

# Dokumentation Praxisprojekt

Dokumentation für das Praxisprojekt: "Munchkin Tutorial"

*Praxisprojekt* im Studiengang Informatik

an der Fakultät für Informatik

der Technischen Hochschule Köln

vorgelegt von: Jonas Benjamin Lüben  
Matrikel-Nr.: 111 032 30  
Adresse: Herreshagener Str. 13  
51643 Gummersbach  
[ben@lueben.net](mailto:ben@lueben.net)

eingereicht bei: Prof. Dr. Dietlind Zühlke

Gummersbach, 09.05.2021

## **Abstract**

Das Ziel dieses Projektes war es ein mobiles Tutorial für das Kartenspiel Munchkin auf dem Android Betriebssystem zu entwickeln. Durch verschiedene Probleme bei der Implementierung für Android wurde das Ziel dann aber auf ein Proof of Concept auf Basis von C++ geändert. Für die Implementierung wurden OpenCV und Aruco Marker benutzt. Das Ergebnis am Ende des Projektes ist ein Proof of Concept mit OpenCV und Aruco Markern in C++. Zum Endstand des Projektes ist es möglich dem Nutzer eine grobe Anleitung und Hilfestellung für Munchkin zu geben und ihn durch verschiedene Züge bis hin zum Sieg zu geleiten. Dem Nutzer werden je nach Karte Hinweise angezeigt wie er auf die Karte reagieren muss und was seine weiteren Möglichkeiten sind.

# Inhaltsverzeichnis

<b>1 Einführung</b>	<b>4</b>
1.1 Munchkin . . . . .	4
1.2 Grundidee des Projektes . . . . .	5
1.3 Layout der Doku . . . . .	6
<b>2 Vergleichbare Projekte</b>	<b>7</b>
2.1 Table Top Simulator . . . . .	7
2.2 Munchkin Quacked Quest . . . . .	8
2.3 Virtual Reality . . . . .	8
2.4 Augmented Reality . . . . .	9
<b>3 Grundlagen</b>	<b>10</b>
3.1 Munchkin . . . . .	10
3.1.1 Karten . . . . .	10
3.1.2 Spielablauf . . . . .	13
3.2 OpenCV . . . . .	14
3.3 Kamera Kalibrierung . . . . .	15
3.3.1 Extrinsiche Parameter . . . . .	15
3.3.2 Intrinsische Parameter . . . . .	15
3.3.3 Kameramatrix . . . . .	16
3.3.4 Schätzung der Parameter . . . . .	17
3.4 Aruco Marker . . . . .	18
<b>4 Vorbereitende Arbeiten</b>	<b>19</b>
4.1 Versuchsaufbau . . . . .	19
4.2 Integration von OpenCV . . . . .	20
4.3 Marker Generierung . . . . .	20
4.4 Marker Erkennung . . . . .	21

<b>5</b>	<b>Programmlogik</b>	<b>23</b>
5.1	Hauptschleife . . . . .	23
5.2	Projektarchitektur . . . . .	25
5.3	Implementierung der Karten . . . . .	26
<b>6</b>	<b>Recap, Probleme und Lösungen</b>	<b>27</b>
6.1	Probleme mit OpenCV und Android . . . . .	27
6.2	Probleme bei der Implementierung . . . . .	29
<b>7</b>	<b>Fortführung Bachelor Arbeit</b>	<b>30</b>
<b>8</b>	<b>Zusammenfassung</b>	<b>30</b>

# 1 Einführung

In diesem Kapitel geht es um die Einführung in das Projekt. Hier soll ein kurzer Überblick über das Kartenspiel Munchkin und die grundlegende Idee hinter dem Projekt erläutert werden.

## 1.1 Munchkin

Munchkin [Jac03] ist ein satirisches Kartenspiel, welches ursprünglich die subkulturelle Szene der Fantasie-Rollenspiele parodiert hat. Der Name Munchkin stammt ebenfalls aus dieser Szene und ist als abwertender Begriff für Spieler gedacht, die in Rollenspielen ihren Fokus darauf legen, möglichst schnell einen möglichst starken Charakter mit der besten Ausrüstung zu erhalten, ohne dabei auf die erzählerischen Aspekte des Spieles einzugehen.



(a) Original Munchkin

(b) Munchkin Cthulhu

(c) Munchkin Apokalypse

Abbildung 1: Verschiede Versionen von Munchkin Quelle: [Jac03] [Jac13] [Jac16]

Inzwischen genießt das Spiel in seinen Spielerkreisen einen Kultstatus und es gibt unzählige offizielle und inoffizielle Erweiterungen zu anderen Genres der Popkultur (zB. Piraten, Cthulhu, Science-Fiction, etc).

Das Spielprinzip ist dabei wie folgt: Jeder Spieler steigt als menschlicher Munchkin der Stufe 1 in ein Verlies herab. Als einzige Ausrüstung hat er hierfür seine Startkarten, die aus 4 Tür- und 4 Schatzkarten bestehen, dabei. Seine Aufgabe ist es sich durch das Verlies zu kämpfen, indem er zu Beginn eines jeden Zuges eine Tür öffnet hinter der viele Gefahren lauern. Meistens lauern hinter diesen Türen Monster die er bekämpfen muss, aber auch Flüche oder andere Ereignisse können auf Ihn wirken.

Wenn der Spieler erfolgreich ein solches Monster besiegen kann, steigt er mindestens eine Stufe auf und erhält Schatzkarten, die dann neue Ausrüstung darstellen mit denen er stärker wird. Ziel des Spieles ist es, als erster Munchkin die Stufe zehn zu erreichen. Das kann entweder durch reinen Kampf geschehen oder aber auch durch die so genannten Steige-eine-Stufe-auf Karten kurz SESAs. Wichtig hierbei ist, dass die finale Stufe normalerweise nur durch einen Kampf erreicht werden kann. Das sorgt auch dafür, dass die anderen Spieler ihr möglichstes versuchen, um dies zu verhindern. Andere Spieler können zum Beispiel durch eigene Flüche oder ähnliches den Kampf beeinflussen und den kämpfenden Munchkin daran hindern den Kampf zu gewinnen.

## 1.2 Grundidee des Projektes

Oftmals werden neue Spieler dazu überredet Munchkin einmal auszuprobieren, was durch die komplexen Spielmechaniken leider meist dazu führt, dass man entweder den Großteil des Abends damit verbringt, den neuen Spieler anzulernen oder aber dieser am Tisch sitzt und einfach nicht weiß was er mit den Karten auf seiner Hand und den Karten die vor ihm auf dem Tisch liegen anfangen soll.

Die Idee des Projektes ist es, eine Art Tutorial für neue Spieler zu entwickeln, das dabei helfen soll die Spielmechaniken möglichst einfach zu erklären und den Spieler durch seine ersten Munchkin Runden zu begleiten.

Der erste Ansatz dafür war, über Projection Mapping mit einem Beamer eine Art Spielfeld auf einen Tisch zu projizieren, auf dem dann die Karten in vorgezeigte Bereiche gelegt werden sollten. Diese Karten sollten dann durch eine Kamera und einen Erkennungsalgorithmus erkannt und ausgewertet werden und dem Spieler gezeigt werden, was der beste Zug wäre.

Das Problem bei dieser Herangehensweise war nicht nur, dass durch die zu diesem Zeitpunkt herrschende Corona Pandemie der Verleih der dafür nötigen Ausrüstung nicht möglich war. Sondern auch, dass diese Herangehensweise für die Praxis eher umständlich umzusetzen gewesen wäre. Für diese Art der Umsetzung bräuchte man nicht nur zum Teil sehr teures Equipment, sondern auch viel Platz um eben dieses Equipment aufzubauen.

Mit Absprache der Betreuer des Projektes wurde die Idee erarbeitet, das Tutorial für eine Androidumgebung umzusetzen in Verbindung mit Augmented Reality kurz AR. Dadurch wäre es nicht nur möglich ein sehr kleines und einfaches Setup mit einem Smartphone und einem Dreibein zu benutzen, sondern auch dem Spieler auf dem Bildschirm seines Smartphones direkt Hinweise zu seinen Karten zu geben. Außerdem wäre es so auch möglich, dass mehrere Einsteiger in der gleichen Runde ihre Smartphones mit der App benutzen können um ihre eigenen Anleitungen zu bekommen.

### **1.3 Layout der Doku**

Die Dokumentation gliedert sich wie folgt. Zuerst wird es einen Einblick in vergleichbare Projekte geben. Darauf folgt eine Einführung in die Spielweise von Munchkin um dem Leser einen Überblick zu geben, worauf bei der Umsetzung des Projektes geachtet wurde. Danach folgt eine kurze Erklärung zu der im Projekt verwendeten Technik beziehungsweise den verwendeten Programmzbibliotheken und wieso diese für das Projekt gewählt wurden. Anschließend gibt es einen detaillierteren Einblick in die Architektur sowohl softwareseitig als auch technisch.

Im Anschluss folgt die Diskussion der wichtigsten Implementierungsdetails. Abschließend werden noch einmal die Probleme und Lösungen im Laufe des Projektes und die Möglichkeiten das Projekt im Rahmen einer Bachelorarbeit fortzuführen erörtert.

## 2 Vergleichbare Projekte

Um zu prüfen, ob es bereits vergleichbare Arbeiten gibt mit denen man das Projekt gegebenenfalls vergleichen kann, gibt es im folgenden Abschnitt einen Überblick über ähnliche Projekte. Darauf folgt dann ein Vergleich zwischen Virtual- und Augmented Reality im Bezug auf das Projekt.

