Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Masterarbeit

Studienrichtung
Informatik

Anatolij Schlotthauer
Evaluierung von Augmented Reality Frameworks für die Implementierung eines Zauberwürfel-Assistenten für Smart Glasses

Erstprüfer: Prof. Dr. Peter Rösch
Zweitprüfer: Prof. Dr.-Ing Alexandra Teynor
Firma: Pixel GmbH
Abgabe der Arbeit am: 23.04.2019

Hochschule für angewandte
Wissenschaften Augsburg
University of Applied Sciences

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info@hs-augsburg.de

Fakultät für Informatik
Telefon: +49 821 5586-3450
Fax: +49 821 5586-3499

Verfasser der Masterarbeit:
Anatolij Schlotthauer
Dietrich Bonhoeffer-Str.1
86399 Bobingen
An.schlotthauer@gmail.com

Hochschule
Augsburg University of
Applied Sciences

Erklärung zur Abschlussarbeit

Hiermit versichere ich, die eingereichte Abschlussarbeit selbständig verfasst und keine andere als die von mir angegebenen Quellen und Hilfsmittel benutzt zu haben. Wörtlich oder inhaltlich verwendete Quellen wurden entsprechend den anerkannten Regeln wissenschaftlichen Arbeitens zitiert. Ich erkläre weiterhin, dass die vorliegende Arbeit noch nicht anderweitig als Abschlussarbeit eingereicht wurde.

Das Merkblatt zum Täuschungsverbot im Prüfungsverfahren der Hochschule Augsburg habe ich gelesen und zur Kenntnis genommen. Ich versichere, dass die von mir abgegebene Arbeit keinerlei Plagiate, Texte oder Bilder umfasst, die durch von mir beauftragte Dritte erstellt wurden.

_____ _____

Ort, Datum Unterschrift des/der Studierenden

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

In den vergangen Jahren hat Augmented Reality (AR) immer mehr an Bedeutung gewonnen. Ein Grund ist der Einzug von Smartphones in unseren Alltag sein. Smartphones eignen sich besonders gut als Plattform für AR-Anwendungen, da sie mit einer Kamera, einem hochauflösenden Display und einer Vielzahl an Sensoren ausgestattet sind. Diese Features sind oft Voraussetzungen für AR-Anwendungen.

Es gibt jedoch auch Hardware, die im Gegensatz zu Smartphones, speziell für AR-Anwendungen entwickelt wurde: AR-Brillen. Diese sind im Stande virtuelle Inhalte direkt in das Sichtfeld des Nutzers einzublenden, wodurch ein komplett anderes AR-Erlebnis entsteht. Außerdem müssen AR-Brillen nicht in den Händen gehalten werden, wodurch diese für andere Aufgaben frei sind. Dies ist insbesondere für den industriellen Einsatz von AR von Interesse sein. Auch in anderen Bereichen, wie z.B. Militär, Medizin, Unterhaltung und Inneneinrichtung wird die Entwicklung der Technologie verfolgt.

Trotz dem momentan vorherrschenden Hype um AR-Brillen, konnte die Hardware noch keinen Einzug in den Alltag der Menschen finden. Ein Grund hierfür liegt in den recht hohen Anschaffungskosten für bekannte AR-Brillen, wie die Microsoft HoloLens (über 3000€ [9]). Es gibt jedoch auch günstigere Alternativen, deren Preis dem eines Mittelklasse-Smartphones gleichkommt.

Um die Fähigkeiten von AR-Brillen zu demonstrieren bedarf es allerdings nicht nur Hardware, sondern auch Software. Für deren Entwicklung gibt es mittlerweile einige Software-Developments-Kits (SDKs). Diese sollen die Entwicklung von AR-Apps erleichtern, indem sie oft benötigte Funktionalität (z.B. Objekterkennung) zur Verfügung stellen. Sie unterscheiden sich in diversen Aspekten, wie z.B. Funkti-

onsumfang, Präzision, Performanz, Gerätekompatibilität und Dokumentation.

Die PIXEL GmbH möchte eine Anwendung (App), die die AR-Fähigkeiten der Epson Moverio BT-300 Smart Glasses demonstriert. Die Anwendung soll dem Anwender dabei helfen einen ”Zauberwürfel” zu lösen. Für die Entwicklung einer solchen App bietet sich die Verwendung eines AR-SDK (fortlaufend auch AR-Framework genannt) an. Die Wahl eines möglichst gut geeigneten Frameworks kann dabei ausschlaggebend für den Erfolg der späteren Anwendungsentwicklung sein. Eine Evaluierung der SDKs soll eine Übersicht von deren Funktionsumfang und Besonderheiten verschaffen. Anhand dieser Übersicht kann man einen passenden Kandidaten für die Implementierung der gewünschten Anwendung auswählen. Zudem können sich die Resultate der Evaluierung bei der zukünftigen Wahl von SDKs für andere Projekte als nützlich erweisen.

1.2 Zielsetzung und Vorgehen

Ziel dieser Arbeit ist die Entwicklung einer prototypischen AR-Anwendung für die PIXEL GmbH. Sie soll dem Träger einer Epson Moverio BT-300 dabei helfen einen ”Zauberwürfel” zu lösen, und muss folglich auf der besagten Hardware funktionieren. Für die Entwicklung der App soll ein AR-Framework verwendet werden. Um sicherzustellen, dass die bestmögliche Entscheidung bei der Wahl des Frameworks getroffen wurde, soll erst eine Sammlung von geeigneten SDK-Kandidaten erstellt werden. Zudem sollen Kriterien definiert werden, die den Funktionsumfang der App abstecken. Anschließend soll anhand dieser Kriterien eine Evaluierung der zuvor gesammelten Kandidaten stattfinden. Das Ziel der Evaluierung besteht darin, ein Framework zu bestimmen, das für die Implementierung der ”Zauberwürfel”-Assistenzanwendung zum Einsatz kommen soll. Abschließend soll die eigentliche App entwickelt werden.

1.3 Aufbau der Arbeit

Diese Arbeit beginnt mit einer Übersicht der Grundlagen von Augemented Reality. Anschließend wird der ”Zauberwürfel” näher beschrieben. Dabei wird insbesondere auf den Aufbau des Würfels und die oft damit verknüpfte Notation ein-

gegangen. Als Nächstes folgen die konkreten Anforderungen, die von der ”Zauberwürfel”-Assistent-App zu erfüllen sind. Hier wird zwischen Grundfunktionalität und Bonusfunktionalität unterschieden.
Darauf folgt die Vorstellung der, später zu evaluierenden, SDKs. Im Weiteren werden die Evaluierungskriterien aus den zuvor definierten Anforderungen abgeleitet. Im Anschluss folgen die Evaluierung und die Auswahl eines SDKs für die Entwicklung der App. Es folgt die Vorstellung der Architektur der entwickelten Anwendung. Dann wird auf die Besonderheiten bei der Umsetzung und die Funktionsweise der App eingegangen.
Abschließend werden Resultate dieser Arbeit nochmals zusammengefasst. Eine Überprüfung der Umsetzung der App-Anforderungen wird durchgeführt und es folgt ein Ausblick auf die Verbesserungsmöglichkeiten der App und potentielle Themen für zukünftige Arbeiten.

2 Grundlagen

Dieses Kapitel stellt die Grundlagen vor, deren Kenntnis für weitere Teile der dieser Arbeit als Voraussetzung angesehen werden.

2.1 Augmented Reality

In diesem Abschnitt wird erst die Definition von Augmented Reality vorgestellt. Dann werden einige Anzeigetechnologien betrachtet. Anschließend werden einige Anwendungsmöglichkeiten von AR kurz dargestellt.

2.1.1 Definition

Eine heute noch gängige Definition für AR wurde 1997 von R. T. Azuma verfasst [17]. Sie gibt vor, dass ein AR-System die folgenden drei Eigenschaften haben muss:

1. Kombination von reellen und virtuellen Inhalten
2. Interaktivität in Echtzeit
3. Ausrichtung von virtuellen und reellen Inhalten im dreidimensionalen Raum

Durch diese Definition ist AR nicht nur auf den Sehsinn beschränkt, sondern kann auch auf andere Sinne angewandt werden. Man kann z.B. haptischen Technologien dazu verwenden, um die Berührung eines Objekts zu simulieren.[21]

Auch für andere Sinne, wie z.B. Hören und Riechen, gibt es bereits Ansätze für AR-Lösungen.[19]

AR beschränkt sich nicht auf das Hinzufügen von virtuellen Inhalten in die reale Welt. Auch das Verstecken von realen Objekten ist ein potentieller Anwendungsfall.[23]

Zum Beispiel wird es, in der Inneneinrichtungsbranche, ermöglicht die Möbel in einem Raum virtuell zu entfernen und den Raum im leeren Zustand zu sehen. Anschließend kann man virtuelle Möbel in dem Raum platzieren, um zu sehen, wie diese im Raum aussehen.

Mixed Reality

Mixed Reality (MR) ist ein Begriff den man heutzutage oft in Zusammenhang mit AR-Technologien zu hören bekommt. Dabei wird er häufig als Synonym für Augmented Reality verwendet. Nach Milgram und Kishino handelt es sich bei MR jedoch um eine Überkategorie von AR[26].
In Abbildung 1 ist das Realitäts-Virtualitäts-Kontinuum zu sehen. Zu Mixed Reality zählen sämtliche Mischformen aus Realität und Virtualität, die zwischen den beiden Grenzwerten liegen. Darunter fällt auch AR. Sie befindet sich im Kontinuum näher am *Real Envirionment* (komplett reale Umgebung) als am *Virtual Environment* (komplett virtuelle Umgebung), was bedeutet, dass der reale Anteil von AR größer ist als der virtuelle Anteil. Das Gegenstück von AR ist die Augmented Virtuality (AV). Dabei handelt es sich um eine primär virtuelle Umgebung, in der reale Inhalte (z.B. Objekte, Personen) integriert werden.

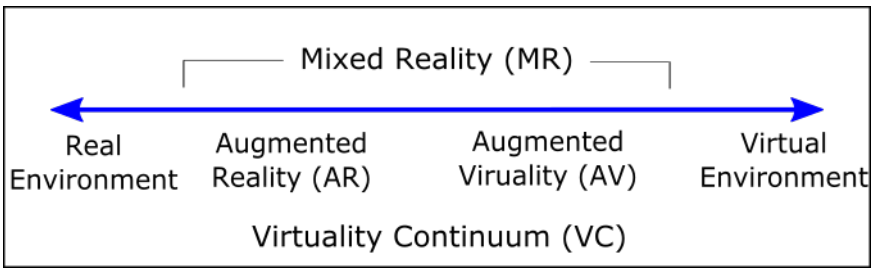

Abbildung 1: Die vereinfachte Repräsentation des Realitäts-Virtualitäts-Kontinuums[26]. Das linke Ende des Kontinuums (*Real Envirionment*) beschränkt sich komplett auf reelle Inhalte. Das andere Ende (*Virtual Environment*) beschränkt sich auf rein virtuelle Inhalte. Dazwischen befinden sich die Mixed Reality. Sie besteht aus Mischformen, bei denen sowohl virtuelle, als auch reelle Inhalte in Kombination eingesetzt werden. Je näher sich diese Mischformen an einem Extremum befindet, desto größer der prozentuale Anteil des Extremums an der besagten Mischform.

2.1.2 Displaytypen für visuelle AR

Diese Arbeit beschränkt sich auf rein-visuelle AR, da das die Art von AR ist, die auch auf AR-Brillen zum Einsatz kommt. Jedoch wird visuelle AR in drei Kategorien unterteilt. Diese unterscheiden sich durch die Technologie, die beim Anzeigen der augmentierten Inhalte verwendet werden.
In AR-Brillen findet typischerweise nur eine dieser drei Anzeigetechnologien Verwendung. Aus Gründen der Vollständigkeit, werden trotzdem alle drei Kategorien vorgestellt und ihre Vor- und Nachteile hervorgehoben.

Video see-through

Die erste Kategorie heißt *video see-through* (dt. Video-Durchsicht). AR-Systeme, die ihr zuordnet werden, verwenden eine oder mehrere Kamera(s) um ein Video der Realität aufzunehmen. Dieser Aufnahme werden die zu augmentierenden Inhalte hinzugefügt. Anschließend wird die Kombination aus Aufnahme der Realität und den hinzugefügten Inhalten auf einem Bildschirm angezeigt (siehe Abbildung 2). Beim Schauen auf den Bildschirm entsteht der Eindruck, dass man durch ihn hindurch sehen kann, da in der Kamerahufnahme Dinge zu sehen sind, die eigentlich vom Bildschirm verdeckt werden. Daraus ergibt sich auch der Name für diese Kategorie.
Die meisten AR-Anwendungen für Smartphones verwenden *video see-through* für die Augmentierung.

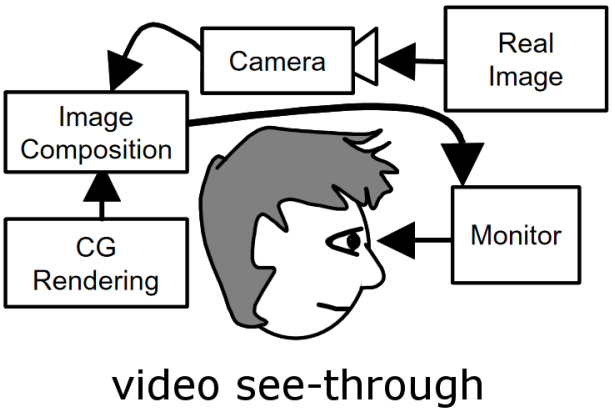

Abbildung 2: Die Abbildung zeigt den groben Aufbau von *video see-through*-Technologien.[23] Die Realität wird durch einen Monitor verdeckt, aber gleichzeitig auch auf ihm angezeigt. Es entsteht der Eindruck, dass man durch den Monitor hindurchsieht.

Nachteile: Ein Kritikpunkt dieses Ansatzes ist, dass der Nutzer keine direkte Sicht auf die Realität hat, da diese durch den Bildschirm verdeckt wird. Auf ihm ist zwar die Kameraaufnahme der Realität zu sehen, aber es handelt sich dabei nur um ein Abbild, und nicht um die Realität selbst.
Dadurch entsteht der Nachteil, dass das Blickfeld durch den Bildschirm eingeschränkt wird, wenn dieser Mal ausfällt. Das kann besonders dann gefährlich werden, wenn die Anwender eine Tätigkeit ausübt, für die Sicht unverzichtbar ist (z.B. medizinische Eingriffe).
Ein weiteres Problem sind Latenzen, die bei der Bildverarbeitung/Bildübertragung auftreten können. Diese führen zu einer Abweichung zwischen Wirklichkeit und dem Bild auf dem Display führen. Der Anwender kann so nicht mehr in Echtzeit reagieren.
Auch die Auflösung der Kamera(s) und des Bildschirms können sich einschränkend auf die Qualität des AR-Erlebnisses auswirken. Des Weiteren kann das Ersetzen des normalen Sichtfelds durch das Kamerabild zu Orientierungsproblemen führen, da die Positionen der Kamera(s) nicht mit den Positionen der Augen des Anwenders übereinstimmen. [31]

Vorteile: *Viedo see-through* ist die am einfachsten zu implementierende der drei Kategorien.[31] Dadurch, dass ein digitales Abbild der Realität verwendet wird, ist es leichter Objekte digital zu entfernen (z.B. Green Screen). Außerdem ist leichter die Helligkeit und den Kontrast der virtuellen Inhalte mit der Realität abzugleichen.

Optical see-through

Optical see-through (dt. optische Durchsicht) ist die Art von visueller AR, die vor allem in AR-Brillen zum Einsatz kommt. Sie verwendet halb-transparente Spiegel für das Anzeigen von virtuellen Inhalten. Diese werden über Displays an der Seiten der Brille auf die Spiegel geworfen und dann in Richtung Nutzer reflektiert. Da die Spiegel halb-transparent sind, ist es möglich sowohl die Reflexion zu sehen, als auch durch die Spiegel hindurchzuschauen. Ein Beispiel dafür ist in Abbildung 3 zu sehen.

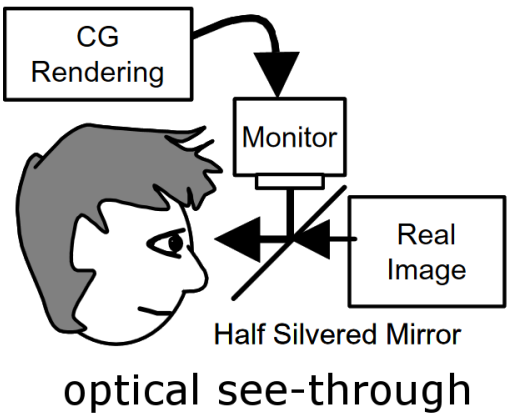

Abbildung 3: Die Abbildung zeigt eine Skizze des groben Aufbaus von *optical see-through*-Technologien zu sehen.[23] Das Einfügen der virtuellen Inhalte erfolgt über die halb-transparenten Spiegel, durch die es immer noch möglich ist die Realität zu sehen.

Nachteile: Virtuelle Inhalte werden nicht mehr in effektiv zweidimensionalen Kamerahufnahmen einer dreidimensionalen Umgebung platziert, sondern müssen anhand des Anwendersichtfeldes positioniert werden. Dies erschwert es die virtuellen Inhalte passend zur Realität auszurichten.
Außerdem muss die Anzeige für die spezifischen Sehbedingungen von verschie-

denen Anwendern angepasst werden.
Ein Weiterer Nachteil ist, dass die Augmentierung durch die Größe der Spiegel beschränkt wird. Inhalte die über den Rand des Spiegels hinausgehen, werden ”abgeschnitten”. Dies führt dazu, dass nur ein Teilbereich des Sehfelds mit Augmentierungen versehen werden kann. Aufgrund der halb-transparenten Natur der Technologie, kann an zu hellen Orten zu Kontrastproblemen bei der Anzeige führen.

Vorteile: Die Sicht des Anwenders wird nicht verdeckt. Ein Ausfall des Systems ist weniger gravierend als bei *video see-through*, da nur die virtuellen Inhalte wegfallen und die Realität aber noch immer ungestört wahrgenommen wird.
Negative Auswirkungen von Displays mit geringer Auflösung beschränken sich auf augmentierte Inhalte. Außerdem entfällt die redundante Anzeige von Kameraaufnahmen. Des Weiteren entfallen mögliche Orientierungsprobleme, da der Anwender seine Umgebung mit seinen eignen Augen wahrnimmt.

Projection based

Projection based (dt. auf Projektion basierend) Technologien bilden die dritte Kategorie der visuellen AR. Der Unterschied zu den vorherigen Kategorien liegt darin, dass bei einem auf Projektionen basierenden Verfahren keine Hardware zwischen Auge und Realität benötigt wird um augmentierte Inhalte sehen zu können. Grund dafür ist, dass die virtuellen Inhalte direkt auf die Oberfläche von Objekten projiziert werden. Die Augmentierung findet also nicht mehr auf einem Bildschirm oder Spiegeln statt. Jedoch bedarf es spezieller Hardware für die Projektionen.

