



Projektdokumentation

MArC
Mixed Reality Architecture Composer

von

Laura Anger (Matrikelnr. 11086356)
Vera Brockmeyer (Matrikelnr. 11077082)
Paul Berning (Matrikelnr. 11068249)
Lukas Kolhagen (Matrikelnr. 11084355)

Durchgeführt im
Master Medientechnologie
im SS 2016 und WS 2016/17

Betreuer:

Prof. Dr. Stefan Michael Grünvogel
Institut für Medien- und Phototechnik

Inhaltsverzeichnis

1 Einleitung	5
1.1 Motivation	5
1.2 Anwendungskontext	5
1.3 Projektziel	5
2 Grundlagen	6
2.1 Architektur von Virtual-Reality-Anwendungen	6
2.2 Haptische Interaktionsmethoden in VR Umgebungen	7
2.3 Handtracking Interaktionsmethoden	7
2.4 Menüführung in VR Umgebungen	7
2.5 Netzwerkverbindung anhand ISO/OSI-7-Schichtenmodell	7
2.6 Bedienkonzepte in VR-Umgebungen	8
2.7 Marker Tracking	8
3 Materialien	11
3.1 Hardware	11
3.1.1 Computer zur Ausführung der Unity-Simulation	11
3.1.2 Computer zur Ausführung der Tracking-Anwendung	11
3.1.3 HTC Vive	12
3.1.4 IDS uEye 164LE-C	13
3.1.5 Leap Motion	13
3.1.6 Leap Motion SDK	14
3.1.7 ArUco Marker	14
3.1.8 Würfel Marker	15
3.2 Obsolete Hardware	16
3.2.1 Ovrvision Pro	17
3.2.2 Webcam	17
3.3 Software	18
3.3.1 Unity	18
3.3.2 Visual Studio 2015	18
3.3.3 OpenCV	19
3.3.4 Orion Beta Software	19

3.3.5	Steam VR	19
3.3.6	Windows Sockets (Winsock)	19
4	System	20
4.1	Aufbau	20
4.1.1	Tracking Aufbau	20
4.2	Systemvoraussetzungen	21
4.3	Netzwerk	21
4.4	Starten des Systems	22
4.5	Menüführung	22
4.5.1	Menüs	23
4.5.2	Ablauf der Menüführung	25
4.6	Kontex Menü	27
4.7	Marker Handles	27
4.7.1	Menüführung	28
4.7.2	Architektur Berechnung	28
4.8	Table Menü	28
4.8.1	Szenen Management	29
4.8.2	Speichern von Szenen	29
4.8.3	Laden von Szenen	30
4.8.4	Match Modus	31
5	Tracking Programm	32
5.1	uEye Ansteuerung	32
5.2	Kalibrierung	33
5.2.1	Kamerakalibrierung	33
5.2.2	Kalibrierung des Arbeitsbereichs	36
5.2.3	Kalibrierungfehler	37
5.2.4	Mögliche Fehlerquellen	37
5.3	Marker Detektion	37
5.4	Green Keying	37
5.5	ArUco Marker Tracking	37
5.6	Marker Objekt	37
5.7	Zuordnung der Identitäten	37

6 Ausblick	38
6.1 Trackingmethoden	38
6.2 AR Erweiterung mit der Webcam	38
6.3 Daten als Excel Tabelle	38
6.4 Validierung	38
7 Projektmanagement	38
7.1 Zeitplanung	38
7.2 Gant Chart	38
7.3 Planungsmethoden	38
7.4 Reflexion	38
7.4.1 Kommuniktion im Team und nach Außen	38
7.4.2 Probleme	39
8 Zusammenfassung	39
9 Steckbrief	39
10 Anhang	39
10.1 <i>MArC</i> ReadMe-Datei	39

1 Einleitung

Paul

Virtuelle und erweiterte Realität ist bereits seit einiger Zeit in aller Munde. Mit dem Erscheinen von Oculus Rift, HTC Vive und anderen Virtual-Reality-Headsets rückt eine neue Art der Immersion beim Genuss von Videospielen in greifbare Nähe.

Wenn es jedoch um die Nutzung dieser Technologien zur Effizienzsteigerung in professionellen Umgebungen geht, so sind verfügbare Anwendungen bisher nur selten anzutreffen. Die Entwicklung von MArC, einem Mixed-Reality-System für die architektonische Planung bei Siedlungsbauten, soll dies ändern.

1.1 Motivation

Paul

1.2 Anwendungskontext

Paul

Die frühe Konzeptionierung zur Erschließung von Wohngebieten findet heute in Architekturbüros meist noch so statt wie vor dem Einzug der weit verbreiteten, digitalen Technik in unsere Leben. Dazu werden simple Modelle aus leicht zu verarbeitenden Materialien – wie etwa Styropor – erstellt und als Platzhalter für die zu planenden Gebäude bei dem Entwurf verwendet.

Diese Herangehensweise macht Änderungen an den Gebäuden aufwendig und führt zu dem Umstand, dass ein bestimmter Zustand der Planung nur umständlich wiederhergestellt werden kann – zum Beispiel durch Fotografieren und späteren manuellen Wiederaufbau.

1.3 Projektziel

Paul

An diesem Punkt setzt MArC an, der „Mixed Reality Architecture Composer“.

Vervollständigen

2 Grundlagen

2.1 Architektur von Virtual-Reality-Anwendungen

Lukas

Bei der Entwicklung von Virtual Reality (VR) Anwendungen gibt es zwei entscheidende Unterschiede im Vergleich zu normalen Software-Anwendungen [7]:

- die virtuelle Umgebung und das damit verbundene Interface sollten auf die vorliegende Aufgabe zugeschnitten werden und
- spezielle Anforderungen an die Performanz der Anwendung müssen erfüllt sein, damit die virtuelle Realität erfolgreich präsentiert werden kann.

Als grundsätzlich unterschiedliche Architekturen bei der Entwicklung von Virtual Reality (VR) kann nach Hardware-Plattformen unterschieden werden:

Desktop-VR-Applikationen: hierbei handelt es sich um die leistungsstärksten VR-Applikationen, die potente Computer-Hardware häufig mit Head-Mounted-Displays wie etwa Oculus Rift oder HTC Vive verwenden.

Mobile-VR-Applikationen: mobile Anwendungen vereinen häufig (jedoch nicht immer) alle notwendige Hardware in einem Gerät, wie etwa einem Smartphone oder Tablet-Computer.

Beispiele für mobile VR-Anwendungen sind Samsung GearVR [42] und Google Cardboard [18]. Im Falle von Samsung GearVR und Google Cardboard wird zusätzlich eine Haltevorrichtung für das verwendete Gerät eingesetzt, die bei GearVR auch als Controller fungiert und durch ein integriertes Linsensystem das Sichtfeld des Benutzers erhöht.

Web-VR-Applikationen: Anwendungen für das Web wie etwa WebVR [31] erlauben den Zugriff auf Virtual-Reality-Geräte wie Head-Mounted-Displays durch einen Browser. Auf diese Weise können Web-Inhalte mit VR-Hardware konsumiert werden.

Die Entscheidung, für das vorliegende Projekt auf eine Desktop-VR-Anwendung zu setzen, wurde getroffen, weil mobile und Web-VR-Applikationen die folgenden, vom Projektteam als unverzichtbar eingestuften, Voraussetzungen nicht erfüllen konnten.

- Es sollte möglich sein, die in 2.3 beschriebenen Handtracking Interaktionsmethoden für die Verwendung von haptischen Würfel-Markern zu verwenden.
- Außerdem sollte die verwendete Architektur ein zuverlässiges Echtzeit-Tracking mit geringer Latenz von mindestens einem Dutzend Markern erlauben, wie es in 2.7 beschrieben wird.
- Und schließlich sollte *MArC* die Basis bereitstellen, um später auch mit mehreren Benutzern verwendet werden zu können.

Nr.	Schicht	Beispiel
7	Anwendung	HTTP, SMTP, FTP, DNS
6	Darstellung	HTTP, SMTP, FTP, NNTP, NetBIOS
5	Sitzung	HTTP, SMTP, FTP, NNTP, NetBIOS, TFTP
4	Transport	TCP, UDP, SPX, NetBEUI
3	Vermittlung	IP IPX
2	Sicherung	Ethernet, ATM, FDDI, TR
1	Bitübertragung	Manchester, 10B5T, Trellis

Tabelle 1: ISO-/OSI-7-Schichtenmodell

2.2 Haptische Interaktionsmethoden in VR Umgebungen

Laura

2.3 Handtracking Interaktionsmethoden

Paul

2.4 Menüführung in VR Umgebungen

Lukas

2.5 Netzwerkverbindung anhand ISO/OSI-7-Schichtenmodell

Laura

Um die Kommunikation zwischen unterschiedlichsten technischen Systemen zu ermöglichen und zu vereinheitlichen dient das *ISO/OSI-7-Schichtenmodell* [23], welches in Tabelle 1 schematisch dargestellt ist. Um die Weiterentwicklung von Kommunikationsmodellen möglichst barrierefrei zu gestalten, sind in dem Modell sieben aufeinanderfolgende Schichten definiert worden, die für einen klar eingegrenzten Teilbereich der Kommunikation zuständig sind. Die Netzprotokolle, die in einer Schicht zum Einsatz kommen, müssen einheitliche Schnittstellen aufweisen, um einen reibungslosen Austausch zu gewährleisten. Entsprechende Beispiele sind der rechten Spalte von Tabelle 1 zu entnehmen.

Während die Schichten 1-4 als transportorientierte Schichten einzustufen sind, können die verbleibenden Schichten 5-7 als anwendungsorientiert angesehen werden. Da

der Austausch von Daten für die Umsetzung von *MaRC* im Vordergrund steht, wird im Folgenden besonders auf die vierte Schicht eingegangen. Dabei werden die beiden Übertragungsprotokolle *Transmission Control Protocol* (TCP) und *User Datagram Protocol* (UDP) vorgestellt. Auf die Aufführung weiterer Übertragungsprotokolle wird bewusst verzichtet, da wie in Kapitel 3.3.6 beschrieben ist, die *Winsock API* die Übertragung über diese beiden Protokolle ermöglicht. Die Transportschicht stellt eine logische Ende-zu-Ende-Verbindungen dar und dient als Bindeglied zwischen den transportorientierten und anwendungsorientierten Schichten [23].

TCP: Dieses Transportprotokoll ist verbindungsorientiert und paketvermittelt. Genaue Details können in dem Standard RFC 793 [3] von 1981 in Erfahrung gebracht werden.

UDP: Dieses Transportprotokoll wurde aus der Notwendigkeit heraus entwickelt, für die Übertragung von Sprache auf ein, im Gegensatz zur TCP, einfacheres Protokoll zurückgreifen zu können. Genaue Details können in dem Standard RFC 768 [39] von 1981 in Erfahrung gebracht werden.

Ein wichtiger Unterschied zwischen den beiden Transportprotokollen ist, dass die Übertragung per TCP im Vergleich zu UDP sicherstellt, dass gesendete Daten korrekt übertragen und empfangen werden. **QUELLE??** Dies geschieht dadurch, dass in einem TCP-Socket-Netzwerk Fehlererkennungs- und Korrekturmechanismen enthalten sind, die fehlerhafte Übertragungen der darunterliegenden Internet Protocol (IP) Schicht ausgleichen, also entweder korrigieren oder dafür sorgen, dass fehlerhafte Daten erneut übertragen werden.

2.6 Bedienkonzepte in VR-Umgebungen

Lukas

Die Bedienung eines Systems – vor allem Dateneingabe und Objektmanipulation – in einer Virtual Reality (VR) Umgebung unterscheidet sich stark von der konventionellen Bedienung mit Maus und Tastatur [8]. Das Hauptziel einer VR-Umgebung ist Immersion, also das „Eintauchen“ des Benutzers in die virtuelle Welt. Ein Teil zur Erreichung dieses Ziels sind Methoden, mit denen der Benutzer mit der virtuellen Welt – möglichst intuitiv – interagieren kann.

2.7 Marker Tracking

Vera

In einer VR oder AR Umgebung ist zur interaktiven Positionierung eines 3D Objektes die präzise Bestimmung der Orientierung und Position im 3D-Zielraum notwendig. Zur Lösung dieses Problems muss zunächst ein physisches Objekt in der

realen Welt erkannt, zugeordnet und verfolgt werden. Demzufolge ist es notwendig dieses physische Objekt mit einer Videokamera auf zu nehmen. In den resultierenden Bildsequenzen werden die physischen Objekte anhand von festgelegten Merkmalen erkannt. Diese Merkmale können sowohl natürlicher Art sein oder als künstlich erstellten Codes definiert sein. Ein Objekttracking mit natürlichen Merkmalen wird in der Literatur auch als *Markerless Tracking* bezeichnet, während die Verwendung von Codes, beziehungsweise Bildmarken, als *Markerbased Tracking* bekannt ist.