### 2.1 Table Top Simulator

Ein bekanntes Projekt zur Umsetzung von Table-Top Spielen in eine virtuelle Umgebung ist der “Table Top Simulator” [TTS15]. In diesem Spiel werden viele Karten- und Brettspiele simuliert. Dabei sind viele Spiele und Erweiterungen von Nutzern umgesetzt und teilweise kostenpflichtig als zusätzliche Inhalte verfügbar. Es ist auch möglich diese Spiele als Virtual Reality Spiele zu spielen.

Leider wurde die Implementierung von Munchkin und sämtlicher, im Tabletop Simulator verfügbarer, Erweiterungen nach einer Aufforderung der ursprünglichen Herausgeber von Munchkin entfernt, da es gegen die Urheberrechte verstieß. Es gibt aber noch verschiedene Videos auf der Internetplattform Youtube, die einen Einblick in die Umsetzung des Table Top Simulator ermöglichen. So zum Beispiel ein Video auf dem deutschsprachigen Youtube Channel Gorobai [Gor19] in dem gezeigt wird, wie drei Spieler den Table Top Simulator verwenden, um zusammen eine Partie Munchkin zu spielen.

Dabei ist schnell zu erkennen, dass es sich dabei um eine reine Implementierung des Spieles auf einem Spielfeld handelt. Die Karten sind je nach Edition vorgegeben und man kann online mit seinen Freunden Munchkin spielen so als wenn man mit ihnen an einem Tisch sitzen würde. Das Spielfeld hilft dabei, indem es verschiedene Felder für Ausrüstung, Rassen und Klassen gibt. Außerdem gibt es am linken Rand neben den einzelnen Feldern für die Karten ein kleines Diagramm auf dem das Prinzip eines einzelnen Zuges in Munchkin aufgezeigt ist.



Abbildung 2: Table Top Simulator Quelle: [Gor19]

Daran kann sich der Spieler orientieren, in welcher Phase des Zuges er sich aktuell befindet und was seine weiteren möglichen Schritte in diesem Zug sein können. Darüber hinaus gibt es keinerlei Hinweise für den Spieler. Jeder muss selbst das grundlegende Wissen mitbringen, welche Karten gezogen werden müssen, was welche Karte bewirkt, welche Einschränkungen es gibt, was das Ziel des Spiels ist etc. Zudem gibt die Implementierung keine Möglichkeit die einzelnen Züge der Spieler zu prüfen, ob diese nach dem Munchkin Regelwerk möglich wären. Dadurch ergibt sich zwar eine gute Möglichkeit Munchkin auch online zu spielen, aber als Tutorial für neue Spieler ist sie ungeeignet.

## 2.2 Munchkin Quacked Quest

Ein weiteres Projekt was in eine Richtung Tutorial geht wäre das von offizieller Munchkin Seite entwickelte Spiel Munchkin Quacked Quest [MQQ19]. Munchkin Quacked Quest ist ein offizielles Videospiel zum Thema Munchkin. Allerdings basiert dieses auf dem Brettspiel Munchkin Quest [Jac19], welches eine komplett andere Spielmechanik als das Kartenspiel aufweist. Dadurch ist auch Munchkin Quacked Quest wieder nur eine andere Version von Munchkin und auch nicht als Tutorial für das eigentliche Kartenspiel geeignet.

## 2.3 Virtual Reality

“Virtual Reality” (kurz VR) findet in immer mehr Bereichen unseres Lebens ihre Anwendung. Sei es nun im Bereich der Videospiele, bei der Planung von Einrichtungen oder auch bei der Besichtigung von Immobilien oder Autos. Auch ist die Technik so weit fortgeschritten, dass die Dimensionen der benötigten Brille und die Leistungsanforderungen für VR inzwischen kein großes Problem mehr darstellen. Zwar ist dies immernoch für leistungintensive Bereiche wie den Gaming Bereich nötig. Für andere Bereiche wie zum Beispiel Besichtigungen oder das immersive Eintauchen in verschiedene Videos oder Filme reicht jedoch inzwischen das eigene Smartphone, eine passende Halterung und ein paar Kopfhörer um in eine virtuelle Realität einzutauchen.

Für die Umsetzung eines Tutorials für ein Kartenspiel ist diese Technik aber nur bedingt sinnvoll. Zwar kann der Spieler dadurch sehr gut in das Spiel eintauchen, aber für den Einsatzbereich an einem Tisch umrundet von Mitspielern ist es sehr unpraktisch wenn man von diesen Mitspielern nichts mehr sieht oder hört. Auch die Steuerung wäre ein großes Problem da diese umgesetzt und angepasst werden müsste. Aktuell ist es nahezu unmöglich mit den verfügbaren Controllern physikalische Karten zu manipulieren. Die Implementierung der Karten wäre auch aufwendig da diese visuell in das Tutorial integriert werden müssten.

## 2.4 Augmented Reality

Eine Alternative ist die Augmented Reality die “erweiterte Realität”. Sie bietet die Möglichkeit zum Beispiel über das Display eines Smartphones und dessen Kamera dem Nutzer zusätzliche Informationen anzuzeigen. Hierbei ist meistens weniger Hardware Leistung nötig und auch das Setup ist sehr viel kompakter, da zu dem Smartphone lediglich ein Dreibein oder eine ähnliche Befestigungsmöglichkeit nötig ist um ein möglichst wackelfreies Bild zu erhalten. Diese Technik wird aktuell zum Beispiel in der Navigation oder auch bei Einrichtungsplanern verwendet, bei denen der Nutzer die Kamera in sein Zimmer halten und nach belieben Möbel hinzufügen oder entfernen kann [ROOMLE]. Hierbei handelt es sich anders als bei der virtuellen Realität wie der Name schon sagt um eine Erweiterung des Umfeldes mit der der Nutzer interagieren kann.

Für das Tutorial eignet sich also AR deutlich besser als die VR da es, wie schon vorher erwähnt, deutlich einfacher für den Nutzer ist eine App auf einem Smartphone einzurichten und zu verwenden. Außerdem können auch mehrere anwesende Spieler ihre eigenen Smartphones verwenden um sich von der App Hilfe geben zu lassen. Das Gemeinschaftsgefühl bleibt dabei aber bestehen, da man immernoch miteinander am gleichen Tisch sitzt und sich direkt sehen und hören kann.

### 3 Grundlagen

Um einen kurzen Überblick über Munchkin, OpenCV und Aruco Marker zu geben wird in diesem Kapitel auf die Grundlagen der einzelnen Punkte eingegangen. Es wird erklärt, wie das Kartenspiel Munchkin funktioniert und wie ein Spielzug aussehen kann. Dabei wurde die Logik des Spiels auf die Aspekte vereinfacht, die zum aktuellen Stand im Projekt implementiert sind. Darauf folgt eine kurze Einführung zu OpenCV und den Aruco Markern.

#### 3.1 Munchkin

Der folgende Abschnitt beschreibt die verschiedenen Karten und den Spielablauf von Munchkin. Hierbei handelt es sich aber nur um eine grobe Beschreibung mit den nötigsten Mechaniken und dem aktuellen Stand der Implementierung des Prototypen. Der genaue Spielablauf sowie sämtliche Funktionen der verschiedenen Karten kann im offiziellen Regelwerk nachgelesen werden<sup>1</sup>.

##### 3.1.1 Karten

Nachdem es in dem Abschnitt 1 schon einen kurzen Einblick zu Munchkin gab, folgt nun eine Erklärung zum Spielverlauf zu Munchkin. In Munchkin gibt es zwei verschiedene Arten von Spielkarten: Die Türkarten und die Schatzkarten.



(a) Türkarte

(b) Schatzkarte

Abbildung 3: Kartenarten Quelle: Spielkarten abgescannt aus [Jac03]

---

<sup>1</sup>Dieser Abschnitt basiert auf [Jac03], verfügbar unter [https://pegassusshop.de/media/pdf/41/85/43/4250231703669\\_de.pdf](https://pegassusshop.de/media/pdf/41/85/43/4250231703669_de.pdf)

## Schatzkarten

Die Schatzkarten erhält ein Spieler meistens nachdem er ein Monster besiegt hat und man kann sie in drei verschiedene Kategorien einordnen.

Als Gegenstände versteht man in Munchkin alles mögliche an Ausrüstung. Gegenstände dienen in Munchkin dazu dem Spieler zusätzliche Stärke im Kampf gegen Monster zu geben. Zu den meisten Arten von Gegenständen gibt es aber Einschränkungen wieviele Gegenstände des Types ausgerüstet werden dürfen. Zum Beispiel kann ein Munchkin nur eine Rüstung tragen.

Manche Gegenstände bieten aber auch eine besondere Funktion für das Spiel. So gibt es zum Beispiel die Karte “gezinkter Würfel” mit der man ein beliebiges Würfelergebnis auf eine Zahl ändern kann. Dadurch kann zum Beispiel ein Fluchtversuch gelingen, indem das Würfelergebnis auf die passende Zahl geändert wird.



Abbildung 4: Karten im Detail Quelle: Spielkarten abgescannt aus [Jac03]

Zusätzliche zu den Gegenständen gibt es noch die Gegenstandsverstärker. Diese werden an den gewünschten Gegenstand angelegt und geben diesem noch mehr Stärke im Kampf gegen Monster. Dabei ist wichtig daran zu denken, dass wenn man einen Gegenstand verliert auch der daran angebrachte Gegenstandsverstärker verloren geht.

