Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
IMS Institut für
Mechatronische Systeme

MSE | MASTER OF SCIENCE
IN ENGINEERING

Vertiefungsarbeit

Master of Science in Engineering

Vorstudie für einen interaktiven Billardroboter

Autor	BSc Men Duri Coray
Hauptbetreuung	Dr. Joachim Wirth
Datum	31. Juli 2014

Zusammenfassung

Men Duri Coray
Vorstudie für einen interaktiven Billardroboter
31. Juli 2014
ZHAW - Zürcher Hochschule für angewandte Wissenschaften

In jeder guten Spielhalle hat es im Minimum einen Billardtisch, bei dem man versuchen kann mit seinem Spiel zu glänzen. Doch wie es meistens im Leben ist, hat man gute oder schlechte Tage. Dann will kein einziger Stoss gelingen, obwohl man doch selbst einen Billardtisch zu Hause hat. Leider ist es so, dass die Lust zum Spielen schnell vergeht, wenn man gegen sich selbst spielt. Wie schön wäre es doch analog zu einem Schachcomputer einen Billardroboter zu besitzen.
Diese Vertiefungsarbeit befasst sich mit eben dieser Aufgabenstellung. Im Projekt sollte abgeklärt werden, ob es möglich und sinnvoll ist, einen Billardroboter umzusetzen. Es wurde aber kein komplettes Spiel aufgebaut, sondern nur der finale Stoss mit zwei Kugeln ausgeführt. Um den Stoss umsetzen zu können, wurden verschiedene Probleme analysiert und gelöst, wie die Kugel-Positionen auf dem Tisch oder die Berechnung des Stosses.

Um eigenständig Billard spielen zu können, wurde der Roboter mit einem Steuerrechner verbunden. Dieser wiederum kann mittels einer Industriekamera ein Bild der aktuellen Lage des Tisches aufnehmen und aus diesem die Position aller Kugeln auslesen. Nachdem die Positionen aller Kugeln bekannt ist, kann eine Ziel-Tasche definiert und die Berechnung gestartet werden. Als Resultat der Simulation wird dem Roboter der zum Stoss benötigte Winkel mittels TCP/IP übergeben. Zusammen mit der errechneten Position der Kugel kann er den Stoss ausführen und die Kugel einlochen.

Zurzeit ist die Genauigkeit noch ein Problem, da die Koordinaten der Bildverarbeitung nicht korrekt sind. Dennoch ist mit dieser Anwendung eine gute Basis für einen interaktiven Billardroboter geschaffen worden, welche in weiteren Arbeiten verfolgt werden kann, bis man auch in einer Spielhalle gegen einen Roboter spielen und trainieren kann.

Inhaltsverzeichnis

1 Einleitung

In der heutigen Zeit sind Computer und Roboter längst zum Begleiter des Menschen geworden. Man spielt gegen einen Computer Schach währendem ein Roboter den Boden säubert. Diese Situation gilt als normal, weil die Entwicklung schon weit fortgeschritten ist. Der erste funktionsfähige Schachcomputer *Belle* wurde beispielsweise schon 1978 [1] entwickelt und auch Serviceroboter gibt es seit Ende der 90er Jahre [2].

Bei grösseren Gesellschaftsspielen wie Billard ist es nicht möglich gegen einen Computer bzw. Roboter zu spielen. Doch gerade ein solches Spiel wäre für eine Roboteranwendung ideal, schliesslich sind hier in erster Linie nur Geometrie und Wiederholgenauigkeit entscheidend. In dieser Arbeit wird die Verknüpfung zwischen Roboter und Billard vollzogen und ein Ansatz für einen Spielaufbau entwickelt.

1.1 Billard

Abbildung 1: Ein Pool Billardtisch [3]

"Billard selbst reicht bis ins 13. Jahrhundert zurück, wo sich Hinweise auf Ballspiele finden, die auf dem freien Feld gespielt und bei dem die Bälle mit einem Schläger oder Stock geschlagen wurden. Um auch in Gegenden mit meist schlechtem Wetter das Spiel betreiben zu können, verlegte man das Geschehen nach und nach in geschlossene Räume und dort schließlich auf einen Tisch. Auch wenn die Spielfläche sich dadurch erheblich verkleinerte, blieb die Grundidee des Spiels die gleiche. Damit die Bälle nicht vom Tisch fielen, befestigte man an den Rändern Leisten. Bei diesen ersten Formen eines Ballspiels auf einem Tisch gehörten diverse Schikanen wie Tore, Bögen, Kegel und Löcher zur Ausstattung, wobei die Bälle mit dem dicken Ende des Schlägers geschlagen wurden, vergleichbar etwa mit dem heutigen Hockey" [4].

"Gespielt wird Billard in der heutigen Zeit auf verschiedenen Tischgrössen, unterteilt in verschiedenen Disziplinen" [5]:

Poolbillard "Poolbillard ist eine Variante des Billards, bei der es darum geht, mit einem weißen Spielball farbige Objektbälle nach bestimmten Regeln in sechs Taschen zu spielen (einzulochen). Der Spielball wird dabei als einzige Kugel mithilfe des Queues [Stock mit dem die Kugeln gespielt werden, Anm. d. Verf.] gespielt. Die Spieler haben abwechselnd je eine Aufnahme. Die Aufnahme ist beendet, wenn der Spieler in einem Stoß keine Kugel regelgerecht lochen konnte" [6].

Snooker "Snooker ist eine Variante des Präzisionssports Billard, die mit speziellen Queues auf

einem Snookertisch gespielt wird. Das Spielprinzip besteht darin, 15 rote und sechs andersfarbige Bälle mit dem weißen Spielball nach bestimmten Regeln in die Taschen zu versenken" [7].

Carambole "Carambole ist der Überbegriff einer Billard-Variante, die mit drei Kugeln gespielt wird. Im Gegensatz zum Poolbillard oder Snooker werden hier keine Kugeln in Taschen versenkt; der Tisch hat keine Löcher" [5].

8-Ball

8-Ball ist die bekannteste Variante des Poolbillards und für diese wird der Roboter auch entwickelt.
"Bei ihr wird mit fünfzehn Objektbällen (die Farbigen) und einem Spielball (die Weisse) auf einem Pool Billardtisch gespielt wird. Die Kugeln mit den Nummern 1 bis 7 sind komplett farbig und werden daher die *Vollen* genannt. Im Gegensatz dazu ist bei den Kugeln 9 bis 15 jeweils nur ein Streifen farbig und der Rest weiss, daher werden diese auch die *Halben* genannt. Beide Spieler müssen zunächst versuchen, ihre Farbgruppe komplett zu lochen, um dann die schwarze Acht versenken zu dürfen, was bei korrekter Ausführung zum Gewinn des Spieles führt" [6].

1.2 Aufgabenstellung

Die Arbeit ist der erste Schritt um einen interaktiven Billardroboter realisieren zu können. Dieser Roboter soll mittels einer geeigneten Methode seine Umgebung in Form des Billardtisches wahrnehmen und aufgrund dieser Ausgangslage einen optimalen Stoss durchführen können. Dabei sollte er selbständig um den Tisch herum navigieren können. Als Roboter wird ein Industrieroboter der Firma *ABB* vom Typ *IRB 120* (siehe Abbildung 2) verwendet.