Nachteile: Projektionsbasierte Verfahren sind stark von der Umgebung, in der sie zum Einsatz kommen sollen abhängig. Dabei spielen besonders die vorherrschenden Lichtverhältnisse eine besonders wichtige Rolle. Ein zu heller Raum führt dazu, dass die Projektion nur schlecht erkennbar ist. Dies ist auch der Grund dafür, dass sich das Verfahren nicht für den Einsatz im Freien eignet.[31] Das Entfernen von Objekten (vgl. Vorteile von `video see-through`) ist durch Projektionsbasierte Verfahren nicht umsetzbar.

Vorteile: Projizierende Technologien ermöglichen mehreren Nutzern die selben Inhalte zu sehen, ohne individuelle Hardware für jeden Nutzer zu benötigen. Außerdem muss die Hardware nicht auf die Sehverhältnisse der Anwender angepasst werden. Es ist z.B. nicht nötig die Pupillen der Anwender zu beobachten um die Augmentierung an den Blickwindel des Anwenders anzupassen.

2.1.3 Anwendungsbereiche von AR

AR kann in vielen verschiedenen Bereichen zum Einsatz kommen. Einige Beispiele dafür werden in diesem Unterkapitel vorgestellt.

In der Medizin kann AR dazu verwendet werden, um Scan-Daten direkt auf dem Patienten anzuzeigen, und damit die Arbeit von Chirurgen o.Ä. zu erleichtern.

Beim Militär kann AR als Hilfsmittel für die Zielerfassung, oder die Informationsdarstellung Verwendung finden. Ein Beispiel hierfür ist das vom US-Militär entwickelte ”Integrated Visual Augmentation System” (IVAS).[6] Es handelt sich dabei um eine modifizierte Version der HoloLens 2. Momentan befindet sich die Technologie noch in der Entwicklung, ist jedoch bereits mit einigen Features ausgestattet. IVAS kann eine Karte der Umgebung mit Markierungen, die die Position und Blickrichtung von Verbündeten darstellen, anzeigen. Es hat einen virtuellen Kompass, der dazu verwendet wird um Wegpunkte besser zu finden. Dazu kommt noch ein Wärmebildmodus, der die Position von Wärmequellen via Augmentierung anzeigt.

Die Tourismusbranche kann AR-Brillen als digitale Reiseführer verwenden. Die Träger bekommen Informationen über besichtigte Gebäude/Sehenswürdigkeiten über die Brille angezeigt.

Auch in der Unterhaltungsindustrie gewinnt AR an Beliebtheit. Im Jahr 2016 feierte das AR-Spiel *Pokémon Go* einen kommerziellen Erfolg.

Ein weiterer Anwendungsfall für AR ist die Inneneinrichtung. Durch AR-Apps ist es möglich virtuelle Möbel in einen realen Raum zu stellen, und sich dadurch ein besseres Bild über die Einrichtung zu machen. Der Einrichtungskonzern IKEA hat eine solche App entwickelt. Sie kann dazu verwendet werden um maßstabsgetreue Möbel, aus dem IKEA-Sortiment, in einem Raum zu platzieren. Des Weiteren ist sie im Stande reale Möbel zu scannen und zu bestimmen, um welchen Artikel es sich dabei handelt.

2.2 Verwendete Hardware

Dieses Unterkapitel soll einen Überblick über die verwendete Hardware verschaffen. Für diese Arbeit wurde die Epson Moverio BT-300 als Plattform ausgewählt. Sie wird von Epson als ”Multimedia-Brille” vermarktet. Jedoch eignet sie sich durch die Kamera, Sensoren und des verbauten *optical see-through*-Displays für AR-Anwendungen. In Tabelle 1 findet man die technischen Eigenschaften der Brille.

Tabelle 1: Technische Daten der Epson Moverio BT-300 [1]

Display:	
Anzeigetechnologie:	*optical see-through*
Auflösung:	1280x720Pixel
Blickfeld:	23°
Farbtiefe:	24Bit
Bildschirmfrequenz:	30Hz
Software:	
OS:	Android 5.1 (API Level: 22)
Hardware:	
Prozessor:	Intel® Atom™x5, 1.44GHz Quad Core
RAM:	2GB
Sensoren:	
Kamera (5 Mio. Pixel)	
GPS-Sensor	
Kompass	
Gyroscope	
Beschleunigungssensor	
Helligkeitssensor	

2.3 Zauberwürfel

Der ”Zauberwürfel” (im englischen Sprachraum auch Rubik’s Cube genannt) spielt eine wichtige Rolle in dieser Arbeit. Dieses Kapitel soll einen Überblick über die wichtigsten Eigenschaften des Würfels verschaffen, da diese in späteren Kapiteln von Relevanz sein werden.

2.3.1 Aufbau

Ein gewöhnlicher 3x3-Zauberwürfel ist ein würfelförmiges Drehpuzzle, dessen Seiten aus neun kleineren Würfeln (von nun an *Cubies* genannt) bestehen (siehe Abbildung 4). Diese sind in einem 3x3-Gitter angeordnet. An der Außenseite des Würfels befinden sich folglich 26 sichtbare Cubies. Im Inneren befindet sich ein ”Kern” an dem die Mittelsteine befestigt sind. Folglich ist die Position der Mittelsteine fest und kann sich nicht ändern.
An den sichtbaren Flächen der Cubies befinden farbige Aufkleber. Die Anzahl der Aufkleber hängt dabei von der Art des Cubie ab. Die Mittelsteine (Center) haben nur eine sichtbare Seite und daher auch nur einen Aufkleber. Die Kantensteine (Edges) haben zwei Aufkleber. Die Ecksteine (Corners) haben drei.
Abbildung 5 soll nochmal verdeutlichen, wo sich welche Art von Cubie in einer Würfelseite befindet.

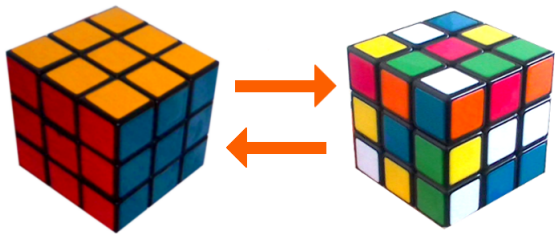

Abbildung 4: Links in der Abbildung ist ein Zauberwürfel im gelösten Zustand zu sehen. Auf der rechten Seite befindet sich ein Beispiel für einen Würfel, der sich im verdrehten Zustand befindet. In beiden Fällen sind drei von sechs Würfelseiten sichtbar.

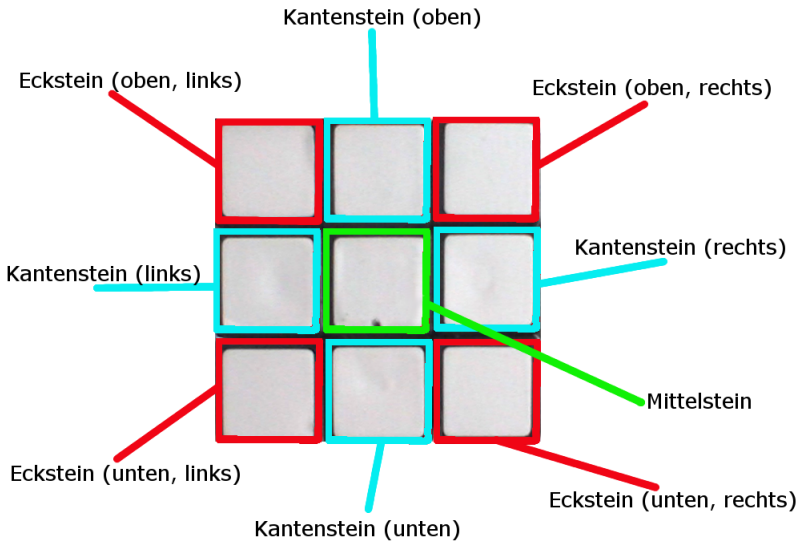

Abbildung 5: Die Abbildung zeigt die Anordnung der verschiedenen Cubie-Arten in der weißen Seite eines Zauberwürfels. Kantensteine sind blau umrandet. Ecksteine haben einen roten Rand. Der Mittelstein ist von einem grünen Rand umgeben.

Die Aufkleber haben sechs verschiedene Farben. Die Standard-Farbkombination ist: Rot, Orange, Gelb, Weiß, Grün, Blau. Es gibt aber auch Zauberwürfel mit anderen Farben. Im Anfangszustand (Abbildung 4, links) ist jede Seite des Würfels einfarbig, d.h. die neun Aufkleber einer Seite haben alle die selbe Farbe.
Die Würfelseiten können in 90°, 180° und 270°-Schritten gedreht werden. Bei jeder Drehung rotieren die Eck- und Kantensteine um den Mittelstein der entsprechenden Seite. Dies führt dazu, dass die Seiten des Würfels nicht mehr einfarbig sind, sondern eine Kombination aus den sechs Seitenfarben annehmen (siehe Abbildung 4, rechts). Ziel des Zauberwürfels ist, den verdrehten Würfel wieder zurück in die Ausgangslage zu bringen. Diese Aufgabe erscheint zwar erstmals trivial, erweist sich aber als relativ komplex, da Drehungen entlang der x-, y- und z-Achsen möglich sind. Das führt dazu, dass sich die sowohl die Position, als auch die Orientierung der Cubies verändert. Die Anzahl an verschiedenen Zuständen, die ein Zauberwürfel annehmen kann, liegt bei $4.3 * 10^{19}$.[25]
Die ist auch der Grund dafür, dass der Würfel für viele Jahre das Interesse von Informatikern und Mathematikern geweckt hat. Es wurde versucht die maximalen Anzahl an Drehungen, die benötigt werden, um den Würfel in jeder beliebigen Position zu lösen zu finden. Über die Jahre hinweg wurde die Zahl immer kleiner.[5]

Mittlerweile liegt sie bei 20.[28]

2.3.2 Notation

Wenn man im Netz nach Lösungen für den Rubik's Cube sucht, stößt man oft auf die inoffizielle "Zauberwürfel-Notation". Die Notation wird auch in dieser Arbeit verwendet und daher in diesem Kapitel erklärt.

Würfelseiten: Die Würfelseiten werden wie folgt referenziert:

- **F**: Vorderseite des Würfels in der Frontalansicht
- **R**: Rechte Seite, grenzt rechts an F an
- **U**: Oberseite, grenzt oben an F an
- **B**: Rückseite, gegenüber von F
- **L**: Linke Seite, grenzt links an F an
- **D**: Unterseite, grenzt unten an F an

Die Bezeichnung ist dabei stets abhängig von der aktuellen Ausrichtung des Würfels. Die Farben spielen hierbei keine Rolle.
Bsp.: Der Würfel befindet sich im gelösten Zustand. Die Seite U ist weiß. Die Seite D ist gelb. Wird der komplette Würfel um 180° umgedreht, ist U gelb und D weiß. Abbildung 6 zeigt ein visuelles Beispiel für die Notation des Würfels.

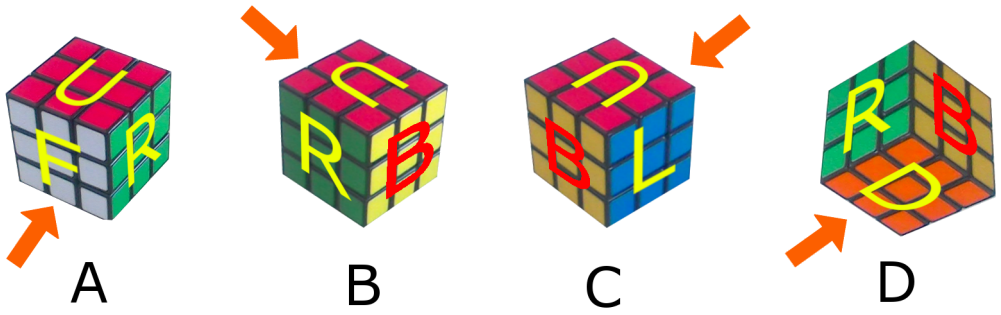

Abbildung 6: Der Würfel ist aus verschieden Perspektiven zu sehen. Die drei sichtbaren Seiten sind stets mit der entsprechenden Notation beschriftet. Die Pfeile zeigen die Blickrichtung des Betrachters an. Es gilt zu beachten, dass der Würfel nicht gedreht wurde. Er wird lediglich aus verschiedenen Perspektiven dargestellt, um alle Seiten darstellen zu können. Wenn der Würfel gedreht wird, muss auch die Notation entsprechend der Drehung angepasst werden.

Drehungen: Analog zu den Seiten gibt es auch eine Notation für Drehungen. Die folgende Namenskonvention wird verwendet:

- Die Drehung einer Seite $*$ um 90°wird als $*$ bezeichnet.
- Die Drehung einer Seite $*$ um -90° 270° wird mit $*'$ notiert.
- Die Drehung einer Seite $*$ um 180° notiert man als $*2$.

Abgesehen von den Seitendrehungen gibt es noch die Drehungen des kompletten Würfels. Hier ist die Drehachse für die Namensgebung verantwortlich. Wird der Würfel auf der x-Achse gedreht, so notiert man die Drehung als x. Das selbe gilt für die y- und z-Achsen. Für 180° und 270°-Drehungen gilt die selbe Namenskonvention wie für die Seitendrehungen. Abbildung 7 stellt die Achsen nochmals bildlich dar.

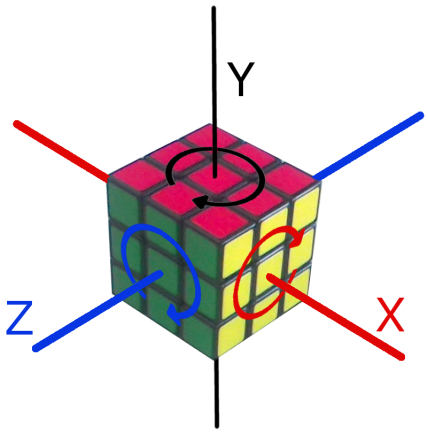

Abbildung 7: Die Abbildung zeigt einen Zauberwürfel auf dem die Achsen für die Drehungen des kompletten Würfels eingezeichnet wurden. Die y-Achse ist schwarz eingezeichnet. Die x-Achse ist rot und die z-Achse ist blau. Die grüne Seite ist die F-Seite. Die rote und gelbe Seite sind jeweils die U-Seite und die R-Seite. Zusätzlich zu den Achsen wurden auch Pfeile eingezeichnet, die eine Drehung im Uhrzeigersinn für jede Achse darstellen sollen.

2.4 Anforderungen an Software

Die zu entwickelnde Applikation soll für Demonstrationen auf Messen oder ähnlichen Veranstaltungen verwendet werden, um das Interesse von potentiellen Kunden und/oder Mitarbeitern an der PIXEL GmbH zu wecken.
Aufgrund der recht umfangreichen Themenstellung dieser Arbeit wurden die Anforderungen in Grundfunktionalität und Bonusfunktionalität unterteilt.

2.4.1 Grundfunktionalität

Die Grundfunktionalität beschreibt minimalen Anforderungen, die an die Anwendung gestellt werden.

- Mithilfe der Brillenkamera soll die relative Position eines Standard-Zauberwürfels im Kamerabild ermittelt werden.

- Für die Positionsbestimmung des Würfels reicht es erstmals aus nur eine Würfelseite in der Frontalansicht zu verwenden.
- Es muss eine Möglichkeit geben den gesamten Würfel zu scannen und dessen Zustand exakt zu bestimmen.
- Die Software soll in der Lage sein einen möglichen Lösungsweg für einen beliebig verdrehten Zauberwürfel zu generieren. Dieser Vorgang soll in einem möglichst kurzen Zeitraum stattfinden. Idealerweise soll der Anwender hiervon nichts merken.
- Der Lösungsweg soll dem Anwender Schrittweise über die Brille mitgeteilt werden. Die Lösungsschritte sollen dabei auf dem Würfel abgebildet werden.
- Bei jedem Schritt muss eine Überprüfung stattfinden, die Aufschluss darüber gibt, ob der aktuelle Lösungsschritt befolgt wurde.
- Der Anwender soll Feedback über die folgenden Dinge erhalten:
  - Wurde der Würfel erkannt?
  - Befindet sich die korrekte Seite vorne?
  - War die letzte erkannte Drehung korrekt?
- Im Falle einer inkorrekten Drehung soll der ermittelte Lösungsweg um einen Korrekturschritt erweitert werden. Alternativ kann auch ein komplett neuer Lösungsweg für den neuen Zustand generiert werden.
- Die App soll weitestgehend ohne die Verwendung der Hände bedienbar sein, da diese den Würfel halten und drehen müssen.

2.4.2 Bonusfunktionalität

Die Bonusfunktionalität beschreibt zusätzliche Features und Erweiterungen, die die Software besser machen sollen. Die Anwendung kann aber auch ohne sie als funktionsfähig angesehen werden.

- Zauberwürfel können verschieden Farben und Farbanordnungen haben. Um die Software unabhängig von einem bestimmten Würfel zu machen, soll die App diverse Farben und Farbanordnungen unterstützen. Die optimale Lösung ist komplett unabhängig von den Würfelfarben und daher mit jedem beliebigen Zauberwürfel kompatibel.
- Die App soll Würfel anderer Größenordnungen (z.B.: 2x2, 4x4, 5x5, etc.) unterstützen.
- Die Software soll um die Erkennung von mehreren Würfeln gleichzeitig erweitert werden.
- Für die Grundfunktionalität reicht es aus nur eine Würfelseite in der Frontalansicht für die Erkennung zu verwenden. Eine Verbesserung bestünde darin, auch das Erkennen von zwei und drei Seiten zu unterstützen.
- Die Anwendung soll um Gestenerkennung erweitert werden, da diese sowohl bei der Bedienbarkeit, als auch beim Erkennen von Drehungen behilflich sein kann.

3 Auswahlprozess für AR-SDKs

Dieses Kapitel behandelt den Auswahlprozess der AR-SDKs. Es gliedert sich wie folgt. Erst werden Kriterien für Erhoben, die die als Voraussetzung der SDKs gelten sollen. Dann folgt eine knappe Vorstellung diverser SDKs. Anschließend folgt die Evaluierungsphase, die in der Auswahl eines Frameworks für die Implementierung der Zauberwürfel-Assistent-App verwendet werden soll.

3.1 Kriterien

Das SDK soll Funktionalität zur Verfügung stellen, um die Position eines dreidimensionales Objektes (= Würfel) im Kamerabild zu verfolgen (engl. 3D-Object Tracking). Da der Würfel in den Händen gehalten wird und die Brille sich auf dem Kopf des Anwenders befindet, wird der Würfel nie weiter entfernt sein, als eine Armlänge. Es ist also nicht nötig sehr kleine (= weit entfernte) Objekte verfolgen zu können.
Für den Fall, dass es nicht möglich sein ist ein dreidimensionales Objekt zu verfolgen, soll es alternativ möglich sein ein zweidimensionales Bild (= eine Würfelseite) zu verfolgen (engl. 2D-Image Tracking).
Ist auch das nicht möglich sein, muss das SDK eine Möglichkeit bereitstellen, die es erlaubt eigene Tracking-Logik zu verwenden. Dabei soll die eigene Tracking-Logik zusammen mit den Standard-Tracking-Mechanismen des SDKs funktionieren. Dadurch wird gewährleistet, dass potentielle Erweiterungen der App auf die Funktionalität des SDKs zurückgreifen können. Schließlich ist diese Funktionalität einer der Gründe, weshalb die Verwendung eines SDKs überhaupt erst in Betracht gezogen wurde.

