



# Projektdokumentation

---

**MArC**  
Mixed Reality Architecture Composer

---

von

Laura Anger (Matrikelnr. 11086356)  
Vera Brockmeyer (Matrikelnr. 11077082)  
Paul Berning (Matrikelnr. xxxxxxxxxxxx)  
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

<b>1 Einleitung</b>	<b>5</b>
1.1 Motivation . . . . .	5
1.2 Anwendungskontext . . . . .	5
1.3 Projektziel . . . . .	5
<b>2 Grundlagen</b>	<b>6</b>
2.1 Architektur VR und MR Anwendungen . . . . .	6
2.2 Haptische Interaktionsmethoden in VR Umgebungen . . . . .	6
2.3 Handtracking Interaktionsmethoden . . . . .	6
2.4 Menüführung in VR Umgebungen . . . . .	6
2.5 Datenaustausch über internes Netzwerk . . . . .	6
2.6 Green-Keying . . . . .	6
2.7 Marker Tracking . . . . .	6
<b>3 Materialien</b>	<b>8</b>
3.1 Hardware . . . . .	8
3.1.1 Computer zur Ausführung der Unity-Simulation . . . . .	8
3.1.2 Computer zur Ausführung der Tracking-Anwendung . . . . .	9
3.1.3 HTC Vive . . . . .	9
3.1.4 IDS uEye 164LE-C . . . . .	10
3.1.5 Leap Motion . . . . .	11
3.1.6 Leap Motion SDK . . . . .	11
3.1.7 ArUco Marker . . . . .	11
3.1.8 Würfel Marker . . . . .	12
3.2 Obsolete Hardware . . . . .	13
3.2.1 Ovrvision Pro . . . . .	14
3.2.2 Webcam . . . . .	14
3.3 Software . . . . .	15
3.3.1 Unity . . . . .	15
3.3.2 Visual Studio 2015 . . . . .	15
3.3.3 OpenCV . . . . .	15
3.3.4 Orion Beta Software . . . . .	16

---

3.3.5	Steam VR . . . . .	16
3.3.6	Winsoc . . . . .	16
<b>4</b>	<b>System</b>	<b>17</b>
4.1	Aufbau . . . . .	17
4.2	Systemvoraussetzung . . . . .	18
4.3	Starten des Systems . . . . .	18
4.4	Menüführung . . . . .	18
4.4.1	Menüs . . . . .	18
4.4.2	Ablauf der Menüführung . . . . .	20
4.5	Toogle . . . . .	21
4.6	Kontex Menü . . . . .	23
4.6.1	Menüführung . . . . .	23
4.6.2	Architektur Berechnung . . . . .	23
4.7	Table Menü . . . . .	23
4.7.1	Szenen Management . . . . .	23
4.7.2	Match Modus . . . . .	23
4.8	Tracking Algorithmus . . . . .	23
4.8.1	Green Keying . . . . .	23
4.8.2	ArUco Marker Tracking . . . . .	23
4.8.3	Marker Objekt . . . . .	24
4.8.4	Zuordnung der Identitäten . . . . .	24
4.9	Kalibrierung . . . . .	24
4.9.1	Kamerakalibrierung . . . . .	24
4.9.2	Kalibrierung des Arbeitsbereichs . . . . .	27
4.9.3	Kalibrierungsfehler . . . . .	28
4.9.4	Mögliche Fehlerquellen . . . . .	28
<b>5</b>	<b>Ausblick</b>	<b>28</b>
5.1	Trackingmethoden . . . . .	28
5.2	AR Erweiterung mit der Webcam . . . . .	28
5.3	Daten als Excel Tabelle . . . . .	28
5.4	Validierung . . . . .	28

---

---

<b>6 Projektmanagement</b>	<b>28</b>
6.1 Zeitplanung . . . . .	28
6.2 Gant Chart . . . . .	28
6.3 Planungsmethoden . . . . .	29
6.4 Reflexion . . . . .	29
6.4.1 Kommuniktion im Team und nach Außen . . . . .	29
6.4.2 Probleme . . . . .	29
<b>7 Zusammenfassung</b>	<b>29</b>
<b>8 Steckbrief</b>	<b>29</b>