Abbildung 2: ABB IRB 120 [8]

1.3 Zielsetzung

Ziel dieser Arbeit ist es, die Möglichkeit zur Realisierung des Billardroboters nachzuweisen. Dafür wird die gesamte Aufgabe in verschiedene Teilaufgaben unterteilt. Die Funktionalität der jeweiligen Lösungsvariante soll dabei mit einem kleinen Demonstrationsprogramm nachgewiesen werden.
Die folgenden Punkte wurden auf ihre Machbarkeit und Umsetzungsart hin untersucht:

- Definition der benötigten Hardware
- Ausarbeitung eines geeigneten Konzepts zur Kugelführung
- Kugelerkennung auf dem Tisch
- Erfassung des Billardtisches mit Taschen und Banden (Kanten)
- Realisierung einer Kollisionsberechnung und Spielplanung über Kugelverlauf
- Überprüfung der Ergebnisse durch versuchte Stösse mit einem ABB IRB 120 Industrieroboter auf einem Miniatur-Billardtisch (Abmessungen ca. 100×50×20 cm)

Basiert auf dieser Auswertung soll der endgültige Durchführungsentscheid getroffen werden, ob eine Realisation des Projektes sinnvoll ist.

2 Projektkonzeption

Zur strukturierten Abarbeitung des Projekts ist die Projektkonzeption notwendig. Diese beschreibt die grundlegende Methodik des Vorgehens während der Projektarbeit. Daraus entstehen die für die Realisierungsphase benötigten Arbeitsschritte sowie den Aufbau der Arbeit.

2.1 Abgrenzungen des Systems

Da sich bei einem normalem 8-Ball Spiel 16 Kugeln auf dem Tisch befinden, würde die Erkennung und Identifizierung aller Bälle den zeitlichen Rahmen der Arbeit sprengen. Deshalb wurde das Umfeld des Roboters für diese Arbeit wie folgt definiert:

Es wird nur mit zwei Kugeln gearbeitet, wobei eine der weisse Spielball ist und die andere die schwarze Acht. Das System selbst soll diese Kugel als Spielball anerkennen, welche weiter von der Tasche (Ziel) entfernt ist (siehe Abbildung 3). Indem auch die Kugel Nr. 8 als Spielball verwendet wird, können komplizierte Stösse über die Bande vermieden werden.

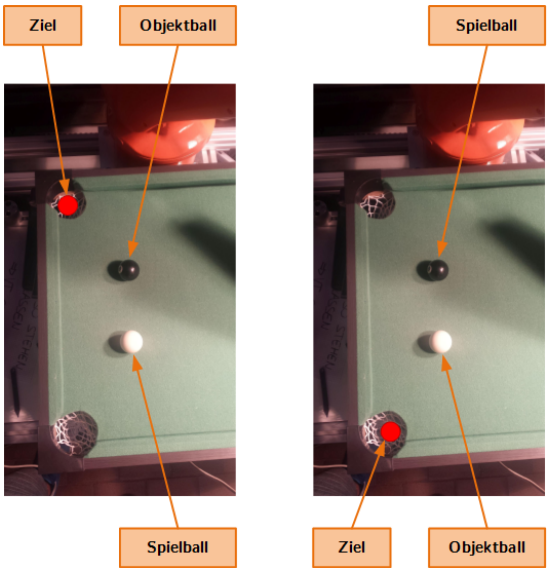

Abbildung 3: Definition des Spiel- und Objektballs

An dieser Tatsache zeigt sich, dass kein komplettes Spiel durchgeführt werden kann, sondern das Hauptaugenmerk auf dem finalen Stoss liegt. Da der verwendete Roboter mit seinem Arm nicht den ganzen Tisch abdecken kann, wird das Spielfeld auf die Hälfte reduziert. Die Kugeln sollen dabei beliebige Positionen in der Tischhälfte einnehmen dürfen.

2.2 Die Kugelführung

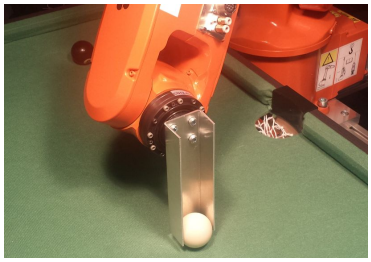
Abbildung 4: Die Kugelführung

Um die Kugel gezielt mit dem Roboter abspielen zu können, musste ein neues Roboter-Werkzeug entworfen werden. Es soll die Kugel möglichst lange stabilisieren und in Bewegungsrichtung des Roboters führen. Um diese Anforderungen zu erfüllen, wurde ein Blechteil in u-Form hergestellt (siehe Abbildung 4). Die Öffnung des u-Profils ist leicht grösser wie der Kugeldurchmesser, so dass die Kugel nicht gebremst wird. Gleichzeitig ist sie so eng, dass die Kugel möglichst wenig Spielraum hat.

2.3 Verbindung zwischen Roboter und Billardtisch

Damit der Roboter die richtigen Positionen auf dem Billardtisch anfahren kann, musste dieser in das Robotersystem eingebunden werden. Dazu wurde der Tisch fest mit dem Roboter verschraubt. Anschliessend wurde ein für alle weiteren Berechnungen gültiges Koordinatensystem auf dem Nullpunkt des Tisches definiert (Abbildung 5 unten links).
Dabei stellt der rote Pfeil die x-Achse dar, der grüne die y-Achse und der blaue (nach oben gerichtete) Pfeil die z-Achse.

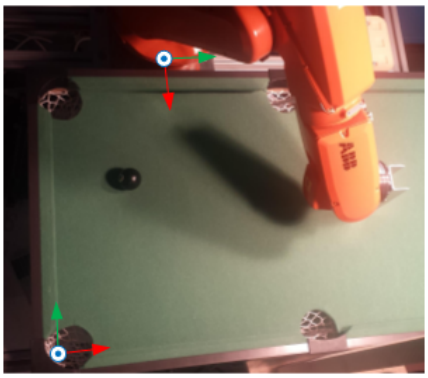

Abbildung 5: Die Koordinatensysteme von Roboter und Billardtisch

Die Umrechnung von dem Welt-Koordinatensystem des Roboters (Abbildung 5 oben Mitte) in das lokale System des Tisches wurde so gelöst, dass ein Werkobjekt in der Robotersteuerung definiert wurde. Ist dieses aktiv, werden die Koordinaten automatisch vom Welt-Roboterkoordinatensystem in das Werkobjekt-System umgerechnet.
"Ein Werkobjekt ist ein Koordinatensystem, das die Stellung eines Werkstücks [in diesem Fall der Tisch, Anm. d. Verf.] beschreibt. Das Werkobjekt besteht aus zwei Koordinatensystemen: Einem Benutzer-Koordinatensystem und einem Objekt-Koordinatensystem. Alle programmierten Positionen beziehen sich auf das Objekt-Koordinatensystem, das sich wiederum auf das durch das Welt-Koordinatensystem definierte Benutzer-Koordinatensystem bezieht" [9].