Da die Anwendung am Ende auf der auf der Epson Moverio BT-300 laufen soll, muss das SDK kompatibel zu deren Hard- und Software sein. Außerdem ist

es vorteilhaft, wenn das SDK die spezifischen Brillen-Funktionen (z.B. stereoskopische Anzeige, Hardwaresteuerung, etc.) unterstützt, da dies den Entwicklungsaufwand weiter reduzieren könnte.

Eine weitere Aufgabe ist die Darstellung von augmentierten Inhalten. Da die Anzeige der Brille auf *optical see-through*-Technologie basiert, müssen die virtuellen Inhalte so ausgerichtet werden, dass sie auf die entsprechenden realen Objekte fallen.
In der finalen Applikation soll der aktuelle Lösungsschritt, in Form eines Bildes, auf dem Würfel angezeigt werden. Nachdem der Schritt ausgeführt wurde, soll der nächste Schritt angezeigt werden. Folglich muss es möglich sein eigene Bilder oder 3D-Modelle zu augmentieren, und diese auch dynamisch auszutauschen.

Es ist davon auszugehen, dass die Funktionalität, die für das Erkennen des Würfelzustands benötigt wird, nicht von den SDKs bereitgestellt wird. Sie muss also selbst implementiert werden. Daher bedarf es einer Möglichkeit auf die erkannte Position des Würfels und das Kamerabild zuzugreifen.
An dieser Stelle ist anzumerken, dass es in Android nicht möglich ist, dass mehrere Anwendungen gleichzeitig auf die Kamera zugreifen. Es ist daher nicht ausreichend nur eine Funktion zur Übergabe der Koordinaten zu haben. Andererseits ist es aber ausreichend nur eine Funktion zur Bereitstellung des aktuellen Kamerabildes zu haben, da die Positionsbestimmung mit dem Kamerabild auch selbst implementiert werden kann.

Zu diesen Hauptkriterien kommen noch sekundäre Kriterien hinzu. Dabei handelt es sich um Aspekte, die zwar nicht unabdingbar für die finale Applikation sind, deren Untersuchung aber einen anderen Mehrwert mit sich bringt.
Ein solcher Aspekt ist der Einfluss des SDKs auf die Portabilität der Anwendung. Im Rahmen dieser Arbeit wird zwar nur eine Anwendung für eine bestimme Plattform entwickelt, dennoch kann eine Betrachtung der Portabilität als sinnvoll angesehen werden, da sie für die Zukunft der App von Interesse sein könnte. Bsp.: Wenn ein SDK die Portierung der Anwendung von einer AR-Brille zu einer anderen AR-Brille (oder Smartphone) einfacher gestaltet als ein anderes SDK, ist das erste SDK favorisiert zu behandeln, sofern es keine signifikanten Unterschiede in den primären Kriterien gibt.
Ein weiteres sekundäres Kriterium ist die verfügbare Dokumentation. In einem

Software Projekt gibt es stets einen vorgegebenen Zeitrahmen, indem eine Anwendung entwickelt werden soll. Je weniger Zeit dafür aufgebraucht wird, sich in die Technologie einzuarbeiten, desto mehr Zeit bleibt um Bugs zu beseitigen, weitere Features zu implementieren und Tests durchzuführen. Eine unvollständige, schwer zugängliche, oder lückenhafte Dokumentation erschwert die Einarbeitungsphase und zieht sie unnötig in die Länge.
Die Dokumentation wurde als sekundäres Kriterium eingestuft, da sie keinen direkten Einfluss auf den Funktionsumfang des SDKs hat.

In der folgen Auflistung sind nochmals alle Kriterien zusammengefasst und mit einer kurzen Begründung versehen:

Primäre Kriterien

- **3D-Object Tracking:** Ein SDK das 3D-Object Tracking für beliebige Objekte unterstützt, ist theoretisch in der Lage einen ”Zauberwürfel” in verschiedenen Situationen und Position zu erkennen.
- **2D-Image Tracking:** Eine Würfelseite kann als zweidimensionales Bild angesehen werden. Mithilfe von 2D-Image Tracking für beliebige Bilder, sollte ein SDK in der Lage sein eine Würfelseite zu erkennen.
- **Eigene Tracking-Logik:** Für den Fall, dass sowohl 2D-, als auch 3D-Tracking nicht zum gewünschten Ziel führen, soll eigene Tracking-Logik verwendet werden.
- **Brillenkompatibilität:** Das SDK solle kompatibel mit der Hard- und Software der Brille sein.
- **Unterstützung von Brillenfunktionalität:** Die AR-Brille verfügt über spezielle Funktionen (z.B. stereoskopische Anzeige). Es wäre vorteilhaft, wenn deren Steuerung vom SDK übernommen wird.
- **Augmentierung von beliebigen Inhalten:** Die Lösungsschritte des Würfels werden vom Entwickler in Form von Bildern oder 3D-Modellen bereitgestellt. Diese sollen mithilfe des SDKs angezeigt werden.

- **Dynamischer Wechsel von augmentierten Inhalten:** Die augmentierten Inhalte sollen sich kontinuierlich und dynamisch ändern. Das soll möglichst flüssig passieren, um Nutzererlebnis nicht negativ zu beeinflussen.
- **Möglichkeit eigene Funktionalität einzupflegen:** Unabhängig davon ob eigene Tracking-Logik zum Einsatz kommt, muss es eine Möglichkeit geben, um zusätzlichen eignen Programmcode (z.B. Lösung von Würfel) zu verwenden.
- **Bereitstellung von Würfelposition und Kamerabild:** Ohne Zugriff auf das Kamerabild kann der Würfelzustand nicht erkannt werden. Daher bedarf es einer Funktion, die das Kamerabild nach Außen gibt. Die Übergabe der Würfelposition ist von Vorteil, kann aber als optional angesehen werden.

Sekundäre Kriterien

- **Portabilität:** Die Portabilität hat keinen direkten Einfluss auf die Implementierung der App. Sie könnte aber vorteilhaft für mögliche zukünftige Arbeiten an der App sein. Je mehr Aufgaben von SDK übernommen werden, desto einfacher ist die Portierung der Anwendung auf eine andere Plattform.
- **Dokumentation:** Die Dokumentation hat indirekten Einfluss auf die Implementierung, da ihre Qualität die Einarbeitungsphase erschweren oder erleichtern kann.

3.2 Vorstellung von AR-SDKs

Es wurde nach gängigen AR-SDKs recherchiert. Dabei wurde erstmal noch nicht auf Kompatibilität mit Brillen Soft- oder Hardware geachtet. Die Ergebnisse der Recherche sind in der nachfolgenden Auflistung zu finden:

Moverio AR: Das Moverio AR SDK befindet sich zur Zeit der Anfertigung dieser Arbeit noch in der Beta-Phase. Der Zugang wurde nur auf Anfrage gewährt. Das Framework unterstützt sowohl 3D-Object-Tracking, als auch 2D-Image-Tracking. Es ermöglicht das Augmentieren von Bildern und dreidimensionalen Modellen.

Das SDK wurde speziell für die Epson Moverio Hardware entwickelt und hat daher auch volle Kompatibilität zu diesem.[10]

MAXST: Das MAXST SDK hat eine kommerzielle Lizenz. Eine freie Testversion ist zwar verfügbar, hat aber einen eingeschränkten Funktionsumfang. Die Pro-Version erlaubt z.B. das Einspielen von eigenen Videoaufnahmen. In der Testversion ist dies nicht möglich.
Im Funktionsumfang sind das Erkennen von Bildern, Markierungen und Szenen enthalten. Des Weiteren können virtuelle Bilder und 3D-Modelle augmentiert werden. Das Framework unterstützt die Betriebssysteme Android, iOS und Windows. Explizite Unterstützdung für Smart Glasses Hardware gibt es für die Epson Moverio BT-200, BT-300, BT-350 und die Osterhout Design Group ODG-R7. Auch Unity wird unterstützt.[8]

Vuforia: Das Vuforia Framework ist das SDK, welches auch auf der HoloLens zum Einsatz kommt. Es unterstützt alle gängigen Betriebssysteme (iOS, Android, Windows) und Unity. Die Lizenz ist kommerziell, aber eine kostenfreie Version mit eingeschränkter Funktionalität wird zum Testen bereitgestellt. Das SDK unterstützt Bilderkennung, Objekterkennung und das Erkennen von Markierungen. Auch die Darstellung von virtuellen Inhalten wird vom SDK übernommen.[13]

Crunchfish: Das Crunchfish SDK hat eine kommerzielle Lizenz. Eine freie Testversion wird nicht angeboten. Das SDK unterstützt vor allem mobile Geräte wie Smartphones und AR-Brillen. Der Fokus des SDKs liegt in der Gestenerkennung und der Entwicklung von berührungslosen Nutzer-Interfaces.[4]

Augmenta: Das Augmenta SDK hat eine kommerzielle Lizenz. Es kann drei Monate lang kostenlos getestet werden. Auch hier liegt der Fokus auf Gestenerkennung von berührunglosen Nutzer-Interfaces. Das SDK unterstützt Android, Windows, Linux und andere POSIX-kompatible Betriebssysteme. Eine Besonderheit ist die Tatsache, dass das SDK direkte Unterstützung zu anderen SDKs (z.B. Vuforia) enthält. Des Weiteren gibt es eine Plugin-API, die die Kompatibilität zu SDKs sicherstellen soll, die nicht direkt unterstützt werden.[3]

Wikitude: Das Wikitude SDK hat eine kommerzielle Lizenz. Es gibt jedoch eine Testversion, die auf unbefristete Zeit verwendet werden kann. Das SDK unterstützt die Betriebssysteme: Android, iOS, Windows. Es ist auch möglich Erweiterungen für die folgenden Frameworks zu entwickeln: Cordova, Titanium, Xamarin, Unity. Es gibt expliziten Support für Epson Moverio und Vuzix Smart Glasses. Der Funktionsumfang des Wikitude SDK enthält das Erkennen von Bilder und Objekten, sowie die Möglichkeit virtuelle Inhalte darzustellen.[14]

ARCore: ARCore ist das AR-SDK von Google. Es ist unter der Apache 2.0 Lizenz verfügbar. Die Zielgeräte des SDKs sind vor allem Smartphones. Daher unterstützt es auch die Betriebssysteme Android und iOS. Des Weiteren werden Unity und die Unreal Engine unterstützt. Der Fokus liegt beim Erkennen der Umgebung (Lichtverhältnisse, Position im Raum, etc.).[2]

3.3 Vorauswahl

Aus zeitlichen Gründen konnten nicht alle SDKs untersucht werden. Stattdessen fand eine Vorauswahl statt, um die Anzahl der Kandidaten auf zwei zu reduzierten. Diese zwei Kandidaten wurden anschließend für eigentliche Evaluierung verwendet.

Vuforia: Vuforia wurde nicht weiter untersucht, da keine Kompatibilität zu der Epson Moverio BT-300 besteht. Die Vorgängerversion (BT-200) wurde zwar in alten Vuforia-Versionen unterstützt, die aktuelle Version (8.1) tut dies allerdings nicht mehr. Da der Support für die BT-200 eingestellt wurde und die BT-300 noch nie unterstützt wurde, ist davon auszugehen, dass Vuforia nicht die benötigte Hardwareunterstützung mit sich bringt.

ARCore: ARCore wurde nicht gewählt, da es sich um ein SDK handelt, dessen primäre Plattform Smartphones sind. Eine Untersuchung wie sich ARCore auf Smart Glasses verhält könnte zwar interessant sein, ist jedoch mit der BT-300 nicht möglich, da sie Android 5.1 verwendet und ARCore Android 7.0 (oder höher) voraussetzt. Selbst wenn es die Möglichkeit gäbe ARCore auf der Brille zu

verwenden, wäre es kein geeignetes SDK, da dessen Fokus auf die Positionsbestimmung im Raum und Umgebungserkennung liegt. Diese Features sind für die zu entwickelnde App jedoch nicht von Bedeutung.

Augmenta: Augmenta bietet keine Funktionen wie z.B. Objekterkennung oder Positionsbestimmung. Stattdessen beschränkt sich der Funktionsumfang auf das Erstellen von GUIs und deren Bedienung via Gestenerkennung. Aufgrund dieser Eigenschaft lässt sich die Annahme treffen, dass das Augmenta-SDK nicht dazu entwickelt wurde um alleine verwendet zu werden. Die zugrundeliegende Idee scheint die Aufteilung von Aufgaben zu sein. Augmenta kann für die Entwicklung des User-Interfaces verwendet werden, während ein anderes SDK die Aufgabe der Objektverfolgung übernimmt. Diese Annahme wird dadurch gestärkt, dass das Augmenta SDK die Verwendung von diversen anderen AR-Frameworks direkt unterstützt. Für Frameworks, die nicht direkt unterstützt werden, wird eine Plugin-API bereitgestellt, um die Kompatibilität mit dem Augmenta SDK zu gewährleisten.

Das Augmenta-SDK eignet sich nicht für die Entwicklung des Zauberwürfel-Assistenten, da wichtige Features fehlen. Es könnte aber in Zusammenhang mit dem anderen SDK dazu verwendet werden, um die implementierte App zu verbessern. Ein weiteres Problem ist die kostenpflichtige Lizenz von Augmenta.

Crunchfish: Crunchfish hat einen Ähnlichen Aufgabenbereich wie Augmenta. Folglich ist auch das Crunchfish-SDK nicht für die Entwicklung der App geeignet.

Wikitude: Das Wikitude-SDK gibt es für verschiedene Plattformen in verschiedenen Ausführungen. Darunter existiert auch eine Version, die speziell für die Epson Moverio BT-Brillen entwickelt wurde. Die Lizenz ist zwar kommerziell, es gibt jedoch eine kostenfreie Testversion die auf unbefristete Zeit verwendet werden kann. Das SDK unterstützt 3D- und 2D-Tracking, sowie die Augmentierung von beliebigen 2D/3D Inhalten. Des Weiteren gibt es eine Schnittstelle für die Erweiterung der SDK-Funktionalität durch einen Code.

Folglich scheint das Wikitude-SDK ein geeigneter Kandidat für die nähere Untersuchung zu sein.

MAXST: Auf den ersten Blick scheint das MAXST-SDK einen ähnlichen Funktionsumfang zu haben wie das Wikitude-SDK. Support für die BT-300 ist vorhanden. 2D- und 3D-Tracking-Verfahren werden unterstützt. Die Lizenz ist zwar kostenpflichtig, aber es gibt eine kostenlose Testversion. Diese ist jedoch in ihrer Funktionalität eingeschränkt. Einige Funktionen (z.B. Einspielen von eigenen Kamerabildern) lassen sich nur in der Pro-Version verwenden. Der wesentliche Unterschied liegt aber in der 3D-Tracking-Vorgehensweise. MAXST verwendet SLAM (simultaneous localization and mapping) um eine Karte des Objektes zu erstellen. Diese Karte besteht aus einer Anzahl von Punkten, die markanten Eigenschaften des Objektes zugeordnet werden. Je mehr Details ein Objekt hat, desto mehr Punkte werden gespeichert. Dies wiederum verbessert die Genauigkeit beim Wiedererkennen des Objektes. Dieses Vorhaben eignet sich vor allem um bestimmte Szenen oder Orte in einem Raum zu finden.

Trotzdem könnte das MAXST-SDK als potentieller Kandidat für die Evaluierung angesehen werden.

Moverio AR: Das Moverio AR-SDK befindet sich zur Zeit der Erstellung dieser Arbeit noch in der Beta-Phase. Zugang zum SDK wird nur auf Anfrage gewährt. Nichtsdestotrotz erscheint dessen Funktionsumfang als vielversprechend. Sowohl 2D- als auch 3D-Tracking-Verfahren werden unterstützt. Des Weiteren wurde das SDK speziell für die Epson Moverio BT-Brillen entwickelt und sollte folglich auch die beste Kompatibilität zur Hardware haben. Es scheint keine Möglichkeit zu geben eigene Tracking-Logik einzupflegen.

Auch das Moverio AR-SDK kann als ein Kandidat für die Evaluierung angesehen werden.

Ergebnisse der Vorauswahl:

Das Wikitude-SDK sieht am erfolgversprechendsten aus. Folglich gehört es auf jeden Fall zu den beiden SDKs, die im nächsten Schritt genauer untersucht wer-

den. Das MAXST-SDK und das Moverio AR-SDK haben ähnliche Vor- und Nachteile. Beide SDKs haben 2D- und 3D-Tracking-Funktionalität, haben aber keine Möglichkeit die Standard-Tracking-Mechanismen zu erweitern. Obwohl sich das Moverio AR-SDK noch in der Beta-Phase befindet, scheint es besser für die geplante Anwendung geeignet zu sein. Grund dafür ist, dass die Objektverfolgung des MAXST-SDKs eher auf die Erkennung von Szenen ausgelegt ist.

3.4 Evaluierungsphase

In diesem Abschnitt werden das Wikitude SDK und das Moverio AR SDK darauf untersucht, wie geeignet sie für die Entwicklung des Zauberwürfel-Assistenten sind. Die beiden Frameworks werden erst genauer Vorgestellt und anschließend anhand der Kriterien aus Kapitel 3.1 evaluiert.

3.4.1 Vorstellung - Wikitude

Dieser Teil des Evaluierungsprozesses beschäftigt sich mit der Vorstellung des Wikitude SDK und der Beschreibung von dessen Funktionsweise. Für die Evaluierung wurde Version 8.2 des Wikitude SDKs betrachtet.

Unterstützte Plattformen

Das Wikitude SDK wird in insgesamt elf verschiedenen Varianten angeboten. Jede Variante ist dabei auf eine andere Plattform oder Framework ausgelegt.[15]

- Mobile Plattformen:
  - Android (nativ)
  - Android (JavaScript)
  - iOS (nativ)
  - iOS (JavaScript)
  - Windows (nativ)
- Erweiterungen:

  - Cordova (Plugin)
  - Titanium (Module)
  - Xamarin (Component)
  - Unity (Plugin

- Smart Glasses:
  - Epson Moverio
  - Vuzix

Diese Arbeit wird sich auf die Epson Moverio-Variante des Wikitude SDKs beschränken.

Kalibrierung

Die Art und Weise wie man durch die Brille schaut unterscheidet sich von Anwender zu Anwender. Die Unterschiede führen zu einer Fehlausrichtung zwischen virtuellen Inhalten und realen Objekten. Eine kleine Fehlausrichtung könnte dem Anwender unbemerkt bleiben. Fällt die Fehlausrichtung jedoch größer aus, leidet das AR-Erlebnis darunter und die App wäre unnutzbar. Aus diesem Grund stellt das Wikitude SDK die Möglichkeit der zur Kalibrierung des Anzeigegerätes zur Verfügung.
Während des Kalibrierungsprozesses muss der Anwender virtuelle Inhalte auf einem ausgedruckten Bild ausrichten. Dies muss für beide Augen aus zwei verschiedenen Distanzen durchgeführt werden. Anschließend wird die optimale Kalibrierung vom Wikitude SDK errechnet. Diese kann bei Bedarf auf dem Gerät gespeichert werden, um zu einem späteren Zeitpunkt wiederverwendet werden zu können.
Des Weiteren besteht die Möglichkeit, den Kalibrierungsprozess über die Wikitude-API zu individualisieren und in die eigene Anwendung einzubauen.