Natürliche Merkmale zur Identifizierung der Objekte sind Textureigenschaften, Kanteninformationen oder sogenannte Keypoints. Eine der robustesten und simpelsten Methoden ist die Verwendung von Keypoints [4][49][9][27], die sowohl in der Ausgangs- als auch in der Kamerawelt bekannt sind. Diese werden mit Hilfe einer Homographie zur Übereinstimmung gebracht. Daraus resultiert die notwendige Transformation zwischen den Welten (siehe Kapitel 5.2). Ein bedeutender Nachteil der Verwendung von Keypoints oder Texturbasierten Verfahren ist, dass sie nur für Objekte mit einem hohen Grad an Texturmerkmalen und großen Gradientenbeträgen geeignet sind. Aus diesem Grund wurden auch Verfahren [20][12][38] entwickelt die sich speziell für texturarme Objekte eignen, wie etwa die in *MArC* verwendeten grünen Rechtecken der Würfel Marker (siehe Kapitel 3.1.8). Andere Autoren verwenden sehr rechenintensive Kantenerkennungsalgorithmen, wie den *Moving Edges Algorithm* [54]. Eine weitere Weiterentwicklung der Kantenbasierten Methoden ist das Modelbased Tracking bei dem die detektierten Kanten eines möglichen Kandidaten mit den 3D-Kantenmodellen des zu verfolgenden Objektes abgeglichen wird [47][53][51][5].

Im Gegensatz zu dem Markerbased Tracking benötigt diese Art des Trackings keine Veränderung der realen Welt und die Parameter, welche den Tracking-Algorithmus beeinflussen können nicht ohne weiteres kontrolliert werden [4]. Dennoch ist die markerbasierte Detektierung von Objekten sehr rechenaufwändig und eine eindeutige Zuordnung beziehungsweise Identifikation von ähnlichen oder gleichförmigen Objekten, wie den Würfel Marker ist sehr aufwändig und nahezu unmöglich. Aus diesem Grund ist es für ein System wie *MArC* zum derzeitigem Zeitpunkt sinnvoller die Würfel Marker mit codebasierten Mustern zu erweitern, die auch nach längerer Verdeckung eine eindeutige Zuordnung von Würfel Marker ermöglichen. In Abbildung 1 sind vielfältige Beispiele von binären Codes zu sehen, die zum Marker Tracking verwendet werden. Für *MArC* sind vor allem rechteckige binäre Muster vorteilhaft, da aus dem äußeren vier Ecken auch die Orientierung des Würfel Markers im Raum abgeleitet werden kann. Während der Inhalt des Muster zur eindeutigen Identifizierung des Würfels beiträgt.

Bekannte markerbasierte Verfahren mit rechteckigen binären Mustern sind das bekannte QR Verfahren, *ARToolKit*[24], *ARToolKit Plus*[50], *ARTag* [14], *BinARYID*[15] und *ArUco*[16]. Wie in Abbildung 1 zu sehen ist, sind bis auf *BinARYID* und *ArUco* alle Muster sehr detailreich und komplex. Diese Eigenschaft ist auf die höhere Bitgröße der Codes zurückzuführen macht die Erkennung in Aufnahmen aus größerem Abstand und gegebenenfalls mit Bewegungsunschärfe ungleich schwerer und es kommt zu häufiger zu Fehlinterpretationen und Ausfällen. Gerade in einem System wie *MArC* tritt Bewegungsunschärfe sehr häufig aus, wenn die Würfel Marker

verschoben werden. Auch der verhältnismäßig große Abstand der Kamera zu den Würfel Markern lässt die Marker Muster im Kamerabild relativ klein werden und somit wirkt sich die Bewegungsunschärfe noch deutlicher auf die feinen Codemuster aus. Darum eignen sich hier vor allem Muster mit geringer Bitanzahl und einfacheren groben Mustern, wie zum Beispiel *ArUco* Marker mit maximal 4 bit Codierung. Diese Marker Bibliothek ist ein *OpenCV* Modul (siehe Kapitel 3.3.3), welche alle notwendigen Funktionalitäten und Ressourcen für das Tracking und die Orientierungsbestimmung enthält. Ein weiterer großer Vorteil dieser Bibliothek ist, dass die Bitgröße und die maximale Anzahl der Identitäten explizit ausgewählt werden kann.

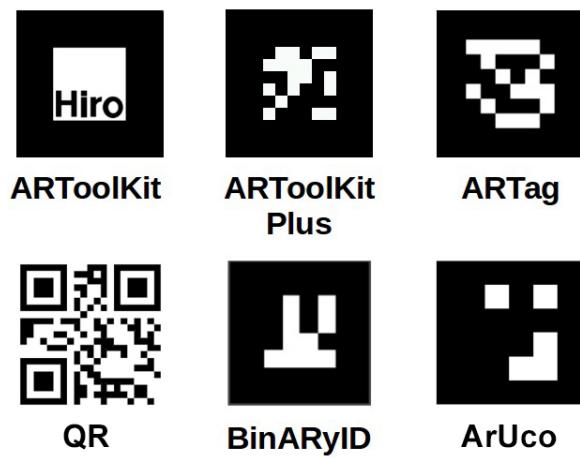


Abbildung 1: Diverse Binäre Muster die als Code für Markerbasiertes Tracking verwendet werden. Quelle: [16]

NAME DES COMPUTERS	Beschreibung
Prozessor	XXX
Arbeitsspeicher	XXX
Grafikkarte	XXX
Betriebssystem	XXX
Schnittstellen	XXX

Tabelle 2: Auszug aus dem technischen Datenblatt des XXXX.

3 Materialien

In den nachfolgenden Abschnitten, werden Werkzeuge und Hilfsmittel beschrieben, die für die Fertigstellung des Projekts vonnöten waren.

Des weiteren werden auch Komponenten vorgestellt, die während der Projektlaufzeit erstellt wurden, wie etwa die Würfel-Marker (s. Abb. 5).

3.1 Hardware

Zur Ausführung der MArC-Software sind diverse Hardware-Komponenten Voraussetzung. Diese Komponenten werden nachfolgend beschrieben und deren Kontext im System näher erläutert.

3.1.1 Computer zur Ausführung der Unity-Simulation

Lukas

Die Anwendung, welche aus Unity [43] heraus erstellt wurde, benötigt einen Host-Computer, welcher sowohl mit dem HTC Vive Head-Mounted Display kompatibel, als auch leistungsstark genug sein muss, um das Rendering der Simulation mit ausreichend hoher Bildrate ausführen zu können.

Für das vorliegende Projekt wurde seitens der Technischen Hochschule Köln ein Computer mit den Eigenschaften aus Tabelle 2 zur Verfügung gestellt:

Beim nächsten Mal in der TH nachschauen...

Die Hard- und Software-Voraussetzungen für die Ausführung der Unity-Anwendung in Verbindung mit der HTC Vive, welche in Tabelle 3 aufgelistet sind, werden von dem verwendeten Computer übertroffen.

3.1.2 Computer zur Ausführung der Tracking-Anwendung

Vera

Auf Grund der begrenzten Bandbreite einer USB-Karte ist es zwingend notwendig

HTC Vive	Systemvoraussetzungen
Prozessor	mindestens Intel Core i5-4590 oder AMD FX 8350
Grafikkarte	mindestens NVIDIA GeForce™ GTX 1060 oder AMD Radeon™ RX 480
Arbeitsspeicher	mindestens 4 GB
Videoausgang	1x HDMI 1.4-Anschluss oder DisplayPort 1.2
USB	1x USB 2.0-Anschluss
Betriebssystem	Windows 7 SP1, Windows 8.1 oder Windows 10

Tabelle 3: HTC Vive Systemvoraussetzungen. [22]

Acer E5-571G-795A	Beschreibung
Prozessor	Intel Core i7-5500U CPU @ 2 × 2.4GHz
Arbeitsspeicher	8.0 GB
Grafikkarte	NVIDIA GeForce 840M
Betriebssystem	Windows 10 Home, 64 bit
Schnittstellen	2× USB 2.0, 1× USB 1.0, Netzwerkport ????

Tabelle 4: Auszug aus dem technischen Datenblatt des Acer E5-571G-795A.

einen weiteren Rechner an das Gesamtsystem zu koppeln, welcher ausschließlich für die Ansteuerung der uEye-Kamera und die Berechnungen des Tracking-Algorithmus zuständig ist. An den Computer zur Ausführung der VR Umgebung sind gezwungenenmaßen viele externe USB Komponenten, wie zum Beispiel die *Leap Motion* und die *HTC Vive*, angeschlossen. Dies führt zu einer hohen Auslastung der Bandbreite der USB-Karte und auf Grund dessen ist es nicht mehr möglich die uEye-Kamera mit der notwendigen maximalen Framerate zu betreiben. Somit wird für ein flüssiges und real-time fähiges Tracking der Acer E5-571G-795A mit den Eigenschaften aus Tabelle 4 verwendet.

3.1.3 HTC Vive

Laura

Bei der *HTC Vive* handelt es sich um ein Head-Mounted Display, welches von *HTC* in Kooperation mit *Valve* [48] produziert wird. Vorgestellt wurde dieses am 1. März 2015 im Vorfeld des *Mobile World Congress* [30].

Die Auflösung des Displays beträgt insgesamt 2160×1200 Pixel, was 2160×1200 Pixeln pro Auge entspricht. Die Brille bietet ein Sichtfeld von bis zu 110° bei einer Bildwiederholrate von 90 Hz [21]. Alle technischen Systemvoraussetzungen können in Tabelle 3 eingesehen werden.

Zur Positionsbestimmung im Raum wird die Lighthousetechnologie von *Valve* genutzt. Zusätzlich sind neben einem Gyrosensor auch ein Beschleunigungsmesser und ein Laser-Positionsmesser verbaut. Mittels speziellen Game-Controllern wird eine Interaktion mit virtuellen Objekten ermöglicht.

3.1.4 IDS uEye 164LE-C

Vera

Die Kamera *uEye 164LE-C* wurde vom Hersteller *IDS Imaging Development Systems* entwickelt. Sie hat eine Auflösung von 1280×1024 Pixel und ermöglicht Live-Video-Aufnahmen im RGB Farbmodus mit maximal 25 fps. Der integrierte CMOS Bildsensor wird im Rolling Shutter betrieben und ermöglicht Belichtungszeiten von $37\mu s$ bis 10s. Weiterführend kann sie universell mit allen gängigen Computern oder Systemen via USB 2.0 Schnittstelle verbunden werden [1].

Die erforderliche Ansteuerung der *uEye 164LE-C* erfolgt mit Hilfe der bereit gestellten *IDS Software Suite*. In diese Suite ist die *uEye-API* integriert, welche die Entwicklung von eigenen Programmen unter *Windows* und *Linux* mit den Programmiersprachen *C++*, *.NET*, *C#* oder *C* ermöglicht [2]. Für das Tracking der Marker in *MArC* wurde eine eigene Schnittstelle in *C++* erstellt, welche die Kamera im Live Modus initialisiert und steuert.

3.1.5 Leap Motion

Paul

Bei der Leap Motion [25] handelt sich um ein $7,6 \times 3 \times 1,3\text{ cm}$ großes Gerät, welches es mit Hilfe von Sensoren möglich macht, Hand- und Fingerbewegungen als Eingabemöglichkeit zu nutzen. Die Idee dahinter ist, eine Eingabegerät analog zu Maus zu schaffen, welches keinen direkten Kontakt bzw. keine Berührung benötigt. Hergestellt wird die Leap Motion von der Firma Leap Motion, Inc., die ihren Hauptsitz in Amerika hat. Gegründet wurde die Firma am 1. November 2010.

Wie auf Abbildung 6 erahnt werden kann, besteht das Gerät im wesentlich aus zwei integrierten weitwinkel Kameras und drei einfachen Infrarot LEDs. Die LEDs haben jeweils eine Wellenlänge von 850 nm . Der durch die beiden Kameras aufgespannte Interaktionsraum der Leap Motion ähnelt einer umgedrehten Pyramide, mit einem Flächeninhalt von knapp 243 cm^2 .

Für das Projekt wurde die Orion beta software, die in 3.3.4 näher beschrieben wird. Diese Software ermöglicht unter anderem eine Erweiterung der Reichweite der Leap Motion von 60 cm auf 80 cm . Diese Reichweite ist durch die Ausbreitung der LED Lichter räumlich begrenzt. Die Lichtintensität der LEDs ist wiederum durch den maximalen Strom, der über die USB-Verbindung fließt beschränkt.



Abbildung 2: Explosionszeichnung der Leap Motion. [26]

3.1.6 Leap Motion SDK

Paul

3.1.7 ArUco Marker

Vera

Die ArUco Bibliothek ist ein Marker Tracking Modul von *OpenCV* (siehe Kapitel 3.3.3) kann für Augmented Reality (AR) Anwendungen genutzt werden und stellt für diese Anwendungen alle notwendigen Funktionalitäten zum Orten und Verifizieren der Codes sowie der anschließenden Pose Estimation der ermittelten Positionen zur Verfügung [16]. Die Marker bestehen ähnlich wie QR-Codes aus einer zweidimensionalen Matrix, mit schwarzen oder weißen Feldern, welche die kodierten Daten binär, wie in Abbildung 3, darstellen. Weiterführend kann die Anzahl der Bits variabel gewählt werden (siehe Abbildung 4), je nachdem wie groß die gefragte Markeranzahl ist oder deren erforderliche Erkennbarkeit in sehr großen Entfernung sowie kleinen Bildern. Um diese Vielzahl an verschiedenen Größen und IDs händeln zu können wurden sogenannte Dictionarys eingeführt [17]. Diese Dictionarys bestehen aus Markern mit gleicher Bit-Anzahl und sind zusätzlich auf eine maximalen Anzahl von IDs begrenzt um ein möglichst hohe Performanz zu gewährleisten.