2.4 Kamera

Damit die Positionen der Kugeln auf dem Tisch detektiert werden können, wurde eine Kamera verwendet. Diese hat den Vorteil, dass sie vielfältig einsetzbar ist und nicht so teuer wie ein komplettes Sensor-System. Für dieses Projekt wurde die Kamera *uEye XS* von *IDS Imaging* eingesetzt. Die Evaluation dazu kann in der 3. Vertiefungsarbeit [10] von J. Krüsi nachgelesen werden. Sie wird vom Hersteller wie folgt beschrieben:

Abbildung 6: uEye XS [11]

"Die kleine, geniale XS mit Autofokus und zahlreichen weiteren Autofunktionen verbindet die Einfachheit einer Konsumentenkamera mit den Anwendungsmöglichkeiten einer Industriekamera. Dank der USB 2.0-Schnittstelle und Mini-B-Buchse lässt sich die XS besonders leicht integrieren. Ausgestattet mit dem 5 Megapixel Aptina CMOS-Sensor und einer Pixelgröße von 1,4 μm bietet die XS höchste Detailgenauigkeit in der Farbwiedergabe und eine kristallklare Bildqualität auch bei schwierigsten Lichtverhältnissen und Umgebungsbedingungen. Die Kamera punktet durch die leichte, kompakte Bauweise, 15 fps bei voller Auflösung (2592×1944 Pixel), die vielfältigen Sonderfunktionen und integrierte Stromversorgung besonders bei Anwendungen in Kiosk- und Embedded Systemen sowie in der Medizintechnik" [11].

3 Projektrealisierung

In diesem Kapitel werden nacheinander die einzelnen Schritte beschrieben, welche benötigt werden um einen Stoss auszuführen. Zuerst muss von der aktuellen Lage ein Bild aufgenommen werden, um anschliessend die Position der Kugeln zu ermitteln (Kapitel 3.1). Dann wird mit den erhaltenen Koordinaten und bekanntem Ziel der Winkel des Stosses ermittelt (Kapitel 3.2). Im letzten Schritt wird ein Kommunikationskanal mit dem Roboter aufgebaut und die neuen Positionsdaten an ihn gesendet, so dass er den Stoss ausführen kann (Kapitel 3.3).

Abschliessend wird noch die Benutzeroberfläche des Steuerprogramms vorgestellt (Kapitel 3.4) und die Genauigkeit des Roboters analysiert (Kapitel 3.5). Die gesamte Programmierung des Berechnungstools erfolgte in der Sprache C++.

3.1 Bildverarbeitung

Die Bildverarbeitung ist der erste Schritt im Prozess. Für sie wurden die C++-Bibliothek *OpenCV* und das SDK der Kamera eingebunden. Der genaue Ablauf für die Installation ist in der Arbeit [10] von J. Krüsi nachgelesen. Mit diesen beiden Bibliotheken können die Koordinaten der Kugeln aus dem Bild herauszulesen werden. Dazu werden der Reihe nach die folgenden Schritte unternommen:

- Zuerst wird ein Bild vom Billardtisch ohne Kugeln als Hintergrund aufgenommen.
- Anschliessend wird ein Bild des Tisches mit Kugeln aufgenommen, was in dem Bild des Vordergrundes gespeichert wird.
- Die Differenz der beiden Bilder wird mittels Background Subtraction berechnet, was in einem Hervorheben der Kugeln resultiert.
- Mittels Hough Circle Detection können im neuen Bild die Positionen aller Kugeln gefunden werden.

Background Subtraction

"Background Subtraction (BS) ist eine weit verbreitete Technik um mit statischen Kameras eine Maske des Vordergrundes (ein Binärbild mit den Pixeln welche sich in der Szene bewegen) zu generieren. Wie der Name schon sagt, errechnet BS eine Maske des Vordergrundes indem eine Subtraktion zwischen dem momentanen Bild und dem Hintergrundmodell durchgeführt wird,

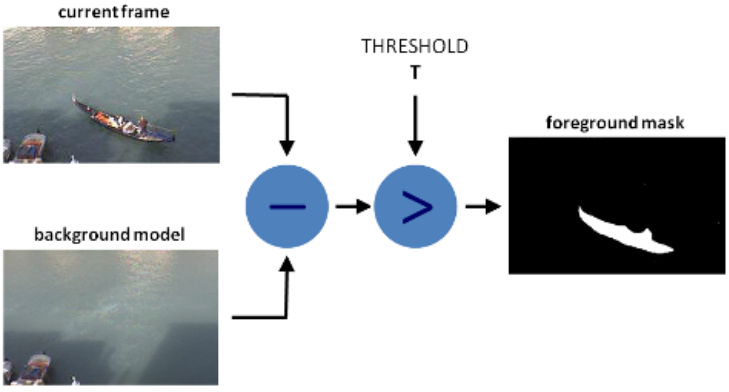

Abbildung 7: Prinzip der Background Subtraction [12]

welches den statischen Teil der Szene beinhaltet oder, mehr generell, alles was aufgrund der Charakteristik der beobachteten Szene als Hintergrund gegeben ist" [12].

So können wie in Abbildung 7 anstatt dem Schiff die Bälle hervorgehoben werden.

Hough Circle Detektion

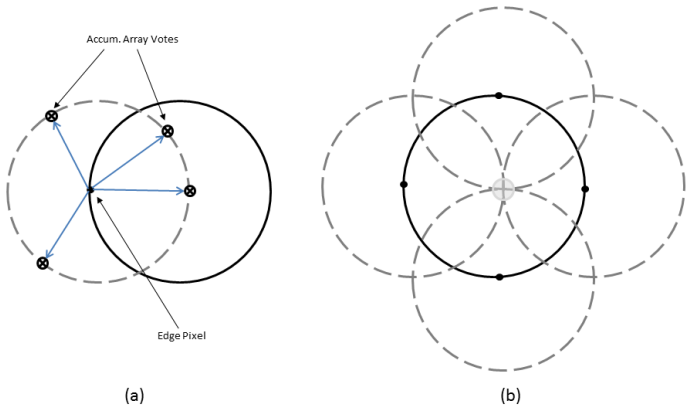

Abbildung 8: Hough Kreis Transformation [13]

Nach dem Hervorheben der Kugeln werden mit der Hough Transformation alle Kreise (sprich die Kugeln) auf dem Bild gesucht.

"Die Hough-Transformation ist ein Verfahren zur Detektion kollinearer Punkte. Mit ihr können Objekte erkannt werden, die sich in geschlossener parametrisierbarer Form darstellen lassen. Beispiele hierfür sind Kreise, Ellipsen oder Linien" [14].

Die Hough Kreiserkennung selbst funktioniert wie folgt: "Durch jeden Punkt im Originalbild wird ein Kreis mit vorher festgelegtem Radius gelegt. Punkte im Hough-Transformationsbereich, in denen sich solche Kreise schneiden, sind Kandidaten für Mittelpunkte von Kreisen (mit dem gewählten Radius) im Originalbereich. Die Punkte in denen sich die meisten Kreise schneiden, werden als Kreismittelpunkte in den Originalbereich rücktransformiert" [14].

3.2 Stossberechnung

Definitionen

Alle Objekte werden mittels linearer Algebra - sprich zweidimensionalen Vektoren - beschrieben. Für das Projekt definiert wurden Kugeln und Linien, welche sich durch folgende Parameter unterscheiden:

Objekt	Parameter		Zeichen
Kugel:	Zentrum der Kugel	Punkt	P oder Q
	Geschwindigkeit der Kugel	Vektor	u, v oder w
	Radius der Kugel	Skalar	r
Linie:	Startpunkt der Linie	Punkt	P oder Q
	Richtung der Linie	Vektor	u oder v

Tabelle 1: Parameterwerte

Eine Kugel-Bewegung wird mit der Punkt-Richtungsform $P + \lambda \cdot u$ dargestellt, wobei der Parameter λ als Zeit t angesehen wird.