Funktionsweise

Für die Verfolgung von Bildern und 3D-Objekten verwendet das Wikitude SDK sogenannte ”Targets” (dt. Ziele). Dabei handelt es ich um Dateien in speziellen

Formaten, die Informationen enthalten, die vom Tracking-Algorithmus des Wikitude SDKs dazu verwendet werden, um die Bilder/Objekte zu erkennen. Für die Bilderkennung müssen Targets im `.wtc`-Format vorliegen. Für die Objekterkennung wird as `.wto`-Format verwendet.
Um Targets erstellen zu können, stellt Wikitude das Wikitude Studio zu Verfügung. Dabei handelt es sich um eine Webanwendung, die Entwicklern dabei hilft Dateien in den beiden Wikitude-Formaten zu generieren.

ARchitect World: Die Hauptlogik für das Erstellen von AR-Anwendung mit Wikitude liegt in der ARchitect JavaScript API. Diese API ist die Hauptschnittstelle für die Entwicklung von AR-Applikationen mit dem Wikitude SDK. Sie wird für die Erstellung von ARchitect World verwendet. Dabei handelt es sich um Webseiten, die die ARchitect API verwenden, um vom Wikitude Framework als AR-Anwendungen verwendet werden zu können.
ARchitect Worlds werden wie gewöhnliche Webseiten erstellt. Über HTML wird die Struktur vorgegeben, CSS-Dateien bestimmen das Aussehen und JavaScript-Dateien enthalten die Logik. Um auf die ARchitekt API zugreifen zu können, muss diese in der Haupt-HTML-Datei geladen werden.
Über ARchitect Worlds steuerte man sowohl die Bild- und Objektverfolgung, als auch die Anzeige von virtuellen Inhalten und deren Platzierung. Da ARchitect Worlds auf dem Einsatz von Webtechnologien aufbauen, sind sie dafür geeignet Cross-Plattform-Anwendungen zu entwickeln. Die verschiedenen Versionen des Wikitude SDKs kümmern sich um die Besonderheiten der jeweiligen Plattform. Dies führt dazu, dass die selbe ARchitect World auf verschiedenen Plattformen verwendet werden kann. Intern leitet Wikitude die JavaScript-Anweisungen an die entsprechende Plattform-spezifische Implementierung weiter.

Objektverfolgung: Um ein Objekt mithilfe von Wikitude erkennen zu können, muss erst ein Target für das besagte Objekt erstellt werden. Für die Erstellung eines Targets werden Bilder des Objekts aus verschiedenen Winkeln in Wikitude Studio hochgeladen werden. Aus diesen Bildern wird ein dreidimensionales Model des Objekts in Form einer Punktwolke erstellt (siehe Abbildung 8).

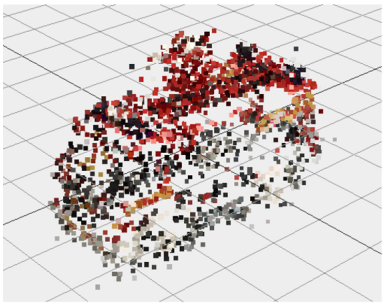

Abbildung 8: Auf der linken Seite der Abbildung ist ein Feuerwehrauto zu sehen, das Teil der offiziellen Wikitude Dokumentation und Beispielanwendungen ist.[16] Auf der rechten Seite befindet sich ein Screenshot vom Punktwolken-Editor in Wikitude Studio.

Die Punktwolke besteht aus markanten Punkten, die auf der Oberfläche des Objektes erkannt wurden. Ein Teil dieser Punkte wird bei erneuter Betrachtung es Objekts wiedererkannt werden. Anhand der Übereinstimmung der Punkte lässt sich eine Aussage darüber treffen, wie wahrscheinlich das Objekt erkannt wurde, in welcher Position es sich befindet und an welcher Stelle es im Kamerabild gefunden wurde.

Um die Objektverfolgung anzustoßen, muss in einer ARchitect World erst eine Instanz der `AR.TargetCollectionResource`-Klasse mit dem Target für das entsprechende Objekt erzeugt werden. Anschließend kann diese Instanz dazu verwendet werden, um einen `AR.ObjectTracker` zu erstellen. Nach dessen erfolgreicher Erstellung läuft die Objektverfolgung.

Folgender Code dient als Beispiel für die Erstellung eines Trackers:

```
1 var World = {
2     //...
3     init: function initFn() {
4         World.createTracker();
5     },
6     createTracker: function createTrackerFn() {
7         this.targetCollectionResource = new
```

```
8           AR.TargetCollectionResource("assets/cube.wto", {});
9           this.tracker = new AR.ObjectTracker(
10                  this.targetCollectionResource, {
11              onError: function(errorMessage) {
12                  console.log(errorMessage);
13              }
14          });
15          //...
16      },
17      //...
18  }
19  World.init();
```

Bildverfolgung: Das Erstellen eines Trackers für Bilder erfolgt weitestgehend identisch zum Erstellen eins Trackers für 3D-Objekte. Erst wird eine Instanz des `AR.TargetCollectionResource`-Klasse erstellt. Dabei muss die `.wto`-Datei durch eine `.wtc`-Datei ersetzt werden. Anschließend wird ein Tracker erstellt. Da ein Bild statt eines Objektes Erkannt werden soll, wird dafür die `AR.ImageTracker`-Klasse verwendet.

Augmentierung: Für das Erstellen von virtuellen Inhalten wird wieder die ARchitect API verwendet. Im den vorherigen zwei Abschnitten wird beschrieben, wie ein `AR.ObjectTracker` und ein `AR.ImageTracker` erstellt werden. Mithilfe dieser Tracker können`AR.ObjectTrackable`- und `AR.ImageTrackable`-Objekte erstellt werden. Ihre Aufgabe besteht darin, virtuelle Inhalte an der Position von erkannten Objekten darzustellen. Diese Inhalte werden ihnen in Form von `AR.Drawable`s übergeben. `AR.Drawable` ist die Basisklasse für alle zu graphischen Elemente, die augmentiert werden sollen.
Für die Augmentierung von Bildern wird die `AR.ImageDrawable`-Klasse verwendet. Diese nimmt über die `AR.ImageResource`-Klasse eine Bilddatei entgegen und zeigt diese an der entsprechenden Stelle an. Das Anzeigen von dreidimensionalen Inhalten erfordert zusätzliche Vorarbeit. 3D-Modelle müssen erst in ein spezielles Format (`.wt3`) konvertiert werden, bevor sie von der ARchitect API verwendet werden können. Das Wikitude SDK enthält das Programm *Wikitude 3D Encoder*. Es

kann dazu verwenden werden um `.fbx`-Dateien in `.wt3`-Dateien umzuwandeln.

Folgendes Beispiel zeigt den Code, der für die Darstellung eines 3D-Objekts benötigt wird. Es ergänzt das Codebeispiel aus dem vorletzten Abschnitt ”Objektverfolgung”.

```
var World = {
    //...
    init: function initFn() {
        World.createTracker();
    },
    createTracker: function createTrackerFn() {
        // assume this.tracker was created here
        this.model = new AR.Model("models/Cube3dModel.wt3", {
        });
        this.objectTrackable = new AR.ObjectTrackable(
        this.tracker, "*", {
            drawables: {
                cam: [model]
            }
        });
    },
    //...
}
World.init();
```

Plugin API: Das Wikitude SDK stellt eine Plugin-API zu Verfügung, die es Entwicklern ermöglicht, die vorhandene SDK-Funktionalität um selbst entwickelte Funktionalität zu erweitern. Die Kommunikation zwischen dem eigenen Code und dem SDK erfolgt über ein Plugin. Plugins können in der Epson Moverio Version des Wikitude SDKs in Java oder C++ implementiert werden. Für die beiden Sprachen wird jeweils eine Plugin-Basisklasse mit den benötigen Methoden bereitgestellt. Durch Vererbung können diese Methoden mit Logik versehen werden. Die Plugins aus der Bibliothek müssen in der `onPostCreate()`-Methode ei-

ner `Activity`-Klasse initialisiert werden:

```
1  public class MainActivity extends AppCompatActivity {s
2     private WearableArchitectView mWearableArchitectView;
3     //...
4     @Override
5     protected void onPostCreate(Bundle savedinstanceState) {
6        //...
7        mWearableArchitectView.registerNativePlugins("PluginName",
8           new PluginManager.PluginErrorCallback() {
9           @Override
10          public void onRegisterError(int i, String s) {
11             Toast.makeText(MainActivity.this, getString(
12                R.string.fail_toast), Toast.LENGTH_SHORT).show();
13             Log.e(TAG, getString(R.string.fail_log_msg) + " " + s);
14          }
15       });
16    }
17    //...
18 }
```

Input Plugin API: Input Plugins werden dazu verwendet, um das Standard-Aufnahmeverhalten an die eignen Bedürfnisse anzupassen. Wie bei den normalen Plugins, gibt es auch für Input Plugins eine Basisklasse `InputPlugin`, in der sich die zu implementierenden Methoden befinden. `InputPlugin` erbt von der `Plugin`-Klasse, was dazu führt, dass die beiden Plugin-Arten bei der Registrierung identisch behandelt werden.
Die wichtigste Funktion von Input Plugins ist das Erstellen von eignen Frames. Diese werden vom Wikitdue SDK identisch zu den normalen Kameraaufnahmen behandelt.

Die folgenden Eigenschaften können angepasst werden:

Inhalt: Der Inhalt der Frames kann vom Anwender beliebig gesetzt werden. Es ist möglich eine externe Kamera zu verwenden, oder externe Videoda-

ten über eine Internetverbindung zu beziehen. Ein weiterer Anwendungsfall ist das Wiedereinspielen von vorherigen Kameraaufnahmen. Dadurch können z.B. zwei verschieden Algorithmen gegen identischen Video-Input getestet werden. Dies ermöglicht auch das Erstellen von Benchmarks durch das Einspielen von externen Datensetzen.

Farbraum: Frames die über den Standard-Input des Wikitude SDKs kommen, befinden sich im `YUV_420_YV12`-Format. Dies ist nicht immer das Format, das man verwendet möchte. Darum erlaubt die Input-Plugin-API es, Input-Frames in einem anderen Farbraum/Format zu erzeugen. Folgende Farbräume und Formate werden unterstützt.

- `YUV_420_NV21` → entspricht `ImageFormat.NV21`-Konstante auf Android-Plattformen
- `YUV_420_YV12` → entspricht `ImageFormat.YUV_420_888`-Konstante auf Android-Plattformen
- `RGB` → entspricht `ImageFormat.FLEX_RGB_888`-Konstante auf Android-Plattformen
- `RGBA` → entspricht `ImageFormat.FLEX_RGBA_888`-Konstante auf Android-Plattformen

Bildgröße: Die Frames des Standard-Inputs haben eine Größe von 640x480 Pixeln. Des Weiteren werden Auflösungen von 1280x720px und 1920x1080px unterstützt. Bei Verwendung eines Input Plugins können diese Werte beliebig gewählt werden.

Verarbeitung des Inputs: Standardmäßig werden die eingehenden Frames von der Wikitude computer vision engine verarbeitet. Wenn dieses Verhalten nicht erwünscht ist, kann es durch das Verwenden eines Input Plugins deaktiviert werden. Das kann unter Anderen auch zu Performance-Gewinn führen.

Anders als bei normalen Plugins, gibt es für Input Plugins keine Java-API. Folglich muss die Implementierung von Input Plugins in C++ erfolgen.

3.4.2 Evaluierung - Wikitude

3D-Object Tracking

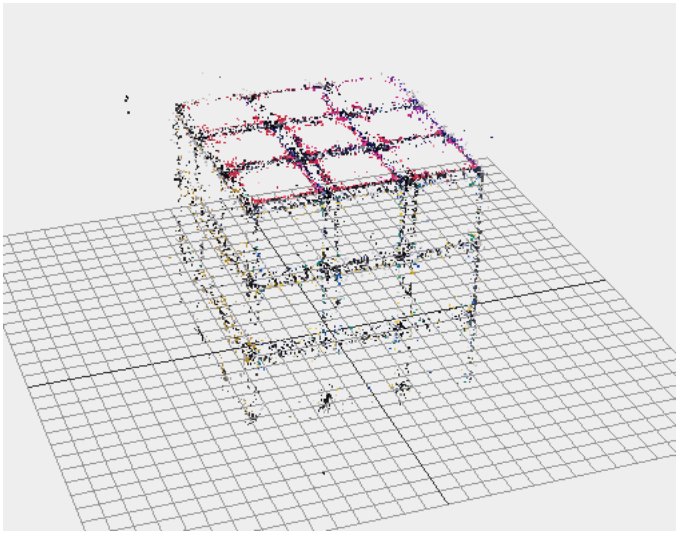

Abbildung 9: Punktwolke für Zauberwürfel

In Abbildung 9 ist die Punktewolke für einen Zauberwürfel zu sehen. Wie man erkennen kann, ist Farbe der einzelnen Aufkleber an bestimmte Positionen gebunden. Das bedeutet, dass eine Punktwolke nur für die Erkennung des Würfels in einem fest vorgegebenen Zustand verwendet werden kann. Da es $4.3 * 10^{19}$ verscheide Zustände gibt, bräuchte man auch die selbe Anzahl an Targets. Dies ist sowohl vom Speicherbedarf als auch von der Performanz her nicht umsetzbar. Die Positionsbestimmung des Würfels anhand der Objekterkennung des Wikitude SDKs scheidet aus.

2D-Image Tracking

Um eine Würfelseite unabhängig von ihren Farben erkennen zu können, muss die Erkennung auf die schwarze Gitterstruktur (siehe Abbildung 10) beschränkt werden. Die farbigen Pixel der Aufkleber können durch transparente Pixel ersetzt werden.

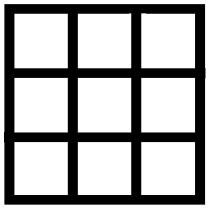

Abbildung 10: Die Schwarze Gitterstruktur bleibt übrig, wenn die Farben der Sticker entfernt werden. Sie sollte für die farbunabhängige Erkennung von Würfelseiten verwendet werden.

Dieses Vorgehen hat sich als nicht umsetzbar herausgestellt, da Wikitude keine Bilder mit transparenten Pixeln unterstützt. Folglich kann auch das 2D-Image-Tracking-Verfahren von Wikitude nicht für die Positionsbestimmung des Würfels verwendet werden.

Eigene Tracking-Logik

Mithilfe der Plugin-API ist es möglich eigene Tracking-Logik parallel zur normalen Wikitude-Tracking-Logik zu verwenden.

Brillenkompatibilität

Die Epson Moverio Version des Wikitude SDKs wurde speziell für Epson Moverio Brillen entwickelt. Daher weißt diese Version des SDKs auch vollständige Kompatibilität zur Moverio BT-300 auf.

Unterstützung von Brillenfunktionalität

Um die Epson Moverio Variante des Wikitude SDKs verwenden zu können, muss das, von Epson bereitgestellte, BT-SDK als Abhängigkeit (engl. dependency) hinzugefügt werden. Hierbei handelt es sich nicht um das Moverio AR SDK, sondern um eine Bibliothek, die dazu verwendet werden kann um die Brillenhardware anzusteuern.
Bei der Auswahl des BT-SDKs ist zu beachten, dass die zur Brille passende Version gewählt wird. (Bsp.: Das BT-200-SDK ist nicht für den Einsatz mit der Epson

Moverio BT-300 ausgelegt.)
Durch diese Dependency ist dafür gesorgt, dass das Wikitude SDK auf die Brillenhardware zugreifen kann. Folglich müssen Entwickler sich nicht mehr selbst um die Hardwaresteuerung kümmern, was auf eine Verminderung des Arbeitsaufwands hinausläuft.
Gleichzeitig bleibt dem Entwickler die Möglichkeit, die Hardware selbst über das BT-SDK zu steuern, erhalten.

Augmentierung von beliebigen Inhalten

Wie aus dem Vorstellungsabschnitt bereits hervorgeht, unterstützt das Wikitude SDK beliebige Inhalte bei der Augmentierung. Bilder können direkt verwendet werden. 3D-Modell müssen erst über das Wikitude Studio in ein spezielles Format konvertiert werden. Dies kann zu Problemen führen, wenn man dynamisch generierte 3D-Modelle verwenden möchte. Dies ist für die Zauberwürfel-Assistent-App allerdings nicht der Fall.

Dynamischer Wechsel von augmentierten Inhalten

Der Wechsel von Augmentierten Inhalten erfolgt über den Austausch der List von `AR.Drawable`-Objekten im `Trackable`. Dies ist auch dann möglich, während das `Trackable` in Betrieb ist. Ein Austausch der `AR.Drawable`s hat nur zufolge, dass sich die angezeigten Inhalte verändern. Die Bild-/Objekterkennung muss nicht pausiert werden, sondern läuft einfach weiter.

Möglichkeit eigene Funktionalität einzupflegen

Durch die Verwendung der Plugin-API (siehe vorheriger Abschnitt) lässt sich die vorhande Funktionalität des Wikitude SDKs beliebig erweitern.

Bereitstellung von Würfelposition und Kamerabild

Nachdem ein Objekt erkannt wurde, befinden sich alle relevanten Informationen über das Objekt in dem `targetBucket_`-Parameter in der Methode `update(const RecognizedTargetsBucket& targetsBucket_)`. Zu diesen Informationen gehören:

- die physische Höhe eines Bildes
- der Abstand zum Bild/Objekt
- die Position des Bildes/Objekts im Kamerabild

Das Kamerabild und diverse Meta-Informationen befinden sich im `Frame`-Parameter der `void cameraFrameAvailable(const Frame& cameraFrame_)`-Methode.

Portabilität

Wie bereits erwähnt, gibt es elf verschiedene Versionen vom Wikitude SDK, die jeweils eine spezifische Plattform bzw. ein Framework unterstützten. Da große Teile der Anwendungen mithilfe von Webtechnologien realisiert werden, ist es ein Leichtes sie auf verschiedenen Plattformen zum laufen zu bringen. Dies gilt nicht für die nativen Varianten des SDKs, da dort keine HTML-Seiten als Oberflächen verwenden werden, sondern Plattform-spezifische Komponenten.
Auch Plugins können auf andere Plattformen portiert werden. Dabei gilt zu beachten, dass dies nur dann möglich ist, wenn kein plattformspezischer Code verwenden wurde. Wenn dies nicht der Fall ist, ist es erforderlich diese Teile des Codes auszutauschen.