Als letzte Art der Schatzkarten gibt es noch die “Steige eine Stufe auf” Karten kurz SESAs. Diese ermöglichen dem Spieler, wie der Name schon sagt, eine Stufe aufzusteigen. Aber auch hier gibt es wieder Einschränkungen. Ein Munchkin kann mit Hilfe von SESAs maximal bis zur Stufe neun aufsteigen und muss teilweise zusätzlich zum Aufstieg eine Aufgabe erfüllen.

## **Türkarten**

Eine Türkarte erhält der Spieler indem er zu Beginn seines Zuges eine solche offen zieht und sie “eintritt”. Außerdem kann der Munchkin, sollte er in seiner Runde gegen kein Monster gekämpft haben, eine zweite Türkarte verdeckt ziehen und der Effekt der Karte trifft ihn nicht. Türkarten lassen sich wiederum in 5 verschiedene Kategorien einteilen.

Rassen und Klassen sind zwar zwei verschiedene Kategorien funktionieren aber auf die gleiche Art und Weise. Sie bieten dem Spieler verschiedene Boni können aber auch Nachteile im Kampf gegen Monster haben.

Eine weitere Kategorie sind die Joker. Sie werden meist in Kämpfen eingesetzt und können für eine der beiden Seiten positive oder negative Auswirkungen haben. Meist werden solche Karten von anderen Munchkins gewirkt um zum Beispiel einen Munchkin daran zu hindern einen Kampf zu gewinnen der den Sieg für den Munchkin bedeuten würde.

Flüche sind eine Kartenkategorie die den Munchkin trifft wenn er ihn vom Türstapel zieht oder ein anderer Munchkin einen Fluch auf ihn spielt. Flüche haben verschiedene negative Effekte auf den Munchkin und können entweder einen einmaligen Effekt oder einen dauerhaften Effekt haben.

Als letzte Kategorie der Türkarten fehlen noch die Monster, die schon einige Male ange- sprochen wurden. Monster müssen vom Munchkin bekämpft werden und falls der Munchkin verliert wird er von den “schlimmen Dingen” des Monster getroffen. Diese sind vergleichbar mit den Flüchen.

### 3.1.2 Spielablauf

Nun folgt eine grobe Beschreibung des Spielablaufs. Der genaue Ablauf kann auch im offiziellen Regelwerk zu Munchkin nachgelesen werden. Zu Beginn des Spiels bekommt jeder Munchkin jeweils vier Tür- und Schatzkarten welcher er verdeckt auf die Hand nimmt. Diese Karten sind seine Startausstattung und können zu Anfang des Zuges angelegt werden und jeder Munchkin startet mit seinem biologischen Geschlecht. Ein Spielzug besteht aus fünf Phasen.

**Phase 1: Tür eintreten** In der ersten Phase tritt der Munchkin eine Tür ein, indem er eine Türkarte vom Türstapel nimmt. Auf diese Karte wird dann reagiert bevor es zu anderen Phasen kommt. zieht er ein Monster wird zur Kampfphase weitergeleitet.

**Phase 2: Auf Ärger aus sein** Die zweite Phase heißt “auf Ärger aus sein” und sie gibt dem Munchkin die Möglichkeit, wenn er als erste Türkarte kein Monster gezogen hat, gegen ein Monster von seiner Hand zu kämpfen. Falls er sich dagegen entscheidet folgt direkt die dritte Phase.

**Phase 3: Raum plündern** In der dritten Phase darf der Munchkin den Raum, dessen Tür er eingetreten hat, plündern indem er eine weitere Türkarte verdeckt zieht und auf seine Hand nimmt.

**Phase 4: Milde Gabe** Danach folgt die vierte Phase. Die so genannte “milde Gabe”. Falls der Munchkin am Ende seines Zuges mehr als 5 Karten auf der Hand hat muss er so lange Karten an den oder die Munchkins mit der niedrigen Stufe abgeben bis er nur noch fünf Handkarten hat.

**Phase 5: der Kampf** Falls ein Munchkin beim eintreten einer Tür einem Monster gegenüber steht muss er gegen dieses kämpfen und für den restlichen Zug werden die Phasen zwei und drei übersprungen. Nun wird verglichen, ob die Stärke des Munchkin die des Monsters um mindestens eins übersteigt. Dabei wird die Stärke des Munchkins durch seine aktuelle Stufe und durch die zusätzliche Stärke durch seine Gegenstände errechnet.

Sollte der Munchkin es nicht schaffen das Monster zu besiegen muss er vor dem Monster fliehen um den schlimmen Dingen zu entgehen. Um erfolgreich fliehen zu können wirft der Munchkin einen Würfel. Sollte das Würfelergebnis kleiner als fünf sein so schlägt der Fluchtversuch fehl und der Munchkin wird von den schlimmen Dingen getroffen.

Das Spiel ist erst dann zuende wenn ein Munchkin die zehnte Stufe erreicht hat. Diese Stufe muss mit einem Kampf erreicht werden.

### 3.2 OpenCV

Da für die Umsetzung des Projektes eine Herangehenweise per Augmented Reality gewählt wurde, musste eine dazu passende Softwarebibliothek gefunden werden, welche die benötigten Funktionalitäten liefert.

Eine der größten Softwarebibliotheken für Computer Vision ist OpenCV (Open Computer Vision) [BK00]. OpenCV ist eine freie Programmbibliothek für Bildverarbeitung mit BSD Lizensierung und somit für Lehrende und Studierende kostenfrei.

Es kann in den Sprachen C, C++, Python und Java programmiert werden und ist komplett plattformunabhängig. Somit sollte auch eine Umsetzung für Android möglich sein. Die Stärke von OpenCV macht insbesondere die Geschwindigkeit und die Menge an Algorithmen aus die auf den neuesten Forschungsergebnissen basieren. Anwendungsbereiche sind unter anderem Bereiche der Gesichtserkennung, der Gestenerkennung und der Mensch-Computer-Interaktion. OpenCV wird auch von verschiedenen NVIDIA GPUs unterstützt unter anderem die CUDA GPUs für den mobilen Bereich.

Für dieses Projekt interessant ist vor allem, dass OpenCV direkt die Kalibrierung von Kameras unterstützt und dass über das opencvcontrib Package eine Vielzahl an Drittanbieter Software verfügbar ist. Darunter auch die Aruco Marker, die im 3D Raum verwendet werden um beispielsweise einem KI gesteuerten Roboter die Navigation ermöglichen. In diesem Projekt werden die Aruco Marker [Gar+15; RMM18] dazu eingesetzt, um die Position der Karten zu erkennen und durch die MarkerId die in einer Bibliothek hinterlegte Karte zu erhalten und zu verarbeiten. Dies vereinfacht den Prozess der Kartenerkennung immens indem nicht die Karte selbst sondern nur der Marker erkannt werden muss.

### 3.3 Kamera Kalibrierung

Für eine korrekte Berechnung der Markerposen wird ein kalibriertes Kameramodell benötigt. Das Kameramodell bildet von den Punkten im Weltkoordinatensystem auf Punkte im Sensorraum, also Pixelkoordinaten ab. Dieses Modell ist definiert durch extrinsische und intrinsische Kameraparameter. OpenCV schätzt die Parameter anhand eines Schachbrettes als Referenzobjekt mittels der Methode von Zhang [Zha00].

#### 3.3.1 Extrinsische Parameter

Die extrinsischen Parameter definieren Position und Orientierung der Kamera im Weltkoordinatensystem und bestehen aus einer Rotationsmatrix  $\mathbf{R}$  und dem Kamerapositionsvektor  $\mathbf{t}$ . Diese beiden Parameter kombiniert man in eine extrinsische Matrix  $\mathbf{M}$ :

$$\mathbf{M} = \begin{pmatrix} \mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & -\mathbf{t} \end{pmatrix} \quad (1)$$

$\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3$  sind hierbei die Spaltenvektoren der Rotationsmatrix.

#### 3.3.2 Intrinsische Parameter

Zusätzlich zu diesen Parametern werden noch die intrinsischen Parameter der Kamera benötigt. Die intrinsischen Parameter sind immer abhängig von der Kamera und enthalten Informationen wie die Brennweite ( $f_x, f_y$ ) und das optische Zentrum ( $c_x, c_y$ ). Diese Parameter können verwendet werden um die so genannte Kalibrationsmatrix  $\mathbf{K}$  für genau diese Kamera zu erzeugen. Da die Kalibrationsmatrix einzigartig für die Kamera ist muss sie für die verwendete Kamera nur einmal berechnet werden und kann danach abgespeichert und wiederverwendet werden. Sie wird durch eine 3x3 Matrix beschrieben.

$$\mathbf{K} = \begin{pmatrix} f_x & 0 & c_x + x_d(r) \\ 0 & f_y & c_y + y_d(r) \\ 0 & 0 & 1 \end{pmatrix} \quad (2)$$

Jede Kamera hat je nach Anzahl und Art der Linsen eine bestimmte Verzerrung. Hierbei unterscheidet man primär zwischen der radialen und der tangentuellen Verzerrung. Bei einer radialen Verzerrung erscheinen eigentlich gerade Linien gebogen und der Effekt wird größer je weiter man sich von dem Punkt der Bildmitte entfernt. Die radiale Verzerrung kann durch die folgenden Gleichungen angenähert werden, wobei  $r$  die Distanz vom optischen Zentrum zum Pixel ist:

$$x_{d_r} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (3)$$

$$y_{d_t} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \quad (4)$$

Zu einer tangentiellen Verzerrung kommt es, wenn die Linsen der Kamera nicht perfekt parallel zum Bild stehen. Dadurch wirken manche Bereiche des Bilds näher als erwartet. Diese Art der Verzerrung kann angenähert werden durch:

$$x_{d_t} = x + (2p_1xy + p_2(r^2 + 2x^2)) \quad (5)$$

$$y_{d_t} = y + (p_1(r^2 + 2y^2) + 2p_2xy) \quad (6)$$

Es werden also insgesamt fünf Parameter benötigt. Diese fünf Parameter sind als Verzerrungskoeffizienten bekannt.

