

# Billiard-AI

Ein intelligenter Billardtisch

**Projekt 2**

Studienrichtung:

Informatik - Computer Perception and Virtual Reality

Autor:

Lukas Seglias, Luca Ritz

Dozent:

Markus Hudritsch

Experte:

Datum:

20. Mai 2021



# Inhaltsverzeichnis

<b>1</b>	<b>Zusammenfassung</b>	<b>1</b>
<b>2</b>	<b>Einführung</b>	<b>3</b>
<b>3</b>	<b>Ziele</b>	<b>5</b>
3.0.1	Planung . . . . .	5
<b>4</b>	<b>Billiard-AI</b>	<b>7</b>
4.1	Aufbau . . . . .	7
4.1.1	Kamera . . . . .	7
4.2	Aufbau und Messung . . . . .	7
4.3	Architektur . . . . .	8
4.4	Modellkoordinatensystem . . . . .	9
4.4.1	Pixelkoordinaten zu Kamerakoordinaten . . . . .	10
4.4.2	Kamerakoordinaten zu Weltkoordinaten . . . . .	10
4.4.3	Weltkoordinaten zu Bandenkoordinaten . . . . .	13
4.4.4	Bandenkoordinaten zu Modellkoordinaten . . . . .	14
<b>5</b>	<b>Resultate</b>	<b>15</b>
5.1	Detektionsgenauigkeit . . . . .	15
5.2	Übersetzungsgenauigkeit der erkannten Kugeln . . . . .	15
<b>6</b>	<b>Weitere Arbeiten</b>	<b>17</b>
<b>7</b>	<b>Fazit</b>	<b>19</b>
<b>8</b>	<b>Anhang</b>	<b>31</b>
8.1	Messgeräte . . . . .	31
8.1.1	Messschieber . . . . .	31
8.1.2	Lasermessgerät . . . . .	31
8.2	Fehlerradius . . . . .	31
8.3	Testbilder . . . . .	34
8.3.1	Detektionstestbilder . . . . .	34
8.3.2	Übersetzungsgenauigkeit . . . . .	34



# 1 Zusammenfassung



## 2 Einführung



# 3 Ziele

Ins Billard-Spiel einzusteigen ist nicht ganz einfach. Zu Beginn kann man schlecht abschätzen, welchen Weg eine Kugel wählen wird, wenn man sie anschlägt und ebenfalls hat man Mühe, den optimalen Stoss zu finden, erst recht, wenn man weiter in die Tiefe sehen will. Wenn es also wichtig wird, die Weisse nach einem Stoss für den Nächsten optimal zu platzieren.

Am Ende wird also ein System entstehen, das einem den optimalen Stoss vorschlägt, basierend auf Kriterien und einer festgelegten Tiefe.

Die Projekt-2 Arbeit hat das grundlegende Ziel der Vorbereitung auf die Bachelor-Thesis. Daher sei zu Beginn erwähnt, dass möglichst viel vorgearbeitet werden kann, auch wenn dies an dieser Stelle keine explizite Erwähnung findet.

Im Wesentlichen geht es aber vor allem um die zugrunde liegenden Basisarbeiten. Diese setzen 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 der Kamera wie auch des Projektors.

**Kalibrierung der Kamera** Bestimme die intrinsische Transformationsmatrix, um genaue Bildanalyse betreiben zu können.

**Erkennung der Kugelpositionen** Die Kugeln sollen einer Position im Pixelkoordinatensystem zugewiesen werden können.

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

**Übersetzung in internes Koordinatensystem** Mittels Marker, welche am Tisch angebracht werden, sollen die auflösungsabhängigen Pixelkoordinaten in ein standardisiertes internes Koordinatensystem überführt 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 direkte Stöße berücksichtigen. Indirekte über die Bande wie auch über andere Kugeln werden vorerst nicht beachtet. Die Ausgabe soll auch nicht unbedingt über den Projektor erfolgen, eine textuelle Präsentation soll hier genügen.