Dokumentation

Die Dokumentation des Wikitude SDKs ist versionsübergreifend nicht konstant. Dies ist wahrscheinlich auf die vielen verschiedenen Versionen zurückzuführen. Insbesondere bei der Dokumentation der Plugin API sind zwischen der Android(JS)-Version und der Epson Moverio-Version größere Unterschiede in Umfang und Vollständigkeit festzustellen. Die wesentlichen Informationen für eine Version sind jedoch immer vorhanden. Informationen, die in der Dokumentation einer Version fehlen, konnten in der Dokumentation einer anderen Version in Erfahrung gebracht werden.

3.4.3 Vorstellung - Moverio AR

Dieser Teil des Evaluierungsprozesses beschäftigt sich mit der Vorstellung des Moverio AR SDKs und der Beschreibung von dessen Funktionsweise. Für die

Evaluierung wurde Version 1.2 des Moverio AR SDKs betrachtet.

Unterstützte Plattformen

Das Epson Moverio AR SDK unterstützt nur Epson Moverio-Produkte. Da die Brille Android 5.1 als Betriebssystem verwendet, besteht theoretisch die Möglichkeit, das SDK auf anderen Android-Geräten zu verwenden. Dies wird jedoch explizit nicht unterstützt und es besteht keine Garantie, dass dieses Vorhaben funktioniert.

Kalibrierung

Das Moverio AR SDK wird mit dem Programm ”Calibration Tool” ausgeliefert. Dabei handelt es sich um eine `.apk`-Datei, die auf der Brille installiert werden muss. Das Calibration Tool wird dazu verwendet, um die Brillenanzeige auf einen bestimmen Anwender einzustellen. Zwar wird eine Standardkalibrierung von SDK bereitgestellt, jedoch ist diese nicht für jeden Anwender geeignet. Unterschiedliche Nutzer werden nicht die selbe geometrische Gesichtsbeschaffenheit haben. Dadurch fällt der Blickwinkel zwischen Objekt und Brillendisplay für jeden Anwender etwas anders aus. Wenn die Anzeige nicht auf den aktuellen Anwender eingestellt ist, kann es zu einer Fehlausrichtung zwischen den virtuellen Inhalten und der Realität kommen. Mithilfe des Calibration Tools lassen sich Profile für einzelne Anwender erstellen und speichern. Die zuvor erstellten Profile können dann je nach Anwender geladen werden. Dies ermöglicht es, dass die selbe Hardware von mehreren Nutzer verwendet werden kann.

Funktionsweise

Objektverfolgung: Das Moverio AR SDK ermöglicht es dreidimensionale Objekte über die Brillenkamera zu erkennen. Es gibt zwei Arten von Objekten:

- Objekte mit Texturen (viele Merkmale)
- Objekte ohne Texturen (wenig Merkmale)

Der Unterschied zwischen den beiden Kategorien kommt während der Verwendung des Capture Tools zu tragen. Bei Objekten mit wenig Merkmalen müssen

die Bilder in 20°-Schritten aufgenommen werden. Objekte mit vielen Merkmalen können in 60°-Schritten abgescannt werden, da die einzelnen Bilder wegen der Texturen einfacher zusammengeführt werden können.

Capture Tool Das Moverio AR SDK wird mit dem Programm ”Capture Tool” ausgeliefert. Es ist eine Applikation, die auf der AR-Brille installiert wird. Die Software wird dazu verwendet um ein Trainingsset für die Objektverfolgung zu erstellen. Dafür müssen Bilder des Objekts von allen Seiten aufgenommen werden. Der Abstand (= Winkel) zwischen diesen Bildern hängt davon ab wie viele Texturen das Objekt hat. Dieses Set wird anschließend dazu verwendet der Anwendung beizubringen das eingescannte Objekt im Kamerabild wiederzuerkennen.

Training Tool Das Training Tool wird dazu verwendet um die zuvor aufgenommenen Bilder in ein Model zu konvertieren, welches von SDK für die Objekterkennung verwendet wird. Dafür wird ein CAD-Model des Objekts benötigt. Dieses kann als `.stl`- oder `.obj`-Datei vorliegen. Über das Training Tool werden Punkte ausgewählt, die dazu benutzt werden, um die Dreidimensionalität des Models auf die zweidimensionalen Bilder abzubilden (siehe Abbildung 11).

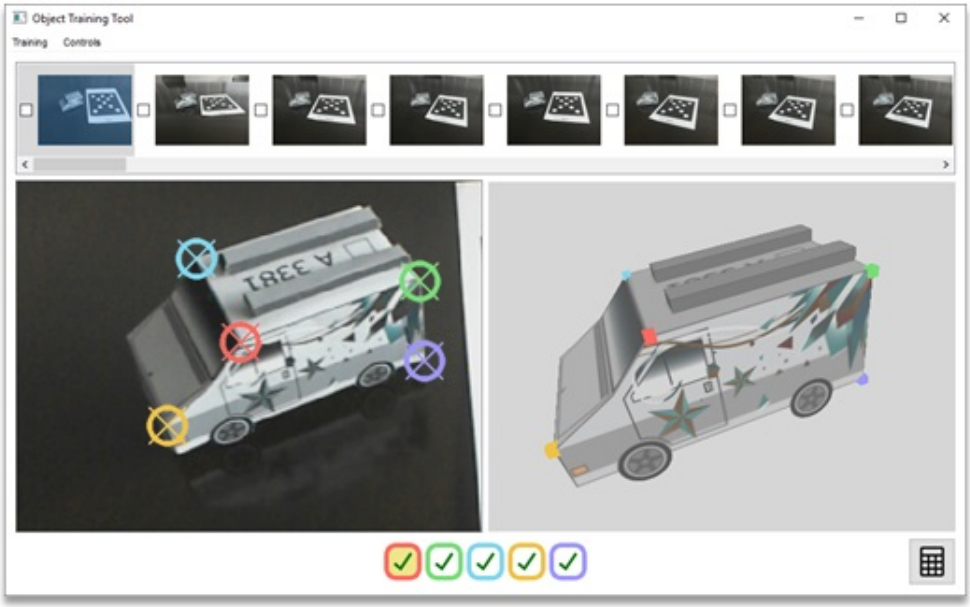

Abbildung 11: Dieses, aus der offiziellen Moverio AR Dokumentation entnommene, Bild [11] zeigt einen Screenshot des Object Training Tools während der Vorbereitung für ein Training.

Es wird zwischen zwei Trainingsarten unterschieden: Texture Based Training und Edge Based Training. Beim Texture Based Training werden Objekt anhand ihrer Texturen erkannt. Es ist besonders für Objekt mit vielen visuellen Details geeignet.
Das Edge Based Training wird für Objekte verwendet, die nur wenige Details haben. Anstatt sie über die Texturen zu erkennen, werden sie über ihre Kanten erkannt. Dieses Verfahren scheint anfälliger für Fehler zu sein und ist weniger zuverlässig.

Folgende Anforderungen gelten für 3D-Objekte:

- Objekte müssen dreidimensionale Festkörper sein
- Objekte dürfen nicht verformbar oder biegsam sein
- Objekte dürfen nicht transparent sein
- Objekte dürfen nicht zu stark reflektieren
- Objekte dürfen keine zylindrische Form haben

Bildverfolgung: Das Erkennen von Bildern erfolgt, ähnlich wie das Erkennen von Objekten, über das Training Tool. Das Capture Tool wird für Bilder nicht benötigt. Es muss lediglich dass zu erkennende Bild vorhanden sein.
Das Framework stellt die folgenden Anforderungen an Bilder, die es erkennen soll:

- Bilder sollen möglichst viele Details und guten Kontrast haben
- Bilder sollen flach sein und nicht reflektieren
- Muster sollen sich nicht zu oft wiederholen
- Texte sind nach Möglichkeit zu vermeiden

Die aus dem Training resultierenden Daten können dazu verwendet werden, um die Erkennung von einem Bild/Objekt anzustoßen. Dafür muss im Code lediglich eine Instanz der `TrackingObject`-Klasse mit dem Pfad zu den Trainingsdaten erstellt werden.

Da ein Trainingsset mehrere Ziele enthalten kann, kann er vorkommen, dass mehrere `TrackingObject`s für ein Trainingsset benötigt werden. Jedes TrackingObject hält die Informationen zu einem der Ziele.

Augmentierung: Die Darstellung von augmentierten Inhalten erfolgt im Moverio AR SDK über Szenen (`Scene`). Den Szenen können `Model`s und `TrackingObject`s zugewiesen werden. Es ist möglich mehrere unterschiedliche Szenen zu erstellen. Jedoch kann immer nur eine aktiv sein. Die Modelle der aktiven Szenen werden an der Position des gefundenen Objektes angezeigt.

3.4.4 Evaluierung - Moverio AR

3D-Object Tracking

Texture Based Training eignet sich nicht für einen ”Zauberwürfel”, da nur mäßig viele Details vorhanden sind. Außerdem wechselt der Würfel permanent die Farbe, was die Erkennung anhand der Texturen unmöglich macht. Das Edge Based Training ist die bessere Wahl für die Würfelerkennung. Der ”Zauberwürfel” wechselt während jeder Drehung die Form und kann in dieser Zeit nicht erkannt werden. Jedoch hat der Würfel vor und nach jeder Drehung die selbe Form. Lediglich die Anordnung der Farben verändert sich. Da das Verfahren auf die Form und nicht auf die Farbe ausgelegt ist, wurde angenommen, dass dies nicht zu Problemen führt. Nach einem Testdurchlauf wurde dieses Verfahren jedoch als zu inkonsistent eingestuft. Daher liegt die Präferenz bei der Verwendung eines anderen Verfahrens, sofern dies möglich ist.

2D-Image Tracking

Das SDK unterstützt keine transparenten Pixel. Folglich ist es nicht möglich die Bilderkennung nur auf das Erkennen des schwarzen Gitters zu trainieren. Es ist daher festzustellen, dass die Bilderkennung des Moverio AR SDK ungeeignet für die Zauberwürfel-Assistent-App ist.

Eigene Tracking-Logik

Das Moverio AR SDK bietet keine direkte Möglichkeit eigene Tracking-Logik zu verwenden.

Brillenkompatibilität

Das Moverio AR SDK ist kompatibel zur Moverio BT-300.

Unterstützung von Brillenfunktionalität

Da das Moverio AR SDK von Epson entwickelt wurde, hat es vollständigen Support für die komplette Brillenhardware, ohne dass ein zusätzliches SDK o.Ä. benötigt wird.

Augmentierung von beliebigen Inhalten

Modelle können sowohl Bilder, als auch 3D-Modelle enthalten. Anders als bei Wikitude, müssen 3D-Modelle nicht erst in ein spezielles Format umgewandelt werden, sondern können in den geläufigen Formaten `.obj` oder `.stl` vorliegen.

Dynamischer Wechsel von augmentierten Inhalten

Ein dynamischer Wechsel von augmentierten Inhalten erfolgt über das Hinzufügen und Entfernen von Modellen einer Szene.

Möglichkeit eigene Funktionalität einzupflegen

Das Moverio AR SDK liefert alle nötigen Methoden, die dafür benötigt werden, um eigene Funktionalität einzupflegen.

Bereitstellung von Würfelposition und Kamerabild

Die Position von erkannten Bildern/Objekten können über die entsprechenden `TrackingObject`s bestimmt werden. Der API-Referenz zufolge können die Bilddaten in Form eines Byte-Arrays über die `ARData`-Klasse bezogen werden. Zugriff auf diese Klasse gibt es nur über die Implementierung der `onPoseFrame(String`

`patternName, float[] poseMat, ARData data)`-Methode aus dem Interface `MoverioARTrackingPoseFrameListerner`. Eine Klasse, die dieses Interface implementiert, kann auf einem `TrackingObject` registriert werden, um über die Methode `onPoseFrame(...)` zugriff auf die Position des gefundenen Zieles und das `ARData`-Objekt zu erhalten.
Das Problem bei diesem Vorgehen ist, dass der Zugriff auf das Kamerabild nur über ein zu erkennendes Objekt/Bild erfolgen kann. Folglich ist es momentan nur möglich eigene Tracking-Logik zu verwenden, wenn die bereits vorhandenen Tracking-Mechanismen zum Einsatz kommen. Dies ist jedoch nicht immer erwünscht.
Das Problem ließe sich dadurch umgehen, dass absichtlich möglichst schlechte Trainingsdaten verwenden werden um ein `TrackinObject` zu erstellen. Dieses kann (fast) nie gefunden werden. Ein Listener, der auf dem `TrackingObject` registriert wird, hätte Zugriff auf die Kamerabilder. Die können dann für selbstentwickelte Trainingsmechanismen verwendet werden. Diese Lösung ist jedoch nicht optimal und ist nach Möglichkeit zu vermeiden.

Portabilität

Das Moverio AR SDK wurde speziell für Epson Moverio-Hardware entwickelt. Dementsprechend ist auch keine Unterstützung für andere Plattformen vorgesehen. Die Hardwareunterstützung beschränkt sich auf BT-300, BT-350, BT-2000 und BT-2200. Des Weiteren wird die Entwicklung in Unity unterstützt.

Dokumentation

Die Dokumentation für das Moverio AR SDK umfasst einen Developer Guide, die API Referenz und einige Codebeispiele. Der Developer Guide dient als erste Anlaufstelle für Entwickler und erklärt die wichtigsten Aspekte des SDKs. Darunter fallen: Inbetriebnahme der Beispiele, Anleitung für Calibration Tool und Capture Tool, Anweisungen für 2D- und 3D-Training, sowie eine Anleitung für die Entwicklung mit Unity.

3.4.5 Ergebnisse

Beide SDKs unterstützten das Verfolgen von dreidimensionalen Körpern. Die Farbabhängigkeit macht das Erkennungsverfahren von Wikitude und das textbasierte Verfahren vom Moverio AR SDK nicht geeignet, um mit dem ”Zauberwürfel” verwendet zu werden. Die Edge based Variante des Moverio AR SDK erschien zwar vielversprechend, konnte aber nicht die benötigte Stabilität mit sich bringen.
Auch das Verfolgen von Bildern wird von beiden SDKs unterstützt. Jedoch konnte keine Möglichkeit gefunden werden, die Bilderkennung zur Würfelerkennung auszunutzen, da beide SDKs keine transparenten Pixel unterstützen.
Das Moverio AR SDK wurde zwar speziell für die Epson Moverio Hardware entwickelt, jedoch wurden auch beim Wikitude SDK keine Probleme mit der Hardwarekompatibilität festgestellt.
Beide SDKs erlauben das Augmentieren von beliebigen Inhalten. Ein dynamischer Wechsel dieser Inhalte ist in beiden Frameworks möglich.
Sowohl das Kamerabild, als auch die Position der erkannten Objekte/Bilder können in beiden SKDs bezogen werden.
Beim Erfüllen der vorgegebenen Kriterien kommt es zu weiten Überlagerungen zwischen den beiden SDKs. Die wesentlichen Unterschiede in den primären Kriterien liegen bei der farbunabhängigen Variante zur Erkennung von Objekten des Moverio AR SDKs, und der Unterstützung für die Verwendung von eigener Tracking-Logik vom Wikitude SDK.
Aufgrund des mäßigen Erfolgs bei der Verwendung der farbunabhängigen Objekterkennung, wird die Möglichkeit eigene Tracking-Algorithmen zu verwenden als ausschlaggebender Faktor für die Auswahl verwendet. Da das Wikitude SDK in diesem Aspekt vorne liegt, fällt die finale Wahl auf das Wikitude SDK.
Da die primären Kriterien für die Auswahl eines SDK ausreichend waren, ist der Einbezug der sekundären Kriterien in die Auswahl optional. Wenn dies nicht der Fall wäre, würde die Auswahl anhand der sekundären Kriterien getroffen werden. Auch dies würde zur Wahl des Wikitude SDKs führen, da es in Betracht des Portabilität-Kriteriums weit besser abschneidet.

4 Zauberwürfel-Assistent-Anwendung

In diesem Kapitel geht es um die Zauberwürfel-Assistent-Anwendung, die im Rahmen dieser Arbeit entwickelt wurde. Der Fokus liegt hier besonders auf der Architektur und der Funktionsweise der Anwendung. Zudem beinhaltet das Kapitel eine kurze Anleitung für die Verwendung der Software.

4.1 Architektur

Abbildung 12 zeigt die Architektur der entwickelten Software.

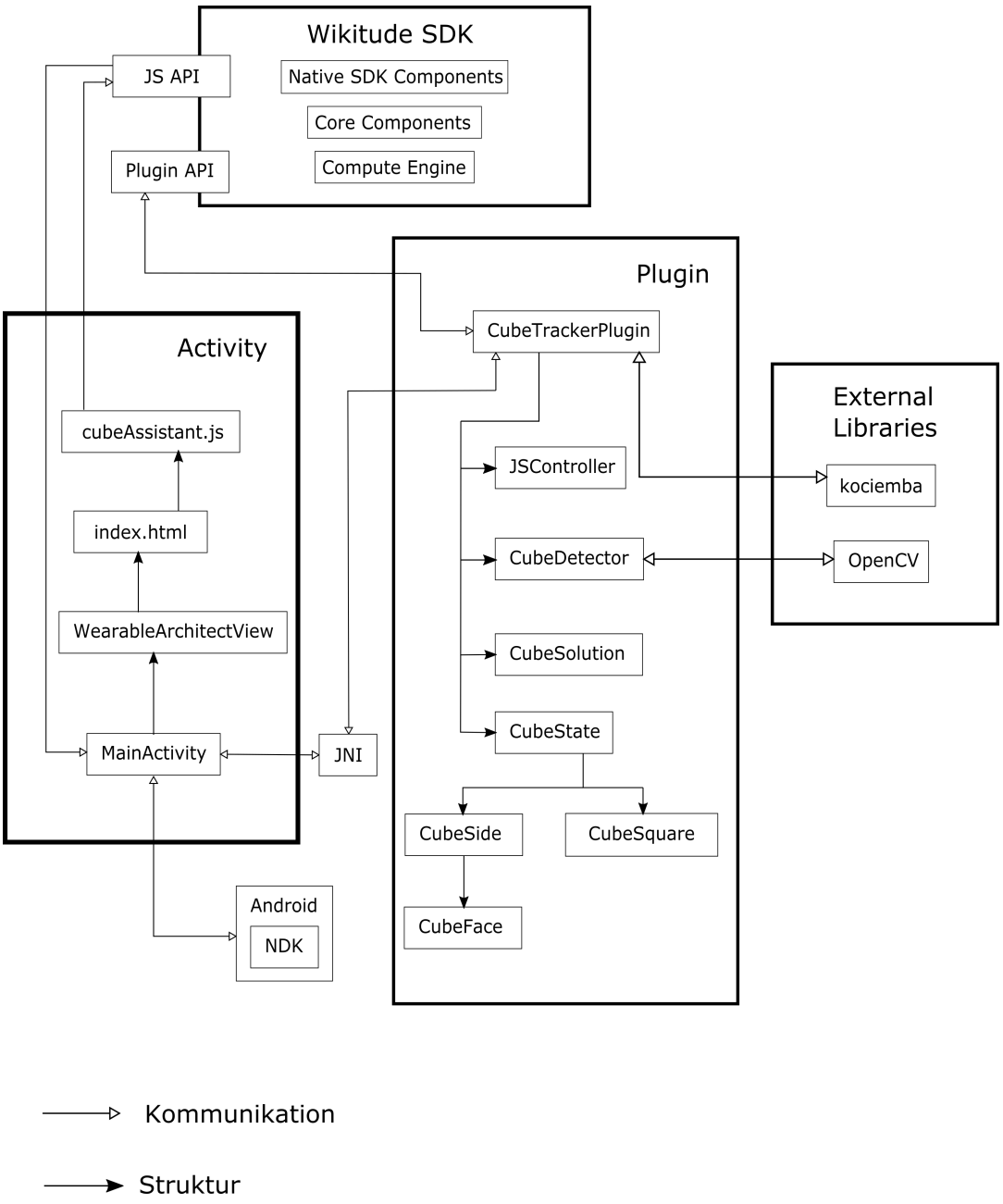

Abbildung 12: Architektur der entwickelten Zauberwürfel-Assistent-Anwendung

4.1.1 Plugin

In der Komponente *Plugin* befinden sich ausschließlich C++ Klassen. Sie ist für sämtliche Interaktionen mit dem Zauberwürfel verantwortlich und enthält die Hauptlogik des Zauberwürfel-Assistenten.

