem. Bildquelle: [Sat20]	11
4.7 Verwendete ArUco markers	12
4.8 Billiardtisch mit ArUco markers	12
4.9 Weltkoordinatensystem	13
4.10 Welt-Ursprung W und Banden-Ursprung R	14
4.11 Banden-Ursprung B und Modell-Ursprung R	15
4.12 Länge und Breite des Spielbereits	15
4.13 Eingabebild mit Schatten	17
4.14 Spielfeld-Maske. Zur Visualisierung auf Abbildung 4.13 angewendet.	18
4.15 Der Farbwertskanal. Sehen Sie die grüne Kugel?	18
4.16 Resultat der Detektion. Die Kreismittelpunkte und -umkreise sind eingezeichnet.	20
4.17 Unity - Spielstand	20
8.1 Fehlerwinkel	37
8.2 Fehlerwinkel vereinfacht	38
8.3 Abweichung über Distanz	39
8.4 Detektionsgenauigkeit - Testbild 1	41
8.5 Übersetzungsgenauigkeit - Testbild 1	41
8.6 Übersetzungsgenauigkeit - Testbild 2	41
8.7 Übersetzungsgenauigkeit - Testbild 3	41

Tabellenverzeichnis

3.1 Ziele	6
5.1 Detektionsgenauigkeit	21
5.2 Messresultate Koordinatensystem	22
5.3 Messresultate Detektion	22

Literatur

- [Can86] J. Canny. "A Computational Approach to Edge Detection". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-8.6* (1986), S. 679–698. doi: 10.1109/TPAMI.1986.4767851.
- [Kyl12] Kyle Simek. *Dissecting the Camera Matrix, Part 2: The Extrinsic Matrix*. [Online; accessed 20.05.2021]. 2012. URL: [https://ksimek.github.io/2012/08/22/extrinsics/#:~:text=The%20Extrinsic%20Camera%20Matrix,the%20modelview%20matrix%22\)..\n](https://ksimek.github.io/2012/08/22/extrinsics/#:~:text=The%20Extrinsic%20Camera%20Matrix,the%20modelview%20matrix%22)..\n)
- [ope21a] opencv.org. *OpenCV - Detection of ArUco Boards*. [Online; accessed 20.05.2021]. 2021. URL: https://docs.opencv.org/4.5.2/db/da9/tutorial_aruco_board_detection.html.
- [ope21b] opencv.org. *OpenCV - Funktion projectPoints*. [Online; accessed 23.05.2021]. 2021. URL: https://docs.opencv.org/4.5.2/d9/d0c/group__calib3d.html#ga1019495a2c8d1743ed5cc23fa0daff8c.
- [Sat20] Satya Mallick. *LearnOpenCV - Geometry of Image Formation*. [Online; accessed 23.05.2021]. 2020. URL: <https://learnopencv.com/geometry-of-image-formation/>.
- [Unk21a] Unknown. *Bosch - GLM 40 Professional Laser-Entfernungsmesser*. [Online; accessed 23.05.2021]. 2021. URL: <https://www.bosch-professional.com/ch/de/products/glm-40-0601072902>.
- [Unk21b] Unknown. *Intel RealSense Depth Camera D435*. [Online; accessed 20.05.2021]. 2021. URL: <https://www.intelrealsense.com/depth-camera-d435>.
- [Unk21c] Unknown. *Intel RealSense Depth Camera D435 Datasheet*. [Online; accessed 20.05.2021]. 2021. URL: <https://www.intelrealsense.com/wp-content/uploads/2020/06/Intel-RealSense-D400-Series-Datasheet-June-2020.pdf>.
- [Unk21d] Unknown. *Intel RealSense SDK 2.0 Github*. [Online; accessed 20.05.2021]. 2021. URL: <https://github.com/IntelRealSense/librealsense>.
- [Unk21e] Unknown. *Matlab - What is Camera Calibration?* [Online; accessed 20.05.2021]. 2021. URL: <https://ch.mathworks.com/help/vision/ug/camera-calibration.html>.
- [Unk21f] Unknown. *OmniVision OV2740*. [Online; accessed 20.05.2021]. 2021. URL: <https://www.ovt.com/sensors/OV2740>.
- [Unk21g] Unknown. *Projector Central - BenQ MW820ST Projector*. [Online; accessed 23.05.2021]. 2021. URL: <https://www.projectorcentral.com/BenQ-MW820ST.htm>.
- [Wik21a] Wikipedia. *Wikipedia - Circle Hough Transform*. [Online; accessed 22.05.2021]. 2021. URL: https://en.wikipedia.org/wiki/Circle_Hough_Transform.
- [Wik21b] Wikipedia. *Wikipedia - HSV-Farbraum*. [Online; accessed 22.05.2021]. 2021. URL: <https://de.wikipedia.org/wiki/HSV-Farbraum>.