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 geht 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

# 4 Billiard-AI

In diesem Kapitel werden die gewählte Architektur wie auch die bisher erarbeiteten theoretischen Ansätze behandelt. Weiterhin wird auf verschiedene Aspekte des Aufbaus und dessen Problematik eingegangen.

## 4.1 Aufbau

### 4.1.1 Kamera

Als Kamera wurde eine Intel RealSense Depth Camera D435 verwendet [Unk21a]. Die intrinsischen Kameraparameter des Farbsensors wurden mithilfe des RealSense SDK 2.43.0 [Unk21c] ausgelesen. Beim Farbsensor handelt es sich um einen *OmniVision OV2740* [Unk21b]. Die Auflösung des Sensors beträgt 1920x1080 und die Pixelgrösse beträgt  $1.4\mu\text{m}$  [Unk21e].

Nachfolgend ist die allgemeine intrinsische Kameramatrix in *column-major order* aufgeführt. Siehe [Unk21d] 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.

## 4.2 Aufbau und Messung

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. TODO: Bild einfügen

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 4.1, 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 4.1, 2 und 3) wird über ein Laser-Messgerät<sup>1</sup> durchgeführt. Die Messungenauigkeit wird mit  $\pm 1.5\text{mm}$  angegeben. Die Abbildung 4.1 zeigt die Messung in X-Richtung, diese Messung wird ebenfalls in gleicher Art und Weise in Y-Richtung durchgeführt.

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 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 übersetzt<sup>2</sup>. Bei diesem befindet sich der Ursprung in der Mitte des

<sup>1</sup>Die genauen Spezifikationen sind im Anhang 8.1 ersichtlich.

<sup>2</sup>Siehe Kapitel 4.4

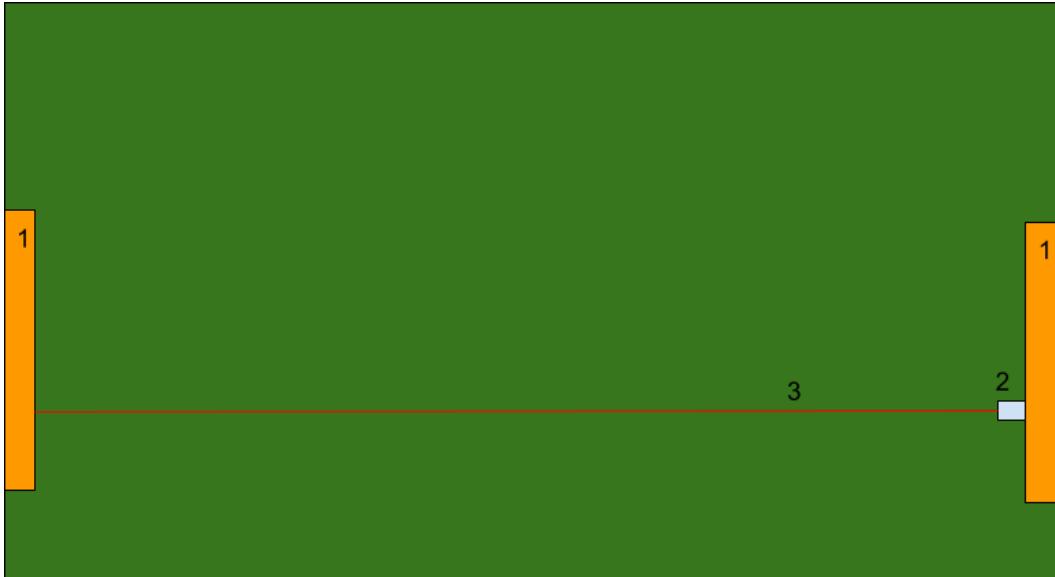


Abbildung 4.1: Messung - Tisch

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, dazuaddiert 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.