CubeTrackerPlugin

Die `CubeTrackerPlugin`-Klasse leitet von `Plugin` ab. Dadurch wird sie zur Hauptschnittstelle zwischen dem Wikitude SDK und der restlichen Software.

JSController

Die Plugin API ermöglicht es JavaScript-Anweisungen über die Methode `void addToJavaScriptQueue(const std::string& javaScriptSnippet)` auszuführen. Dies ermöglicht es die GUI aus C++ heraus zu manipulieren.
Die `JSController`-Klasse wird von `CubeTrackerPlugin` dazu verwendet, um Funktionen in der ARchitect World aufzurufen. Für die Instanziierung von `JSController` wird eine Referenz auf ein Plugin benötigt, da die `addToJavaScriptQueue(...)`-Methode nur über eine Instanz der `Plugin`-Klasse aufgerufen werden kann. In diesem Fall wird `CubeTrackerPlugin` verwendet.

CubeSquare

`CubeSquare` repräsentiert die Aufkleber des Würfels. Die Klasse hält Informationen über die Farbe eines Aufkleber und dessen finale Position auf dem Würfel. (Sprich: Auf welcher Seite wird sich der Aufkleber befinden, wenn der Würfel den gelösten Zustand angenommen hat.)

CubeFace

Die `CubeFace`-Klasse stellt eine Würfelseite dar. So wie eine Würfelseite 3x3 Aufkleber hat, hat auch die `CubeFace`-Klasse neun `CubeSquare`-Attribute. Jedem dieser `CubeSquare`s wird dabei eine Position innerhalb der Seite zugewiesen (oben-links, oben, oben-rechts, links, zentral, rechts, unten-links, unten, unten-rechts). Durch dieser Zuordnung ist es möglich den Zustand einer Würfelseite abzubilden.

CubeSide

`CubeSide` ist eine Klasse die von `CubeFace` ableitet. Zusätzlich den bereits bekannten Attributen der `CubeFace`-Klasse, hat `CubeSide` noch zwölf Referenzen auf die

angrenzenden Aufkleber der Nachbarseiten. Abbildung 13 verdeutlicht den Unterschied zwischen `CubeFace` und `CubeSide` nochmal visuell.

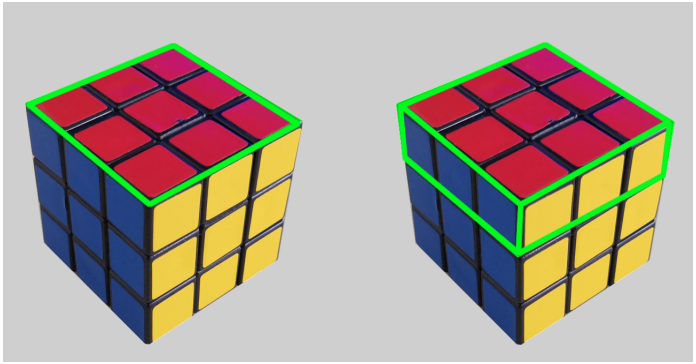

Abbildung 13: Auf dem linken Würfel wurde ein Bereich markiert, der von der `CubeFace`-Klasse abstrahiert wird. Auf dem rechten Würfel wurde das selbe für die `CubeSide`-Klasse getan. Wie zu sehen ist, besteht der Unterschied zwischen den beiden darin, dass `CubeFace` tatsächlich nur eine Seite des Würfels darstellt, während `CubeSide` ein komplette Schicht (aus neuen Cubies) darstellt.

`CubeSide` wird verwendet, um den Gesamtzustand bei einer Drehung korrekt upzudaten. Die `CubeFace`-Klasse ist hierfür nicht ausreichen, da bei einer Drehung nicht nur die Aufkleber einer Seite die Position wechseln, sondern auch die Aufkleber, die an die besagte Seite angrenzen. Grund hierfür ist, dass sich die Aufkleber der Seite und die angrenzenden Aufkleber auf den selben Cubies befinden.

CubeState

Der Würfelzustand wird von der `CubeState`-Klasse repräsentiert. Sie hält sechs Instanzen von `CubeSide`-Objekten, die die sechs Würfelseiten darstellen.

CubeSolution

`CubeSolution` ist eine Hilfsklasse, deren primäre Aufgabe darin besteht, die Lösungsphase vorzubereiten. Sie wird mit einem String, der den Lösungsweg in der Würfelnotation (siehe Kapitel 3) darstellt und dem ermittelten Würfelzustand (= Startzustand) initialisiert. Anhand dieser Informationen lassen sich die Zustände zwischen dem Start- und Endzustand (= gelöster Würfel) bestimmen. Durch diese Zwischenzustände ist es möglich zu überprüfen, ob der Anwender dem vorgegebenen Lösungsweg folgt. Dies passiert durch einen Vergleich zwischen der von

der Kamera aufgenommenen Ist-Würfelseite, und der Soll-Würfelseite des aktuellen Zwischenzustands.
Eine weitere Aufgabe der `CubeSolution`-Klasse besteht darin, den Lösungsweg so zu manipulieren, dass alle darin vorhandene Schritte überprüft werden können. Dies stellt besonders für Drehungen der B-Seite (Rückseite) ein Problem dar, da sie nicht von der Kamera gesehen werden können. Um dieses Problem zu umgehen, werden Zwischenschritt eingefügt, die den Würfel so drehen, dass die B-Seite sichtbar ist.

CubeDetector

Die `CubeDetector`-Klasse wird dazu verwendet um den Zauberwürfel im Kamerabild zu finden und die Farben seiner Aufkleber zu erkennen. Die Klasse hat nur zwei Methoden:

- `bool findCube(cv::Mat srcImg)`
- `CubeFace getCurrentFace()`

Die `findCube(...)`-Methode wird dazu verwendet, um die Position des Würfels zu ermitteln. Wenn der Würfel nicht erkannt wurde, gibt die Methode `false` zurück. Andernfalls resultiert die Methode in `true` und die Würfelposition kann abgefragt werden.
Über `getCurrentFace()` wird die Repräsentation der erkannten Würfelseite als `CubeFace`-Objekt übergeben. Die `CubeSquare`s innerhalb dieses Objekts sind mit dem durchschnittlichen Farbwert für den entsprechenden Aufkleber versehen. Sie haben jedoch noch keine finale Position, da diese erst nach dem Scan-Prozess bestimmt wird.

4.1.2 External Libraries

Unter *External Libraries* (externe Bibliotheken) wurden die Komponenten zusammengefasst, die nicht selbst entwickelt wurden, aber auch nicht Teil der AR SDKs sind.

kociemba

Für die Generierung des Lösungsweges für den Zauberwürfel wurde eine Open-Source C-Implementierung des Kociemba-Algorithmus verwendet. Sie nimmt den Würfelzustand in Form einer Strings in der Würfelnotation (siehe Kapitel 2) entgegen, und generiert einen Lösungsweg. Dabei handelt es sich um einen String, der aus einzelnen Lösungsschritte in Würfelnotation besteht.

OpenCV

OpenCV ist eine Open-Source-Bibliothek für Bildverarbeitung. Sie steht unter einer BSD-Lizenz und ist somit für akademische und kommerzielle Verwendung gratis verfügbar. Sie stellt API für die Programmiersprachen Java, C++ und Python zur Verfügung. Die Platzformunterstützung umfasst Windows, Linux, MacOS, iOS und Android.[18]
In der Zauberwürel-Assistent-App kommt OpenCV in der `CubeDetector`-Klasse zum Einsatz.

4.1.3 Activity

Activities (Aktivitäten) sind Teile einer Android-App, die dafür zuständig sind, den Nutzer eine bestimme Tätigkeit ausüben zu lassen. Die `Activity`-Klasse ist dafür zuständig ein Fester zu erzeugen, indem die UI (User Interface) platziert wird.[1] In diesem Fall besteht die UI nur aus der `WearableArchitectView`.

JNI

Java Native Interface (JNI) ist ein Framework, das als Bindeglied zwischen Java und nativen Anwendungen fungiert. Es ermöglicht Java-Code, der in der JVM (Java Virtual Machine) läuft, aus Plattform-spezifischen Code (z.B.: C/C++) heraus aufzurufen. Auch die andere Richtung wird unterstützt, wodurch Plattform-spezifischer Code aus Java-Code heraus aufgerufen werden kann.
Das JNI wird zwischen der `MainActivity` und dem `CubeTrackerPlugin` verwendet, und ermöglicht es den beiden miteinander zu interagieren.

MainActivity

Die `MainAcitivity`-Klasse ist der Einstiegspunkt der Android-Anwendung. Über sie wird das CubeTrackerPlugin registriert und die GUI (= index.html) geladen.

WearableArchitectView

Bei der `WearableArchitectView` handelt es sich um eine Wikitude-spezifische View für Android-Anwendungen. Sie stellt die Schnittstelle zur Java-API des Wikitude SDKs dar und dient gleichzeitig als zentrale Visuelle Komponente, in der die ARchitect World angezeigt wird.

index.html

Die `index.html`-Datei stellt den visuellen Teil der ARchitect World dar, welche die Anzeige der App bildet.

cubeAssistant.js

Die `cubeAssistant.js`-Datei beinhaltet den Logik-Teil der ARchitect World. Das Erkennen von Objekten und Bildern, sowie das Anzeigen von virtuellen Inhalten werden in dieser Datei gesteuert.

4.1.4 Android - NDK

Das NDK (Native Development Kit) wird dafür verwendet, um native Bibliotheken aus C/C++-Code zu erzeugen, die dann von Android-Anwendungen benutzt werden können.

4.2 Funktionsweise

Die ursprüngliche Idee war es, die Position des Würfels durch ein Framework bestimmen zu lassen und nur die Zustandserkennung selbst zu implementieren. Da dies jedoch nicht nicht möglich war, musste auch die Positionsbestimmung des

Würfels selbst implementiert werden. Hierfür wurde die Bildverarbeitungsbibliothek *OpenCV* verwendet.
Dieses Kapitel beschreibt die Positionsbestimmung, Zustandserkennung und Lösungsgenerierung.

4.2.1 Positionsbestimmung

Das Plugin-API des Wikitude-SDKs bietet die Methode `cameraFrameAvailable( const wikitude::sdk:: Frame &cameraFrame_)` als Einstiegspunkt für die Erweiterung des Wikitude-Funktionsumfangs an. Im Parameter `cameraFrame_` befinden sich diverse Meta-Informationen über das aktuellen Kamerabild, sowie die eigentlichen Bilddaten im YUV420-Format.
Das Verfahren zur Positionsbestimmung besteht im wesentlichen aus den folgenden zwei Schritten:

1. Schritt: Erkennen von Quadraten

Die Erkennung von Quadraten basiert auf dem OpenCV-Beispiel mit der selbigen Aufgabe[12]. Das Vorgehen hat diese fünf Schritte:

1. Konvertierung zu Schwarz-Weiß-Bild
2. Glättung des Bildes mithilfe eines Gauß-Filters
3. Erkennung der Kanten mithilfe von Canny-Algorithmus
4. Abbildung von Kanten zu Formen (Vielecken) durch Approximation
5. Aussortieren von allen nicht-quadratischen, und zu kleinen Formen.

2. Schritt: Anordnung von Quadraten in 3x3-Gitter

Im zweiten Schritt wird versucht in den zuvor erkannten Quadraten ein 3x3-Gitter zu finden. Der Algorithmus besteht aus den folgenden drei Schritten:

1. Berechnung von den Abständen D_{ij} zwischen den Mittelpunkten von allen zuvor gefunden Quadraten Q_i zu den Mittelpunkten von allen anderen Quadraten Q_j. Zum besseren Verständnis dient folgender Pseudo-Code:

```
ssquares = [Q1, Q2, ..., Qn] #array of n squares
distances = [n][n] #n*n array for distances between squares
for Qi in squares:
    for Qj in squares:
        if Qi == Qj: #skip calculation for identical squares
            distances[Qi][Qj] = 0
            continue
        Dij = dist(Qi.center, Qj.center) #calculate distance
        distances[Qi][Qj] = Dij
```

2. Über die berechneten Abstände lassen sich Aussagen darüber treffen ob zwei Quadrate Qi und Qj Teil einer Würfelseite sein können. Der Abstand zwischen zwei direkt benachbarten Quadraten entspricht etwa der Breite eines Quadrates. Ist der Abstand wesentlich größer oder kleiner, handelt es sich nicht um benachbarte Quadrate. Des Weiteren sollen die Fläche von Qi und Qj nahezu identisch sein. Ist dies nicht der Fall, sind Q_i und Q_j nicht als benachbarte Quadrate anzusehen, da alle Quadrate die selbe Fläche haben müssen. Jeder der drei Steintypen hat eine bestimme Anzahl an Nachbarsteinen (siehe Tabelle 2). Wenn Q_i nicht die korrekte Anzahl an Nachbarn hat, wird es im nächsten Schritt nicht weiter untersucht.

3. Über die Nachbarschaft lässt sich die Position der Quadrate bestimmen. Berechnet man den normierten Richtungsvektor v_{ij} von Qi zu dessen Nachbarn Qj, so kann dieser nur einen von vier Werten annehmen: $(0,1)$ $\uparrow$, $(1,0)$ $\rightarrow$, $(0,-1)$ $\downarrow$, $(-1,0)$ $\leftarrow$. In Tabelle 2 kann beobachtet werden, dass die Position eines Steines eindeutig über die Richtungsvektoren zu seinen Nachbarn bestimmt werden kann.

Tabelle 2: Nachbarschaftsbeziehungen zwischen Cubie-Arten

Position	Nachbarzahl	Richtungsvektoren
oben links	2	$\downarrow, \rightarrow$
oben	3	$\leftarrow, \downarrow, \rightarrow$
oben rechts	2	$\leftarrow, \downarrow$
links	3	$\uparrow, \rightarrow, \downarrow$
zentral	4	$\uparrow, \leftarrow, \rightarrow, \downarrow$
rechts	3	$\uparrow, \leftarrow, \downarrow$
unten links	2	$\uparrow, \rightarrow$
unten	3	$\leftarrow, \uparrow, \rightarrow$
unten rechts	2	$\leftarrow, \uparrow$

Die eigentliche Implementierung befindet sich auf der beiliegenden CD im Verzeichnis: `CubeAssistant/app/src/main/cpp/src/detector/CubeDetector.cpp`

4.2.2 Zustandsbestimmung

Ein Lösungsweg für den verdrehten Würfel kann nicht generiert werden, wenn dessen exakter Zustand nicht bekannt ist. Dieser Abschnitt beschäftigt sich mit den Ansätzen, die für die Erkennung des Würfelzustands verfolgt wurden.
Für die eindeutige Bestimmung des Würfelzustands sind die folgenden Eigenschaften zu bestimmen:

- Farbe der Aufkleber auf jeder Seite
- Position der Aufkleber auf der entsprechenden Seite
- Anordnung der Seitenfarben zu einander (festgelegt durch die Farbe der Center-Steine)
- Farbe der F-Seite (die Farben der anderen Seiten lassen sich über die zuvor bestimmte Anordnung der Seitenfarben ermitteln)

Bei Verwendung einer Kamera, liegt die Anzahl an gleichzeitig sichtbaren Würfelseiten bei drei. Es werden aber Information von allen sechs Seiten benötigt,

um den Zustand rekonstruieren zu können. Folglich muss der Scan-Prozess in mindestens zwei Schritten erfolgen.

1. Ansatz: Der erste Ansatz bestand darin, drei Seien gleichzeitig einzuscannen (siehe Abbildung 4), den Würfel um $180°$ zu drehen und die anderen drei Seiten zu scannen. Dieses Vorgehen wurde verworfen, weil die Reflexionen an der Oberseite zu Problemen bei der Farberkennung führten.

2. Ansatz: Der zweite Ansatz sah vor, nur eine Seite gleichzeitig zu scannen. Dafür wird der Würfel in der Frontalansicht betrachtet (siehe Abbildung 5). Dies hat den Vorteil, dass alle Seiten in nahezu der selben Position befinden. Winkelbedingte Reflexionen können so leichter vermieden werden.
Das Problem bei diesem Verfahren ist jedoch, dass stets nur eine Seite des Würfels sichtbar ist. Bei der Farberkennung führt dies zwar nicht zu Problemen, aber erschwert später die Zustandsverfolgung während des Lösens des Würfels. Zudem erfolgt der Scan-Prozess in sechs Schritten. Dies macht ihn anfälliger für potentielle Fehler.
Trotz der Nachteile, wurde der zweite Ansatz gewählt, da dort die Farberkennung besser funktioniert hat.

Farberkennung

Die präzise Erkennung von Farben ist eine der größten Herausforderungen in der Bildverarbeitung.[20] Grund hierfür ist, dass die Farberkennung von vielen Faktoren (z.B. Lichtverhältnisse) abhängig ist. Dies konnte auch während der Implementierung der Zauberwürfel-Assistent-App beobachtet werden. Um die bestmögliche Lösung zu finden, wurden diverse Ansätze ausprobiert. Sie werden in diesem Kapitel aufgelistet und kurz erklärt.

1. Ansatz: RGB Das Plugin-API von wikitude liefert die Bilddaten standardmäßig im YUV420-Format. Mithilfe von OpenCV lässt sich das Bild in den bekannten RGB-Farbraum konvertieren. Im RGB-Farbraum setzen sich die Farben aus einem Rotwert (R), einem Grünwert (G) und einem Blauwert (B) zusammen. Je größer ein Wert ist, desto größer ist dessen Anteil an der Gesamtfarbe.

Ein Versuch die RGB-Werte für jede Würfelfarbe auszulesen, abzuspeichern und künftige Messwerte mit den gespeicherten Farben abzugleichen, kann nicht erfolgreich ausfallen. Zwei aufeinander folgende Kamerabilder werden selten die exakt selben RGB-Werte liefern, da diese von äußeren Einflüssen (z.B. Lichtverhältnisse, Winkel, etc.) beeinflusst werden. Daher bietet es sich an, einen Farbbereich für jede Farbe zu definieren. Wenn ein Messwert z.B. in den blauen Farbbereich fällt, kann angenommen werden, dass der zugehörige Aufkleber blau ist. Dies erwies sich jedoch als Problem, da die RGB-Farben zu stark von den einzelnen Komponenten abhängig sind. Dies hat die Definition von Farbbereichen erheblich erschwert.