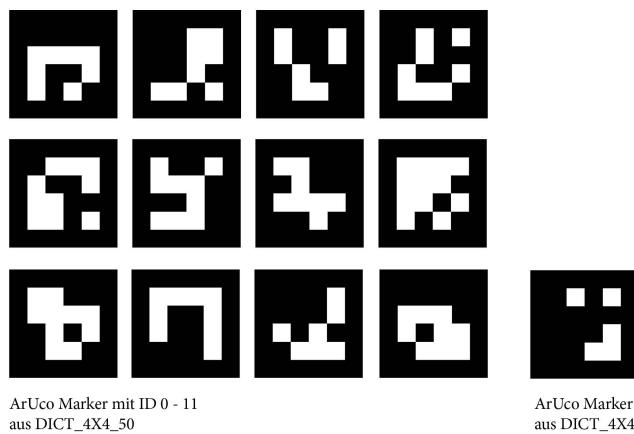


Abbildung 3: Alle genutzten 16 bit ArUco Marker des Prototypen. Links die zwölf IDs der Würfel Marker und rechts die ID, welche zur Kalibrierung benötigt wird. Die maximale Anzahl der Marker ist auf 50 begrenzt.



Abbildung 4: ArUco Marker mit unterschiedlicher Bitgröße. Von Links nach Rechts: $n = 5$, $n = 6$ und $n = 8$. Quelle: [16]

3.1.8 Würfel Marker

Vera

In vielen VR oder AR Umgebungen müssen die Nutzer eines Systems häufig nach virtuellen Objekten zur Interaktion greifen, die nicht real existieren. Demzufolge greifen die Personen ins Leere, was häufig die Immersion stört und zu Irritationen und Unsicherheit führt. Um dem Nutzer des Systems *MArC* für die Positionierung und Orientierung ein reales haptisches Feedback zu ermöglichen wurden zwölf Aluminiumwürfel mit aufgeklebten Markern designt. Diese Würfel können beliebig innerhalb eines zuvor festgelegten Bereiches auf dem realen Tisch verschoben und rotiert werden. An der aktuellen Position und Orientierung des jeweiligen Markers wird in der VR ein explizit zugeordnetes Objekt gerendert. Diese Position wird mit Hilfe des Tracking-Algorithmus (siehe Kapitel 5) aus den Bildern der uEye Kamera ermittelt. Alle zwölf Marker stimmen mit der Form und Farbe, sowie Material und Oberflächenbeschaffenheit aus Abbildung 5 überein. Sie haben eine Kantenlänge von 46 mm und sind in einem Winkel von 45° an allen Kanten gefräst. Das Aluminium ist glasperlgestrahlt um eine matte Oberfläche zu erzeugen, welche ungewollte Reflexionen und Überstrahlungen vermeidet, die unter Umständen den Tracking-Algorithmus beeinflussen können. Auf die Oberseite des Markers ist mittig ein leuchtend grünes Quadrat mit einer Kantenlänge von 40 mm aufgebracht. Diese grüne Fläche wird für

ein Green-Keying benötigt, welches die Verfolgung der Marker auch bei Bewegungsunschärfe ermöglicht. Ebenfalls mittig ist jeweils ein individueller 35 mm großer ArUco-Marker plan befestigt. Alle verwendeten ArUco-Marker haben einen Rand von einem Bit hat und wurden jeweils aus dem Marker Dictionary DICT_4X4_50 des Arcuo Moduls [34] generiert.

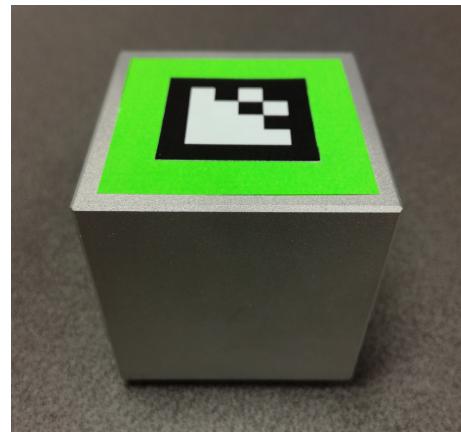


Abbildung 5: Würfel Marker mit grüner Fläche und einem ArUco Marker, die mittig auf den Aluminiumwürfel aufgebracht sind.

3.2 Obsolete Hardware

Lukas

Im Laufe eines Projekts nach Art von MArC ist es kaum vermeidbar, dass die gesetzten Projektziele reevaluiert werden müssen. Die Gründe hierfür können vielfältig sein. Beispielsweise könnte die Fertigstellung eines bestimmten Teils des Projekts deutlich länger gedauert haben als geplant, oder es könnte sich herausgestellt haben, dass bestimmte Komponenten zueinander nicht kompatibel sind.

Im vorliegenden Projekt trat eine Kombination der beiden oben genannten Gründe auf. Das Betreiben der Ovrvision Pro am US-Bus verschiedener während der Entwicklung verwendeter Rechner stellte sich als unberechenbar und damit leider unbenutzbar heraus. Die Kamera sorgte während der Ausführung von Unity dafür, dass mit allen anderen Geräten, die ebenfalls per USB angeschlossen waren, unterschiedlichste Probleme auftraten. Als die Situation nach dem Verbinden der Kamera in der teilweisen Zerstörung eines Mainboards gipfelte, wurde die Entscheidung getroffen, die Ovrvision nicht länger als Gerät in der Entwicklung zu verwenden.

Stattdessen war zu diesem Zeitpunkt die Idee, eine gewöhnliche Webcam zu verwenden, um die Realisierung von Augmented Reality dennoch zu ermöglichen, wenn auch ohne den Stereo-3D-Effekt, welchen die Ovrvision nativ bereitgestellt hätte.

Im weiteren Verlauf des Projekts führte eine lange Zeit ungeklärte, starke Abweichung der Positionen der realen und virtuellen Marker zur Neuordnung der Projekt-prioritäten. Dies hatte zur Folge, dass letztendlich auch die Webcam als Plattform für die Umsetzung der AR-Fähigkeiten von MArC aufgegeben wurde.

Örtliche Auflösung pro Auge	Zeitliche Auflösung	Bildwinkel	
		Horizontal	Vertikal
2560 × 1920 px	15 fps	115°	105°
1920 × 1080 px	30 fps	87°	60°
1280 × 960 px	45 fps	115°	105°
1280 × 800 px	60 fps	115°	90°
960 × 950 px	60 fps	100°	98°
640 × 480 px	90 fps	115°	105°
320 × 240 px	120 fps	115°	105°

Tabelle 5: Bildmodi der Ovrvision Pro Stereokamera. [37]

Nachfolgend werden die Eigenschaften und technischen Daten beider Geräte kurz beschrieben.

3.2.1 Ovrvision Pro

Lukas

Die Ovrvision Pro ist eine kompakte Stereokamera, welche über USB 3.0 mit dem Rechner verbunden werden kann [35]. Sie ist kompatibel mit Programmen wie Unity, welches für das Projekt benutzt wurde und in 3.3.1 beschrieben wird.

Die Bildmodi der Kamera sind in Tabelle 5 aufgeführt.



Abbildung 6: Explosionszeichnung der Leap Motion.[26]

3.2.2 Webcam

Vera

Die *Creative Senz3D* ist eine RGB Kamera mit einer zusätzlichen Infrarot-Tiefenkamera. Das generierte RGB-Bild hat eine Auflösung von 1280×720 Pixel und das Tiefenbild

von 320×240 Pixel bei einer Reichweite von 15 – 99 cm sowie einem Sichtfeld von 74°. Die Kamera wird über eine USB 2.0 Schnittstelle mit einem Computer verbunden und nimmt Videos mit einer Framerate von bis zu 30 fps auf [10]. Weiter ist es möglich die Kamera direkt aus Anwendungen per *Intel Perceptual Computing SDK* anzusteuern.

3.3 Software

Vera

Zur Entwicklung der *MArC*-Software sind diverse Software-Komponenten und Bibliotheken notwendig. Die Funktionalitäten und Verwendung dieser Komponenten werden in diesem Kapitel kurz erläutert.

3.3.1 Unity

Lukas

Unity ist eine sogenannte Spiel-Engine, also eine Entwicklungs- und Laufzeitumgebung, die speziell auf die Entwicklung von 3D-Spielen ausgelegt ist. Die Software wurde am 6. Juni 2005 veröffentlicht [19] und wird von Unity Technologies [43] entwickelt und vertrieben. In der Spieleentwicklung ist Unity weit verbreitet, so werden beispielsweise 34 % der kostenfreien Top-1000-Spiele im mobilen Sektor mit Unity entwickelt [45].

Unity bietet eine sehr breite Plattformunterstützung [44] und erlaubt ebenso die Entwicklung für Head-Mounted-Displays, wie etwa die Oculus Rift [32][46] oder auch die in diesem Projekt verwendete HTC Vive [46].

Die zu Beginn des Projekts verwendete Stereo-Kamera Ovrvision Pro stellt ein Software-Development-Kit (SDK) für Unity (Version 5) zur Verfügung [36]. Da das endgültige Resultat des Projekts die Verwendung der Ovrvision Pro nicht mehr vorsieht, wie in 3.2 beschrieben, wird auf eine weitere Beschreibung dieses SDKs verzichtet.

3.3.2 Visual Studio 2015

Vera

Micosoft Visual Studio 2015 ist eine verbreitete integrierte Entwicklungsumgebung (IDE), welche unter anderem die Programmiersprachen Visual Basic, Visual C#, und Visual C++ unterstützt. Mit Hilfe dieser IDE kann ein Entwickler Win32/Win64 Anwendungen sowie weitere WebApps und Webservices [29] programmieren sowie anschließend compilieren. Für *MArC* wurde mit der Version 14.0.25123.00 Update2 gearbeitet.

3.3.3 OpenCV

Vera

Open Source Computer Vision (OpenCV) ist eine Open Source Bibliothek für Bild- und Videoverarbeitung in der Programmiersprache *C++*. Vorgestellt wurde sie vor über zehn Jahren von *Intel* und wird seitdem stetig von verschiedenen Programmierern weiterentwickelt. Diese Bibliothek stellt die gängigsten Algorithmen sowie aktuelle Entwicklungen der Bildverarbeitung zur Verfügung [11].

Für dieses System ist vor allem das Modul `calib3d` [33] und das extra Modul `aruco` [34] verwendet. Das erste Modul `calib3d` bietet alle notwendigen Funktionen zur Erstellung, Verwendung und Weiterverarbeitung von intrinsischen und extrinsischen Kamerakalibrierungen an (siehe Kapitel 5.2). Während das Zweite alle benötigten Ressourcen und Funktionalitäten zum Tracken von *ArUco* Markern zur Verfügung stellt (siehe Kapitel 3.1.7).

3.3.4 Orion Beta Software

Paul

3.3.5 Steam VR

Paul

3.3.6 Windows Sockets (Winsock)

todo[inline, color=yellow] Lukas Windows Sockets (abgekürzt Winsock) ist eine API für den Zugriff auf Netzwerkkomponenten in Microsoft Windows Betriebssystemen [40]. Winsock wird nativ in Microsoft Windows bereitgestellt.

Für die unkomplizierte Übertragung zwischen zwei Anwendungen in einem lokalen Netzwerk bieten sich sowohl das TCP, als auch das UDP an. Das Erstellen von Netzwerk-Sockets für die Übertragung per TCP und UDP wird von Winsock ermöglicht.

4 System

Lukas

Die Benutzung von *MarC* ist in dem Programm mitgelieferten ReadMe-Datei (vgl. 10.1) beschrieben. Darin wird erklärt, welche Hard- und Softwarekomponenten erforderlich sind und wie das System gestartet und kalibriert wird. Auf diese Aspekte wird in den folgenden Abschnitten näher eingegangen.

Des weiteren enthält die ReadMe-Datei eine Übersicht über die enthaltenen Quellcode-Dateien.

4.1 Aufbau

Lukas und Vera

Der Aufbau von *MarC* kann in zwei Teile aufgeteilt werden. Zum Einen den Part der für das Tracking der Würfel Marker verantwortlich ist und zum Anderen den zweiten Teil, welcher die VR Umgebung erzeugt und die notwendige Peripherie für die Interaktionen stellt.