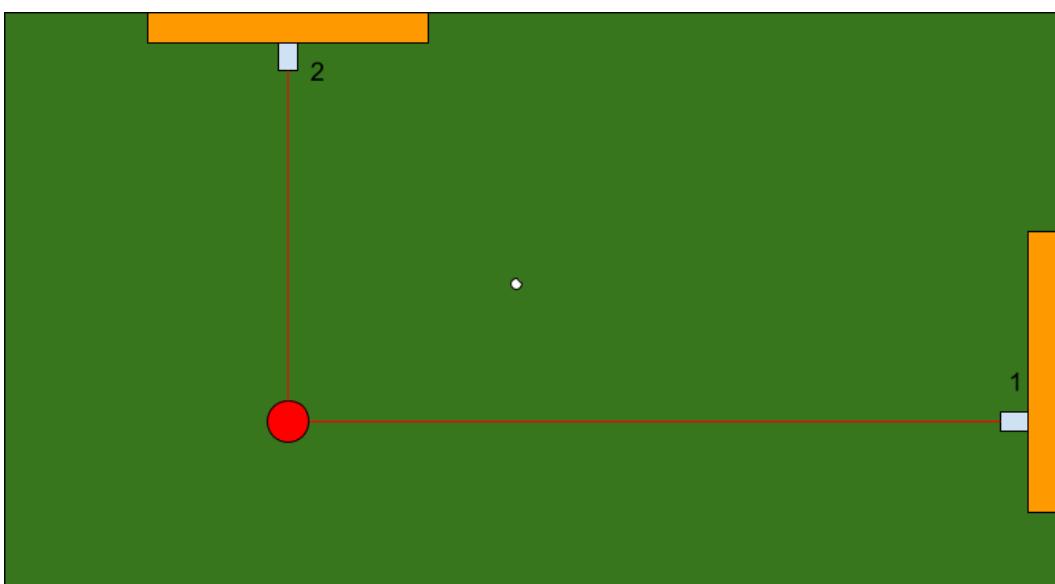


Abbildung 4.2: Messung - Kugel

### 4.3 Architektur

Die eigentliche Funktionalität wird in einer Core-Library untergebracht, welche nativ in C++ entwickelt wird. Das Endprodukt soll aber weitaus mehr zu bieten haben, wie aus den Zielen ersichtlich wird. Deswegen wird in Unity eine Interaktionsmöglichkeit geschaffen, worüber der Benutzer einerseits die Resultate visualisiert erhält und andererseits der Core-Library seine nächsten Schritte mitteilen kann. Um dies zu erreichen, wird eine weitere native Komponente erstellt, welche die Interaktion zwischen

Unity und der Core-Library ermöglicht. Es ist dies eine C/C++-Bibliothek, die ein C-Interface bereitstellt, welches in Unity geladen wird. Unity selbst erhält ebenfalls eine Abstraktionsschicht, um die Nativen auf die Applikationsmodelle zu mappen. Eine Darstellung kann in Abbildung 4.3 entnommen werden.

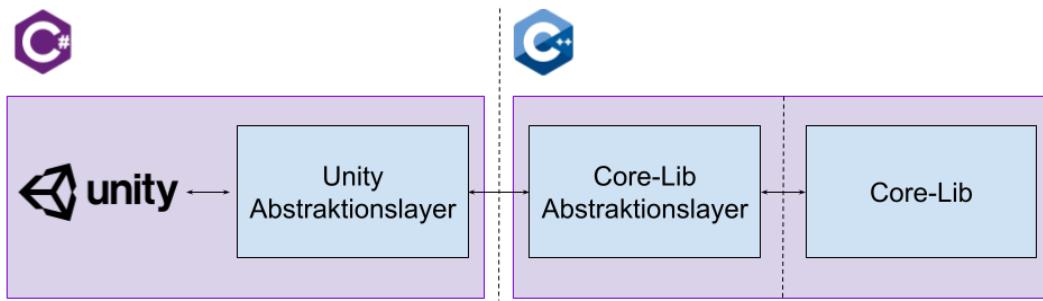


Abbildung 4.3: Grobarchitektur - Applikationsumgebung

Die Core-Library selbst besteht aus mehreren Teilstücken. Es sind dies namentlich mitsamt Funktionalität die folgenden:

**billiard\_capture** Erfasst den aktuellen Spielstand und stellt diesen im OpenCV-Format bereit.

**billiard\_detection** Erstellt aus dem Spielstand eine interne Repräsentation, welche den Status beschreibt.

**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.

## 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 und ist wie folgt definiert:

1. Das Koordinatensystem ist zweidimensional.
2. Der Ursprung befindet sich in der Mitte des Billiardtisches 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 Billiardtisches sehr einfach in diesem Koordinatensystem definiert werden können. In Abbildung 4.4 ist der Ursprung und die Achsen des Koordinatensystems eingezeichnet.

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

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



Abbildung 4.4: Modellkoordinatensystem

#### 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.

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

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

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

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

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

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 bekannte Grösse, welches im Bild erkannt werden kann.

Als Objekt bekannter Grösse und Position wurde ein ArUco-Board [ope21] mit vier Markern verwendet. Die verwendeten Marker mit einer Seitenlänge von 50mm und einer Grösse von 3x3 bits sind in Abbildung 4.5 aufgeführt.

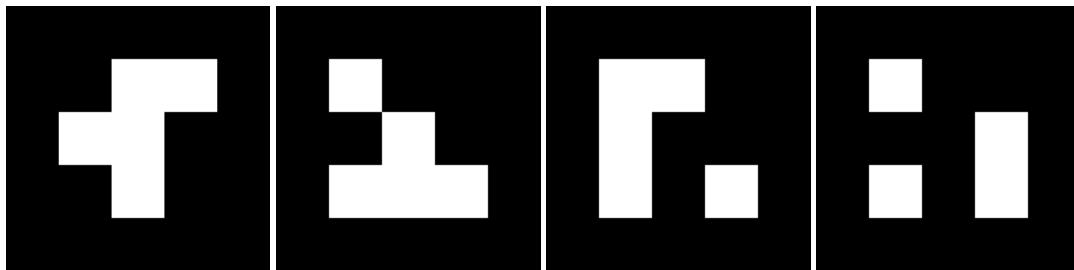


Abbildung 4.5: Verwendete ArUco markers

Diese Marker wurden auf dem Tisch wie in Abbildung 4.6 angebracht und deren relative Position zueinander ausgemessen.



Abbildung 4.6: 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 4.7 unten links gewählt. Die X-Achse ist parallel zur langen Seite des Tisches und die Y-Achse ist parallel zur kurzen Seite des Tisches.

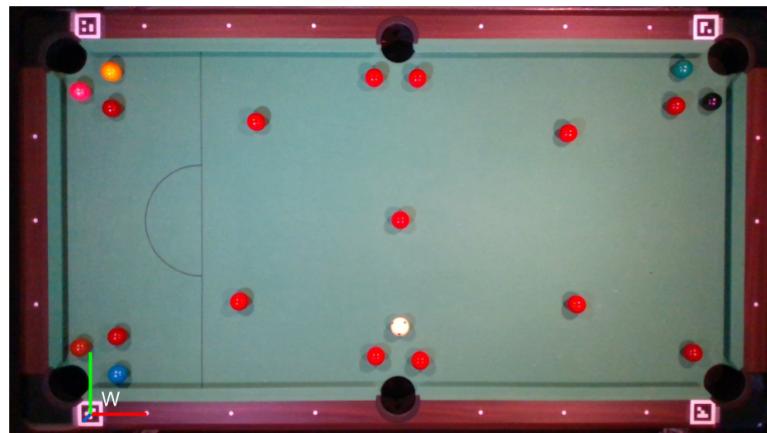


Abbildung 4.7: Weltkoordinatensystem

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  $E^{-1}$  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 des 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  $\Sigma$ , 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  $\lambda$  definiert. Sei  $B$  eine Ebene in der Normalenform mit dem Stützvektor  $\vec{a}$  und dem Normalenvektor  $\vec{n}$ , dann gilt:

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

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

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

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

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

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

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

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

$$(4.22)$$

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 4.8. 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. Er wurde belassen, da dies eine kleine Operation darstellt und dadurch das Ausmessen des ArUco boards und dessen Verschiebung zum Bandenursprung klar getrennt ist.



Abbildung 4.8: Welt-Ursprung W und Banden-Ursprung R

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

$$B_R = B_W + \begin{pmatrix} \Delta x \\ \Delta y \\ 0 \end{pmatrix} \quad (4.31)$$

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 4.9.

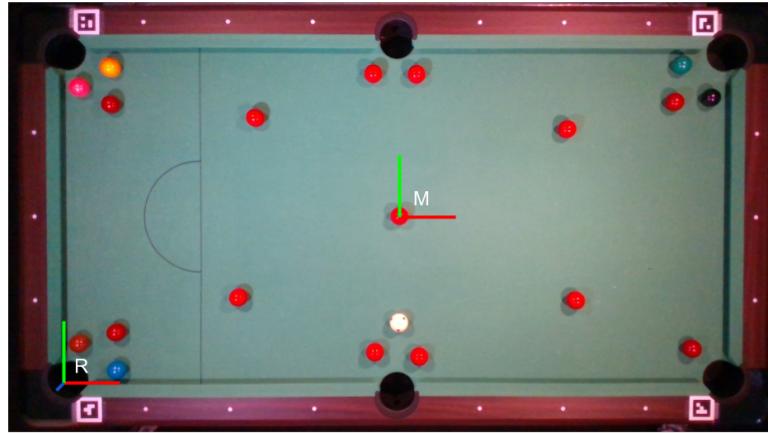


Abbildung 4.9: Banden-Ursprung B und Modell-Ursprung R

Für diese Verschiebung muss lediglich die Länge und Breite des Bereichs, welcher von den Banden begrenzt wird, gemessen werden, siehe Abbildung 4.10.

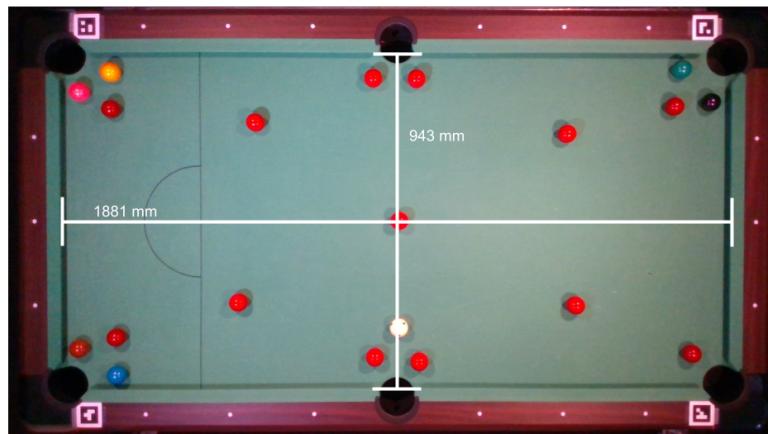


Abbildung 4.10: Länge und Breite des Spielbereits

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

$$\Delta x = \frac{L}{2} \quad (4.32)$$

$$\Delta y = \frac{B}{2} \quad (4.33)$$

$$B_M = B_R - \begin{pmatrix} \Delta x \\ \Delta y \\ 0 \end{pmatrix} \quad (4.34)$$

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

# 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 Testbilder<sup>1</sup> 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  $2\text{mm}^2$ . 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  $4.12\text{mm}$ , wobei allfällige Messfehler von Hand nicht berücksichtigt werden.