3.2.1 Geometrische Zusammenhänge

Für eine Simulation des Stosses mussten diverse Probleme gelöst werden. Bevor eine Simulation umgesetzt werden konnte, musste eine Grundlage der Geometrie programmiert werden. Dazu wurde unterschieden, was alles für verschiedene Kollisionen möglich sind und diese dann separat behandelt.

Grundsätzlich ist der erste Schritt immer derselbe. Es muss der Schnittpunkt zwischen zwei Linien ermittelt werden. Dies wird umgesetzt, indem die zwei Objekte in Punkt-Richtungsform gleichgesetzt werden. Dieses lineare Gleichungssystem kann nach den beiden Parameterwerten λ und μ aufgelöst werden.

$$\left.\begin{array}{l} g: P + \lambda \cdot \vec{u} \\ h: Q + \mu \cdot \vec{v} \end{array}\right\} \quad \Rightarrow \quad \text{Schnittpunkt } S = g = h \quad \Rightarrow \quad \begin{cases} P_x + \lambda \cdot u_x = Q_x + \mu \cdot v_x \\ P_y + \lambda \cdot u_y = Q_y + \mu \cdot v_y \end{cases}$$

$$\begin{aligned} \lambda &= \frac{\text{Det}(Q, \vec{v}) - \text{Det}(P, \vec{v})}{\text{Det}(\vec{u}, \vec{v})} \\ \mu &= \frac{\text{Det}(Q, \vec{u}) - \text{Det}(P, \vec{u})}{\text{Det}(\vec{u}, \vec{v})} \end{aligned} \tag{3.1}$$

Abschliessend kann der Schnittpunkt errechnet werden, indem beispielsweise der Parameter λ in die Gerade g eingesetzt wird. λ entspricht also der Zeit t.

Kollision zwischen Kugel und Bande

Sobald festgestellt wurde, dass eine Kollision zwischen Kugel und Bande stattfindet, muss anschliessend der neue Geschwindigkeitsvektor der Kugel bestimmt werden. Geometrisch gesprochen ist dieser Vorgang eine Spiegelung des Vektors an der um 90° gedrehten Bande (siehe Abbildung 9). Die Spiegelung selbst wird erreicht, indem zuerst das zum Ursprung $(x_0|y_0)$ möglicherweise gedrehte Koordinatensystem $(x_1|y_1)$ der Bande errechnet wird. Dies kann aus dem Richtungsvektor $\vec{u}$ der Bande wie folgt abgeleitet werden.

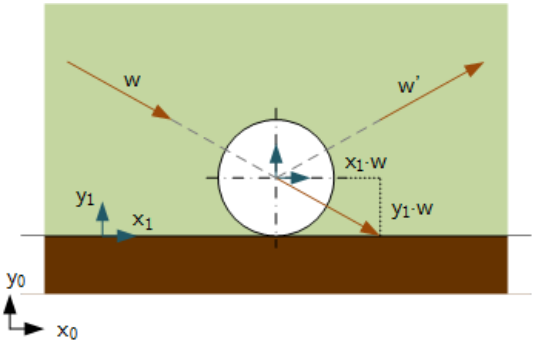

Abbildung 9: Bandenkollision

$$\vec{x_1} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \qquad \vec{y_1} = \begin{pmatrix} -u_2 \\ u_1 \end{pmatrix} \qquad \text{wobei gilt:} \quad \|\vec{x_1}\| = \|\vec{y_1}\| = 1$$

Mit diesen Vektoren kann die Projektion des Kugelvektors $\vec{w}$ auf das neue Koordinatensystem errechnet werden und anschliessend die Spiegelung $\vec{w}'$ davon, welche durch Ändern des Operators erreicht wird. Dieses Vorgehen resultiert in dem neuen Geschwindigkeitsvektor:

$$\vec{w} = \begin{pmatrix} \vec{x_1} \cdot \vec{w} \\ \vec{y_1} \cdot \vec{w} \end{pmatrix} = (\vec{x_1} \cdot \vec{w})\vec{x_1} + (\vec{y_1} \cdot \vec{w})\vec{y_1}$$

$$\Rightarrow \quad \vec{w}' = (\vec{x_1} \cdot \vec{w})\vec{x_1} - (\vec{y_1} \cdot \vec{w})\vec{y_1} \tag{3.2}$$

Kollision zwischen Kugel und Kugel

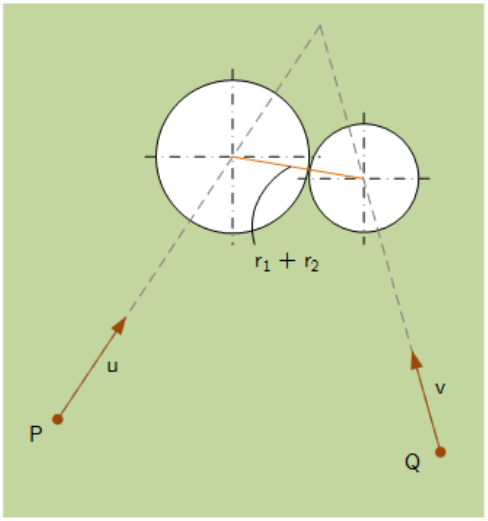

Abbildung 10: Kugelkollision

Um zu überprüfen, ob und wann 2 Kugeln wie in Abbildung 10 zusammenstossen, muss die Formel 3.1 der Kollision zweier Linien leicht abgeändert werden. Hier muss nicht der Zeitpunkt errechnet werden, wann sich die beiden Geschwindigkeitsvektoren treffen, sondern wann der Abstand der Kugeln genau der Summe der beiden Radien entspricht. Dieser Zusammenhang mit den Unbekannten λ und μ definiert sich wie folgt:

$$\|(P + \lambda \cdot \vec{u}) - (Q + \mu \cdot \vec{v})\| = r_1 + r_2$$

Dieses unterbestimmte Gleichungssystem kann gelöst werden, indem der Zusammenhang berücksichtigt wird, dass λ und μ genau gleich gross sein müssen - nämlich t. Es entsteht eine Gleichung der 2. Ordnung mit der Form $A\lambda^2 + B\lambda + C = 0$, welche mit der Mitternachtsformel gelöst werden kann. Die gesuchte Zeit t entspricht dem Minimum von λ_1 und λ_2:

$$\left.\begin{aligned} A &= (\vec{u} - \vec{v})^2 \\ B &= 2 \cdot (P - Q) \cdot (\vec{u} - \vec{u}) \\ C &= (P - Q)^2 - (r_1 + r_2)^2 \end{aligned}\right\} \quad \Rightarrow \quad \lambda_{1,2} = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

$$t = \min(\lambda_1, \lambda_2) \tag{3.3}$$

Neue Geschwindigkeitsvektoren nach der Kugelkollision

Nachdem der Berührungspunkt der beiden Kugeln bekannt ist, müssen die neuen Richtungsvektoren bestimmt werden. Dies wird erreicht, indem die beiden Kugel-Impulse gemäss dem Impul-