$$\text{Distortion coefficients} = (k_1 k_2 p_1 p_2 k_3) \quad (7)$$

### 3.3.3 Kameramatrix

Mit den extrinsischen und intrinsischen Parametern lässt sich jetzt die Kameramatrix  $\mathbf{P}$  definieren:

$$\mathbf{P} = \mathbf{K}\mathbf{M} \quad (8)$$

Um jetzt die Pixelkoordinaten von einem beliebigen Punkt im Weltkoordinatensystem zu berechnen setzt man die Koordinaten  $X, Y, Z$  in folgende Gleichung ein:

$$\begin{pmatrix} x_H \\ y_H \\ w_H \end{pmatrix} = \mathbf{P} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (9)$$

Damit erhält man die homogenen Bildkoordinaten. Um die Pixelkoordinaten zu erhalten, müssen wir diese noch durch die dritte Komponente  $w$  teilen:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{x_H}{w_H} \\ \frac{y_H}{w_H} \end{pmatrix} \quad (10)$$

### 3.3.4 Schätzung der Parameter

Um die Parameter zu berechnen braucht man Passpunkte im Weltkoordinatensystem mit bekannten Koordinaten und die korrespondierenden Pixelkoordinaten dieser Punkte im Kamerabild.

Ein Schachbrett eignet sich hierfür gut da die Eckpunkte im Kamerabild einfach zu identifizieren sind. In der Methode von Zhang wird jetzt der obere linke Eckpunkt eines solchen Schachbrettes mit vordefinierten Maßen als Ursprung des Weltkoordinatensystems definiert, wodurch auch die anderen Eckpunkte festgelegt werden. Es wird davon ausgegangen dass das Schachbrett eben ist und in der XY-Ebene des Weltkoordinatensystem liegt. Das Schachbrettmuster kann einfach ausgedruckt werden und muss dann auf ein möglichst festes Objekt aufgebracht werden beispielsweise Brett oder fester Karton, damit bei der Kalibrierung das Schachbrett nicht verfälscht wird.



Abbildung 5: Links: Eingabebild zur Kalibrierung. Mitte: Lokalisierung der Passpunkte auf dem Schachbrett. Rechts: Entzerrtes Eingabebild - die Krümmung der Linien ist verschwunden. Quelle: [OCVCC]

Dadurch sind dann die Weltkoordinaten der Punkte und auf dem Bild bekannt und können verwendet werden, um die Kameramatrix zu berechnen. Für gute Ergebnisse werden hierfür mindestens zehn verschiedene Bilder benötigt. Je mehr Bilder für die Kalibrierung verwendet werden, desto präziser wird die Erkennung der Marker. Für das Projekt wurden ungefähr 50 Bilder in verschiedenen Abständen und Winkeln aufgenommen um eine präzise Erkennung der Marker zu gewährleisten.

Im Code bietet OpenCV einige Funktionen an, die benötigt werden, um die Kameramatrix zu erzeugen. Für die Kalibrierung des Projektes habe ich ein Schachbrett der Größe 9x6 verwendet, wobei die erste Reihe an Quadranten nicht für die Größe mit eingerechnet wird, sondern erst ab der zweiten Zeile gewertet wird. Die für die Kalibrierung benötigten Daten sind ein Satz von 3D Punkten in der realen Welt und diesen entsprechende Punkte im 2D Koordinatensystem des Bildes. Die 2D Punkte werden vom Schachbrett genommen und zwar immer dort wo sich zwei der schwarzen Quadrate berühren. Die 3D Punkte werden dazu von einer statischen Kamera in verschiedenen Winkeln und Positionen genommen und für jeden Punkt die Werte für X, Y und Z Koordinate geschätzt. Dadurch, dass die Größe der Quadrate bekannt ist, können alle drei Werte der Koordinaten geschätzt werden. Mit der Funktion `cv.findChessboardCorners()` werden diese Punkte zurückgegeben, wenn das Muster erkannt wurde. Die Funktion benötigt nur die Größe des Schachbretts (9x6). Dann lassen sich die Punkte mit `cv.drawChessboardCorners()` visualisieren und in einem Videostream anzeigen.

Bei der gesamten Kalibrierung ist es dem Nutzer unbekannt wieviele der Bilder wirklich “gut” waren und für die Kalibrierung verwendet werden können, deswegen sollte man immer einige Bilder mehr machen als von der OpenCV Dokumentation zur Kamerakalibrierung[OCVCC] gefordert wird. Für das Projekt wurde der Code so umgesetzt, dass per Tastendruck die Werte der einzelnen Punkte in einem Array gespeichert wurden und nachdem der Nutzer der Meinung ist, genug Bilder aufgenommen und genug Daten für die Kalibrierung zusammen zu haben, kann per anderer Taste die Berechnung der Kameramatrix gestartet werden. Dies geschieht über die Funktion `cv.calibrateCamera()` und gibt die Kalibrationsmatrix, die Verzerrungskoeffizienten, die Translations- und die Rotationsvektoren zurück, die dann gespeichert werden können.

### 3.4 Aruco Marker

Aruco Marker sind zweidimensionale quadratische und binäre Passermarken die primär in der Navigation von Robotern und dem Bereich der Augmented Reality verwendet werden [OCVAM], um Punkte im dreidimensionalen Raum zu erkennen. In diesen Bereichen ist es sehr wichtig Positionen im 3D-Raum zu schätzen. Da dieser Prozess auf Korrespondenzen zwischen Punkten in der realen Umgebung und einer Projektion auf ein 2D Bild basiert, ist es sehr kompliziert gut funktionierende Algorithmen dafür zu schreiben.

Um dieses Problem zu umgehen, werden meist synthetische Marker oder Passermarker eingesetzt. Meistens sind diese, so wie die Aruco Marker, binäre und quadratische Marken, da dies den Vorteil hat, dass schon die 4 Kanten des Markers ausreichen um die Pose, die die räumliche Lage des Markers beschreibt, für die Kamera zu bestimmen. Die innere binäre Codierung der Marker, die die ID des einzelnen Markers darstellt und aus einzelnen Bits besteht, ist sehr robust und ermöglicht auch verschiedene Techniken zum Korrigieren von Fehlern die beim Erkennen der Aruco Marker auftauchen können.

Durch den breiten schwarzen Rand des Markers ist es sehr einfach, Kandidaten für mögliche Marker zu erkennen. Innerhalb des Randes befindet sich die binäre Kodierungsmatrix welche aus schwarzen und weißen Bits besteht. Die Größe eines Aruco Markers bestimmt die Innere Matrix. So hat ein Marker der Größe 6x6 eine Matrix von 6 x 6 Bits also insgesamt 36 Bits.

Bei der Generierung eines Markers kann zusätzlich zu der Größe der inneren Matrix noch die Größe des Markers selbst in Pixeln angegeben werden. Um dann von dieser Größe die Matrix richtig zu bilden wird der Marker in gleichgroße Quadrate aufgeteilt. In diesem Beispiel würde die Pixel Anzahl also durch acht geteilt werden, da im Normalfall der Rand die Breite eines internen Bits hat. Dieser Rand wird dann komplett schwarz gefärbt und die einzelnen Quadrate im inneren entsprechend der Matrix schwarz oder weiß. Da der Marker aus Sicht der Kamera auch gedreht sein kann und dadurch die Ausrichtung der Markerkanten nicht immer gleich sind muss der Erkennungsprozess in der Lage sein auch gedrehte Marker zu erkennen. Auch dafür hilft die Kodierung der inneren Matrix.

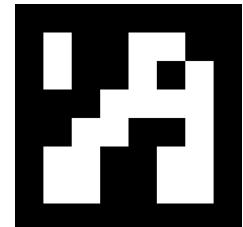


Abbildung 6: Aruco Marker

Um die Aruco Marker generieren zu können, muss man sich vorher dazu entscheiden, welche Bibliothek man verwendet. So ist auch die Geschwindigkeit zum Erkennen der Marker davon abhängig, wie groß die gewählte Bibliothek ist. Je größer die Bibliothek desto mehr Bits müssen in der Matrix untergebracht werden, wodurch der Marker selbst komplizierter wird und es dadurch für den Algorithmus schwieriger ist den Marker zu erkennen. Noch hinzu kommt, dass Marker die bei gleicher physikalischer Größe mehr Bits beinhalten nicht so robust bei der Erkennung einzelner Bits sind wie Marker die weniger Bits beinhalten. Dadurch, dass bei einem Marker mit weniger Bits bei gleicher Größe die einzelnen Bits größer sind ist es auch leichter diesen auf eine größere Entfernung zu erkennen. Eine solche Bibliothek wäre zum Beispiel die DICT\_6X6\_50. Diese Bibliothek beinhaltet 50 Marker der Größe 6x6. Die ID des Markers wird zwar über die innere Matrix erkannt, ist jedoch nicht der konvertierte numerische Wert der Matrix sondern die Position des Markers innerhalb der Bibliothek.

## 4 Vorbereitende Arbeiten

Im folgenden Abschnitt wird erläutert, wie der Versuchsaufbau des Projektes aussieht und die Integration von OpenCV und die Generierung der Aruco Marker im Projekt funktioniert und stellt somit die Software Seite des Projektes dar.