Um die Genauigkeit zu messen, wurden mehrere Testbilder<sup>3</sup> 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.

Der durchschnittliche Messfehler beträgt:  $1.956\text{ [mm]}$

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

<sup>1</sup> Alle Testbilder sind im Anhang 8.3.1 ersichtlich

<sup>2</sup> Für die Herleitung des Fehlerradius siehe Anhang 8.2

<sup>3</sup> Alle Testbilder sind im Anhang 8.3.2 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

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	Messung - Tisch . . . . .	8
4.2	Messung - Kugel . . . . .	8
4.3	Grobarchitektur - Applikationsumgebung . . . . .	9
4.4	Modellkoordinatensystem . . . . .	10
4.5	Verwendete ArUco markers . . . . .	11
4.6	Billiardtisch mit ArUco markers . . . . .	11
4.7	Weltkoordinatensystem . . . . .	11
4.8	Welt-Ursprung W und Banden-Ursprung R . . . . .	13
4.9	Banden-Ursprung B und Modell-Ursprung R . . . . .	14
4.10	Länge und Breite des Spielbereits . . . . .	14
8.1	Fehlerwinkel . . . . .	31
8.2	Fehlerwinkel vereinfacht . . . . .	32
8.3	Abweichung über Distanz . . . . .	33
8.4	Detektionsgenauigkeit - Testbild 1 . . . . .	34
8.5	Übersetzungsgenauigkeit - Testbild 1 . . . . .	34
8.6	Übersetzungsgenauigkeit - Testbild 2 . . . . .	35
8.7	Übersetzungsgenauigkeit - Testbild 3 . . . . .	35



# Tabellenverzeichnis

5.1 Detektionsgenauigkeit . . . . .	15
5.2 Messresultate Koordinatensystem . . . . .	16
5.3 Messresultate Detektion . . . . .	16



# Literatur

- [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/extrinsic/#:~:text=The%20Extrinsic%20Camera%20Matrix,the%20%22modelview%20matrix%22\) ..](https://ksimek.github.io/2012/08/22/extrinsic/#:~:text=The%20Extrinsic%20Camera%20Matrix,the%20%22modelview%20matrix%22) ..)
- [ope21] 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](https://docs.opencv.org/4.5.2/db/da9/tutorial_aruco_board_detection.html).
- [Unk21a] Unknown. *Intel RealSense Depth Camera D435*. [Online; accessed 20.05.2021]. 2021. URL: <https://www.intelrealsense.com/depth-camera-d435>.
- [Unk21b] 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>.
- [Unk21c] Unknown. *Intel RealSense SDK 2.0 Github*. [Online; accessed 20.05.2021]. 2021. URL: <https://github.com/IntelRealSense/librealsense>.
- [Unk21d] Unknown. *Matlab - What is Camera Calibration?* [Online; accessed 20.05.2021]. 2021. URL: <https://ch.mathworks.com/help/vision/ug/camera-calibration.html>.
- [Unk21e] Unknown. *OmniVision OV2740*. [Online; accessed 20.05.2021]. 2021. URL: <https://www.ovt.com/sensors/OV2740>.



# 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

TODO: Messgerät beschreiben

## 8.2 Fehlerradius

Der maximal zulässige Fehlerradius  $\beta$  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  $\beta$  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  $T$  und  $M$  gegeben ist.