4.1.1 Tracking Aufbau

Vera

Wie in Abbildung 7 dargestellt wird senkrecht über einem beliebigen Tisch eine Kamera installiert, die Würfel Marker aus der Vogelperspektive filmt. Der Abstand zum Tisch sollte so gewählt werden, dass die Aufnahmen noch scharf sind und sich die Nutzer nicht den Kopf daran stoßen können. Hier ist besondere Vorsicht geboten, da die Nutzer durch die *HTC Vive* nicht die reale Umgebung wahrnehmen können. Für den Prototypen wurde eine *IDS uEye 164LE-C* (siehe Kapitel 3.1.4) verwendet und über eine USB 2.0 Schnittstelle an den Computer mit dem Tracking Algorithmus verbunden. Diese Kamera wird mit Hilfe der uEye-API vom Tracking Algorithmus im Live-Bild-Modus initialisiert und gesteuert. In diesen Live Bildern werden die Würfel Marker erkannt und verfolgt. Für jeden erkannten Würfel Marker werden alle relevanten Informationen über die TCP Netzwerkverbindung (siehe Kapitel 2.5) an den Computer zur Ausführung der *Unity*-Simulation (siehe Kapitel 3.1.1) übertragen. Damit der Algorithmus einen Würfel Marker erkennt muss er in dem vorab definierten Spielfeld bewegt werden. Diese Festlegung findet während der Kalibrierung des Arbeitsbereiches statt. Für diese Kalibrierung ist es notwendig den einzelnen *ArUco* Marker aus Abbildung 3 mit der ID 49 exakt wie in Abbildung 8 auf den Controller der *HTC Vice* zu montieren, da nur so gewährleistet werden kann, dass die erkannte *ArUco* Marker Position in der Kamerawelt mit den Controller Positionen in der *Unity* Welt korrespondieren.

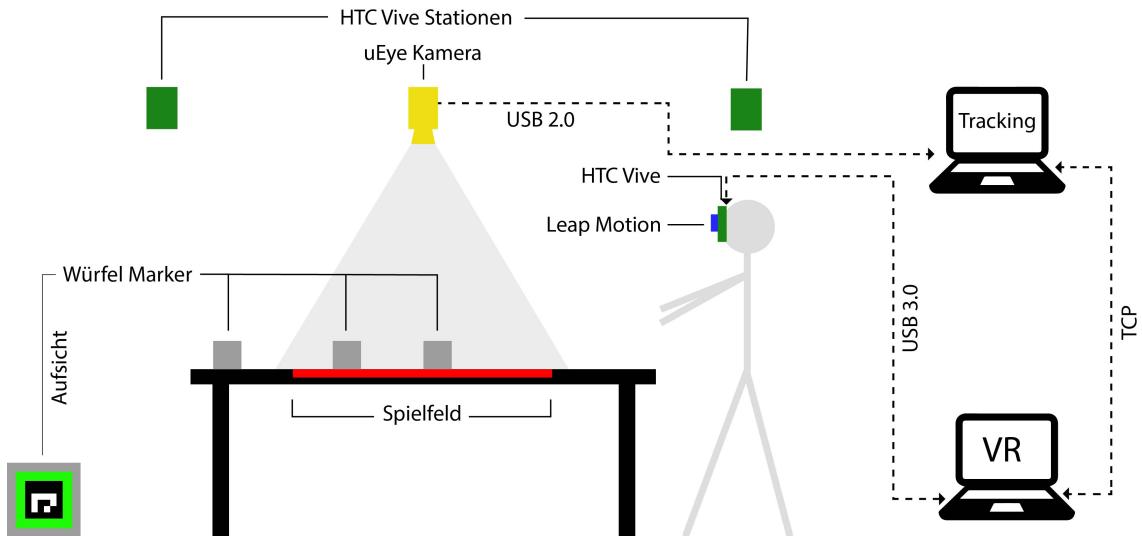


Abbildung 7: Aufbau des *MArC* System.

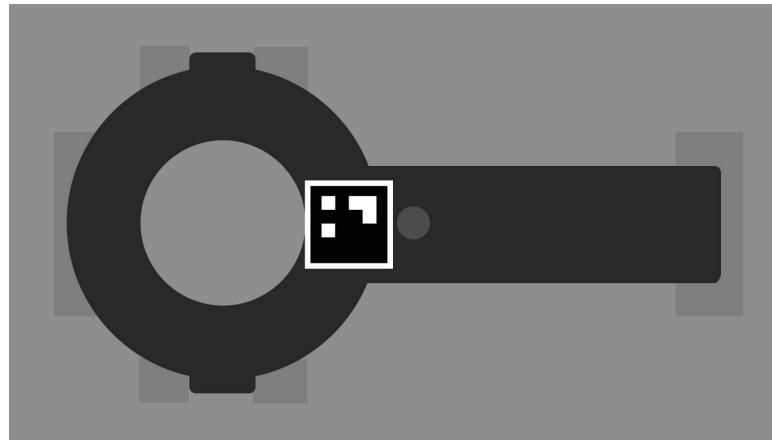


Abbildung 8: Kalibrierungscontroller des *MArC* System mit ArUco Marker.

4.2 Systemvoraussetzungen

Lukas

Die Systemvoraussetzungen für *MArC* sind in der ReadMe-Datei beschrieben, welche der im Projekt erstellten Software mitgeliefert ist. Die gesamte ReadMe-Datei ist in Abschnitt 10.1 zu finden, außerdem enthält Abbildung 9 einen Ausschnitt der ReadMe-Datei, welcher die Voraussetzungen beschreibt, die vor dem Starten des Systems erfüllt sein müssen.

4.3 Netzwerk

Lukas und Laura

Starting the System

The following software is needed:

- IDS uEye Driver and SDK (<https://de.ids-imaging.com/download-ueye-win32.html>)
- Steam VR (<http://store.steampowered.com/about/>)
- Leap Driver (<https://developer.leapmotion.com/windows-vr>)
- The OpenCV Framework (<http://opencv.org/downloads.html>)

In order to start the system, make sure the following requirements are met:

- 2 Computers are available:
 - one that runs the IDS uEye tracking application [A] and
 - one for the Unity application [B]
- The HTC Vive head-mounted display has been connected to computer [B]
- The Leap Motion controller has been connected to computer [B]
- The IDS uEye software suite has been installed on computer [A]
- Computers [A] and [B] are connected via a network cable
- The IP address of the ethernet adapter of computer [A] has been set to 192.168.0.7 with the standard subnet mask of 255.255.255.0
- The IP address of the ethernet adapter of computer [B] has been set to 192.168.0.13 with the standard subnet mask of 255.255.255.0
- It is necessary to use the uEye camera with another computer than the rendering, because the performance could be worse while the computer renders the scene and does the tracking at the same time.

Abbildung 9: Auszug aus der *MaRC* ReadMe-Datei (vgl. 10.1).

Die in Kapitel 3.3.6 beschriebene API verfügt, wie dort geschildert, über die Möglichkeit den Datentransport mittels TCP oder UDP umzusetzen. Für *MaRC* wurde ein TCP-Socket-Netzwerk bestehend aus einem Server und einem Klienten verwendet, weil der in 2.5 beschriebene Fehlerschutz des TCPs ausgenutzt werden sollte. Da die zu übertragenden Daten von *MaRC* in der Größe exakt definiert und hinreichend klein sind, wurde die etwas höhere Geschwindigkeit einer UDP-Socket-Verbindung als unnötig erachtet.

Kann noch angepasst werden und muss noch erweitert werden

4.4 Starten des Systems

Lukas

Nachdem sichergestellt wurde, dass alle in 4.2 beschriebenen Voraussetzungen bestehen, kann das System gestartet werden, indem zunächst die Tracking-Anwendung (auf dem einen Computer, vgl. 3.1.2) und anschließend die aus Unity heraus erstellte Anwendung (auf dem anderen Computer, vgl. 3.1.1) gestartet wird. Auf letzterem Computer beginnt darauffolgend die Menüführung, welche in 4.5 beschrieben ist.

4.5 Menüführung

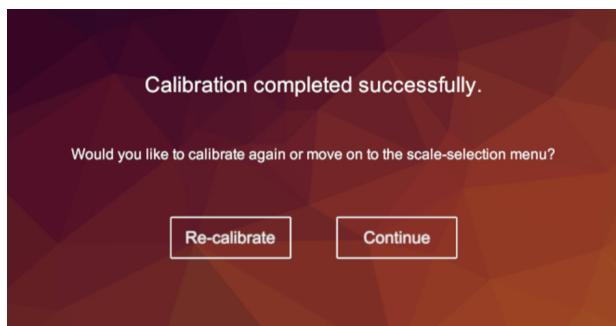
Lukas

Die Menüführung dient dazu, den Benutzer durch alle notwendigen Schritte zu leiten, die vor dem Starten der eigentlichen Simulation erforderlich sind. Im nachfolgenden Abschnitt 4.5.1 werden alle verfügbaren Menüs der Anwendung aufgelistet und kurz beschrieben, während im Abschnitt 4.5.2 der Ablauf der Menüführung erläutert wird.

4.5.1 Menüs

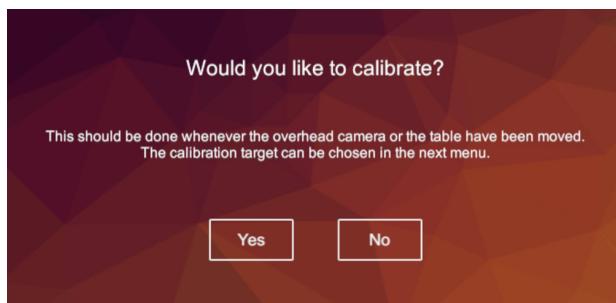
Laura

Die folgenden Menüs sind Bestandteil der Menüführung:



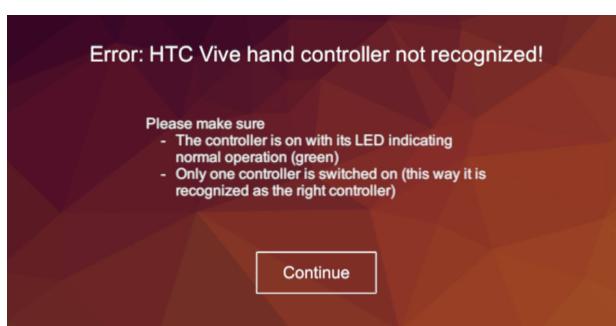
CalibDone:

Wird aufgerufen, wenn die Kalibrierung des Arbeitsbereichs abgeschlossen ist. Es informiert den Benutzer, dass die Kalibrierung erfolgreich war und der Vorgang fortgesetzt werden kann.



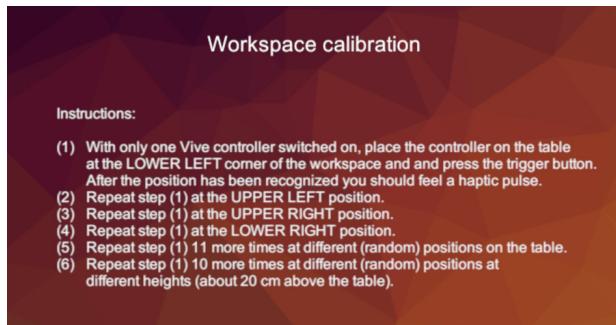
CalibrateOrNot:

Erscheint nach dem Verlassen des Welcome-Menüs und erlaubt dem Benutzer eine Kalibrierung durchzuführen oder eine bereits durchgeführte Kalibrierung zu laden.

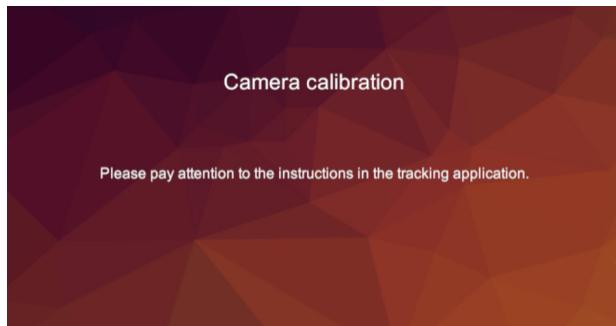


ControllerNotFound:

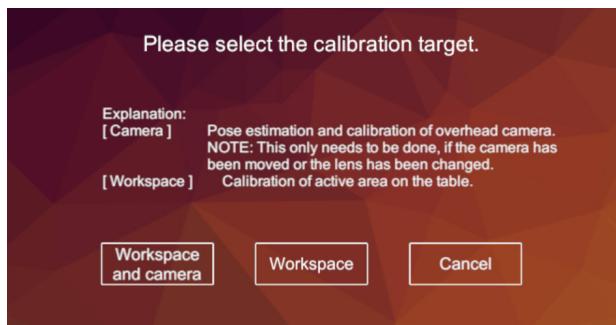
Warnt den Benutzer nach dem Starten der Kalibrierung, dass der HTC Vive Controller, welcher für die Kalibrierung benötigt wird, nicht eingeschaltet ist. Während das Menü angezeigt wird, kann der Benutzer den Controller einschalten und anschließend auf Continue klicken.

**doPlaneCalibInVS:**

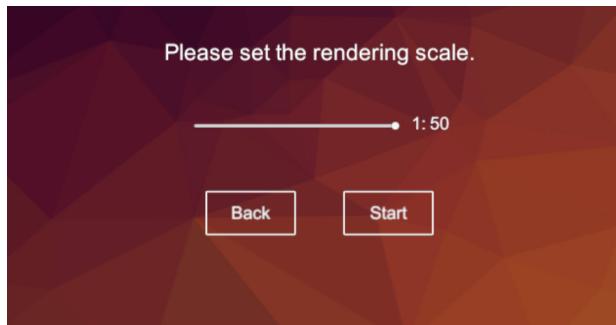
Dient dem Benutzer als Anleitung für die Durchführung der Arbeitsbereich-Kalibrierung. Diese wird in [5.2.2](#) genauer beschrieben.