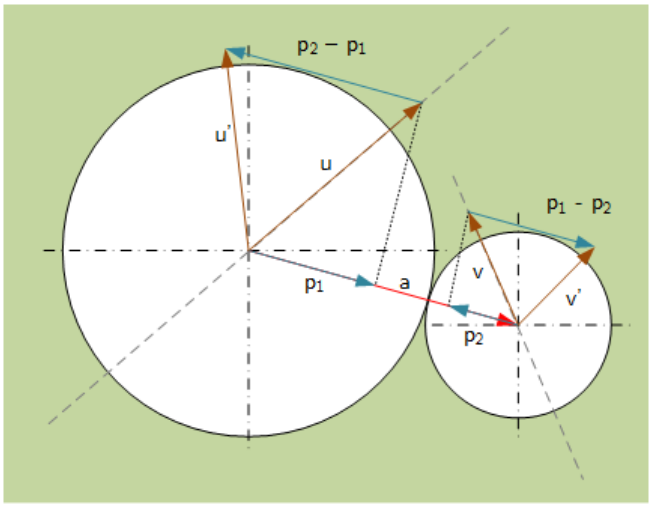

Abbildung 11: Neue Geschwindigkeitsvektoren

serhaltungssatz voneinander addiert bzw. subtrahiert werden (Abbildung 11). Da beide Kugeln immer gleich schwer sind und der Impuls $p = m \cdot v$ ist, kann nur mit den Geschwindigkeitsvektoren gerechnet werden. Diese werden auf den Verbindungsvektor $\vec{a}$ der beiden Mittelpunkte projektiert und dann gemäss der Impulserhaltung miteinander addiert.

$$\vec{u}' = \vec{u} - \vec{u} \cdot \left(\frac{\vec{a} \cdot \vec{u} - \vec{a} \cdot \vec{v}}{\|a\|^2} \right) \tag{3.4}$$

3.2.2 Simulation eines Stosses

Zielgrösse

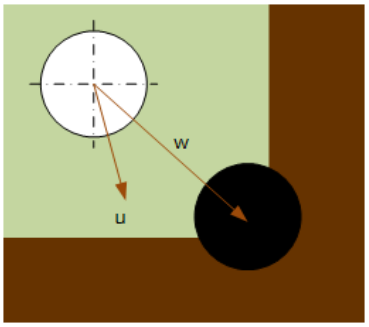

Abbildung 12: Winkelberechnung

Um zu berechnen, in welchem Winkel ein Stoss auf den Spielball ausgeführt werden muss, wird geprüft wie gross der Winkel zwischen dem direkten Verbindungsvektor $\vec{w}$ und dem Geschwindigkeitsvektor $\vec{u}$ des Objektballs ist (Abbildung 12). Dieser wird mit der folgenden Formel ermittelt:

$$\varphi = \arccos \left(\frac{\vec{u} \cdot \vec{w}}{\|\vec{u}\| \cdot \|\vec{w}\|} \right) \tag{3.5}$$

Diese Grösse φ soll nun mit der Simulation so minimiert werden, dass mach dem Stoss der Objektball möglichst genau das Zentrum der Tasche trifft.

Ablauf

Mit dieser Grundlage der Geometrie konnte der Ablauf der Simulation definiert werden. Dabei wird zuerst überprüft, ob eine Kollision zwischen Spielball und Objektball stattfindet. Erst wenn dies der Fall ist, wird der Geschwindigkeitsvektor des Objektballes neu berechnet und der Zwischenwinkel mit dem Zielvektor überprüft. Der genaue Ablauf ist wie folgt:

- Spielball, Objektball und Ziel werden definiert.
- Zwischen Ziel und Position des Objektballs wird ein Zielvektor berechnet.
- Der Winkel des Spielball-Geschwindigkeitsvektors wird auf 0° gesetzt.
- Der Stoss wird berechnet:
  - Mit der Formel 3.1 werden alle Zeitpunkte von Kollisionen zwischen Kugeln mit der Bande registriert (Zeitpunkt t_B wird gespeichert).
  - Mit der Formel 3.3 werden alle Zeitpunkte von Kollisionen von Kugeln untereinander registriert (Zeitpunkt t_K wird gespeichert).
  - Das Minimum der beiden Zeitpunkte t_B und t_K ist der nächste Zeitschritt t.
  - Die Kugeln werden alle um den errechneten Zeitschritt t bewegt.
  - Je nachdem welcher Zeitpunkt kleiner war, wird mit der Formel 3.2 oder 3.4 der neue Ball-Geschwindigkeitsvektor berechnet.
- Bei einer allfälligen Kollision zwischen den Kugeln wird der Winkel zwischen dem Geschwindigkeitsvektor des Objektballs und dem Zielvektor mit der Formel 3.5 berechnet und zusammen mit dem Spielball-Geschwindigkeitsvektor (als Winkel) gespeichert.

- Der Winkel des Spielball-Geschwindigkeitsvektors wird inkrementiert.
- Sobald dieser 360° erreicht hat, wird die Berechnung abgeschlossen.
- Der minimale Zwischenwinkel von Ziel- und Objektballvektor wird mit dem dazugehörigen Spielball-Geschwindigkeitsvektor ausgelesen.
- Der neue Spielball-Geschwindigkeitsvektor wird an das Steuerprogramm zurückgegeben.

3.3 Roboter

3.3.1 Kommunikation

Damit das Steuerprogramm die berechneten Koordinaten an den Roboter senden kann, musste ein Kommunikationskanal aufgebaut werden. Dieser basiert auf der Ethernet-Technologie, genauer dem TCP/IP-Protokoll. Auf Basis eines Sockets, bei welchem die Robotersteuerung die Serverrolle übernimmt, werden Roboterkoordinaten und -quaternionen übertragen. Dabei sind beide Parteien des Sockets fähig, Daten zu senden und zu empfangen. Für die Verwendung des Sockets wurde auf Seite des Verarbeitungsrechners eine eigene Klasse implementiert, während die Robotersteuerung dies innerhalb des Abarbeitungsprogramms regelt. Damit die Übertragung der Koordinaten richtig interpretiert werden kann, musste das folgende Übertragungsprotokoll verwendet werden:

Befehl	Bedeutung
`gPos`	Fordert die momentane Position des Roboters an. Diese wird im Format `pos;x;y;z;` retourniert, wobei x, y und z die Koordinaten des Roboters in mm sind.
`gRot`	Fordert die momentane Ausrichtung des Roboter-Werkzeugs an. Wird im Format `rot;q1;q2;q3;q4;` retourniert, wobei q1, q2, q3 und q4 die Quaternionen des Roboterwerkzeugs sind.
`sPos;x;y;z;`	Definiert ein neues Ziel für den Roboter und übergibt nacheinander die Koordinaten x, y und z in mm.
`sRot;q1;q2;q3;q4;`	Definiert eine neue Ausrichtung des Roboter-Werkzeugs und übergibt nacheinander die Quaternionen q1, q2, q3 und q4.
`move`	Startet die Bewegung mit neuen Koordinaten und Quaternionen.
`stop`	Schliesst die Verbindung zwischen den beiden Sockets.

Tabelle 2: TCP/IP Übertragungsprotokoll

3.3.2 Umrechnung des Drehwinkels

Die Ausrichtung des Roboterwerkzeugs wird in Quaternionen ausgegeben. “Quaternionen sind Elemente einer algebraischen Struktur, die als eine Erweiterung der komplexen Zahlen angesehen werden kann. Man kann Addition und Multiplikation als innere Operationen der Struktur definieren, so dass die Körperaxiome mit Ausnahme der allgemeinen Kommutativität der Multiplikation gelten. Man nennt einen solchen Körper auch Schiefkörper. Quaternionen sind vierdimensionale Objekte, man schreibt sie jedoch zweckmäßigerweise als Skalarkomponente und dreidimensionale Vektorkomponente” [15].