2. Ansatz: HSV Im nächsten Ansatz wurde der HSV-Farbraumes anstelle des RGB-Farbraumes verwendet. HSV-Farben bestehen aus Hue (Farbwert), Saturation (Sättigung) und Value (Hellwert). Die H-Komponente repräsentiert die eigentliche Farbe (z.B. blau, rot, grün) und die beiden andern Komponenten sind für die Nuancen (z.B. hellblau, dunkelblau) zuständig. Da in den gängigsten Würfelfarben (siehe Kapitel 2.1) keine Nuancen der selben Farbe vorkommen, können die Farben allein durch ihre unterschiedlichen H-Werte von einander unterschieden werden. Da die H-Werte auf einem Spektrum liegen, ist es einfach Bereiche für die Einzelnen Farben zu definieren. Den Ausnahmefall bildet die weiße Farbe, da sie einen beliebigen H-Wert annehmen kann. Dies ist darauf zurückzuführen, dass Weiß als Fehlen von Sättigung angesehen werden kann. Reduziert man den S-Wert einer HSV-Farbe stark genug, so wird diese zu Weiß. Zusätzlich zu der Betrachtung der H-Werte müssen also auch die S-Wert überprüft werden, um auch die weiße Farbe erkennen zu können. Ein Beispiel für HSV-Farbbereiche kann der Tabelle 3 entnommen werden.

Tabelle 3: HSV-Farbbereiche für: rot, blau, weiß, orange, grün, gelb

Farbe	H	S	V
rot	$0-19, 340-360$	$20-100$	$50-100$
blau	$200-250$	$20-100$	$50-100$
weiß	$0-360$	$0-19$	$50-100$
orange	$20-44$	$20-100$	$50-100$
grün	$90-180$	$20-100$	$50-100$
gelb	$45-70$	$20-100$	$50-100$

Beim Testen kam es zu Problemen bei der Unterscheidung von den rot- und orangefarbenen Aufklebern. Je nachdem welche Lichtverhältnisse herrschten, konnte es vorkommen, dass orangefarbene Aufkleber als rot erkannt werden und rote Aufkleber als orange. Eine Vergrößerung der Farbbereiche war nicht möglich, da die beiden Farben zu nah beieinander liegen und es dadurch zu einer Überschneidung der beiden Farbbereiche kommt. Auch dieser Ansatz wurde verworfen.

3. Ansatz: YUV + Normierung Der nächste Ansatz war an das Verfahren von Kasprzak[22] angelehnt. Dabei wird der YUV-Farbraum dazu verwendet, um die Farben im Bild zu normieren und dadurch zu vereinheitlichen. Die Annahme war, dass die normierten Farben sich stärker von einander unterscheiden werden und dadurch weniger Überschneidungen zwischen Orange und Rot auftreten. Außerdem kann man sich die Konvertierung des Farbraums sparen, da das Kamerabild bereits im YUV-Format vorliegt. Auch hier kam es zu Problemen bei der Unterscheidung von Rot und Orange, und der Ansatz wurde nicht weiter verfolgt.

4. Ansatz: CIEDE2000 Der zweite und dritte Ansatz wiesen die selben Probleme auf, hatten aber auch eine ähnliche Grundidee: Statische Farbbereiche für jede Würfelfarbe definieren und den Aufklebern anhand dieser Bereiche eine Farbe zuordnen. Aufgrund der bisherigen Ergebnisse wurde die Annahme getroffen, dass solche (oder ähnliche) Methoden nicht erfolgreich sein werden.
Folglich wurde für den vierten Ansatz eine Methode gewählt, bei der die Farbwerte dynamisch festgelegt werden. Dafür müssen die Bilddaten in den Lab-Farb-

raum konvertiert werden. Dieser Farbraum wurde speziell dafür entwickelt um die menschliche Farbempfindung besser zu beschreiben als andere Farbräume. Durch diese Eigenschaft ist es auch möglich den Farbunterschied zwischen zwei Farben so zu bestimmen, wie er vom Menschen wahrgenommen wird. Dafür verwendet man die CIEDE2000-Formel [27]. Die Formel nimmt zwei Lab-Farben als Parameter entgegen und berechnet den Wert ΔE. Dieser repräsentiert die Farbdifferenz zwischen den beiden Farben. Je größer ΔE ist, desto unterschiedlicher werden die beiden Farbwerte wahrgenommen.
Unter Verwendung der CIEDE2000-Formel ist folgender Algorithmus für die Farberkennung entwickelt worden:

Scan-Prozess

1. Reihenfolge der Seiten beim Scann-Prozess festlegen (Der Anwender muss die Seiten in dieser Reihenfolge Scannen um am Ende einen korrekten Würfelzustand zu erhalten)
2. YUV-Bild in Lab-Bild konvertieren.
3. Scannen einer Würfelseite (vorgegeben durch Reihenfolge aus 1.)
4. Speichern der Seitenart (F, U, R, B, D, L) und der durchschnittlichen Farbwerte für jedes Quadrat in der aktuellen Seite.
5. Scannen der nächsten Seite (vorgegeben durch Reihenfolge aus 1.)
6. Wiederholen der Schritte 3. - 5. bis alle Seiten eingescannt wurden.

An dieser Stelle im Algorithmus, befinden sich alle sechs Würfelseiten-Modelle im Speicher. Jedes Seitenmodel hat die Rohfarbwerte und die Position für die neuen Quadrate, welche Teil der besagten Seite sind. Des Weiteren gibt es eine Zuordnung, die angibt wo sich welche Seite am Würfel befindet. Für die Bestimmung des vollständigen Würfelzustands bedarf es nur noch der Zuordnung von den Quadraten an eine Endposition(Position die jedes Quadrat hat sobald der Würfel sich wieder im gelösten Zustand befindet). In den Ansätzen 2. und 3. erfolgt dies über die Zuweisung von vordefinierten Farben. Da die Position der

mittleren Quadrate (Center) fest ist, ist eine Festlegung der Endposition der äußeren Quadrate (Edges und Corners) anhand dieser Farben möglich. Bsp.: Ein blaues äußeres Quadrat muss sich im gelösten Würfel immer in der Seite befinden, in der sich auch das blaue mittlere Quadrat befindet.
Durch betrachten dieses Vorgehens zur Endpositionsbestimmung lies sich die folgende Annahme treffen: Wenn man eine Möglichkeit finden, die die Zuweisung von äußeren zu mittleren Quadraten ermöglicht, ohne ihnen eine Farbe anhand von vordefinierten Farbbereichen zuzuordnen, werden die Probleme aus dem zweiten und dritten Ansatz nicht auftreten.
Dafür wurde dieser Algorithmus entwickelt:

Seitenzuweisung

1. Die Endposition jedes mittleren Quadrates wird auf die Seite gesetzt, in der sich das Quadrat bereits befindet.
2. Bestimme die Farbdifferenz $\Delta E_{Q_i,M_i}$ zwischen allen äußeren Quadrate Q_i und alle mittleren Quadrate M_i.
3. Für jedes äußere Quadrat Q_i wird der kleinste Wert für $\Delta E_{Q_i,M_i}$ bestimmt.
4. Jedes Q_i nimmt die selbe Endposition an, wie das mittlere Quadrat M_i, das zu dem kleinsten $\Delta E_{Q_i,M_i}$-Wert gehört.

Diese Methode funktioniert stabiler als die Anderen, da sie unabhängig davon den aktuellen Lichtverhältnissen ist. Bei weniger Beleuchtung wird jeder Farbe dunkler wahrgenommen und bei mehr Beleuchtung wird jede Farbe heller wahrgenommen. Bsp.: Ein hell-/dunkelrotes Quadrat hat stets eine kleinere Farbdistanz zu einem anderen hell-/dunkelroten Quadrat als zu einem hell-/dunkelblauen Quadrat.
Ein weiterer Vorteil dieses Ansatzes ist, dass er fast komplett unabhängig von den Farben eines Zauberwürfels ist. Wie in Kapitel 2 bereits erwähnt, gibt es Zauberwürfel mit verschiedensten Farben, Farbnuancen und Farbschemata (Anordnung der Farben auf dem Würfel). Ein statisches Vorgehen, wie Ansatz 2 und 3, kann solche Würfel nur dann unterstützen, wenn man einen Farbbereich für jede mögliche Farbe definiert. Dies ist in der Praxis jedoch kaum unabsetzbar sein.

Ansatz 4 hat einen großen Nachteil: Während des Scann-Prozesses gibt es keine Möglichkeit Quadrate einer Seite zuzuordnen bis alle Seiten eingescannt wurden. Da die Kamera schneller filmt als der Nutzer den Würfel drehen kann, kommen pro Seite mehrere Bilder an. Um zu verhindern, dass eine Seite mehrere Male gespeichert wird, muss festgestellt werden ob eine Seite bereits gescannt wurde, oder ob es sich um eine neue Seite handelt, die noch gespeichert werden muss. Bei fest definierten Farbbereichen besteht die Möglichkeit die Farben einer Seite sofort zu bestimmen, ohne die anderen Seiten angesehen zu haben. Dadurch kann man zwei aufeinanderfolgende Seiten auf Gleichheit der Farbe der Center überprüfen. Wenn diese die selbe Farbe haben, so ist davon auszugehen, dass in den beiden Bildern die selbe Würfelseite zu sehen ist und es daher keiner neuen Speicherung bedarf.
Diese Möglichkeit existiert bei der Farbdistanz-Variante nicht. Die einzige Möglichkeit zwei Würfelseiten auf Gleichheit zu überprüfen bevor alle Seiten gescannt wurden, ist die Farbdifferenz der Center zu bestimmen. Da zwei Bilder in der Regel aber nicht die exakt selben Farbwerte haben, wird die Farbdifferenz nicht 0 sein. Es muss also ein Toleranzwert gewählt werden, der ausschlaggebend darüber sein soll, ob zwei Farben als gleich oder ungleich angesehen werden sollen.

Problem: Wenn dieser Wert zu klein gewählt wird, kommt es vor, dass zwei gleiche Seiten als ungleich angesehen werden. Wird der Wert zu groß gewählt, führt dies dazu, dass zwei ähnliche Farben, während des Scan-Prozesses, nicht mehr unterschieden werden können.
Es konnte beobachtet werden, dass die Farbdifferenz in hellen Räumen generell größer was als in dunkleren Räumen. Dies verkomplizierte die Wahl eines sinnvollen Toleranzwertes weiter. Besonders problematisch war die Unterscheidung von Rot und Orange in Räumen mit wenig Licht.
Um sicherzustellen, dass die Probleme nicht auf die Hardware zurück zu führen waren, wurden die Ansätze auf einem Rechner getestet. Dafür wurden Bilder verwendet, die mit der Brillenkamera aufgenommen wurden, um zu gewährleisten, dass die Tests nicht durch verschiedene Kameras verfälscht werden. Das Resultat war, dass alle Ansätze wesentlich besser auf dem Rechner funktionierten, als auf der Brille. Da beide Geräte mit den selben OpenCV-Code verwenden, bestand

der einzige Unterschied darin, dass die Wikitude Plugin-API auf der Brille verwendet wurde und nicht auf dem Rechner. Ein direkter Vergleich zwischen dem Feed der normalen Kamera-App und des vom Plugin-API gelieferten Bildes zeigte, dass das Bild vom Plugin-API wesentlich dunkler war. Das Bild wirkt unterbelichtet.

Lösung: Eine simple Aufhellung des Bildes führt oft dazu, dass das Bild an den bereits hellen Bereichen unnatürlich wirkt. Das ist darauf zurückzuführen, dass ist darauf zurückzuführen, dass Menschen die Steigerung der Helligkeit in dunklen Bereichen anders wahrnehmen als in hellen Bereichen. Da sich die Berechnung der Farbdifferenz stark an der menschlichen Wahrnehmung von Farben orientiert, führt dies unweigerlich zu Problemen.
Die Gamma-Korrektur ist ein Verfahren, das dazu verwendet werden um unter-/überbelichtete Bilder so zu korrigieren, dass es für den Menschen natürlich wirkt. Bei unterbelichteten Bildern werden die dunkler Flächen stärker aufgehellt als die helleren Flächen. Bei überbelichteten Bildern werden die dunkleren Flächen schwächer abgedunkelt als die Helleren.
Durch die zusätzliche Verwendung der Gamma-Korrektur kann der Toleranzwert groß genug gewählt werden, dass gleiche Farben nicht als ungleich angesehen werden. Die daraus resultierende Aufhellung führt dazu, dass die Farbdifferenz zwischen zwei ungleichen Farben, auch bei schlechtem Licht, groß genug ist um als ungleich erkannt zu werden. Gleichzeitig werden die Bilder bei gutem Licht nicht so stark aufgehellt, dass alle Farben weiß werden.

4.2.3 Lösungsprozess

Nach erfolgreicher Vollendung des Scan-Prozesses befindet sich ein eine exakte Darstellung des Würfelzustands im `CubeState`-Objekt. Sie kann dazu verwendet werden, um einen Lösungsweg zu finden, der den Würfel wieder in den Lösungszustand bringt. Aufgrund der hohen Komplexität ist dies allerdings nicht trivial. Glücklicherweise gibt es bereits Algorithmen, die speziell für diesen Zweck entwickelt wurden. Die folgenden drei Algorithmen wurden für den Einsatz in der App in Erwägung gezogen: Thistlethwaites Algorithmus[29], Kociembas Algorithmus[7] und Korfs Algorithmus (IDA*)[24].

Der Lösungsweg, der von Thistlethwaites Algorithmus generiert wird, ist länger als die Lösungswege der beiden anderen Algorithmen. Dafür braucht dessen Generierung aber auch weniger Zeit als die bei den beiden Andren. Korfs Algorithmus generiert die kürzeste Lösung, die sehr nah an die optimale Lösung kommt. Es ist aber auch der langsamste der drei Algorithmen. Kociembas Algorithmus ist in beiden Aspekten zwischen den anderen zwei Algorithmen.
Für die Software wurde der Kociemba-Algorithmus verwendet, da er einen Kompromiss zwischen kurzen Weg und kurzer Berechnungszeit darstellt. Anstatt den Algorithmus selbst zu implementieren, wurde eine bereits existierende Lösung verwendet.[30]

4.3 Anleitung

Diese Kapitel ist eine Anleitung für die Bedienung der Software dar. Es soll das Verhalten der Software beschreiben und aufzeigen, was es während ihrer Verwendung zu beachtet gibt.

4.3.1 Interface

In Abbildung 14 ist das Schema des Interfaces zu sehen. Seine Aufgabe besteht darin, die den Anwender mit den notwendigen Informationen zu versorgen. Eine Beschreibung der einzelnen Komponenten und ihren Aufgaben findet sich in diesem Kapitel.

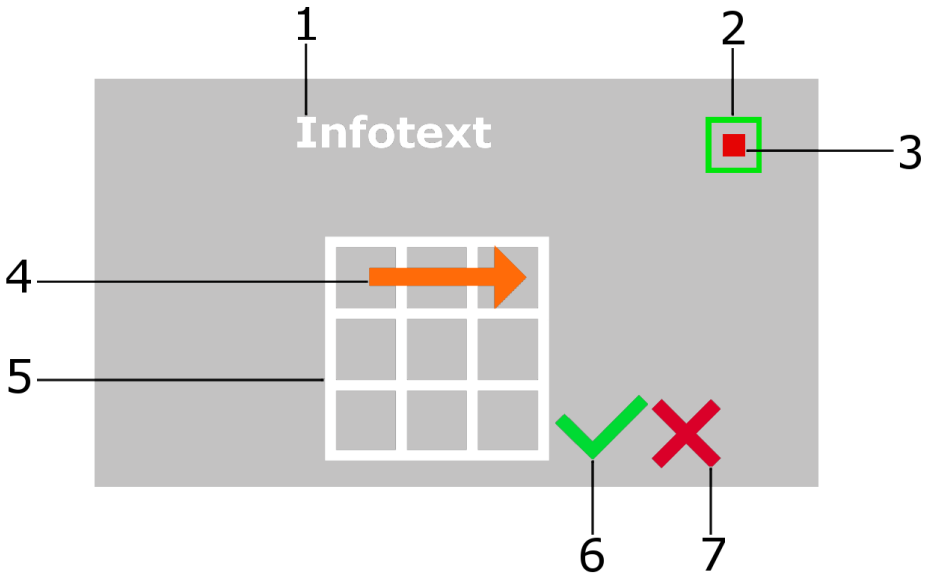

Abbildung 14: Die Abbildung zeigt das Schema vom Nutzer-Interface. Der graue Hintergrund ist auf der Brille transparent. Hier dient er der besseren Sichtbarkeit. Das einzelnen Komponenten des Interface sind wie folgt nummeriert:
1: Textfeld
2: Würfel-Erkannt-Indikator
3: Korrekte-Seite-Indikator
4: Nächste-Drehung-Indikator
5: Gitter
6: Korrekte-Drehung-Indikator
7: Falsche-Drehung-Indikator

Textfeld: Das Textfeld wird dazu verwendet, um dem Anwender Informationen zukommen zu lassen und ihm Anweisungen zu geben.

Würfel-Erkannt-Indikator: Der Würfel-Erkannt-Indikator gibt Auskunft darüber, ob ein Zauberwürfel im aktuellen Kamerabild gefunden wurde. Wenn dies nicht der Fall ist, wird der Indikator rot eingefärbt. Wenn der Würfel erkannt wurde, ist der Indikator grün.

Korrekte-Seite-Indikator: Mithilfe dieses Indikators kann der Anwender erkennen, ob es sich bei der erkannten Vorderseite um die korrekte Seite handelt. Wenn eine Seite erkannt wird, die sich an einer anderen Stelle befinden soll, wird der In-

dikator rot eingefärbt. Der Indikator ist grün, solange sich die korrekte Seite vorne befindet. Der Indikator wird während des Scan-Prozesses nicht angezeigt, da der Lösungsweg erst nach dem erfolgreichen Scannen des gesamten Würfels generiert werden kann.

Nächste-Drehung-Indikator: Der Nächste-Drehung-Indikator zeigt die Drehung, welche dem aktuellen Lösungsschritt entspricht, in Form eines Pfeiles an. Eine Übersicht aller Pfeile und ihrer Bedeutungen kann in Anhang A gefunden werden.

Gitter: Das Gitter repräsentiert die 3x3-Struktur einer Würfelseite. Es wird dazu verwendet, um die Nächste-Drehung-Indikatoren an der korrekten Stelle anzuzeigen. Dafür müssen Gitter und die Vorderseite des Würfels übereinander liegen. Die Position und die Größe des Gitters werden an die Position und Größe des erkannten Würfels angepasst.

Korrekte-Drehung-Indikator: Dieser Indikator gibt dem Anwender Feedback darüber, ob die letzte Drehung, die von der App erkannt wurde, mit dem generierten Lösungsweg übereinstimmt. Der Indikator ist normalerweise nicht sichtbar und wird nach der Drehung nur kurz eingeblendet.