**doPoseCalibInVS:**

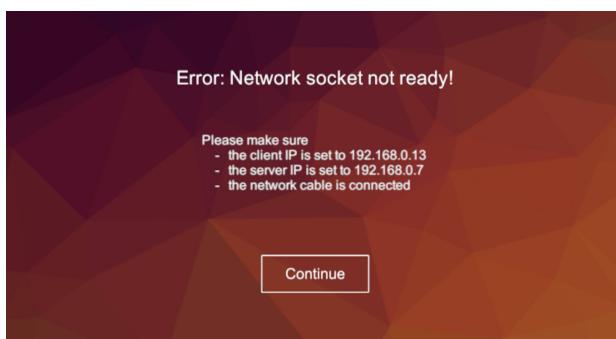
Dient dem Benutzer als Anleitung für die Durchführung der Kamera-Kalibrierung. Diese wird in [5.2.1](#) genauer beschrieben.

**SelectCalibrationTarget:**

Erlaubt die Auswahl der Art der Kalibrierung. Es kann hier entweder nur der Arbeitsbereich oder sowohl der Arbeitsbereich, als auch die Kamera kalibriert werden. Die Kalibrierung ist näher in [5.2](#) beschrieben.

**SetScale:**

Stellt das letzte Menü vor dem Starten der Simulation dar. In diesem kann der Benutzer den Maßstab der Gebäudesimulation einstellen und anschließend die Simulation starten.

**SocketNotReady:**

Warnt den Benutzer nach dem Verlassen des **Welcome**-Menüs, dass die Netzwerkverbindung zwischen den beiden Computern nicht bereit ist. Nach Bestätigung dieses Hinweises durch einen Klick auf **Continue**, kehrt der Benutzer zum **Welcome**-Menü zurück.

**Welcome:**

Erscheint als erstes Menü. Hier erhält der Nutzer eine kurze Information darüber, wie die Anwendung heißt und wozu sie dient.

4.5.2 Ablauf der Menüführung

Lukas

Der Ablauf der Menüführung von *MArC* ist in Abbildung 10 dargestellt. Die einzelnen Menüs sind bereits in 4.5.1 beschrieben worden.

Nach dem Starten der Anwendung wird zunächst das Menü **Welcome** angezeigt. Dieses enthält nur einen Button *Get started*. Sobald dieser gedrückt wird, prüft die Anwendung, ob eine Netzwerkverbindung zu dem Computer mit der Tracking-Anwendung besteht. Sollte dies nicht der Fall sein, wird das Menü **SocketNotReady** angezeigt. Dieses verlässt der Benutzer über einen Klick auf **Continue**, anschließend wird erneut das Menu **Welcome** angezeigt. Wenn zu diesem Zeitpunkt die Netzwerkverbindung korrekt hergestellt wurde, gelangt der Benutzer zum Menü **CalibrateOrNot**, anderenfalls wird wiederholt **SocketNotReady** angezeigt.

In **CalibrateOrNot** hat der Benutzer die Auswahl zwischen den Schaltflächen *Yes* und *No*. Bei einem Klick auf *Yes* wird anschließend **SelectCalibrationTarget** angezeigt, bei einem Klick auf *No* lädt das System eine zuvor durchgeführte Kalibrierung und das Menü **SetScale** wird geöffnet.

SelectCalibrationTarget stellt den Benutzer vor die Wahl entweder nur den Arbeitsbereich (*Workspace*) oder sowohl den Arbeitsbereich als auch die Kamera zu kalibrieren (*Camera and Workspace*). Außerdem besteht die Möglichkeit über *Cancel* zum Menü **CalibrateOrNot** zurückzukehren.

Wählt der Benutzer *Camera and Workspace* in **SelectCalibrationTarget** aus, so informiert die Anwendung die Tracking-Anwendung auf dem anderen Computer

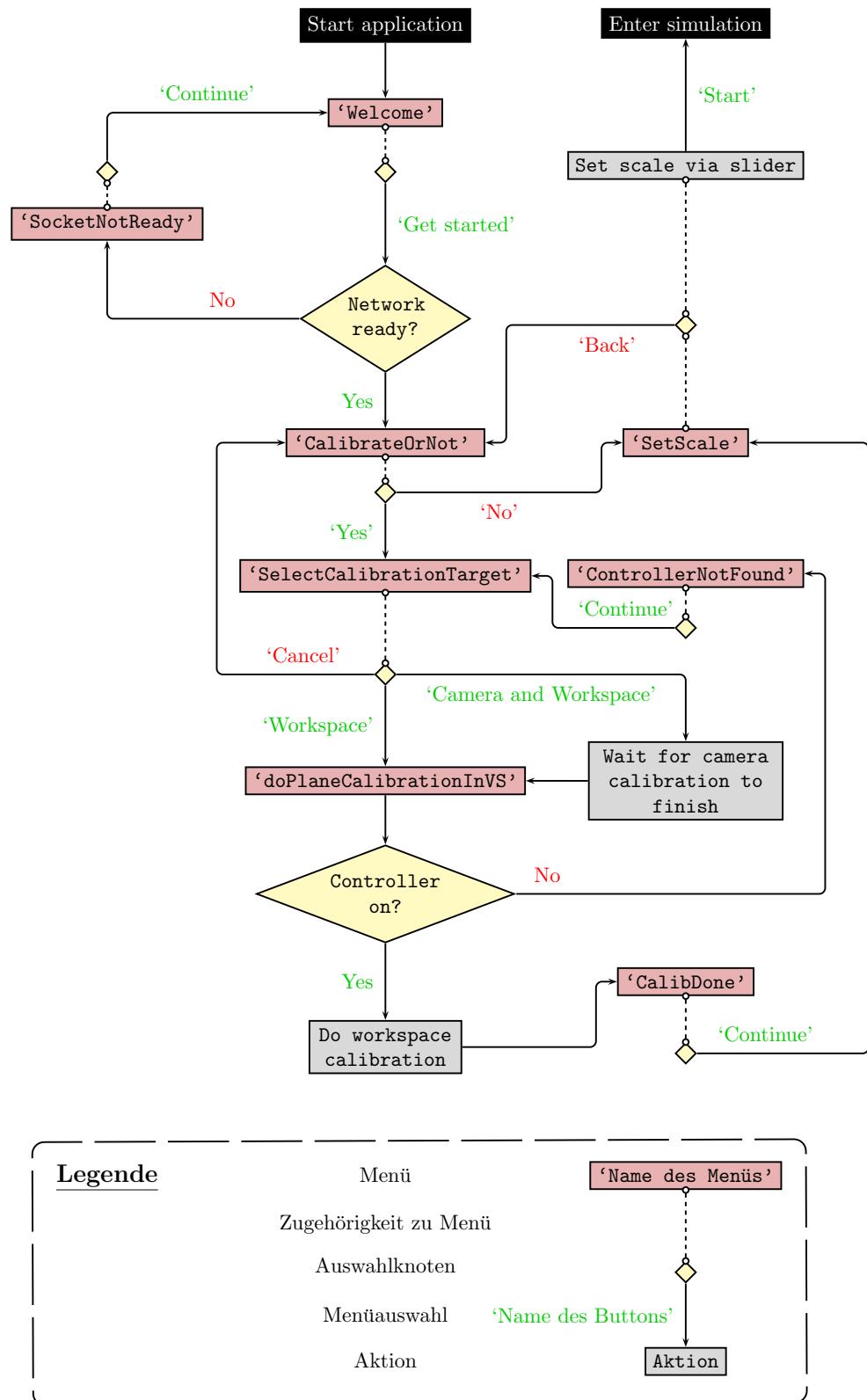


Abbildung 10: Flussdiagramm der Menüführung.

und wartet anschließend darauf, dass von dort die Bestätigung gesendet wird, dass die Kamerakalibrierung abgeschlossen ist. Anschließend wird das Menü `doPlaneCalibrationInVS` angezeigt, welches auch aufgerufen wird, wenn der Benutzer *Work-space* in `SelectCalibrationTarget` wählt.

Im Menü `doPlaneCalibrationInVS` wird zunächst geprüft, ob der für die Kalibrierung notwendige HTC Vive Controller eingeschaltet ist. Sollte dies nicht der Fall sein, wird `ControllerNotFound` aufgerufen. Dieses kann mit einem Klick auf *Continue* verlassen werden, woraufhin wieder `SelectCalibrationTarget` angezeigt wird. Sofern der HTC Vive Controller beim Aufruf von `doPlaneCalibrationInVS` eingeschaltet ist, wird nach Durchführung der Kalibrierung des Arbeitsbereichs das Menü `CalibDone` angezeigt.

`CalibDone` kann über einen Klick auf *Continue* verlassen werden und führt den Nutzer anschließend zu `SetScale`. Aus diesem Menü kann über den Button *Back* entweder zu `CalibrateOrNot` zurückgekehrt oder die Simulation mit dem im Menü über den Slider eingestellten Maßstab gestartet werden.

4.6 Kontex Menü

Laura und Lukas

??

4.7 Marker Handles

Paul

Jeder Marker verfügt über die Möglichkeit in seiner Größe in X-, Y- und Z-Achse verändert zu werden. Diese sog. "Handles" sind sichtbar, wenn das Kontextmenü angezeigt wird (s. Abschnitt ??). Die Handles bestehen aus Zylindern, die mit einem Collider versehen wurden. Die X- und Y- Achse wird betätigt, indem der Benutzer mit einem Finger das jeweiligen Handle berührt und dann in die entsprechende Richtung zieht. Die Größe in Z-Achse wird über "+ und "Button gesteuert. Drückt der Benutzer diese Buttons, wird das Gebäude um je ein Stockwerk erhöht oder verniedrigt.

Die Interaktion der Handles mit der Hand des Nutzers wurde über das Leap-Handmodell umgesetzt; dieses besitzt Collider, die an den Fingern des Modells angebracht sind und sich simultan mit der Hand des Nutzers bewegen. Berührt die Hand nun einen der Handler, wird dieser Collider des Handlers angesprochen". Im Falle der X-, und Y-Handles wird in der `update` Funktion des Skripts `contextMenuTrigger.cs` ein Vektor berechnet, der die Start und Endposition der Bewegung miteinander verrechnet. Im Falle einer Bewegung in X Position wird die sog. "localDifference" berechnet, dies ist der Abstand des Fingers zur Position des Markers. Die eigentliche Skalierung wird dann wie folgt berechnet:

$$pos = (startPosition.x - localDifference.y)/2, startPosition.y, startPosition.z) \quad (1)$$

Das Gebäude wird dann um den Vektor pos im lokalen Objektkoordinatensystem vergrößert.

Für die Bewegung in Y berechnet sich der Vektor wie folgt:

$$pos = (startPosition.x, startPosition.y, (startPosition.z - localDifference.y)/2) \quad (2)$$

Es fällt auf, dass in der zu verändernden Achse verschiedene Achsen zur Berechnung verwendet werden müssen. Dies kommt daher, dass das Koordinatensystem der Leap Motion nicht equivalent zu dem in Unity ist.

Dass der Wert der zu verändernden Achse halbiert wird, hat sich empirisch als Sinnvoll herausgestellt. Ohne diese "Übersetzung" der Bewegung würde das Gebäude nicht equivalent zur Fingerbewegung vergrößert oder verkleinert werden.

4.7.1 Menüführung

Laura

4.7.2 Architektur Berechnung

Laura

4.8 Table Menü

Paul

Zur intuitiven Bedienung des *MArC* wurden die Menüs, die während der Laufzeit benutzt werden können, auf dem Arbeitstisch platziert. Dies hat den Vorteil, dass zum einen diese Menüs permanent sichtbar sein können, ohne den Nutzer während der Arbeit zu stören und zum anderen ein haptisches Feedback bei der Nutzung möglich ist.

Möchte der Nutzer die Menüs bedienen, so drückt er mit dem Finger, der durch die *Leap Motion* getrackt wird auf die Buttons und damit gleichzeitig auf den Tisch. Dies erleichtert die Benutzung deutlich im Vergleich zu der Nutzung von Menüs die im Raum schweben.

Im Laufe der Entwicklung hat sich für das rechte Menü der Name *Table Menü* etabliert. Im folgenden wird dessen Funktion im Detail erläutert.

4.8.1 Szenen Management

Paul

Eine Anforderung an das System war, dass der Nutzer erstellte Szenen abspeichern und wieder aufrufen können soll. Anhand des *Table Menüs* ist dies möglich. Die Funktionen des Menüs sind:

- Speichern von Szenen
- Laden von Szenen und aufrufen des *Match Modus* (s. Abschnitt 4.8.4)
- Scrollen durch die gespeicherten Szenen

4.8.2 Speichern von Szenen

Paul

Möchte der Nutzer eine Szene speichern, so drückt er mit dem Finger auf den Button „Save“. *MArC* speichert die aktuelle Szene automatisch in dem Ordner „\Resources\saves\Dateiname.xml“.

Der Dateiname setzt sich aus aktuellem Datum und aktueller Zeit zum Speicherzeitpunkt wie folgt zusammen:

Tag-Monat-Jahr-Stunde-Minute-Sekunde.xml

Durch diese Markierung können später Dateien identifiziert werden, die auch nur Sekunden hintereinander gespeichert wurde.