Glossar

Versionskontrolle

Version	Datum	Beschreibung	Autor
0.1	01.03.2021	Eröffnung Dokumentation	Luca Ritz
0.2	04.03.2021	Definition Ziele	Lukas Seglias & Luca Ritz

8 Anhang

8.1 Messgeräte

Es werden die verwendeten Messgeräte und alle bekannten Spezifikationen erläutert.

8.1.1 Messschieber

TODO: Messgerät beschreiben

8.1.2 Lasermessgerät

Als Messgerät für Distanzen bis zu maximal 2m wurde ein Bosch GLM 40 Professional, siehe [Unk21a], verwendet. Dieses weist eine Messgenauigkeit von $\pm 1.5\text{mm}$ auf.

8.2 Fehlerradius

Der maximal zulässige Fehlerradius β wird als Winkel zwischen der optimalen sowie der fehlerhaften Bahn der Kugel definiert, welche eingenommen wird, wenn der Mittelpunkt falsch erkannt oder umgerechnet wurde. Effektiv wird β als Annäherung berechnet, da der Punkt P' nicht der Mittelpunkt der Kugel zum Auftrittszeitpunkt darstellt. Dazu müsste in einem ersten Schritt P'' berechnet werden wie in Abbildung 8.1 ersichtlich wird. Weiterhin kann erkannt werden, dass dieser Winkel anscheinend maximal wird, wenn der falsch erkannte Mittelpunkt M' orthogonal zur optimalen Bahn liegt, welche wiederum über den Zielpunkt T und den wirklichen Kugelmittelpunkt M gegeben ist.

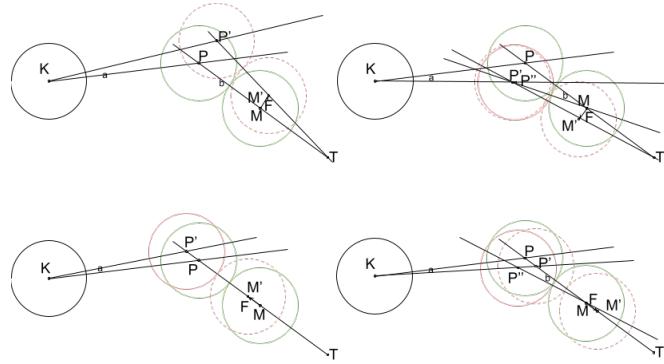


Abbildung 8.1: Fehlerwinkel

Wichtig zu beachten ist auch, dass wenn der Winkel $\beta = 0$ ist, dies nicht bedeutet, dass kein Fehler aufgetreten ist. In dem Fall wird die Kugel entweder nicht getroffen (Fehler) oder sie wird korrekt getroffen (kein Fehler).