Falsche-Drehung-Indikator: Der Falsche-Drehung-Indikator hat starke Ähnlichkeit zur Funktionsweise des letzten Indikators. Der Unterschied liegt darin, dass dafür verwendet wird, um anzuzeigen, dass die erkannte Drehung nicht mit der generierten Lösung übereinstimmt.

4.3.2 Scan-Phase

Die Anwendung fängt nach dem Start sofort an die Kamerabilder nach Zauberwürfeln zu untersuchen. Um den Scann-Prozess zu starten, muss lediglich ein Zauberwürfel ins Kamerabild gehalten werden. Dies soll dazu führen, dass auf dem Gitter ein Pfeil angezeigt wird. Er gibt an, wie der Würfel gedreht werden muss, um die nächste Seite zu scannen. Nachdem eine Seite eingescannt wurde, wird der Korrekte-Drehung-Indikator kurz eingeblendet. Anschließend folgt die

nächste Drehanweisung. Es ist von äußerster Wichtigkeit, dass die Drehungen während der Scan-Phase exakt befolgt werden, da die Würfelseiten in einer Bestimmen Reihenfolge eingescannt werden müssen.
Während des Scann-Prozesses werden nur komplette Würfeldrehungen an der x- und y-Achse durchgeführt. Wenn alle sechs Seiten erfolgreich eingescannt worden sind, wird der Korrekte-Seite-Indikator sichtbar. Nun versucht die Software eine Lösung anhand des Würfelzustandes zu generieren. Wenn dies nicht gelingt, wird ein Fehlertext über das Textfeld ausgegeben.
Wenn es zu Problemen bei der Lösungsgenerierung kommt, sind diese erfahrungsgemäß immer auf einen fehlerhaften Zustand zurückzuführen. Dementsprechend muss es zu einem Fehler während des Scann-Vorgangs von einer oder mehrere Seiten gekommen sein. Die möglichen Fehlerquellen sind: absichtliches oder versehentliches Nicht-Befolgen der vorgegebenen Drehanweisungen und fehlerhaftes Zuweisen der `CubeSquare`-Positionen aufgrund von suboptimalen Lichtverhältnissen, oder Reflexionen auf den Aufklebern.

Während der Verwendung der App müssen alle Aufkleber einer Würfelseite sichtbar sein damit die Software den Würfel erkennen kann.

4.3.3 Lösungsphase

Nachdem die Scan-Phase erfolgreich durchgeführt wurde, findet ein lückenloser Übergang in die Lösungsphase statt. Während der Lösungsphase zeigt der Nächste-Drehung-Indikator die Drehanweisung für den aktuellen Lösungsschritt an. Dies ist solange der Fall, bis eine neue Drehung erkannt wurde. Handelt es sich hierbei um die Drehung, die von der Drehanweisung vorgegeben wurde, so wird sie als korrekt angesehen und der Korrekte-Drehung-Indikator wird kurz eingeblendet. Anschließend wird die nächste Drehanweisung angezeigt.
Bei Erkennen einer falschen Drehung wird der Falsche-Drehung-Indikator angezeigt und die Software versucht die durchgeführte Drehung anhand der sichtbaren Würfelseite zu rekonstruieren. Wenn dies nicht gelingt, muss der Würfel erneut eingescannt werden, da der Interne Zustand der Software inkorrekt ist. Falls die Rekonstruktion gelingt, wird eine neuer Lösungsweg berechnet und der Nächste-Drehung-Indikator entsprechend angepasst.

5 Fazit

In diesem Kapitel werden die Ergebnisse der Arbeit nochmals zusammengefasst. Anschließend folgen ein kritischer Rückblick und der Ausblick auf mögliche weiterführende Arbeiten.

5.1 Zusammenfassung

Für die Entwicklung einer Zauberwürfel-Assistent-Anwendung bietet sich die Verwendung einer AR SDKs an. Um die bestmögliche Entscheidung bei der Auswahl eines solchen SDKs treffen zu können, muss eine Auswahl an verfügbaren SDKs erstellt werden. Anschließend bedarf es einer Evaluierung der Frameworks bezüglich ihrer Eignung für den Einsatz in der zu entwickelnden App.
Für die optimale Wahl des Frameworks wurde erst eine Liste mit Anforderungen an die finale Anwendung erstellt. Anschließend wurden aus dieser Liste Kriterien für die Evaluierung abgeleitet. Es fand eine Vorauswahl zwischen den siben Frameworks: Moverio AR, Wikitude, MAXST, Crunchfish, Augmenta, ARCore und Vuforia. Aus der Vorauswahl ergaben sich die zwei Kandidaten für die eigentliche Evaluierung. Das Wikitude SDK und das Moverio AR SDK wurden aufgrund ihrer Unterstützung von Bild- und Objekterkennung, sowie ihrer Kompatibilität zur verwendeten Hardware ausgewählt. Das MAXST SDK war der drittbeste Kandidat, wurde aber aus Zeitgründen nicht mehr in die Evaluation aufgenommen. Die anderen Kandidaten wurden wegen Inkompatibilität zur Hardware, Lizenzgründen oder fehlender Funktionalität nicht ausgewählt.
Die gewählten SDKs wurden anhand der zuvor definierten Kriterien evaluiert. Dabei wurde ersichtlich, dass der Zauberwürfel nicht für die Verwendung mit der Bild- oder Objekterkennung des Wikitude SDKs geeignet war. Eine ähnliche Situation zeigt sich auf beim Moverio AR SDK. Jedoch gab es ein, auf Kantenerkennung

basierendes, Verfahren das in der Lage war den Würfel zu erkennen. Die finale Wahl fiel trotzdem auf das Wikitude SDK, da das besagte Verfahren zu inkonsistent war und es die Möglichkeit bot, seine inkompatible Tracking-Funktionalität um eigene, selbst einwickelte Tracking-Logik zu erweitern.
Bei der anschließenden Entwicklung der Applikation zeigte sich die Farberkennung unter verschiedenen Lichtbedingungen als eine der größten Herausforderungen. Um sie zu bewältigen wurden diverse Ansätze durchprobiert.

Der Ansatz, der am besten funktionierte war eine Kombination aus Lab-Farbraum und Gamma-Korrektur. Der Lab-Farbraum wird für die Berechnung von Farbdistanzen verwendet. Die Gamma-Korrektur passt die Farben so an, dass diese besser voneinander unterschieden werden können.

5.2 Kritischer Rückblick

Es wäre wünschenswert gewesen weitere SDKs in die Evaluierung einzubeziehen. Die Anzahl musste jedoch aus zeitlichen Gründen auf zwei Kandidaten beschränkt werden. Grund dafür ist, dass ein ausreichend großes Zeitfenster für die eigentliche Implementierung der Anwendung eingeplant werden musste. Aufgrund von unvorhergesehenen Problemen bei der Farberkennung (siehe Kapitel 4.2.3), ist der Zeitaufwand größer ausgefallen als ursprünglich angenommen. Dementsprechend konnten keine weiteren SDKs mehr evaluiert werden.
Des Weiteren wurde die Auswahl von potentiellen SDK-Kandidaten durch weitere Faktoren eingeschränkt. Die Kompatibilität zur Epson Moverio AR ist einer dieser Faktoren. Einerseits musste die Auswahl auf SDKs beschränkt werden, die mit Android 5.1 kompatibel sind. Andererseits solle auch die optical see-through-Technologie der Brille zum Einsatz kommen. Außerdem wurde die Auswahl an Kandidaten durch nicht frei verfügbaren SDKs weiter eingeschränkt. Auch die Zielfunktionalität der App wirkte einschränkend auf die Anzahl der möglichen Kandidaten, da nur SDKs in Erwägung gezogen wurden, die Objekt- und/oder Bilderkennung unterstützen. Die Tracking-Mechanismen der beiden evaluierten SDKs waren größtenteils nicht mit dem Zauberwürfel kompatibel. Dadurch konnte eines der wichtigsten Features für AR SDKs (Bild-/Objekterkennung) nur bedingt in die

Auswahl des, für die App zu verwendenden, Frameworks einbezogen werden.
Zu Beginn der Arbeit war geplant die Tracking-Funktionalität der beiden SDKs mithilfe eines Benchmark-Tests zu vergleichen. Ein solcher Test hätte konkrete Auskünfte darüber geben können, welches SDK den Würfel in verschiedenen Situationen besser erkennt. Es wurde auf die Erhebung des Benchmarks verzichtet, da keine Möglichkeit gefunden wurde, aussagekräftige und unverfälschte Ergebnisse zu erzielen. Die Inkompatibilität des Würfels mit den Tracking-Mechanismen der SDKs machte einen direkten Vergleich unmöglich.

5.2.1 Erfüllung der Anforderungen

In Kapitel 2.3 wurden die Anforderungen für die Anwendung definiert. Sie wurden in Grund- und Bonusfunktionalität unterteilt. Alle Punkte der Grundfunktionalität wurden erfüllt. Aus der Bonusfunktionalität wurde nur die Farbschema-unabhängige Würfelerkennung umgesetzt (erster Punkt der Bonusfunktionalität). Die anderen Punkte wurden nicht umgesetzt. Grund dafür waren entweder Zeitprobleme oder Probleme bei der Umsetzung.

5.3 Ausblick

Der Ausblick unterteilt sich in zwei Kategorien: Verbesserungspotenzial des Zauberwürfel-Assistenten und weiterführende Arbeiten in Bereich von AR und Smart Glasses.

5.3.1 Verbesserungspotenzial des Zauberwürfel-Assistenten

Die Würfelerkennung und Positionsbestimmung erfolgen jeweils über die Erkennung einer Würfelseite. Sie funktioniert nur dann, wenn die Würfelseite frontal zur Kamera ausgerichtet ist. Die Seitenerkennung kann so umgeschrieben werden, dass Würfelseiten mit beliebiger Neigung erkannt werden. Dies macht nicht nur die bisherige Seitenerkennung flexibler, sondern kann auch dazu verwendet werden, um sie auf zwei oder drei Seiten gleichzeitig zu erweitern. Dies wiederum kann zu einer Verbesserung bei der Zustandserkennung des Würfels führen, da mehr Information pro Kamerabild ausgelesen werden kann.

Die Seitenerkennung funktioniert nur, wenn alle neun Quadrate sichtbar sind. Eine Verbesserung bestünde darin, eine Seite auch dann zu erkennen, wenn nicht alle Quadrate sichtbar sind. Wenn nur acht oder sieben Quadrate erkannt werden, sollen die Position und Größe der fehlenden Quadrate anhand der gefundenen Quadrate approximiert werden.
Das größte Problem, das während der Implementierungsphase auftrat, war die Farberkennung. Es wurden diverse Ansätze verfolgt um sie zu verbessern. Die Anwendung funktioniert zwar relativ stabil, ist aber immer noch stark von den vorherrschenden Lichtverhältnissen abhängig. Eine weiterführende Arbeit könne sich mit der Untersuchung von weiteren Ansätzen zur Verbesserung der Farberkennung beschäftigen.

5.3.2 Weiterführende Arbeiten

Die App wurde für die Epson Moverio BT-300 entwickelt. Die Untersuchung von anderer Hardware kann durchaus interessant sein, da diese eventuell zusätzliche Sensoren oder Softwarekompatibilität bereitstellen. Beispielsweise kann ARCore nicht auf der Moverio BT-300 verwendet werden, da deren Android-Version zu niedrig ist. Eine andere AR-Brille kann jedoch eine aktuellere Version von Android haben und wäre somit mit ARCore kompatibel. Außerdem kann man so auch den Aufwand untersuchen die Anwendung auf eine andere AR-Brille oder ein Smartphone zu portieren.
Des Weiteren hatten die spezifischen Anforderungen der entwickelten App zufolge, dass nicht der komplette Funktionsumfang der SDKs benutzt wurde. Insbesondere die Tracking-Funktionalität konnte nicht verwendet werden. Dabei ist sie eines der wichtigsten Features die für den Einsatz eines SDKs sprechen. Die Entwicklung einer App, die mit unveränderlichen Körpern und/oder Bildern arbeitet wäre interessant, da sie die eingebaute Tracking-Funktionalität der SDKs verwenden kann. Dadurch kann ein besseres Verständnis über das Potenzial der SDK-Tracking-Funktionalität unter realen Bedingungen gewonnen werden.
Ein weiteres Feature das in dieser Arbeit nicht betrachtet wurde ist die geographische Positionsbestimmung, die von manchen SDKs zur Verfügung gestellt wird. Diese kann z.B. bei der Entwicklung einer Reiseführer-App zum Einsatz kommen. Eine Arbeit die sich mit diesem Thema beschäftigt, kann untersuchen welches

SDK diesen Task am besten bewältigt. Dabei wird vor allem die Genauigkeit der Positionsbestimmung und das korrekte Erkennen der Blickrichtung relevant sein. Ein weiteres Feature, das sich für eine Reiseführer-App eignet, ist die Identifikation von Sehenswürdigkeiten und anderen Touristenattraktionen. Dafür kann die Untersuchung der Erkennung von größeren Objekten (z.B. Gebäude, Brücken, Statuen, etc.) durchgeführt werden. Hierbei können einige Probleme auftreten. Die Objekte müssen aus verschiedenen Blickwinkeln und Distanzen erkannt werden. Außerdem werden, je nach Blickwinkel und Distanz, verschiedene Teile des Objektes verdeckt werden. Da es sich um eine Anwendung handelt, die vor allem im Freien zum Einsatz kommt spielen auch Licht- und Wetterverhältnisse eine große Rolle.

Literaturverzeichnis

[1] Android activity. https://developer.android.com/reference/android/app/Activity.html. letzer Zugriff: 09.04.2019.

[2] ARCore. https://developers.google.com/ar/. letzer Zugriff: 24.03.2019.

[3] Augmenta. http://augumenta.com/sdk/. letzer Zugriff: 24.03.2019.

[4] Crunchfish. https://www.crunchfish.com/. letzer Zugriff: 24.03.2019.

[5] History of god's number. letzer Zugriff: 16.04.2019.

[6] Hololens 2 für die Armee. https://heise.de/-4365506. letzer Zugriff: 09.04.2019.

[7] Kociemba's Algorithm. http://kociemba.org/twophase.htm. letzer Zugriff: 16.04.2019.

[8] MAXST. http://maxst.com/. letzer Zugriff: 24.03.2019.

[9] Hololens. https://www.microsoft.com/de-de/p/microsoft-hololens-development-edition/8xf18pqz17ts?icid=VRMR_Cat_Nav_1-Hololens&activetab=pivot%3aoverviewtab. letzer Zugriff: 04.04.2019.

[10] Moverio AR SDK. https://tech.moverio.epson.com/restriction/en/arsdk/, . letzer Zugriff: 24.03.2019.

[11] Image of Moverio AR SDK - Training Tool. https://tech.moverio.epson.com/restriction/en/arsdk/developers_guide/_images/ObjectTrainingToolPointClick.png, . letzer Zugriff: 24.03.2019.

[12] Opencv square detection. https://github.com/opencv/opencv/blob/master/samples/cpp/squares.cpp. letzer Zugriff: 08.04.2019.

[13] Vuforia Engine for Digital Eyewear. https://library.vuforia.com/content/vuforia-library/en/articles/Training/Vuforia-for-Digital-Eyewear.html. letzer Zugriff: 24.03.2018.

[14] Wikitude. https://www.wikitude.com, . letzer Zugriff: 24.03.2019.

[15] Different versions of the wikitude SDK. https://www.wikitude.com/documentation/, . letzer Zugriff: 16.04.2019.

[16] Wikitude target images und object recognition samples. https://www.wikitude.com/external/doc/documentation/latest/epson/targetimages.html, . letzer Zugriff: 09.04.2019.

[17] R. Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[18] G. Bradski and A. Kaehler. *Learining OpenCV: Computer vision with the OpenCV library*. 2008.

[19] J. Carmigniani, B. Furht, and M. A. und P. Ceravolo und E. Damiani und M. Ivkovic. Augmented reality technologies, systems and applications. *Multimedia tools and applications*, 51(1):341–377, 2011.

[20] D. Crandall and J. Juo. Robust Color Object Detection using Spatial-Ccolor Joint Probability Functions. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, 2004.

[21] V. Hayward, O. Astley, and M. C.-H. und D. Grant und G. Robles-De-La-Torre. Haptic interfaces and devices. *Sensor Review*, 24(1):16–29, 2004.

[22] W. Kasprzak, W. Szynkiewicz, and L. Czajka. Rubik's cube reconstruction from single view for Service robots. *Machine Graphics & Vision International Journal*, 15(3), 2006.

[23] K. Kiyokawa, Y. Kurata, and H. Ohno. An optical see-through display for mutual occlusion of real and virtual environments. In *Proceedings IEEE and ACM International Symposium on Augmented Reality (ISAR 2000)*, pages 60–67. IEEE, 2000.

[24] R. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial intelligence*, 27(1):97–109, 1985.

[25] R. Korf. Finding optimal solutions to Rubik's cube using pattern databases. In *AAAI/IAAI*, pages 700–705, 1997.

[26] P. Milgram and F. Kishino. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329, 1994.

[27] M. R.Luo, G. Cui, and B. Rigg. The development of the CIE 2000 colour-difference formula: CIEDE2000. *Color Research & Application: Endorsed by Inter-Society Color Council, The Colour Group (Great Britain), Canadian Society for Color, Color Science Association of Japan, Dutch Society for the*

Study of Color, The Swedish Colour Centre Foundation, Colour Society of Australia, Centre Français de la Couleur, 26(5):340–350, 2001.

[28] T. Rokicki, H. Kociemba, M. Davidson, and J. Dethridge. The diameter of the rubik's cube group is twenty. *SIAM Review*, 56(4):645–670, 2014.

[29] M. Thistlethwaite. The 45-52 move strategy. *London CL VIII*, 1981.

[30] M. Tsoy. kociemba C-Impl. `https://github.com/muodov/kociemba.git`. letzer Zugriff: 04.03.2018.

[31] D. van Krevelen and R. Poelman. Augmented reality: Technologies, applications and limitations. *Vrije Univ. Amsterdam, Dep. Comput. Sci*, 2007.

A Drehanweisungen

Bild:	Textuelle Beschreibung:
	Ausgangszustand
	D-Seite um 90° im Uhrzeigersinn drehen
	D-Seite um 90° gegen den Uhrzeigersinn drehen
	F-Seite um 90° im Uhrzeigersinn drehen
	F-Seite um 90° gegen den Uhrzeigersinn drehen
	L-Seite um 90° im Uhrzeigersinn drehen
	L-Seite um 90° gegen den Uhrzeigersinn drehen
	R-Seite um 90° im Uhrzeigersinn drehen

	R-Seite um 90° gegen den Uhrzeigersinn drehen
	U-Seite um 90° im Uhrzeigersinn drehen
	U-Seite um 90° gegen den Uhrzeigersinn drehen
	Den gesamten Würfel entlang der X-Achse um 90° im Uhrzeigersinn drehen
	Den gesamten Würfel entlang der X-Achse um 90° gegen den Uhrzeigersinn drehen
	Den gesamten Würfel entlang der Y-Achse um 90° im Uhrzeigersinn drehen
	Den gesamten Würfel entlang der y-Achse um 90° gegen den Uhrzeigersinn drehen