Da innerhalb von *Unity* eine hierarchische Anordnung sämtlicher Elemente in Form eines Szenengraphen angewendet wird, bot sich eine Speicherung der Daten ebenfalls hierarchisch in form eines XML Dokumentes an.

Die *Extensible markup language (XML)* beschreibt eine Klasse von Daten Objekten (XML Documents) und ermöglicht das hierarchische Abspeichern von geparssten oder ungeparssten Daten [6]. In C# sind Verarbeitungsklassen bereits implementiert. Mittels dieser sog. *XML Prozessoren* wird ein Datenzugriff erleichtert.

Dateiaufbau einer gespeicherten Szene Das gesamte Dokument wird mit einem Hauptknoten <AR2_COMPOSERSCENE> umspannt. Innerhalb diesem wird mit texttt <time> ein Zeitstempel mit gespeichert, falls der Dateiname einmal umbenannt werden sollte. Anschließend folgt der texttt <TableObject> Knoten, innerhalb dessen die Marker gespeichert werden. Jeder Marker wird nach dem folgenden Schema gespeichert, welches den Aufbau in Unity repräsentiert:

```
<Name>
<Kindknoten>
<PositionX>
<PositionY>
```

```

<PositionZ>
<RotationX>
<RotationY>
<RotationZ>
<ScaleX>
<ScaleY>
<ScaleZ>
</Name>

```

Wobei sich dies als Kurzschreibweise versteht, innerhalb der Positions/Rotations-/Scale Knoten ist der entsprechende Wert eingetragen.

Vorgehen der Speicherung Das Skript `save.cs` ist für die Speicherung der Szenen verantwortlich. Zunächst wird der Zeitstempel gesichert und der Dateiname generiert. Anschließend wird ein neues XML Dokument erstellt. An dieses wird zunächst der äußerste Knoten angehängt sowie der Zeitstempel. Anschließend wird die rekursive Funktion `traverseHierarchie` mit dem TableObject der Unity Szene aufgerufen und wird für jeden Kindknoten ausgeführt.

Diese Funktion erstellt jeweils die Knoten für den Namen, die Position, Rotation und Skalierung. Die Werte werden mittels `Node.InnerText` in die Knoten gespeichert.

Ist diese Funktion durch alle Kindknoten gegangen, wird am Ende noch die Skalierung der Szene in dem Knoten `globalBuildingScale` abgespeichert, damit dieser beim Laden der Szene rekonstruiert werden kann.

Darstellen von gespeicherten Szenen Die gespeicherten Szenen werden in Listenform im Table Menü dargestellt. Zu oberst sind immer die aktuellen Szenen. Über die Buttons "1-6", "7-12", "13-18" und "19-24" kann umgeblättert werden und weitere, in der Vergangenheit liegende Szenen dargestellt werden. Für diese Darstellung ist das Skript `Timeline.cs` zuständig. Bei jedem Speichern wird die Darstellung aktualisiert, so dass immer alle neuesten gespeicherten Szenen sichtbar sind.

4.8.3 Laden von Szenen

Paul

Das Laden der Szenen ist ein komplexer Vorgang. Dies kommt daher, dass auf der einen Seite Markerdaten in der zu öffnenden .xml Datei gespeichert sind und geladen werden müssen. Auf der anderen Seite gibt es evtl. noch Marker, die auf dem Arbeitsfeld liegen und aktuell getrackt werden. Beide Informationsströme müssen beim laden der Szene berücksichtigt werden und korrekt verarbeitet werden. Ein einfaches übertragen von Positions-, Rotations- und Skalierungsdaten auf vorhandene Markerobjekte ist nicht möglich, da diese permanent durch den TCP Datenstrom überschrieben werden würden.

Vorgehen beim Laden Durch Auswählen der Datei im Table Menü mit dem Finger wird das Skript `open.cs` gestartet. Dieses setzt zuerst den Pfad aus dem aktuellem Applikationspfad mit dem Unterordner Resources und saves zusammen und fügt den Namen der zu öffnenden .xml Datei aus dem Table Menü hinzu. Anschließend wird die Datei eingelesen, die internen Knoten liegen dann in einer XmlNodeList bereit. Es wird die Funktion `crawlXML` aufgerufen.

Da aus organisatorischen Gründen alle Marker in der Datei gespeichert werden müssen, wird die XmlNodeList bearbeitet und geprüft, welche der Marker als aktiv" gespeichert wurden. Aktive Marker sind solche, die zum Zeitpunkt des Speichern sichtbar waren. Alle aktiven Marker werden in der ArrayList `activeMarkerIDs` gespeichert.

Diese ArrayList wird nun, nachdem sie vollständig gefüllt ist, bearbeitet: Für jeden Eintrag, also jeden aktiven Marker wird ein Vorlagemarker instantiiert. Dies ist ein Marker, der bereits alle Komponenten die benötigt werden angeheftet hat und dem nur noch die Parameter zugewiesen werden müssen. Dies wird dann getan; jeder Marker bekommt die OriginalID + 100 als Namen zugewiesen. Dies ist wichtig um später die aus der .xml Datei gelesenen Marker von den aktuell über TCP gesteuerten Marker unterscheiden zu können. Anschließend wird werden die weiteren Parameter wie Position, Rotation und Skalierung an den Marker übergeben. Schließlich wird der Marker an das "TableObject in Unity gehangen und sichtbar gemacht. Als letzter Schritt wird die Markervariable "MatchMode auf "true" gesetzt und das Skript `matchMode.cs` aktiviert. Dieses wird detailliert im Abschnitt 4.8.4 beschrieben. Die Marker die sich in diesem Match Modus befinden blinken grün und rot, um zu signalisieren, dass sie auf einen echten"Marker warten. Sind alle Marker auf diese Weise geladen und parametrisiert, wird der globale Maßstab an das Skript `setupScene.cs` übergeben.

Zu diesem Zeitpunkt sind also zwei Arten von Markern sichtbar: Die geladenen aus der .xml Datei, die grün und rot blinken und eine ID > 100 haben, und die aktiven Marker die getrackt werden.

4.8.4 Match Modus

Paul

Der Match Modus ist ein Systemzustand, der nach dem Laden von gespeicherten Szenen aktiviert wird. Es werden wie im Abschnitt 4.8.3 beschrieben die geladenen Marker und die getrackten Marker gleichzeitig angezeigt. Der Benutzer muss jetzt die echten Marker an die Position der geladenen virtuellen Marker schieben. Ist ein Marker im Match Modus, sind vier Collider an den Ecken des virtuellen Markers wichtig. Die Markerinterne Variable "matchMode ist = "false" gesetzt. Schiebt der Benutzer nun einen Marker an die Position des virtuellen Markers, werden bei genauer Positionierung alle vier Collider des geladenen virtuellen Markers mit den vier Collidern des anderen virtuellen Markers, der von dem Nutzer bewegt wurde, aktiviert. Ist dies der Fall, und nur dann, wird die Markerinterne Variable "matchMode-Ready auf "true" gesetzt. Dieses Verhalten wird von dem Skript `StopMatchMode.cs` kontrolliert. Sind alle Marker bereit, werden die Parameter aus den gespeicherten

Markern auf diejenigen Marker übertragen, die der Benutzer an die Position der virtuellen Marker geschoben hat. Um diese Funktionalität kümmert sich das Skript `DataHandler.cs`. Die geladenen Marker mit der ID > 100 werden im Anschluss gelöscht. Am Ende des Match Modus existieren wieder nur die TCP gesteuerten Marker, die jetzt allerdings die Parameter aus der geladenen Datei übernommen haben und somit in der Gesamtheit die geladene Szene repräsentieren.

5 Tracking Programm

Vera

Zur Verfolgung und Positionsbestimmung der Würfel Marker (siehe Kapitel 3.1.8) wurde ein Programm entwickelt, welche alle erforderlichen Ressourcen und Funktionen bereit stellt und verknüpft. Zunächst muss die Initialisierung und der Zugriff der uEye Kamera (siehe Kapitel 3.1.4) ermöglicht werden. Im Anschluss ist das Erstellen beziehungsweise laden aller relevanten Kalibrierungsinformationen unabdingbar. Nach erfolgreichem Abschluss beider Schritte führt das Programm das Tracking der Aruco Marker und der grünen Rechtecke aus und steuert die Verwaltung und TCP Übertragung (siehe Kapitel 2.5) der Registrierten Würfel Marker Informationen. Diese Programm wurde in der IDE *Visual Studio* (siehe Kapitel 3.3.2) in der Programmiersprache *C++* entwickelt. Für die Entwicklungen wurden die uEye SDK, *OpenCV* (siehe Kapitel 3.3.3) sowie die Standardbibliothek *Winsock* zur Hilfe genommen. In den folgenden Abschnitten wird die Vorgehensweise im Detail erklärt.

5.1 uEye Ansteuerung

Vera

Der erste Prozess des Tracking Programms ist die Initialisierung der uEye Kamera im Live-Bild-Modus. Mit Hilfe der vom Hersteller bereit gestellten SDK kann die Kamera mit allen benötigten Eigenschaften parametrisiert und alle notwendigen Speicher allokiert. Dies übernimmt die Funktion `inituEyeCam` in der Klasse `uEye_input.cpp`. Die verwendeten Parameter können aus der Tabelle 6 entnommen werden. Die Kamera wird mit dem maximalen Pixel Clock und der maximalen Framerate betrieben. Aus diesen Einstellungen ergibt sich auch die Belichtungszeit. Um einen höheren Kontrast zur Erkennung der Marker zu erzielen wurde zusätzlich das Hardware Signal zwanzigfach verstärkt. Im Live-Bild-Modus werden die generierten Aufnahmen fortlaufend in der selben Speicheradresse überschrieben. Auf diesen Speicherplatz kann das Programm jederzeit mit der Funktion `getCapturedFrame` zugreifen, die die aktuellste Aufnahme zur Weiterverarbeitung zurück gibt. Nach der Beendigung des Tracking Algorithmus wird zusätzlich noch die Funktion `exitCamera` bereit gestellt, welche den verwendeten allokierten Speicher wieder freigibt.

Parameter	Wert
Pixel Clock	37 ms
Frame Rate	23 fps
Belichtungszeit	40 ms
Gamma	2.2
Digitale Verstärkung	20

Tabelle 6: Parametrisierung der uEye Kamera für das Tracking.

5.2 Kalibrierung

Laura

Um das Ziel von *MArC* zu erreichen, an den Positionen der Aluminiumwürfel im Arbeitsbereich in der virtuellen Realität von Unity gerenderte Würfel darzustellen, muss das System kalibriert werden. Die Kalibrierung hat zum Ziel, eine Koordinatentransformation zu finden, die Positionen im Kamera-Koordinatensystem in das Unity-Koordinatensystem transformiert.

Zu diesem Zweck muss eine zweistufige Kalibrierung durchgeführt werden. Zunächst sorgt die Kamerakalibrierung dafür, dass Bildkoordinaten auf dem Sensor der Kamera in das 3D-Kamera-Koordinatensystem transformiert werden. Dafür wird sich einiger OpenCV-Funktionen in Verbindung mit Aruco-Markern bedient. Dieser Vorgang wird nachfolgend in [5.2.1](#) genauer beschrieben.

Der nächste Schritt, die Kalibrierung des Arbeitsbereichs, bestimmt über Punkt-Korrespondenzen – also in zwei verschiedenen Koordinatensystemen bekannte Punkte – eine affine 3D-Transformation, welche die Abbildung vom Kamera-Koordinatensystem auf das Unity-Koordinatensystem ermöglicht. Dieser Kalibrierungsschritt wird nachfolgend in [5.2.2](#) näher beschrieben.

Trotz einer sorgfältigen Umsetzung der in den nächsten beiden Kapiteln beschriebenen Kalibrierungsschritte, ist es nicht gelungen, den Aluminiumwürfel und den gerenderten Würfel vollständig zur Deckung zu bringen. Der entstandene Fehler sowie mögliche systembedingte Fehlerquellen werden in [5.2.3](#) und in [5.2.4](#) beschrieben.

5.2.1 Kamerakalibrierung

Laura

Es gibt viele verschiedene wissenschaftliche Ausführungen über die Durchführung einer Kamerakalibrierung, wie z.B. [41], [52] und [13]. Dabei unterscheidet man häufig zwischen automatischen und manuellen Kalibrierungen.

In Abbildung [11](#) sind die Zusammenhänge zwischen dem Projektionszentrumkoordinatensystem der Kamera, sowie deren Bildebene und dem Weltkoordinatensystem zu sehen. Im Gegensatz zu der Abbildung und den erwähnten Kalibrierungsansätzen reicht die Umrechnung von Bildkoordinaten in Weltkoordinaten, im vorliegenden Fall, nicht aus. Es muss eine Umrechnung der Bildkoordinaten in den Unity Raum

erfolgen, um zu gewährleisten, dass die gerenderten Würfel anschließend deckungsgleich mit den Aluminiumwürfeln sind. Koordinaten des Unity Raumes werden im Folgenden Unitykoordinaten genannt.