Mit der Rotationsformel nach Hamilton lässt sich gemäss [15] die benötigte Drehung des Werkzeugs ausrechnen, welche mit der Simulation bestimmt wurde. Dabei ist $\vec{u}$ die normierte, dreidimensionale Rotationsachse (in Fall des Billardroboters ist es immer die x-Achse $[1, 0, 0]$) und φ der Rotationswinkel. Mit diesen beiden wird ein neues Quaternion q gebildet, wobei zu beachten gilt, dass der Winkel φ negiert übergeben werden muss. Dies lässt sich auf die z-Achse des Tisches zurückführen, weil diese entgegen der x-Achse des Werkzeugs verläuft.

$$q = \left[\cos\left(\frac{\varphi}{2}\right), \sin\left(\frac{\varphi}{2}\right) \cdot \vec{u}\right]$$

Dieses Quaternion multipliziert[1] mit der momentanen Ausrichtung Q_0 des Roboterwerkzeugs ergibt das neue gesuchte Quaternion q_{neu}.

$$q_{neu} = q \star Q_0 \tag{3.6}$$

3.4 Steuerprogramm

3.4.1 Benutzeroberfläche

Die grafische Benutzereingabe war für dieses Projekt sehr wichtig, da es als Schnittstelle zwischen dem Benutzer und der Anwendung dient. Deshalb wurde eine umfangreichere Applikation erstellt, welche folgende Funktionen beinhaltet:

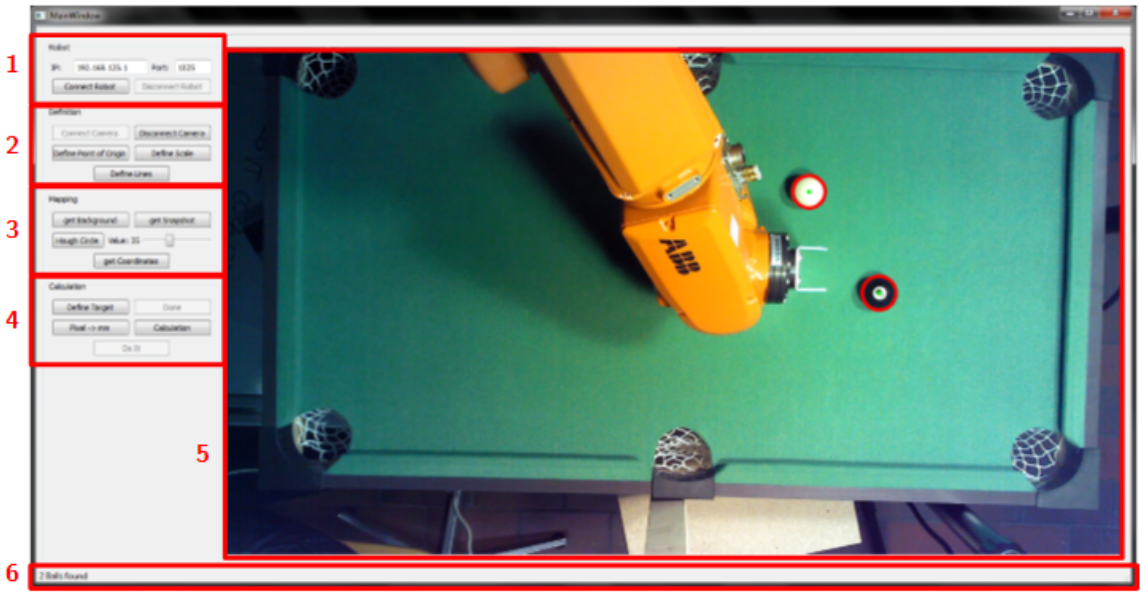

Abbildung 13: Benutzeroberfläche des Programms

- **Robotersteuerung** *[1]*
  Die Verbindung zum Roboter wird mit einer TCP/IP-Verbindung hergestellt, welche über die Benutzeroberfläche sowohl aktiviert als auch deaktiviert werden kann. Zusätzlich können IP-Adresse und Port definiert werden.

[1] Die Herleitung und Durchführung der Multiplikation zwischen zwei Quaternionen kann nachgeschlagen werden unter: [15]

- **Definitionen** *[2]*
  In diesem Bereich kann die Kamera gestartet bzw. beendet werden. Zusätzlich werden hier alle benötigten Elemente wie die Banden oder der Ursprung des Tisch-Koordinatensystems definiert.
- **Mapping** *[3]*
  Im Mapping-Bereich wird die Bildverarbeitung gesteuert. Es können Vorder- und Hintergrund für die Background-Subtraction aufgenommen, sowie die Hough-Transformation durchgeführt werden. Ausserdem werden hier die Ergebnisse der Transformation an die Simulation übergeben.
- **Berechnungen** *[4]*
  In letzten Steuerbereich wird sowohl das Ziel des Stosses definiert, als auch die Simulation gestartet.
  Mit der letzten Schaltfläche wird bei einer aktiven Verbindung mit dem Roboter der Stoss ausgeführt.
- **Ausgabefenster** *[5]*
  Im Ausgabefenster werden die letzte Aufnahme der Kamera oder das Ergebnis der Hough-Transformation angezeigt. Sie bietet aber kein Live-Bild der Kamera.
- **Ereignismeldungen** *[6]*
  Auf der Statuszeile werden Informationen zu den unterschiedlichen Aktivitäten angezeigt. Sobald eine Verbindung (Kamera oder Roboter) hergestellt oder eine Berechnung durchgeführt wurde, erscheint hier eine Meldung.

3.4.2 Standard Template Libary

Für die Abarbeitung der Simulation wie in Kapitel 3.2.2 beschrieben, wurde mit der Standard Template Libary (STL) gearbeitet. “Die STL ist eine Sammlung von verschiedenen Klassentemplates und generischen Algorithmen. Diese Bibliothek wurde geschaffen, um oft wiederkehrende Aufgaben zu generalisieren und wurde als Erweiterung in den C++-Standard aufgenommen. Das hat den Vorteil, dass ein Standard geschaffen wird, der als Referenz für viele Aufgaben dienen kann. Beispielsweise muss ein Programmierer nicht mehr über die Begrenzungen eines Feldes nachdenken. Benutzt er stattdessen den Standardcontainer zum Speichern beliebiger Datentypen, bekommt er diese Funktionalität automatisch zur Verfügung gestellt. [...] Weitere wichtige Bestandteile der STL sind neben den Containern Iteratoren, adaptierte Container, Funktionsobjekte und Algorithmen” [16].

Um die Eigenschaften des Containers im C++-Programm voll auszunutzen, wurden zwar alle 16 Kugeln initialisiert, aber nur diejenigen dem Container zur Manipulation übergeben, welche sich auf dem Tisch befinden. Dasselbe wurde mit den 4 Banden-Objekten gemacht. So wurden nur die Eigenschaften der aktiven Elemente in dem Container verändert.