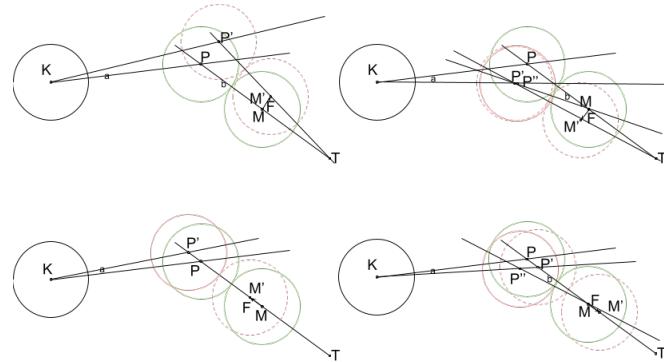


Abbildung 8.1: Fehlerwinkel

Wichtig zu beachten ist auch, dass wenn der Winkel  $\beta = 0$  ist, dies nicht bedeutet, dass kein Fehler passiert 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.

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

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

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

$$T\vec{M}'_0 = \frac{T\vec{M}'}{|T\vec{M}'|} \quad (8.4)$$

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

$$f(\lambda)' = \vec{OM}' + \lambda \cdot T\vec{M}'_0 \quad (8.6)$$

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

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

Der Winkel  $\beta$  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  $\beta$  nur eine Annäherung darstellt, da der effektive Winkel ein wenig grösser sein wird.  $M'$  kann nun mittels dem

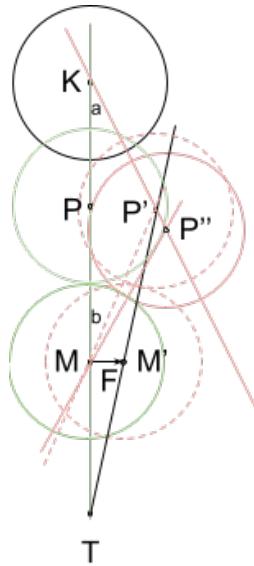


Abbildung 8.2: Fehlerwinkel vereinfacht

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 Gleichung ausgedrückt, um ihn direkt über den Fehlervektor zu definieren.

$$f(\lambda)' = \vec{OM}' + \lambda \cdot T\vec{M}'_0 \quad (8.12)$$

$$T\vec{M}' = \begin{pmatrix} M_x + F_x - T_x \\ M_y + F_y - T_y \end{pmatrix} \quad (8.13)$$

$$T\vec{M}'_0 = \begin{pmatrix} \frac{M_x + F_x - T_x}{\sqrt{(M_x + F_x - T_x)^2 + (M_y + F_y - T_y)^2}} \\ \frac{M_y + F_y - T_y}{\sqrt{(M_x + F_x - T_x)^2 + (M_y + F_y - T_y)^2}} \end{pmatrix} \quad (8.14)$$

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

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

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

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

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

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

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  $\beta$  zusammensetzt. Die Abbildung 8.3 zeigt die Ausgangslage, wobei  $D$  und  $\beta$  bekannt sind. Bei  $d$  handelt es sich um die Gegenkathete, bei  $D$

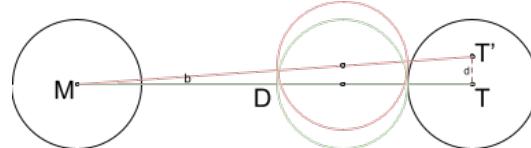


Abbildung 8.3: Abweichung über Distanz

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  $\beta$  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 Testbilder