Es folgen einige Definitionen, die für die Herleitung von Relevanz sind.

$$T(T_x, T_y), K(K_x, K_y), M(M_x, M_y), M'(M'_x, M'_y), F(F_x, F_y) \quad (8.1)$$

$$\vec{TM} = \begin{pmatrix} M_x - T_x \\ M_y - T_y \end{pmatrix}, \vec{TM'} = \begin{pmatrix} M'_x - T_x \\ M'_y - T_y \end{pmatrix} \quad (8.2)$$

$$\vec{TM}_0 = \frac{\vec{TM}}{|\vec{TM}|} \quad (8.3)$$

$$\vec{TM}'_0 = \frac{\vec{TM}'}{|\vec{TM}'|} \quad (8.4)$$

$$f(\lambda) = \vec{OM} + \lambda \cdot \vec{TM}_0 \quad (8.5)$$

$$f(\lambda)' = \vec{OM}' + \lambda \cdot \vec{TM}'_0 \quad (8.6)$$

$$P = f(2R) \quad (8.7)$$

$$P' = f(2R)' \quad (8.8)$$

Der Winkel β ist also durch Formel 8.9 gegeben.

$$\cos(\beta) = \frac{\vec{MP} \cdot \vec{MP}'}{|\vec{MP}| \cdot |\vec{MP}'|} \quad (8.9)$$

Um die maximale Fehlerdistanz zu bestimmen, wird angenommen, dass eine orthogonale Verschiebung des Kugelmittelpunkts das Worst-Case-Szenario bildet. Dies wird in Abbildung 8.2 verdeutlicht. Es ist nochmals ersichtlich, dass der berechnete Fehlerwinkel β nur eine Annäherung darstellt, da der effektive Winkel ein wenig grösser sein wird. M' kann nun mittels dem

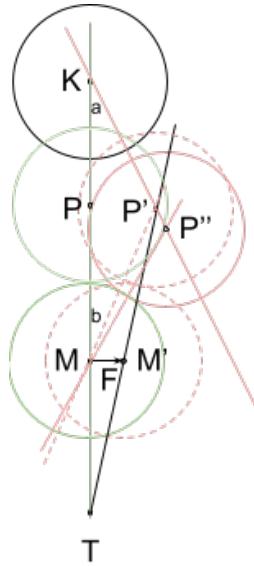


Abbildung 8.2: Fehlerwinkel vereinfacht

Fehlervektor \vec{F} und M ausgedrückt werden.

$$\vec{M}' = \vec{OM} + \vec{F} \quad (8.10)$$

$$\vec{M}' = \begin{pmatrix} M_x + F_x \\ M_y + F_y \end{pmatrix} \quad (8.11)$$

Der Punkt P' wird über eine Geradengleichung ausgedrückt, um ihn direkt über den Fehlervektor zu definieren.

$$f(\lambda)' = \vec{OM}' + \lambda \cdot \vec{TM}'_0 \quad (8.12)$$

$$\vec{TM}' = \begin{pmatrix} M_x + F_x - T_x \\ M_y + F_y - T_y \end{pmatrix} \quad (8.13)$$

$$\vec{TM}'_0 = \begin{pmatrix} \frac{M_x + F_x - T_x}{\sqrt{(M_x + F_x - T_x)^2 + (M_y + F_y - T_y)^2}} \\ \frac{M_y + F_y - T_y}{\sqrt{(M_x + F_x - T_x)^2 + (M_y + F_y - T_y)^2}} \end{pmatrix} \quad (8.14)$$

$$P' = f(2R)' = \begin{pmatrix} M_x + F_x \\ M_y + F_y \end{pmatrix} + 2R \cdot T \vec{M}'_0 \quad (8.15)$$

Der Fehlerwinkel β wird nun zwischen den Vektoren \vec{MP} und \vec{MP}' berechnet.