Für die Veränderung der Eigenschaften wurde mit den STL-Algorithmen gearbeitet. Da viele Operationen mit jeder Kugel genau einmal gemacht werden muss, wurde vor allem der `for_each`-Algorithmus (Listing 3.1 in Zeile 16) verwendet, welcher die gewünschte Operation mit jedem Objekt genau einmal durchführt.

Um bei vielen Kugeln auf dem Tisch eine mögliche Kollision zweier effizient zu überprüfen, soll der Funktions-Aufruf pro Kugelpaar auch nur einmal stattfinden. Darum wurden Iteratoren verwendet, welche auf ein Objekt im Container zeigen. Mit diesen konnte eine doppelte for-Schlaufe generiert werden, bei welcher die Überprüfung zwischen zwei Objekten nur einmal stattfindet (Listing 3.1 zwischen Zeile 19 und 26).

```
#include <algorithm>
#include <set>

// Create new Ball pointer
Ball *b1 = new Ball( 1 ); // Ball with id = 1
Ball *b2 = new Ball( 2 ); // Ball with id = 2
Ball *b3 = new Ball( 3 ); // Ball with id = 3

// Create new container and insert all Ball pointer
std::set<Ball*> ballSet;
ballSet.insert( b1 );
ballSet.insert( b2 );
ballSet.insert( b3 );

// Print all characteristics (function toString()) of every Ball in the
   container
std::for_each( ballSet.begin(), ballSet.end(), toString );

// Double loop to check which Ball pointers are handeled togethter
std::set<Ball*>::iterator jt;
for( std::set<Ball*>::iterator it = ballSet.begin(); it != ballSet.end()
   ; ++it )
{
  jt = it;
  ++jt;
  for( jt; jt != ballSet.end(); ++jt )
    qDebug() << QString( "Ball " + id( *(*it) ) + " with " + id( *(*jt)
       ) );
}

// Output from the double loop:
// Ball 1 with 2
// Ball 1 with 3
// Ball 2 with 3
```

Listing 3.1: Beispiel für das Arbeiten mit STL

3.5 Genauigkeit

Die Genauigkeit der Anwendung ist ein entscheidender Faktor über die Qualität der Arbeit, wenn nicht gar der entscheidendste. Wie bereits anfangs in Kapitel 1 angesprochen, muss der Roboter korrekte Daten von der Software bekommen, um den Stoss erfolgreich auszuführen.

Aus diesem Grund wurden nach der Hough Transformation die Koordinaten der Kugeln überprüft. Die nachfolgende Tabelle 3 stellt einige dieser Messungen dar.

Kugel Nr.	Berechnete Position	Gemessene Position	Differenz
Kugel 1	(609 230)	(615 239)	(6 9)
Kugel 2	(697 143)	(706 150)	(9 7)
Kugel 3	(599 346)	(605 358)	(6 12)
Kugel 4	(777 95)	(788 104)	(11 9)
Kugel 5	(597 267)	(603 277)	(6 12)
Kugel 6	(690 157)	(697 168)	(7 11)
Kugel 7	(629 225)	(636 235)	(7 10)
Kugel 8	(737 81)	(743 93)	(6 12)

Tabelle 3: Messprotokoll der Positionsbestimmung aller Kugeln in mm

Es ist darin ersichtlich, dass die durchschnittliche Differenz in x-Richtung 7.25 mm beträgt. In y-Richtung sind es sogar 10.25 mm. Dies zeigt, dass die Krümmung der Kameralinse für eine genaue Berechnung zu gross ist und somit berechnet werden muss. Diese Linsen-Krümmung selbst lässt sich auf die Grösse der Kleinstkamera *uEye XS* zurückführen.
"[...] Seit der Einführung von billigen Kleinstkameras im späten 20. Jahrhundert, sind sie in unserem täglichen Leben weit verbreitet. Leider zieht diese Geringwertigkeit mit einem Preis mit sich: signifikante Verzerrung. Glücklicherweise ist diese konstant und kann mittels Kalibrierung und Neu-Kartographie korrigiert werden. Ausserdem kann man mit der Kalibrierung die Beziehung zwischen der Kamera (Pixel) und weltlichen Einheiten (zum Beispiel Millimeter) bestimmen" [17].
Für die Kalibrierung bietet *OpenCV* ein komplettes Modul *(calib3d)* an, welches die Krümmung der Kamera berechnet und anschliessend korrigiert.

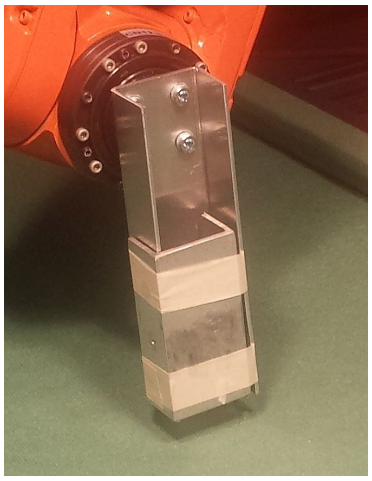

Abbildung 14: Neuer Stossfläche

Weil die Positionsdaten zu ungenau waren, war die Kugel beim Stoss nie innerhalb des U-Profils, sondern traf immer die dünnen Seitenbleche des Roboterwerkzeugs. Um diese Fehlausführung des Stosses zu korrigieren, wurde in einem ersten Schritt versucht, das Werkzeug des Roboters zu verbessern. Dazu wurde die Öffnung des U-Profils mit einer Metall-Platte wie in Abbildung 14 versehen, so dass neu eine Fläche die Kugel im richtigen Winkel berührt. Es konnte nur eine leichte Verbesserung der Kugel-Laufrichtung festgestellt werden.

Trotz der Verbesserungen traf im Durchschnitt nur jeder 10. Stoss das Ziel und die richtige Kugel wurde versenkt. Diese Zahlen sprechen nicht für eine Weiterführung des Projekts, aber ein endgültiger Entscheid sollte erst nach der Implementation der Kalibrierung und erneuter Genauigkeitsmessung gefällt werden.

4 Projektabschluss

4.1 Rückblick

Die zweite Vertiefungsarbeit war entgegen der ersten nicht mehr im Bereich der mobilen Robotik angesiedelt. Darum mussten viele Anwendungsgebiete der Industrierobotik erst erarbeitet werden, wie das Rechnen mit Quaternionen. Auch die Arbeit mit der Bildverarbeitungs-Bibliothek *OpenCV* war neu. Da der Umfang des Projekts von Anfang an gross war, musste schnell ein Abstrich gewisser Funktionen gemacht werden. So konnte trotz der Ungenauigkeiten ein kleines Demonstrationsprogramm erarbeitet werden.

Auch diese Arbeit beschäftigte sich zu einem grossen Teil mit der Programmierung in C++ unter Verwendung der *Qt* Klassenbibliothek. Schon zum zweiten Mal wurde eine komplette Applikation eigenständig in C++ mit *Qt* erstellt, was einen enormen Lernerfolg gebracht hat. Auch die Arbeit mit der Standard Template Libary brachte viele neue Erfahrungen zum Thema C++-Programmierung. Schlussendlich bietet die Arbeit eine gute Basis für die weitere Entwicklung des Billardroboters.