### 8.3.1 Detektionstestbilder

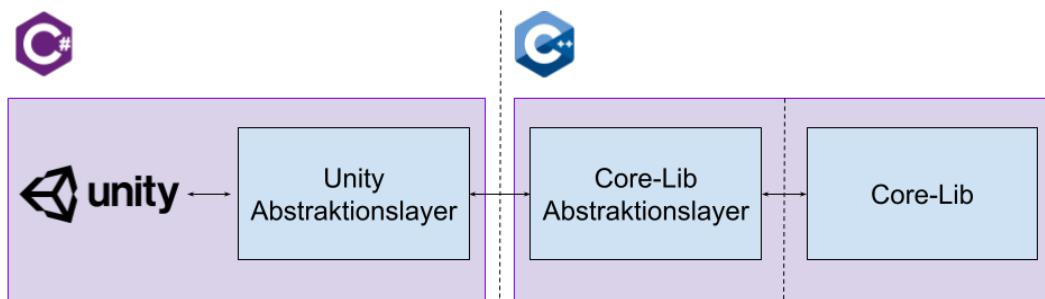


Abbildung 8.4: Detektionsgenauigkeit - Testbild 1

TODO: Add testbilder

### 8.3.2 Übersetzungsgenauigkeit

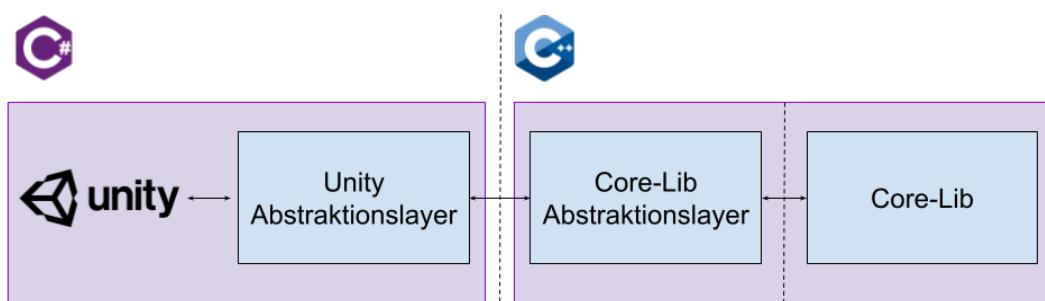


Abbildung 8.5: Übersetzungsgenauigkeit - Testbild 1

TODO: Add testbilder

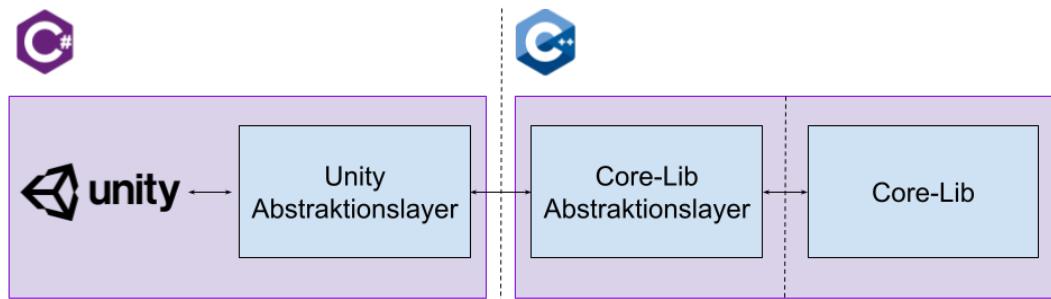


Abbildung 8.6: Übersetzungsgenauigkeit - Testbild 2

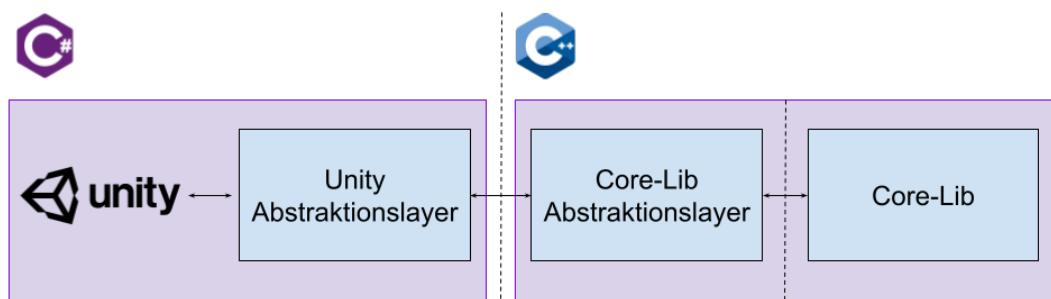


Abbildung 8.7: Übersetzungsgenauigkeit - Testbild 3