$$\vec{MP} = \begin{pmatrix} M_x + 2R \cdot \frac{M_x - T_x}{\sqrt{(M_x - T_x)^2 + (M_y - T_y)^2}} - M_x \\ M_y + 2R \cdot \frac{M_y - T_y}{\sqrt{(M_x - T_x)^2 + (M_y - T_y)^2}} - M_y \end{pmatrix} \quad (8.16)$$

$$\vec{MP}' = \begin{pmatrix} 2R \cdot \frac{M_x - T_x}{\sqrt{(M_x + F_x - T_x)^2 + (M_y + F_y - T_y)^2}} \\ 2R \cdot \frac{M_y - T_y}{\sqrt{(M_x + F_y - T_x)^2 + (M_y + F_y - T_y)^2}} \end{pmatrix} \quad (8.17)$$

$$\vec{MP}' = \begin{pmatrix} (M_x + F_x) + 2R \cdot \frac{M_x + F_x - T_x}{\sqrt{(M_x + F_x - T_x)^2 + (M_y + F_y - T_y)^2}} - M_x \\ (M_y + F_y) + 2R \cdot \frac{M_y + F_y - T_y}{\sqrt{(M_x + F_y - T_x)^2 + (M_y + F_y - T_y)^2}} - M_y \end{pmatrix} \quad (8.18)$$

$$\vec{MP}' = \begin{pmatrix} F_x + 2R \cdot \frac{M_x + F_x - T_x}{\sqrt{(M_x + F_x - T_x)^2 + (M_y + F_y - T_y)^2}} \\ F_y + 2R \cdot \frac{M_y + F_y - T_y}{\sqrt{(M_x + F_y - T_x)^2 + (M_y + F_y - T_y)^2}} \end{pmatrix} \quad (8.19)$$

Mittels 8.17 und 8.19 kann nun der Fehlerwinkel definiert werden. Der Vektor \vec{f} steht dabei für den Fehlervektor, welcher zum effektiven Kugelmittelpunkt addiert werden muss. Er wird bei der Lokalisierung von MP' verwendet.

$$\alpha(\vec{f}) = \arccos\left(\frac{\vec{MP} \cdot \vec{MP}'}{|\vec{MP}| \cdot |\vec{MP}'|}\right) \quad (8.20)$$

Um die Auswirkung eines Fehlervektors der Länge 3mm darzustellen, wird die vereinfachte Situation aus Abbildung 8.2 vorausgesetzt. Weiterhin wird verdeutlicht, wie sich die Abweichung d über eine Distanz D und einen Winkel β zusammensetzt. Die Abbildung 8.3 zeigt die Ausgangslage, wobei D und β bekannt sind. Bei d handelt es sich um die Gegenkathete, bei D

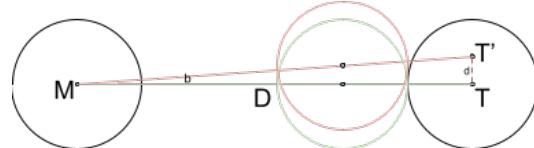


Abbildung 8.3: Abweichung über Distanz

um die Ankathete. Daher gelten die Gleichungen 8.23 und 8.26.

$$\cos(\beta) = \frac{D}{Hyp} \quad (8.21)$$

$$\cos(\beta) \cdot Hyp = D \quad (8.22)$$

$$Hyp = \frac{D}{\cos(\beta)} \quad (8.23)$$

$$\sin(\beta) = \frac{d}{Hyp} \quad (8.24)$$

$$\sin(\beta) \cdot Hyp = d \quad (8.25)$$

$$Hyp = \frac{d}{\sin(\beta)} \quad (8.26)$$

Die Formeln 8.23 und 8.26 können nun gleichgesetzt und nach d aufgelöst werden. Siehe Formel 8.28.

$$\frac{D}{\cos(\beta)} = \frac{d}{\sin(\beta)} \quad (8.27)$$

$$\frac{D \cdot \sin(\beta)}{\cos(\beta)} = d \quad (8.28)$$

Für eine Abweichung um $\vec{f} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$ auf ein Ziel T der Entfernung $T = \begin{pmatrix} 0 \\ -1800 \end{pmatrix}$ ergibt sich ein Winkel β von 3.378 deg. Dies

ergibt eine Gesamtabweichung von $\frac{\sin(3.378) \cdot 1800 \text{ mm}}{\cos(3.378)} = 106.251 \text{ mm}$. Es wird davon ausgegangen, dass die Distanzen, welche eine Kugel zurücklegen, tendenziell kleiner sein werden, da Stöße über grössere Distanzen schwieriger sind. Daher wird die Distanz auf maximal die halbe Tischlänge gekürzt und die Abweichung sinkt auf 27.31mm.