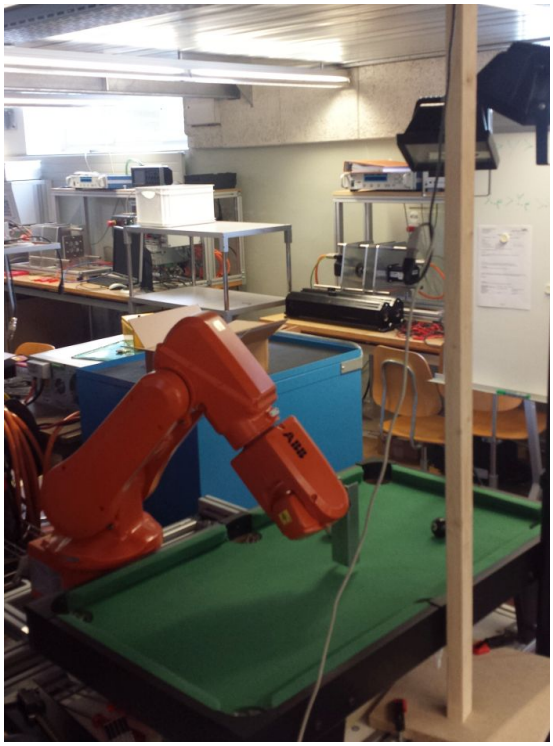

Abbildung 15: Der Billardroboter mit Tisch, Greifer und Scheinwerfer

4.2 Aufgetretene Probleme

Beleuchtung

Bei der Implementation der Hough Transformation konnten anfangs keine Ergebnisse erzielt werden. Dieses Problem liess sich über längere Zeit nicht lösen, weil die Schatten der Kugeln nicht mit der Background Subtraction entfernt werden konnten. So erschien die Form der Kugeln als Ellipse oder überhaupt nicht kreisförmig. Auch das Kalibrieren der Hough-Funktionsparameter zeigte keine nennenswerte Verbesserung. Erst als zwei Baustrahler oberhalb des Tisches montiert wurden, konnte der Kugel-Schattenwurf so minimiert werden, dass die Transformation brauchbare Ergebnisse lieferte.

Linsenkrümmung der Kamera

Bei der Genauigkeits-Überprüfung der Hough-Transformation wurde festgestellt, dass die effektiven Kugelzentren zwischen 6 und 10 mm von den berechneten abweichen. Dieser Versatz wurde auf die Krümmung der Kameralinse zurückgeführt und konnte innerhalb der Projektzeit nicht mehr gelöst werden.

4.3 Ausblick

Die Arbeit ist zu diesem Zeitpunkt noch nicht abgeschlossen. Das Ziel sollte sein, einen eigenständigen Roboter zu haben, welcher gegen Personen 8-Ball spielt. Die ist nur möglich, wenn die folgenden Punkte abgearbeitet werden:

- Der erste Schritt zur Verbesserung der Arbeit sollte die Korrektur der Kameralinse sein. Dazu muss das *calib3d*-Modul von *OpenCV* geladen und für die Kamera implementiert werden. Anschliessend kann nochmals die Genauigkeit analysiert werden.
- Momentan kann die Erkennung der Kugeln nur bei starkem Licht durchgeführt werden. Ohne Baustrahler, welche den Tisch von oben beleuchten, können keine Ergebnisse mit der Hough Transformation erreicht werden. Dieses Problem könnte man beispielsweise lösen, indem rund um sie Spielfläche an der Bande LEDs befestigt werden, welche das Fremdlicht überdecken.
- Für einen eigenständigen Roboter ist es notwendig, dass die Position der Taschen und Banden nicht jedes Mal neu vom Benutzer definiert werden muss. Es sollte ein Algorithmus erarbeitet werden, welcher diese Aufgabe löst oder es wird mit einer fest installierten Kamera gearbeitet und die Daten werden einmal manuell ausgelesen.
- Im jetzigen Stand der Anwendung, muss dem Roboter gesagt werden, in welche Tasche er die Kugel einlochen soll. Dieser Task sollte mit einen eigenen Algorithmus abgearbeitet werden.
- Um komplett eigenständig spielen zu können, muss die Software in der Lage sein, zwischen vollen und halben Kugeln unterscheiden zu können, sowie entscheiden können, welche der zur Auswahl stehenden Kugeln am sichersten eingelocht werden kann.
- Damit auf dem ganzen Tisch gespielt werden kann, müsste ein grösserer Roboter verwendet werden oder der jetzige Roboter müsste sich mobil um den Tisch bewegen können.

Literaturverzeichnis

[1] J. H. Condon und K. Thompson. „Belle Chess Hardware“. In: *Advances in Computer Chess 3*. Hrsg. von M. R. B. Clarke. Pergamon Press, 1982.

[2] International Federation of Robotics. *Service Robots*. 2014. URL: http://www.ifr.org/service-robots/ (besucht am 26. 07. 2014).

[3] Arcahouse. *5 Tips on Choosing a high quality billard table*. 2014. URL: http://www.arcahouse.com/5-tips-on-choosing-a-high-quality-billiard-table.html (besucht am 28. 07. 2014).

[4] swisspool. *Die Geschichte des Billardsports*. 2014. URL: http://www.swisspool-billard.ch/billardsport.asp?rub=3 (besucht am 28. 07. 2014).

[5] swisspool. *Der Billardsport*. 2014. URL: http://www.swisspool-billard.ch/billardsport.asp (besucht am 28. 07. 2014).

[6] A. Huber. *Richtig Billard*. 2. Aufl. BLV Buchverlag, 2007. ISBN: 978-3-8354-0132-7.

[7] W. Grewatsch und M. Rosenstein. *Snooker... Billard "Made in England"*. 5. Aufl. Berlin: Weinmann Verlag, 2004. ISBN: 3-87892-061-X.

[8] ABB Robotics. *IRB 120 - Für eine flexible und effiziente Produktion*. 2014. URL: http://new.abb.com/products/robotics/de/industrieroboter/irb-120 (besucht am 28. 07. 2014).

[9] ABB Robotics. *Bedienungsanleitung Robotstudio*. a5. 2004.

[10] J. Krüsi. „SimpleTeach - Einfaches Anlernen eines Pick-and-Place Industrieroboters“. Vertiefungsarbeit. Zürcher Hochschule für angewandte Wissenschaften, 2014.

[11] IDS Imaging. *USB 2 uEye XS Industirekamera*. 2014. URL: http://de.ids-imaging.com/store/produkte/kameras/usb-2-0-kameras/ueye-xs/show/all.html (besucht am 28. 07. 2014).

[12] OpenCV. *How to Use Background Subtraction Methods*. 2014. URL: http://docs.opencv.org/trunk/doc/tutorials/video/background_subtraction/background_subtraction.html (besucht am 29. 07. 2014).

[13] Mathworks. *Find circles using circular Hough transform*. 2014. URL: http://www.mathworks.ch/ch/help/images/ref/imfindcircles.html (besucht am 29. 07. 2014).

[14] S. Hubert. *Hough-Transformation / Hough-Algorithmus*.

[15] A. Formella und D. Fellner. *Rotation mit Quaternionen*. 2005. URL: http://trevinca.ei.uvigo.es/~formella/doc/ig04/node97.html (besucht am 30. 07. 2014).

[16] A. Willms. *C++ STL: Verstehen, anwenden, erweitern*. Galileo Press, 2000. ISBN: 978-3934358201.

[17] OpenCV. *Camera calibration With OpenCV*. 2014. URL: http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html (besucht am 31. 07. 2014).

Abbildungsverzeichnis

Tabellenverzeichnis