### 4.1 Versuchsaufbau

Die Architektur des Projektes setzt sich aus der verwendeten Technik und der Software zusammen. In diesem Fall besteht die technische Seite nur aus einer Webcam die mit einer Halterung an einem Bildschirmarm befestigt wurde um einen möglichst stabilen Winkel zu ermöglichen. Zudem wurden Aruco Marker ausgedruckt und auf die Munchkinkarten aufgebracht.



Abbildung 7: Webcam Setup

## 4.2 Integration von OpenCV

Um OpenCV in einem Projekt benutzen zu können muss dieses zuerst eingerichtet werden und als Bibliothek zum Beispiel in Visual Studio hinterlegt werden. Da für dieses Projekt die Aruco Marker aus dem contrib Package [OCVCon] von OpenCV benötigt werden und es für dieses Paket keine direkte Quelle gibt muss das Paket zuerst kompiliert werden. Dazu muss sowohl das OpenCV Paket als auch das dazugehörige contrib Paket heruntergeladen werden. Danach muss das Paket kompiliert werden. Dazu rät es sich eine grafische Oberfläche zu verwenden um die Makefiles zu generieren. In diesem Fall wurde die CMake GUI verwendet.

Nachdem man das zu kompilierende Paket von OpenCV ausgewählt und die Attribute generiert hat, kann man diese nach Belieben verändern und für einzelne Pakete oder auch Beispielprojekte auswählen ob diese im fertigen Build integriert sein sollen oder nicht. Hier muss dann auch der Pfad zum contrib Paket hinzugefügt werden, damit auch diese Pakete mitgeladen werden. Nach erneutem Generieren der Attribute können diese dann noch passend zum contrib Paket angepasst werden.

Wenn alles soweit ausgewählt ist, kann man die Makefiles für verschiedene Build Engines generieren. In diesem Fall habe ich mich für Visual Studio entschieden. Nachdem dann die Makefiles für Visual Studio generiert wurden, kann man diese in Visual Studio ausführen. Dabei ist zu beachten, dass noch einige Einstellungen verändert werden müssen, bevor alle Makefiles gebaut werden können. Diese Informationen kann man aber alle aus der OpenCV Dokumentation entnehmen. Nachdem dann der build korrekt kompiliert wurde, muss dieser noch in Visual Studio integriert werden. Dafür ist es für Windows 10 Systeme ratsam, den Ordner des Builds auf der C Partition zu speichern und der Systemvariable PATH hinzuzufügen [OCVI]. Dann müssen noch in einem neu erstelltem Visual Studio Projekt alle OpenCV Pakete einzeln als Bibliotheken hinterlegt werden um diese dann in diesem Projekt zu verwenden.

## 4.3 Marker Generierung

Für dieses Projekt war es auch wichtig die richtige Größe der Marker zu wählen. Es musste ein Kompromiss gefunden werden zwischen einer guten Erkennbarkeit des Markers und dem Platz, der durch den Marker auf der Spielkarte verdeckt wird. Dem Spieler sollte es noch möglich sein, alle wichtigen Texte auf der Karte lesen zu können. Hierfür wurden zuerst Marker in vier verschiedenen Größen ausgedruckt und dann getestet wie gut diese in verschiedenen Entfernung und Winkeln zur Kamera erkannt wurden. Der größte Marker hatte eine Kantenbreite von 4cm und für die anderen drei Marker wurde die Breite jeweils halbiert.

Der Marker mit 2cm Breite wurde dabei noch sehr gut erkannt und der Marker der 1cm breit war nur noch sehr schlecht. Daraufhin wurde dann noch ein Marker ausgedruckt der 1,5cm breit war. Da die Rate, mit der dieser Marker erkannt wurde, zufriedenstellend war, wurde diese Größe für das weitere Projekt verwendet. Um die einzelnen Spielkarten zu schützen wurden diese in Schutzhüllen gesteckt und der dazugehörigen Marker mit doppelseitigem Klebeband an eine Position auf der Karte geklebt, die am wenigsten verdeckt.

Um dann die Marker in der gewählten Größe im Projekt verwenden zu können, müssen diese für das Projekt generiert werden. Dafür bietet jede OpenCV Version in der Aruco Marker integriert sind, Funktionen um die Aruco Marker passend für die OpenCV Version zu generieren. Hierfür wird die `cv::aruco::drawMarker()` Funktion verwendet.

Die Funktion nimmt 5 Parameter an. Als erstes die Bibliothek aus der die Marker generiert werden sollen zum Beispiel `DICT_6X6_50`. In dieser Bibliothek befinden sich 50 Aruco Marker der Dimension 6x6 also insgesamt 36bits pro Marker. Als zweiter Parameter wird die `MarkerId` angegeben für den Marker der generiert werden soll. Die angegebene `MarkerId` muss auch innerhalb der Bibliothek liegen, darf also in diesem Beispiel nur zwischen 0 und 49 liegen, da sonst ein Fehler zurück gegeben werden würde. Als dritter Parameter wird dann die Größe des Markers in Pixel x Pixel angegeben. Hierbei ist zu beachten, dass die Größe mindestens der Größe des Markers plus zwei, für die Grenzen des Markers, betragen muss.

Für eine bessere Erkennung des Markers ist es sinnvoll die Größe des Markers deutlich größer zu wählen als die Größe der inneren Matrix um Deformationen zu vermeiden. Der vierte Parameter ist dann der Name des ausgegebenen Bildes. Der fünfte Parameter ist optional und gibt die Breite der Umrandung des Markers an. Dabei ist der angegebene Wert immer die Anzahl an internen Bits die als Rand verwendet werden. Der Standardwert ist 1 und wenn man zum Beispiel 2 als Wert eingibt wird ein 2 Bits großer schwarzer Rand um den Marker gelegt.

#### 4.4 Marker Erkennung

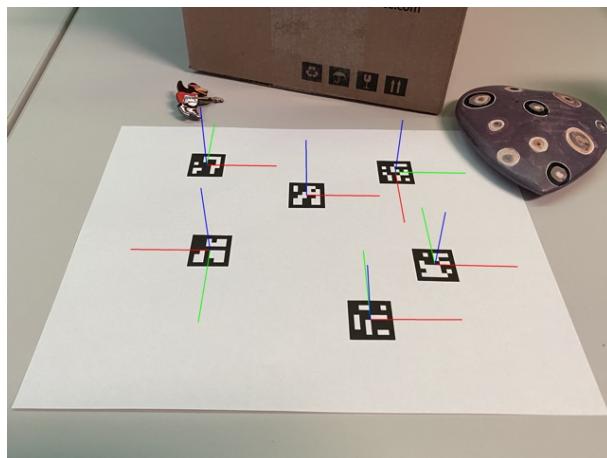
Um einen solchen Marker auf einem einzelnen Eingabebild zu erkennen zu können, zwei Hauptschritte verwendet [OCVAM]. Im ersten Schritt werden alle Kandidaten, die einen Aruco Marker darstellen könnten, aufgelistet. Dafür wird das Bild analysiert und alle Objekte die einem Quadrat ähnlich sind werden als Kandidaten gespeichert. Dann wird mit diesen Kandidaten ein adaptiver Schwellenwert gebildet und dieser dann auf die Konturen der Kandidaten angewendet. Dadurch werden dann alle Kandidaten extrahiert wobei die Kandidaten die nicht konvex oder kein Quadrat sind aussortiert werden.

Im zweiten Schritt werden dann die übrigen Kandidaten bewertet. Hier wird die innere Kodierung der Marker verwendet, um Kandidaten auszuschließen. Damit das funktioniert, werden zuerst die Marker Bits jedes Markers extrahiert. Hierzu wird zuerst eine perspektivische Transformation angewendet um die perspektivische Verzerrung des Markers zu entfernen und ein echtes Quadrat zu erhalten. Solch eine Verzerrung kann zum Beispiel durch einen flachen Kamerawinkel entstehen. Auf diese wird dann das Schwellenwertverfahren von Otsu angewendet [Ots79] bei dem das Maß an Streuung der Grauwerte verwendet wird. Mit dieser Streuung wird versucht einen Schwellenwert zwischen Zwei Klassen, in diesem Fall zwei Bits, zu finden, bei dem die Streuung innerhalb der Klasse möglichst klein aber zwischen den beiden Klassen möglichst groß ist. Hierfür wird der Quotient zwischen den beiden Varianzen gebildet und ein Schwellenwert gesucht, bei dem dieser möglichst groß ist.

Nachdem dieser Schwellenwert gefunden wurde, wird dann ein Raster über das Bild gelegt, welches der Größe der einzelnen Bits entspricht. Dann werden innerhalb dieses Rasters die schwarzen und weißen Pixel gezählt und dadurch entschieden ob es sich um ein schwarzes oder ein weißes Bit handelt. Nachdem dann die innere Matrix erstellt wurde, wird diese mit der entsprechenden Bibliothek abgeglichen, um zu bestimmen ob der Kandidat ein Marker aus der verwendeten Bibliothek ist. Die dadurch entstandenen Marker werden dann in einer Liste zurückgegeben und erhalten zwei Parameter. Die Position aller vier Ecken des Markers und seiner ID.

Um die Pose des Markers in Relation zur Kamera zu erhalten wird zuerst die Pose der Kamera mithilfe der Eigenschaften der Kamera aus der Kamerakalibrierung geschätzt. Diese Eigenschaften sind durch die Kameramatrix und die Verzerrungskoeffizienten gegeben. Die Kamera Pose aus der Perspektive der Marker ist dann eine 3D Transformation des Koordinatensystems des Markers und des Koordinatensystems der Kamera und wird durch die Rotations- und Translationsvektoren spezifiziert.