Es wird nun eine Länge des Fehlervektors gesucht, der auf eine Distanz von 900mm eine Abweichung von 2mm zulässt. Der Winkel ist demnach ca. 0.1273 deg. Bei einer Fehlervektorlänge von 0.1mm ergeben sich 1.82mm Abweichung zur optimalen Bahn.

8.3 Fehler - Grundwahrheit

Als Grundwahrheit zur Bestimmung der Gesamtabweichung in [mm] dient ein manuell annotiertes Testbild. Auf diesem wird also von einer Person der Kugelmittelpunkt in ganzen Pixeln eruiert. Die Grundwahrheit ist gegeben durch 8.29.

$$G = (G_x, G_y) \quad (8.29)$$

Weiterhin ist bekannt, wie vielen Millimetern ein Pixel entspricht. Diese Information ist einerseits über die Masse des Tisches T wie auch über die Auflösung des Bildes B gegeben.

$$T = (T_x, T_y) [\text{mm}] \quad (8.30)$$

$$B = (B_x, B_y) [\text{pixel}] \quad (8.31)$$

$$TB = (T_x/B_x, T_y/B_y) \quad (8.32)$$

Unter der Annahme, dass bei einem Pixel das Zentrum angegeben wird, kann die Abweichung der Detektion D wie in 8.34 dargestellt werden.

$$D = (D_x, D_y) [\text{subpixel}] \quad (8.33)$$

$$\Delta = TB - D \quad (8.34)$$

Die Informationen T wie auch B sind gegeben.

$$T = (1881, 943) [\text{mm}] \quad (8.35)$$

$$B = (1280, 720) [\text{pixel}] \quad (8.36)$$

$$TB = (1.469, 1.309) \left[\frac{\text{mm}}{\text{pixel}} \right] \quad (8.37)$$

Unter der Annahme, dass eine Pixelkoordinate dem Zentrum des Pixels entspricht, der Kugelmittelpunkt subpixelgenau erkannt wird und als Grundwahrheit das korrekte Pixel angegeben wurde, kann die maximale Abweichung Δ_{\max} wie in 8.38 angegeben werden.