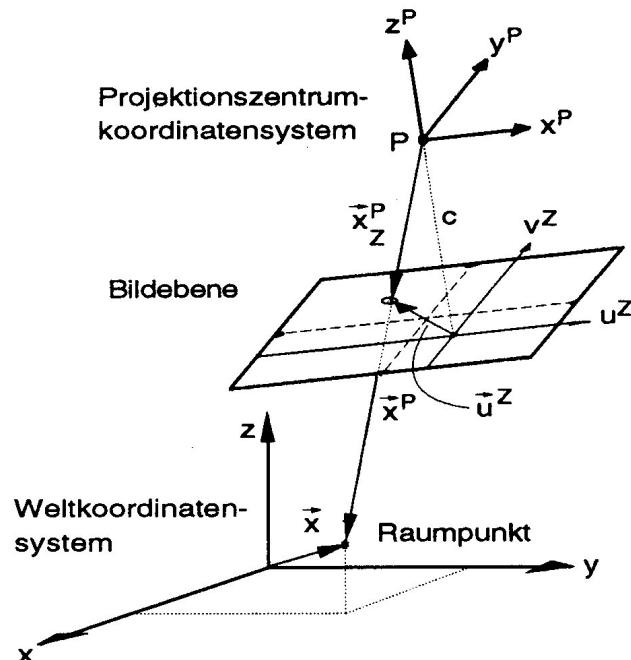


Abbildung 11: Lage des Kamerakoordinatensystems in Bezug auf Projektionsebene und Weltkoordinatensystem. [28]

Die Kamerakalibrierung, die diesem Projekt zu Grunde liegt, ist als manuell einzustufen und kann, wenn gewünscht bzw. benötigt, zu Beginn der eigentlichen Anwendung durchgeführt werden. Um die verschiedenen Koordinatensysteme auseinander zu halten, wird zu Beginn eine Notation festgelegt, die in Tabelle 7 eingesehen werden kann. Zusätzlich können in der Tabelle sowohl die einzelnen Koordinatensysteme, als auch die einzelnen Berechnungsschritte der Kamerakalibrierung nachvollzogen werden, die im Folgenden erläutert werden. Lässt man die ersten zwei Zeilen der Tabelle weg, so ist eine Transformation von Weltkoordinaten \vec{x} in Kamerakoordinaten \vec{u} schematisch dargestellt. Dieses Schema muss für die Kamerakalibrierung von *MaRC* invertiert werden und um die Koordinatentransformation in Unitykoordinaten ergänzt werden.

Die einzelnen Schritte aus Tabelle 7 kann man in vier bzw. drei Abschnitte einteilen: Im ersten Abschnitt wird der Hauptpunkt so versetzt, dass der Koordinatenursprung des Kamerakoordinatensystems mit dem Bildkoordinatensystem übereinstimmt. Dies geschieht, indem man für die eine Achse $u^V = u - \Delta u$ und für die andere Achse entsprechend $v^V = v - \Delta v$ berechnet.

Zusätzlich wird die Bildebene verkippt, so dass gilt $\vec{x}^V = R_v \cdot \vec{x}_D^P$. Bezeichnet φ den Drehwinkel um die x^P -Achse und ϑ den Drehwinkel um die y^P -Achse, so lässt sich R_v wie folgt berechnen:

	Koordinaten	Komponenten	Transformation
\vec{x}^U	Unity	x^U, y^U, z^U	
\downarrow			Koordinatentransformation
\vec{x}	Welt	x, y, z	
\downarrow			Koordinatentransformation
\vec{x}^P	Projektionszentrum	x^P, y^P, z^P	
\downarrow			Projektion
\vec{x}_Z^P	Projektionszentrum	$u^Z, v^Z, -c$	
\downarrow			Linsenverzeichnung
\vec{x}_D^P	Verzeichnung	$u^D, v^D, -c$	
\downarrow			Bildebenenverkippung
\vec{x}^V	Verkippung	$u^V, v^V, -c^V$	
\downarrow			Bildhauptpunktverschiebung
\vec{u}	Sensor	u, v	

Tabelle 7: Parameter und Berechnungsschritte der Kamera Kalibrierung.[\[28\]](#)

$$R_v = \begin{pmatrix} r_{v11} & r_{v12} & r_{v13} \\ r_{v21} & r_{v22} & r_{v23} \\ r_{v31} & r_{v32} & r_{v33} \end{pmatrix} = \begin{pmatrix} \cos \vartheta & \sin \vartheta \sin \varphi & -\sin \vartheta \cos \varphi \\ 0 & \cos \varphi & \sin \varphi \\ \sin \vartheta & -\cos \vartheta \sin \varphi & \cos \vartheta \cos \varphi \end{pmatrix} \quad R_v \in \mathbb{R}^{3 \times 3} \quad (3)$$

Die negative verkippte Kamerakonstante $-c^V$, die in Tabelle 7 aufgeführt ist, berechnet sich wie in [\[28\]](#) beschrieben nach der Formel:

$$-c^V = -\frac{c + u^V \cdot r_{v13} + v^V \cdot r_{v23}}{r_{v33}} \quad (4)$$

Im zweiten Schritt wird im Allgemeinen die Linsenverzeichnung herausgerechnet. Im vorliegenden Fall wurde bewusst auf diesen Schritt verzichtet und die zugehörigen Entzerrungskoeffizienten werden für alle weiteren Berechnungen auf null gesetzt. Dieses Vorgehen wurde gewählt, da einige Tests gezeigt haben, dass die berechneten Entzerrungskoeffizienten stark voneinander abgewichen sind, obwohl dies beim Benutzen der gleichen Kameraeinstellungen und dem gleichen Objektiv nicht der Fall sein dürfte.

Weil wie bereits beschrieben, die Linsenverzeichnung vernachlässigt wurde, kann man Schritt 1 und 2 zu einem Schritt zusammenfassen und vereinfacht darstellen. Dieser erste Schritt, also die Hauptpunktverschiebung und Bildebenenverkippung, lässt sich mit Hilfe der intrinsischen Kameramatrix $M_{\text{intrinsisch}}$ zusammenfassen. Diese beinhaltet neben den Brennweiten f_u und f_v noch die Koordinaten des Hauptpunktes u^V und v^V in Bildkoordinaten und hat somit vier Freiheitsgrade.

$$M_{intrinsisch} = \begin{pmatrix} f_u & 0 & 0 & u^V \\ 0 & f_v & 0 & v^V \\ 0 & 0 & 1 & 0 \end{pmatrix} \quad M_{intrinsisch} \in \mathbb{R}^{3x4} \quad (5)$$

Zu diesem Zeitpunkt sollte nun das Kamerakoordinatensystem mit dem Weltkoordinatensystem übereinstimmen und es kann sich im nächsten Schritt darum gekümmert werden, dass das Kamerakoordinatensystem verschoben und gedreht wird. Während eine dreidimensionale Rotation im Allgemeinen mit Matrix R aus Formel 6 beschrieben werden kann, reicht für die Translation ein Vektor, wie t aus Formel 7 aus.

$$R = R_\gamma \ R_\beta \ R_\alpha = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \quad R \in \mathbb{R}^{3x3} \quad (6)$$

$$t = (t_x \ t_y \ t_z)^T \quad t \in \mathbb{R}^{3x1} \quad (7)$$

Diese beiden Transformationen können wie in Formel 8 zur extrinsischen Kameramatrix $M_{extrinsisch}$ zusammengefasst werden. Diese hat sechs Freiheitsgrade, nämlich drei für den Translationsvektor t und drei für die Eulerwinkel der Rotationsmatrix R für die dementsprechend gelten muss $R \in SO(3)$.

$$M_{extrinsisch} = (R \ | \ t) \quad M_{extrinsisch} \in \mathbb{R}^{3x4} \quad (8)$$

Schritt 1 und 2 lassen sich vereinfachen, indem man die intrinsische Kameramatrix $M_{intrinsisch}$ und die extrinsische Kameramatrix $M_{extrinsisch}$ nach der Formel 9 zu einer Matrix M zusammenfasst.

$$M = M_{intrinsisch} \cdot M_{extrinsisch} \quad M \in \mathbb{R}^{3x4} \quad (9)$$

Die Umrechnung von Kamerakoordinaten in Weltkoordinaten kann dann mit der invertierten Matrix M , wie in Formel 10 berechnet werden.

$$\vec{x} = M^{-1} \cdot \vec{u} \quad (10)$$

An diesem Punkt hat man die Kamerakoordinaten vollständig in Weltkoordinaten überführt und sucht nun eine Transformationsmatrix um diese Punkte auf ihre korrespondierenden Punkte in Unitykoordinaten abzubilden. Die dazu benötigte affine 3D-Transformation wird in Kapitel 5.2.2 erläutert.

5.2.2 Kalibrierung des Arbeitsbereichs

Laura

5.2.3 Kalibrierungsfehler

Laura

In den Kapiteln [5.2.1](#) und [5.2.2](#) wird der Algorithmus zur Berechnung der Koordinatentransformation von Kamerakoordinaten in Unitykoordinaten beschrieben. Beim Benutzen des fertig kalibrierten System

5.2.4 Mögliche Fehlerquellen

Laura

Der in [5.2.3](#) beschriebene Fehler der bei der Abbildung von

5.3 Marker Detektion

5.4 Green Keying

Vera

5.5 ArUco Marker Tracking

Vera

5.6 Marker Objekt

Vera

5.7 Zuordnung der Identitäten

Vera

6 Ausblick

6.1 Trackingmethoden

Vera

6.2 AR Erweiterung mit der Webcam

Laura

6.3 Daten als Excel Tabelle

Paul

6.4 Validierung

Luke

7 Projektmanagement

7.1 Zeitplanung

Paul

7.2 Gant Chart

Paul

7.3 Planungsmethoden

Paul

7.4 Reflexion

7.4.1 Kommuniktion im Team und nach Außen

Paul

7.4.2 Probleme

Paul

8 Zusammenfassung

Lukas

9 Steckbrief

Vera

10 Anhang

10.1 *MArC* ReadMe-Datei

MArC ReadMe

Masterprojekt Lukas Kolhagen, Vera Brockmeyer, Laura Anger, Paul Berning

Starting the System

The following software is needed:

- IDS uEye Driver and SDK (<https://de.ids-imaging.com/download-ueye-win32.html>)
- Steam VR (<http://store.steampowered.com/about/>)
- Leap Driver (<https://developer.leapmotion.com/windows-vr>)
- The OpenCV Framework (<http://opencv.org/downloads.html>)

In order to start the system, make sure the following requirements are met:

- 2 Computers are available:
 - one that runs the IDS uEye tracking application [A] and
 - one for the Unity application [B]
- The HTC Vive head-mounted display has been connected to computer [B]
- The Leap Motion controller has been connected to computer [B]
- The IDS uEye software suite has been installed on computer [A]
- Computers [A] and [B] are connected via a network cable
- The IP address of the ethernet adapter of computer [A] has been set to 192.168.0.7 with the standard subnet mask of 255.255.255.0
- The IP address of the ethernet adapter of computer [B] has been set to 192.168.0.13 with the standard subnet mask of 255.255.255.0
- It is necessary to use the uEye camera with another computer than the rendering, because the performance could be worse while the computer renders the scene and does the tracking at the same time.

The start procedure is as follows:

- Start the application [Masterprojekt.exe] located in the [MArC_Tracking] directory on computer [A]
- Make sure the opened command window indicates that the server socket on port 10000 is set to listen
- Start the Unity build [MArC.exe] located in the [MArC_Unity] directory on computer [B]
- Follow the instructions on the screen of computer [B]
- If there is an offset between the table plane and the leap hand model, place the right hand onto the table and press the trigger button of the Vive Controller. This eliminates the height offset

Calibrating the System

- After starting the application on computer [B], the user will be asked whether he wants to calibrate the system
 - The user can choose between calibrating “camera and workspace” or only the “workspace”
- 1) Calibration of the uEye camera:
- NOTE: It is explicitly recommended to calibrate the camera only when it is needed!
 - Once chosen, a camera window will be opened on computer [A]. The following steps can be controlled on this computer, while computer [B] will be activated again, when it comes to the calibration of the workspace
 - Use the elevated chessboard to calibrate the system
 - Follow the instructions on the console of computer [A] and press “g”
 - The system will do 25 captures automatically and the user will hear a beep after each capture. (Taking the first 15 captures, while placing the elevated chessboard on the desk and the last 10 in a height of 20 cm leads to the best results)
 - After a longer beep (signal that the user has finished the capturing. Process) the calculation of the intrinsic camera parameters and distortion coefficients is made and a text file will be saved
 - This text file allows the user to use the system without making a new camera calibration by loading the parameters directly from the text file
 - After the calculation is completed, the calibration of the workspace can be done (see 2) Calibration of the workspace)
- 2) Calibration of the workspace:
- Follow the instructions displayed on computer [B]
 - Use the HTC Vive controller with the attached Aruco Marker (ID 49, dictionary: DICT_4X4_50) and press the button as described
 - 25 captures are necessary (each capture is confirmed by a haptic impulse)
 - NOTE: The order of the first four captures is important!
 - The user have to make sure to form up a rectangle which is equal to his desired workspace. Starting in the lower left corner and going clockwise
 - it is recommended to place the controller on different positions on the table for the next 11 captures as well
 - the last 10 captures can be done lifting the controller up to 20 cm and vary its position
 - After capturing all positions follow the instructions on computer [B]

Glossary of recurring terms:

Marker Cubes:

- There are two types of marker cubes: real ones that the user can touch and interact with and virtual ones that are rendered at the positions of the real world marker cubes

Context Menu:

- Every (virtual) marker cube has a menu that lets you scale the building associated with the marker and offers context information on the building (such as the living area or the number of floors). This is called the context menu.