Durch die Funktion `cv::aruco::estimatePoseSingleMarker()` werden dann die Rotations- und Translationsvektoren zurück gegeben und mit der Funktion `cv::aruco::drawAxis()` kann man die Achsen der Marker visualisieren. Hierbei wird die x-Achse rot, die y-Achse grün und die z-Achse blau gefärbt. Dadurch wird auch die Rotation der Marker ersichtlich.



(a)

Abbildung 8: Marker mit gezeichneten Achsen. Quelle: [OCVAM]

## 5 Programmlogik

Um die Programmlogik zu erläutern wird der folgende Abschnitt in die Hauptschleife des Programmes, die Projektarchitektur und die Implementierung der Karten aufgeteilt. Zur Veranschaulichung wurde ein Klassendiagramm und ein Flowchart angefertigt auf dem die Erläuterung der Implementierung basiert. Der Quellcode zum Projekt wird via GitHub zur Verfügung gestellt<sup>2</sup>.

### 5.1 Hauptschleife

In Abb. 9 wird die Hauptschleife des Programms in einem Flowchart dargestellt. In jedem Durchlauf wird abgefragt ob ein neuer Frame von der Kamera verfügbar ist. Nachdem ein Frame der Kamera geladen wurde wird die Erkennung der Marker durchgeführt und die dazu gehörende Grafik gerendert. Falls eine Eingabe des Nutzers erforderlich ist, wird diese über die Buttons angefordert. Ansonsten wird die Gamelogik durchgeführt. Nach jeder Runde wird überprüft, ob der Nutzer das Spiel gewonnen oder beendet hat.

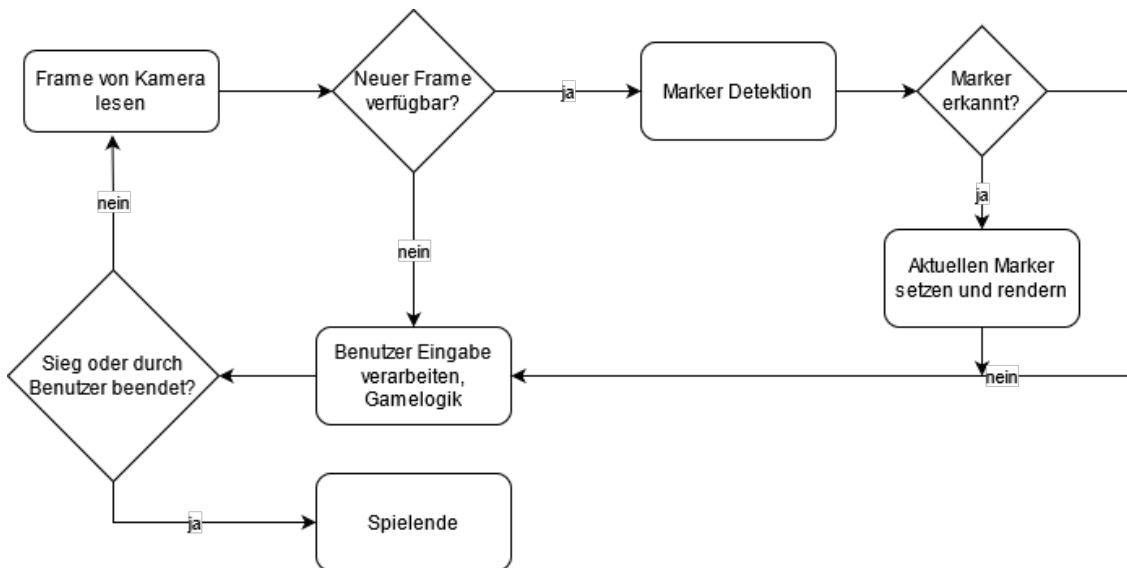
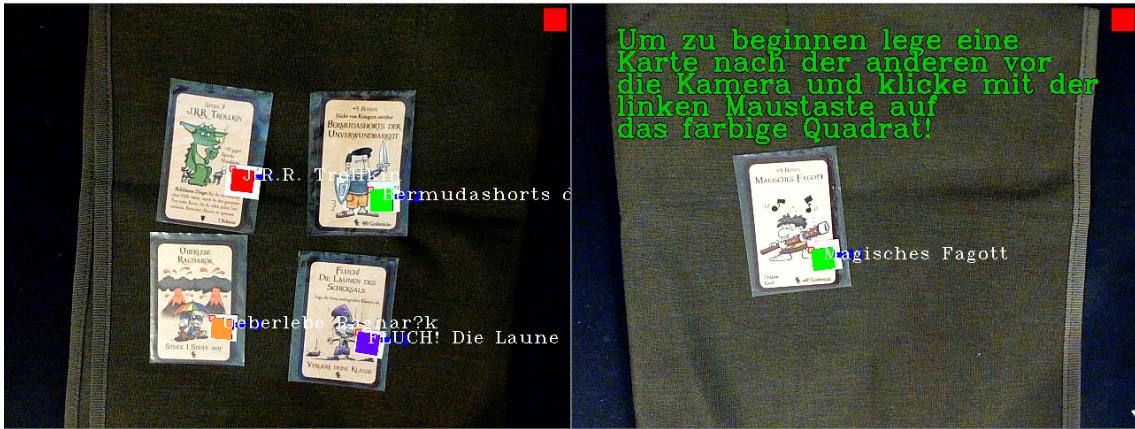


Abbildung 9: Flowchart des Programmablaufs

Für die Umsetzung des Projektes wurde, nachdem die Kamera kalibriert wurde und 50 Aruco Marker der Größe 4X4 generiert und ausgedruckt wurden, wie folgt vorgegangen. Das Videosignal der Webcam wird abgegriffen, jedes Frame geladen und bearbeitet. Hierfür wird eine while Schleife verwendet, die so lange läuft, bis entweder Escape gedrückt wird um das Programm zu beenden, oder aber auf Grund eines Fehlers die Funktion kein Frame vom Videosignal erhält. In jedem Frame wird dann nach Aruco Markern gesucht. Wenn im Frame mindestens ein Aruco Marker gefunden wurde, wird für die gefundenen Aruco Marker eine Funktion aufgerufen, die die Farbe der einzelnen Aruco Marker anhand des Kartentyps, welchen die MarkerId repräsentiert, auswählt. Diese Farbe wird dann zurück gegeben um auf den Marker ein Quadrat in dieser Farbe zu zeichnen.

<sup>2</sup>GitHub Repository erreichbar unter: <https://github.com/BLU1337/PPMunchkinTut>



(a) verschiedene Farben der Polygone

(b) Tutorialtext

Um das Quadrat zu erhalten wird ein Polygon aus den vier Marker Ecken gebildet. Zusätzlich wird das Zentrum des Polygons errechnet und als Mittelpunkt gespeichert. Dieser Mittelpunkt wird verwendet um den Namen der Karte in den Frame zu schreiben beginnend am Mittelpunkt des Markers. Zudem wurde der Text für das Tutorial im oberen Bereich des Frames angezeigt. Um die Sichtbarkeit der Texte auf den Frame zu verbessern, wurde der Text zuerst in doppelter Größe und in schwarz gerendert. Dann wurde der gleiche Text an die gleiche Stelle in der halben Größe und einem hellen Grün gerendert. Dadurch wird verhindert, dass der Text entweder nur gut auf hellen oder nur gut auf dunklen Bereichen des Bildes zu lesen ist.

Für die Umsetzung der Gamelogik wird für die Verarbeitung und Speicherung der wichtigen Parameter für das Spiel ein Gamestate Objekt angelegt in dem alle wichtigen Events und Parameter gespeichert werden. Wenn nun auf einen Marker geklickt wird speichert die Callback Funktion nur den aktuellen Klick in das Gamestate und die ID des vor der Kamera liegenden Markers.



(a) Veränderter Tutorialtext nach Anlegen von Ausrüstung

(b) Spiel gewonnen

Am Ende jedes Frames wird eine update Funktion aufgerufen. Diese Funktion liest dann die Karte aus der Bibliothek mit der MarkerId die im Gamestate steht und startet dann die Verarbeitung. Hierfür wird nun der Kartentyp der Karte gelesen und die dazugehörige Funktion zur Verarbeitung aufgerufen. Da die Funktionen in einer eigenen Datei ausgelagert sind wird sie auf den einzelnen Kartentyp gemappt und dann ausgeführt.

Jeder Funktion wird hierfür ein Pointer auf das Gamestate und auf die Karte übergeben. Für den Fall, dass ein Input des Nutzers benötigt wird ändert die Funktion einen der Parameter des Gamestates und gibt diesen zurück und aktiviert einen Button der dem Spieler angezeigt wird. Zudem wird der Tutorialtext verändert und dem Nutzer wird gezeigt, was sein nächster Schritt ist. Wenn der Nutzer dann den grünen Button drückt wird dies wieder registriert und die Verarbeitung der Logik wird an dem Punkt fortgeführt, an dem sie pausiert wurde. Durch verschiedene Parameter ist es auch möglich, an mehreren Stellen in der Verarbeitung zu pausieren und auf eine Eingabe des Nutzers zu warten.

Auch die Schlimmen Dinge eines Monsters oder die Effekte von Flüchen erhalten Funktionen in einer separaten Klasse die auch auf die Karte gemappt werden.

Nachdem der aktuelle Zug bearbeitet wurde überprüft die Funktion, ob der Spieler das Spiel gewonnen hat und gibt dies dann als Text aus.