$$\Delta_{\max} = \frac{1 [\text{pixel}]}{2} \cdot TB \quad (8.38)$$

$$\Delta_{\max} = (0.7345, 0.654) [\text{mm}] \quad (8.39)$$

$$|\Delta_{\max}| = \sqrt{0.7345^2 + 0.654^2} = 0.983 \text{ mm} \quad (8.40)$$

8.4 Testbilder

8.4.1 Detektionstestbilder

TODO: Add testbilder

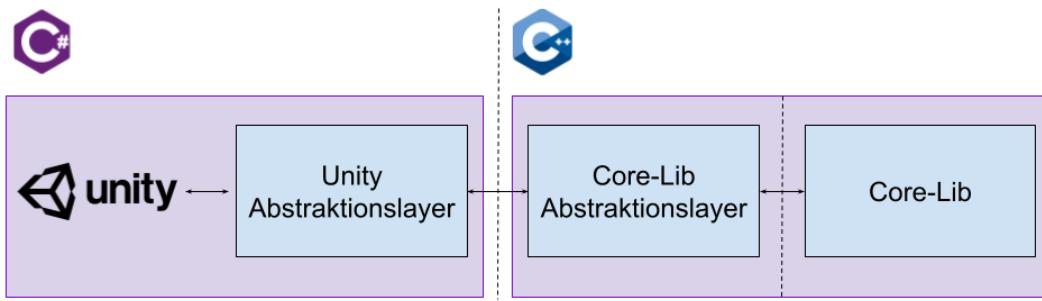


Abbildung 8.4: Detektionsgenauigkeit - Testbild 1

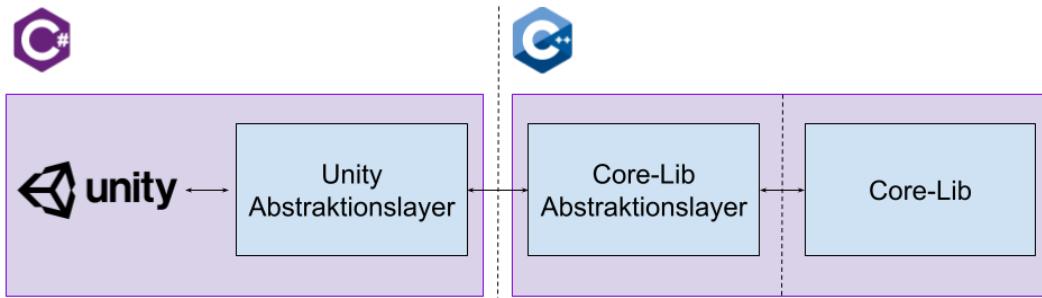


Abbildung 8.5: Übersetzungsgenauigkeit - Testbild 1

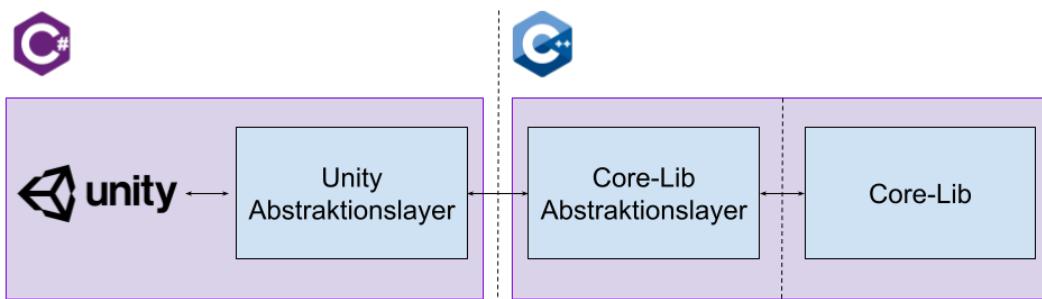


Abbildung 8.6: Übersetzungsgenauigkeit - Testbild 2

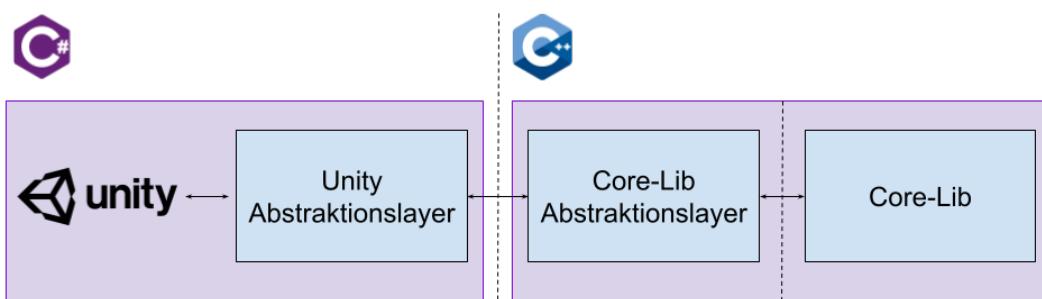


Abbildung 8.7: Übersetzungsgenauigkeit - Testbild 3

8.4.2 Übersetzungsgenauigkeit

TODO: Add testbilder