Table Menu:

- On the right side of the workspace is a menu dedicated to saving and loading scenes. This is called the table menu.

Match Mode:

- When loading a previously saved scene, the match mode is entered. This mode is necessary to ensure that each virtual marker cube can be associated with a marker cube in the real world. In order to connect a virtual marker with a real one, position the real marker cube so that its position coincides with the virtual marker cube that is pulsating in red. As soon as the color of the virtual marker turns to green, the position is close enough and you may move on to the next marker (if there are more). Once this has been done for all markers the “Apply”-button appears on the table menu, which lets the user exit the match mode.

File overview of Unity assets:

1. Plug-In assets

/Assets/LeapMotion	Leap Motion core assets
/Assets/SteamVR	SteamVR core functionality necessary to run the HTC Vive as part of the Unity simulation

2. Script assets

/Assets/ObjectMenu/Scripts

ContextMenu.cs	Fills the object menu canvas with informations
contextMenuTrigger.cs	Handles the user input for the context menu
markerScale.cs	Changes the size of the cube when the context menu handlers are moved
MatchMode.cs	Controls the MatchMode of the markers
MatchModeReady.cs	Is attached to markers at the MatchMode, tells the system if a tcp-controlled marker is set correctly onto a loaded marker position
StopMatchMode.cs	Tells the system the status of the 4 triggers inside every cube while the MatchMode is running

/Assets/Skripts/

ControllerPos.cs	Handles the Vive controller button input
Marker.cs	Getter and setter for the markers
OutputNormalizedControllerPos.cs	Calculates the normalized position of the Vive controller
printFPS.cs	Debug script, shows FPS on an GUI
readInNetworkData.cs	Handles the incoming TCP datastream

setupScene.cs	Dynamic scene setup based on the calibration data
TableCalibration.cs	Calibrates the table in unity
DataHandler.cs	Handles the data copy process to tcp-controlled markers during the MatchMode
LeapHandCalib.cs	If there is an offset between the leap tracked hand and the plane, the offset is removed by this script

/Assets/Menus/

	Contains the StartMenu GUI scenes
/scripts/	Contains scripts for the startMenu GUI

/Assets/TableMenu/scripts

FileBrowser.cs	Contains functions for loading files
open.cs	Loads scenes from the saved xml documents
save.cs	Saves scenes as XML documents in /Resources/saves/
Timeline.cs	Handles the filebrowser ("timeline") in the table menu
TableMenuTrigger.cs	Handles the user input for the table menu

File overview Tracking:

1. External Sources

OpenCV Library	Image Processing Library
Aruco Library	part of OpenCV Library
uEye SDK	SDK runs, actuates and configures the uEye Camera
TCP Lib	Winsock

2. Framework

Classes

TCP.cpp	configures and runs TCP data connection, normalizes the marker pixel coordinates in order to the workspace size
uEye_input.cpp	configures the uEye parameters and runs the capturing
PoseEstimation.cpp	runs the OpenCV camera calibration and saves or loads the camera matrix and the distortion coefficients files
PlaneAndAffineCalibration.cpp	generates the workspace in picture space; calculates affine transformation from uEye camera space and unity world
CoordsTransformation2Unity.cpp	transforms marker center in uEye camera space to unity with affine Transform mat
Calibration.cpp	offers access to the Camera Calibration and Plane Calibration, offers all required attributes of both calibrations
MarkerManagement.cpp	organizes the current Markers and actualises the current position and angle; either matches recently detected Markers with current Markers or register new Marker; delete Markers if they out of workspace
MarkerDetection.cpp	detects the green rectangles and the

	aruco markers
IdMapping.cpp	offers all requested functions to compute matches of recently detected Markers and current Markers
Marker.cpp	Objectclass with all marker attributes
CreateCharucoBoard.cpp	generates CharucoBoard for Camera Calibration
ArucoCodeGenerator.cpp	generates new Aruco Codes to print

Literatur

- [1] Unity. https://www.1stvision.com/cameras/IDS/dataman/uEyeLE_Brochure_english_ND.pdf. Aufgerufen: 13. März 2017.
 - [2] Unity. <https://de.ids-imaging.com/ids-software-suite.html>. Aufgerufen: 13. März 2017.
 - [3] RFC 793: Transmission Control Protocol. <https://rfc-editor.org/rfc/rfc793.txt>, 1981. Aufgerufen: 21. März 2017.
 - [4] Iñigo Barandiaran, Céline Paloc, and Manuel Graña. Real-time optical markerless tracking for augmented reality applications. *Journal of Real-Time Image Processing*, 5(2):129–138, 2010.
 - [5] G. Blasko and P. Fua. Real-time 3d object recognition for automatic tracker initialization. In *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, pages 175–176, 2001.
 - [6] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and François Yergeau. Extensible markup language (xml). *World Wide Web Consortium Recommendation REC-xml-19980210*. <http://www.w3.org/TR/1998/REC-xml-19980210>, 16:16, 1998.
 - [7] Steve Bryson. Approaches to the successful design and implementation of vr applications. *Virtual reality applications*, pages 3–15, 1995.
 - [8] Chi-Cheng P Chu, Tushar H Dani, and Rajit Gadh. Multi-sensory user interface for a virtual-reality-based computeraided design system. *Computer-Aided Design*, 29(10):709–725, 1997.
 - [9] A. I. Comport, E. Marchand, M. Pressigout, and F. Chaumette. Real-time markerless tracking for augmented reality: the virtual visual servoing framework. *IEEE Transactions on Visualization and Computer Graphics*, 12(4):615–628, July 2006.
 - [10] Creative. Creative Senz3D, Tiefen- und Gestenerkennungskamera für PCs . <http://de.creative.com/p/web-cameras/creative-senz3d>. Aufgerufen: 19. März 2017.
 - [11] I. Culjak, D. Abram, T. Pribanic, H. Dzapo, and M. Cifrek. A brief introduction to opencv. In *2012 Proceedings of the 35th International Convention MIPRO*, pages 1725–1730, May 2012.
 - [12] Dima Damen, Pished Bunnun, Andrew Calway, and Walterio Mayol-cuevas. Realtime learning and detection of 3d texture-less objects: A scalable approach. In *in British Machine Vision Conference (BMVC)*, 2012.
 - [13] O. Faugeras. *Three-dimensional Computer Vision: A Geometric Viewpoint*. Artificial intelligence. MIT Press, 1993.
-

- [14] M. Fiala. Designing highly reliable fiducial markers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(7):1317–1324, July 2010.
 - [15] Daniel Flohr and Jan Fischer. A lightweight id-based extension for marker tracking systems. In *Eurographics Symposium on Virtual Environments (EG-VE) Short Paper Proceedings*, pages 59–64, 2007.
 - [16] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280 – 2292, 2014.
 - [17] S. Garrido-Jurado, R. Mu noz Salinas, F.J. Madrid-Cuevas, and R. Medina-Carnicer. Generation of fiducial marker dictionaries using mixed integer linear programming. *Pattern Recognition*, 51:481 – 491, 2016.
 - [18] Google. Cardboard. <https://vr.google.com/cardboard/>. Aufgerufen: 20. März 2017.
 - [19] John Haas. *A History of the Unity Game Engine*. PhD thesis, Worcester Polytechnic Institute, 2014.
 - [20] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, May 2012.
 - [21] HTC. HTC Vive. <https://www.vive.com/>. Aufgerufen: 30. November 2016.
 - [22] HTC. HTC Vive – Für Vive geeignete Computer. <https://www.vive.com/de/ready/>. Aufgerufen: 18. März 2017.
 - [23] ITU-T. Data Networks and Open System Communication. Open Systems Interconnection - Model and Notation. <http://handle.itu.int/11.1002/1000/2820>, 1994. Aufgerufen: 21. März 2017.
 - [24] H. Kato and M. Billinghurst. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In *Augmented Reality, 1999. (IWAR '99) Proceedings. 2nd IEEE and ACM International Workshop on*, pages 85–94, 1999.
 - [25] Leap Motion. Leap Motion. <https://www.leapmotion.com/>. Aufgerufen: 30. November 2016.
 - [26] Leap Motion. Leap Motion Blog. <http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/>. Aufgerufen: 2. Januar 2017.
 - [27] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
-

- [28] Andreas Meisel. *3D-Bildverarbeitung für feste und bewegte Kameras*. PhD thesis, Braunschweig [u.a.], 1994. Zugl.: Aachen, Techn. Hochsch., Diss., 1993 u.d.T.: Meisel, Andreas: 3D-Bildverarbeitung für feste und bewegte Kameras auf photogrammetrischer Basis.
- [29] Microsoft. Introducing Visual Studio. [https://msdn.microsoft.com/en-us/library/fx6bk1f4\(v=vs.90\).aspx](https://msdn.microsoft.com/en-us/library/fx6bk1f4(v=vs.90).aspx). Aufgerufen: 18. März 2017.
- [30] Mobile World Congress. Mobile World Congress. <https://www.mobileworldcongress.com/>. Aufgerufen: 30. November 2016.
- [31] Mozilla. Introducing the WebVR 1.0 API Proposal. <https://hacks.mozilla.org/2016/03/introducing-the-webvr-1-0-api-proposal/>. Aufgerufen: 20. März 2017.
- [32] Oculus. Oculus – Unity Intro. <https://developer3.oculus.com/documentation/game-engines/latest/concepts/unity-intro/>. Aufgerufen: 14. März 2017.
- [33] OpenCV. Camera Calibration and 3D Reconstruction. http://docs.opencv.org/3.1.0/d9/d0c/group__calib3d.html. Aufgerufen: 18. März 2017.
- [34] OpenCV. Detection of ArUco Markers. http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html. Aufgerufen: 18. März 2017.
- [35] Ovrvision Pro. Ovrvision Pro. <http://ovrvision.com/>. Aufgerufen: 30. November 2016.
- [36] Ovrvision Pro. Ovrvision Pro – Informationen für Entwickler. <http://ovrvision.com/setup-en/>. Aufgerufen: 14. März 2017.
- [37] Ovrvision Pro. Ovrvision Pro – Produktdetails. <http://ovrvision.com/product-en/>. Aufgerufen: 9. März 2017.
- [38] Y. Park, V. Lepetit, and W. Woo. Texture-less object tracking with online training using an rgb-d camera. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 121–126, Oct 2011.
- [39] J. Postel. RFC 768. User Datagram Protocol. <https://tools.ietf.org/html/rfc768>, August 1980. Aufgerufen: 21. März 2017.
- [40] Bob Quinn. *Windows sockets network programming*. Addison-Wesley Longman Publishing Co., Inc., 1998.
- [41] T. Rahman and N. Krouglisoff. An efficient camera calibration technique offering robustness and accuracy over a wide range of lens distortion. *IEEE Transactions on Image Processing*, 21(2):626–637, Feb 2012.
- [42] Samsung. Samsung Explores the World of Mobile Virtual Reality with Gear VR. <http://www.samsungmobilepress.com/press/Samsung-Explores-the-World-of-Mobile-Virtual-Reality-with-Gear-VR?2014-09-03>. Aufgerufen: 20. März 2017.
-

- [43] Unity Technologies. Unity. <https://unity3d.com/de>. Aufgerufen: 8. März 2017.
 - [44] Unity Technologies. Unity – Multiplatform. <https://unity3d.com/unity/multiplatform>. Aufgerufen: 14. März 2017.
 - [45] Unity Technologies. Unity – Public Relations. <https://unity3d.com/public-relations>. Aufgerufen: 14. März 2017.
 - [46] Unity Technologies. Unity – VR Overview. <https://unity3d.com/de/learn/tutorials/topics/virtual-reality/vr-overview>. Aufgerufen: 14. März 2017.
 - [47] L. Vacchetti, V. Lepetit, and P. Fua. Stable real-time 3d tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1385–1391, Oct 2004.
 - [48] Valve. Valve Software. <http://www.valvesoftware.com/>. Aufgerufen: 30. November 2016.
 - [49] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, and Dieter Schmalstieg. Real-time detection and tracking for augmented reality on mobile phones. *IEEE Transactions on Visualization and Computer Graphics*, 16(3):355–368, May 2010.
 - [50] Daniel Wagner and Dieter Schmalstieg. Artoolkitplus for pose tracking on mobile devices, 2007.
 - [51] Li-Chen Wu, I-Chen Lin, and Ming-Han Tsai. Augmented reality instruction for object assembly based on markerless tracking. In *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D ’16, pages 95–102, New York, NY, USA, 2016. ACM.
 - [52] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, Nov 2000.
 - [53] H. Álvarez, I. Aguinaga, and D. Borro. Providing guidance for maintenance operations using automatic markerless augmented reality system. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 181–190, Oct 2011.
 - [54] Éric Marchand and François Chaumette. Feature tracking for visual servoing purposes. *Robotics and Autonomous Systems*, 52(1):53 – 70, 2005. Advances in Robot Vision.
-