## 5.2 Projektarchitektur

Bezüglich der Architektur des fertigen Projektes ist ein Augenmerk auf objektorientierte Programmierung gelegt worden, damit der Code möglichst übersichtlich bleibt und man für einzelne Änderungen nicht unnötig viel Code verändern muss. Dafür ist die Logik des Tutorials für die einzelnen Kartenarten in verschiedene Funktionen ausgelagert, die dann auf die einzelnen Karten gemappt werden. Sämtliche Parameter, die für den Spieler benötigt werden um die verschiedenen Stati des Spieles zu durchlaufen, sind in einem Gamestate Objekt gespeichert und stehen global zur Verfügung, sodass jede Funktion darauf zugreifen und den Gamestate verändern kann.

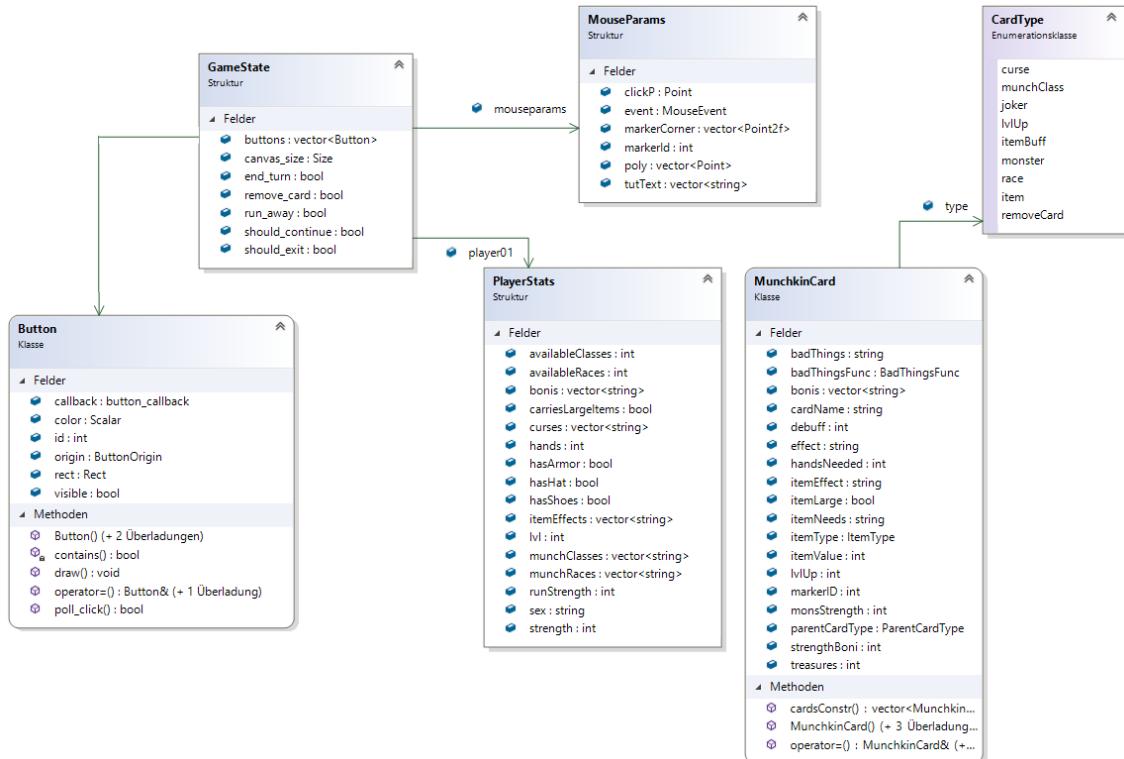


Abbildung 12: Vereinfachtes Klassendiagramm des fertigen Projektes

### **5.3 Implementierung der Karten**

Um die Karten zu implementieren wurden diese für das Proof of Concept in einer eigenen Klasse als Bibliothek angelegt. Sie wurden allerdings so implementiert das es in einem späteren Schritt einfacher wäre die Daten zum Beispiel von einer Datenbank einzulesen. Die Karten sind jeweils ein neues Objekt des Typs `MunchkinCard` und das Objekt wurde so angelegt, dass es alle Parameter beinhaltet kann, die eine Spielkarte benötigt. Die angelegten Objekte werden zum Start des Programms in einen Vektor aus `MunchkinCards` gepusht und können von dort mit der ID ausgelesen werden, da sich die Karte im Vector immer an der Stelle der ID befindet.

Dabei ist darauf zu achten, dass wenn man einige Marker in der Reihenfolge auslässt, diese trotzdem einen Platzhalter erhalten müssen, da sonst die Position im Vektor nicht mehr mit der ID übereinstimmt und so das Auslesen der einzelnen Karten per ID zu Fehlern führt und die falschen Karten gelesen werden bzw auf ein Element des Vektors zugegriffen wird, welches nicht belegt ist.

Eine genauere Dokumentation zum Code des Projektes wird als Anhang an diese Dokumentation beigefügt.

## 6 Recap, Probleme und Lösungen

Bei diesem Projekt kam es zu einigen Problemen, primär war die Kompatibilität verschiedener Bibliotheken nicht so gegeben wie Anfangs erwartet. Eine genauere Erläuterung zu den Problemen und zu den daraus folgenden Änderungen an Architektur, Umsetzung oder der Projektidee selbst folgt in diesem Abschnitt.

### 6.1 Probleme mit OpenCV und Android

Eigentlich sollte das Projekt eine App für Android werden, mit der man das Tutorial überall benutzen könnte. Aber schon im frühen Projektverlauf wurde klar das diese Herangehensweise nicht so einfach werden sollte, wie zuerst angenommen. Zwar ist laut der offiziellen OpenCV Dokumentation das integrieren von OpenCV in Android sehr einfach aber im Verlauf des Projektes wurde festgestellt, dass dem nicht so ist. Erste Hinweise darauf gab es schon, da in der Dokumentation zum einrichten von OpenCV für Android der aktuellste Stand sich noch auf Windows 7 bezieht.

Ferner wurde erklärt, dass wenn man zum ersten Mal ein solches Projekt beginnt, es das einfachste wäre, das Nvidia Tegra Android Development Pack zu verwenden, kurz TADP. Also wurde die neueste Version der Nvidia TADP heruntergeladen und installiert. Inzwischen wurde das Nvidia TADP in das neue Nvidia Code Works integriert. Nachdem alles heruntergeladen und installiert war, musste festgestellt werden, dass in der neuesten Version von Nvidia Codeworks OpenCV nicht mehr verfügbar ist.

Nachdem die Archive von alten Versionen durchsucht wurden, wurde die letzte verfügbare Version des Nvidia TADP installiert, in der OpenCV verfügbar war. Leider war zu diesem Zeitpunkt schon ein Fehler unterlaufen, denn auf Grund der eher unübersichtlich aufgebauten Release Notes von Code Works wurde übersehen, dass für alle drittanbieter Software, die aus älteren Versionen übernommen wurde, ein extra Link in der Liste von Änderungen versteckt war und OpenCV noch bis zur Version 1R6 von Codeworks unterstützt wurde. Die aktuelle Version die getestet wurde war Version 1R8. Wie sich bei der erneuten Recherche zur Dokumentation aber ergab, war auch zu dieser Version das OpenCV contrib Paket und somit Aruco Marker nicht integriert.

Als nächstes wurde dann versucht, mit einer alten Version von Nvidia TADP eine Entwicklungs-umgebung einzurichten. Nachdem nach mehreren Fehlschlägen mit alten Versionen von Visual Studio und Eclipse eine erste Version auf dem Android Testgerät lauffähig war, musste festgestellt werden, dass in dieser Version zwar OpenCV verfügbar und auch verwendbar war, nur leider diese Version von Nvidia TADP so veraltet war, dass das Testgerät, auf dem Android 9 installiert ist, die Anwendung nicht mehr benutzen konnte, weil sie nicht kompatibel ist.

Darauf folgte der Versuch, eine aktuellere Version von OpenCV manuell, mit der Anleitung aus der OpenCV Dokumentation zu installieren. Nachdem auch hier mehrere Versuche unternommen wurden, eine lauffähige Entwicklungsumgebung mit verschiedenen Visual Studio und Eclipse Versionen einzurichten wurde auch hier festgestellt, dass es nicht möglich war dieses Ziel zu erreichen. Da die offizielle Dokumentation von OpenCV noch immer auf einem sehr alten Stand ist, war die Anleitung unbrauchbar da viele Schritte nicht mehr so ausführbar waren wie dort beschrieben. Auch der Versuch andere Lösungen in verschiedenen Foren zu finden blieb zu Anfang ohne Erfolg.

Zuletzt wurde jedoch herausgefunden, dass es möglich war für die von Android selbst zur Verfügung gestellte Entwicklungsumgebung Android Studio OpenCV zu integrieren und so Android Projekte mit OpenCV zu realisieren. Es war dann auch möglich ein erstes lauffähiges Testprojekt der OpenCV Samples auf dem Testgerät ans laufen zu bekommen. Allerdings fiel erst dann auf, dass das normale OpenCV Paket nicht ausreicht, da dort die Aruco Marker nicht integriert sind. Benötigt wurde also das OpenCV\_contrib Paket, was sämtliche Drittanbieter Software beinhaltet.

Da es aber keine offiziellen SDKs des OpenCV\_contrib Pakets gibt, wurde versucht, mit verschiedenen Anleitungen aus verschiedenen Foren, dieser build selbst zu erstellen. Leider war auch dies ohne Erfolg, da auch diese Anleitungen wieder veraltet waren und selbst mit den alten Versionen von OpenCV nicht mit der aktuellen CMake Version oder der aktuellen MinGW Edition zu realisieren war.

Nach weiterer Recherche konnte zwar ein vorgefertigtes Paket von OpenCV\_contrib gefunden werden und das Projekt war wieder auf dem alten Stand, dass eine lauffähige Entwicklungsumgebung und ein Testprojekt verfügbar war, welches auf dem Testgerät lief. Allerdings musste dann nach einiger Zeit festgestellt werden, dass die Dokumentation seitens OpenCV für Android beziehungsweise Java so schlecht ist, dass es nicht möglich war mit den Aruco Markern unter Android zu arbeiten. Selbst das generieren der Aruco Marker über Java war nicht möglich.

Nach Absprache mit den Projektbetreuern wurde dann beschlossen, dass das Ziel für dieses Projekt ein Proof of Concept unter Windows 10 sein soll bei dem unter Verwendung einer simplen Webcam gezeigt werden sollte, dass die Idee eines Tutorials für Munchkin unter Verwendung von OpenCV und Aruco Markern in einer C++ Entwicklungsumgebung zumindest generell realisierbar ist.

Für die weitere Arbeit nach dem Projekt stand auch im Raum, die in dem Projekt geschaffene Struktur und die grundlegende Funktionalität des Tutorials in C++ als Bibliothek auszulagern und in Android Studio zu verwenden, um die benötigten Funktionen im Hintergrund von Android ansprechen zu können und nur die Visualisierung über Android zu realisieren.

## 6.2 Probleme bei der Implementierung

Für die Implementierung der Spielelogik wurden im ersten Anlauf die für die Logik benötigten Parameter wie markerId, Polygon, markerCorner und Tutorialtext in einen struct geladen, welcher dann einer MouseCallback Funktion mitgegeben wurde. Diese Callback Funktion wurde auf jeden Frame geladen und konnte nur die Daten des letzten erkannten Markers enthalten. Dadurch durfte der Nutzer immer nur eine Karte nach der anderen vor die Kamera legen, um diese zu verarbeiten. Wenn dann die Callback Funktion aufgerufen wurde, erhielt diese direkt die Position der Maus auf dem Bild und es konnte in der Funktion abgeglichen werden, ob der Klick auf einem Marker war oder nicht.

Wenn auf den Marker geklickt wurde, wurde die Verarbeitung der Spielelogik durchgeführt. Zuerst wurde mit der übergebenen MarkerId die dazugehörige Karte als eigenes Objekt aus der angelegten Bibliothek geladen. Danach wurde über den Kartentyp ein Switch Case Aufruf abgearbeitet, um zum Beispiel bei einem Fluch dem Spieler den dazugehörigen Effekt als Text anzuzeigen.

Zusätzlich wurde ein Struct für Spielerparameter angelegt, in dem alle wichtigen Attribute des Spielers, wie dessen Stufe, Anzahl an Händen, ob er eine Rasse/Klasse hat etc, hinterlegt und wenn benötigt verändert. Diese Parameter wurden dann zum Beispiel beim Anlegen von Rüstung oder Waffen abgefragt um dem Spieler dann mitzuteilen, ob er den gewählten Gegenstand noch anlegen kann oder ob er ihn auf die Hand nehmen muss.

Bei dieser Herangehensweise die Spielelogik zu implementieren kam es zu einigen Problemen, sobald es nötig war auf den Input eines Spielers zu warten. Da jedes Frame einzeln geladen wurde musste, um auf den Input zu warten, diese Schleife unterbrochen werden. Außerdem wurden die Anweisungen des Tutorials ohne eine Unterbrechung immer nur für einen Frame angezeigt und somit für den Nutzer nicht sichtbar gewesen. Wenn nun also die Schleife unterbrochen wurde stand das Videosignal still, sodass der Nutzer nicht wusste, ob das Programm noch ordnungsgemäß lief. Teilweise wurde auch der Text des Tutorials nicht korrekt gespeichert. Dadurch wusste der Nutzer auch nicht was er als nächstes tun muss.

Dieses Problem wurde gelöst indem die gesamte Struktur des Codes nochmal überarbeitet wurde und eine Update Funktion geschrieben wurde die am Ende jedes Frames die Spielelogik parallel abarbeitet. Dabei wurde der Großteil der Logik in eigene Dateien bzw Funktionen ausgelagert.

## **7 Fortführung Bachelor Arbeit**

Zur Fortführung des Projektes in einer Bachelor Arbeit gibt es zum aktuellen Entwicklungszeitpunkt zwei Möglichkeiten. Die erste Möglichkeit wäre, zu versuchen das Projekt nun so weiter zu entwickeln, dass es als C++ Bibliothek von Android Studio verwendet werden kann und in Zuge dessen eine funktionierende App für das Tutorial zu entwickeln.

Ein zweiter Ansatz wäre es, andere Techniken zur Erkennung der Karten zu implementieren und zu vergleichen, welche Art der Erkennung besser für das Projekt geeignet wäre und diese dann final für das Projekt zu verwenden. Dabei muss man in zwei Schritten vorgehen, da man zuerst das verzerrte Bild der Karte je nach Winkel “entzerren” muss um diese Karte danach mit einer Datenbank an Karten zu vergleichen.

## **8 Zusammenfassung**

Im Laufe des Projektes wurde ein Proof of Concept eines Tutorials für Munchkin geschaffen. Das Tutorial gibt dem Nutzer Hinweise darauf, wie er Munchkinkarten verwenden kann und was sein nächster Zug sein sollte. Das Ergebnis des Projektes zeigt, dass zum Zeitpunkt des Projektes eine Implementierung mit OpenCV in einer Android Umgebung nicht möglich war, da die Dokumentation von OpenCV für Android veraltet und lückenhaft war.

## Literatur

- [BK00] G. Bradski und A. Kaehler. „OpenCV“. In: *Dr. Dobb's journal of software tools* 3 (2000).
- [Gar+15] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas und R. Medina-Carnicer. „Generation of fiducial marker dictionaries using Mixed Integer Linear Programming“. In: *Pattern Recognition* 51 (Okt. 2015).
- [Gor19] youtube.com/Gorobai. Youtube Video. Jan. 2019. URL: <https://www.youtube.com/watch?v=kqZHtoxe6js>.
- [Jac03] S. Jackson. *Munchkin Kartenspiel*. Pegasus Spiele, 2003. URL: [https://pegasusshop.de/media/pdf/41/85/43/4250231703669\\_de.pdf](https://pegasusshop.de/media/pdf/41/85/43/4250231703669_de.pdf).
- [Jac13] S. Jackson. *Munchkin Cthulhu*. Pegasus Spiele, 2013. URL: <https://pegasusshop.de/Sortiment/Spiele/Funspiele/73/Munchkin-Cthulhu-1-2#>.
- [Jac16] S. Jackson. *Munchkin Apokalypse*. Pegasus Spiele, 2016. URL: <https://pegasusshop.de/Sortiment/Spiele/Funspiele/92/Munchkin-Apokalypse-1-2>.
- [Jac19] S. Jackson. *Munchkin Quest*. Pegasus Spiele, 2019. URL: <https://pegasusshop.de/Sortiment/Spiele/Funspiele/516/Munchkin-Quest-Big-Box>.
- [MQQ19] *Munchkin Quacked Quest*. [PC CD-ROM]. Berserk Games, 2019. URL: <https://www.nintendo.de/Spiele/Nintendo-Switch-Download-Software/Munchkin-Quacked-Quest-1670400.html>.
- [OCVAM] *OpenCV: Detection of ArUco Markers*. [Online; accessed 2. Apr. 2021]. Apr. 2021. URL: [https://docs.opencv.org/master/d5/dae/tutorial\\_aruco\\_detection.html](https://docs.opencv.org/master/d5/dae/tutorial_aruco_detection.html).
- [OCVCC] *OpenCV: Camera Calibration*. [Online; accessed 2. Apr. 2021]. Apr. 2021. URL: [https://docs.opencv.org/master/dc/dbb/tutorial\\_py\\_calibration.html](https://docs.opencv.org/master/dc/dbb/tutorial_py_calibration.html).
- [OCVCon] *OpenOpencv\_contrib*. OpenCV. URL: [https://github.com/opencv/opencv\\_contrib](https://github.com/opencv/opencv_contrib).
- [OCVI] *OpenCV: Installation in Windows*. [Online; accessed 16. Apr. 2021]. Apr. 2021. URL: [https://docs.opencv.org/master/d3/d52/tutorial\\_windows\\_install.html](https://docs.opencv.org/master/d3/d52/tutorial_windows_install.html).
- [Ots79] N. Otsu. „A Threshold Selection Method from Gray-Level Histograms“. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), S. 62–66.
- [RMM18] F. Romero-Ramirez, R. Muñoz-Salinas und R. Medina-Carnicer. „Speeded Up Detection of Squared Fiducial Markers“. In: *Image and Vision Computing* 76 (Juni 2018).
- [ROOMLE] *Roomle - 3D / AR Raumplaner*. [Online; accessed 1. Apr. 2021]. Apr. 2021. URL: <https://www.netural.com/de/projekte/roomle-3d-ar-raumplaner>.

- [TTS15] *Table Top Simulator*. [PC CD-ROM]. Berserk Games, 2015. URL: [https://store.steampowered.com/app/286160/Tabletop\\_Simulator/](https://store.steampowered.com/app/286160/Tabletop_Simulator/).
- [Zha00] Z. Zhang. „A flexible new technique for camera calibration“. In: *IEEE Transactions on pattern analysis and machine intelligence* 22.11 (2000), S. 1330–1334.