# 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 VR und MR Anwendungen

Lukas

### 2.2 Haptische Interaktionsmethoden in VR Umgebungen

Laura

### 2.3 Handtracking Interaktionsmethoden

Paul

### 2.4 Menüführung in VR Umgebungen

Lukas

### 2.5 Datenaustausch über internes Netzwerk

Laura

### 2.6 Green-Keying

Vera

### 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 muss diese physische Objekt mit einer oder mehreren Videokamera aufgenommen werden. In den resultierenden Bildsequenzen können die physischen Objekte anhand von festgelegten Merkmalen erkannt werden. Diese Merkmale können sowohl natürlicher Art sein oder in Form von 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, Kan teninformationen oder bekannte Keypoints. Einige Autoren [3][32][5][15] arbeiten mit Keypoints, 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. Andere verwenden sehr rechenintensive Kantenerkennungsalgorithmen, wie den *Moving Edges Algorithm* [36]. Eine weitere Weiterentwicklung der Kantenbasierten Methoden ist das Modelbased Tracking bei dem die detektierten Kanten eines möglichen Kandidaten mit dem 3D-Kantenmodellen des zu verfolgenden Objektes abgeglichen wird [30][35][33][4].

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 [3].

sogenannte Marker benötigt. Mit Hilfe dessen können die aktuellen Positionen in der Realität in einem Kamerabild erfasst werden und in den erforderlichen 3D-Raum transformiert werden. Diese Marker bestehen aus eindeutigen Codes, welche in der Regel in Form von binären Muster verwendet werden. In Abbildung 1 sind vielfältige Beispiele von binären Codes zu sehen, die zum Marker Tracking verwendet werden. Jedem Code wird eine unverwechselbaren ID zu geordnet um Überschneidungen zu vermeiden und jeden Marker auch nach längerer Verdeckung eindeutig identifizieren und orten zu können.

Zur eindeutigen Identifizierung von den Codes sind vielfältige Schritte notwendig. Zunächst muss das Muster innerhalb eines Bildes oder Videos erkannt, geortet und verifiziert werden. Im Anschluss wird das gültige Muster der entsprechenden Identifikation (ID) zu geordnet sowie die Orientierung des Markers im Kameraraum berechnet. Diese umfangreichen Ermittlungsprozesse werden in der Regel von entsprechenden Bibliotheken zur Verfügung gestellt, wie zum Beispiel das *ArUco* Modul (siehe Kapitel 3.1.7) aus dem Bildverarbeitungsbibliothek *OpenCV* (siehe Kapitel 3.3.3).

---

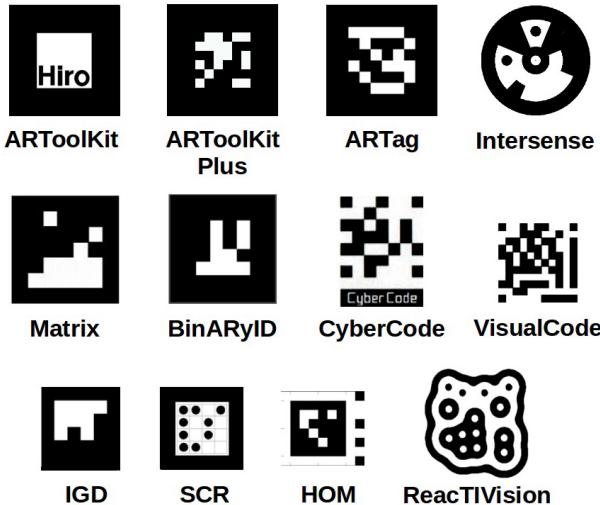


Abbildung 1: Diverse Binäre Muster die als Code für Marker Tracking verwendet werden. Quelle: [8]

## 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. 4).

### 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 [26] 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 1 zur Verfügung gestellt:

Beim nächsten Mal in der TH nachschauen...

Die Hard- und Software-Voraussetzungen für die Ausführung der Unity-Anwendung

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

Tabelle 1: Auszug aus dem technischen Datenblatt des XXXX.

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	1× HDMI 1.4-Anschluss oder DisplayPort 1.2
USB	1× USB 2.0-Anschluss
Betriebssystem	Windows 7 SP1, Windows 8.1 oder Windows 10

Tabelle 2: HTC Vive Systemvoraussetzungen. [12]

in Verbindung mit der HTC Vive, welche in Tabelle 2 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 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 gezwungenermaßen 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 3 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[31] produziert wird. Vorgestellt wurde dieses am 1. März 2015 im Vorfeld der Mobile World Congress [18].

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 3: Auszug aus dem technischen Datenblatt des *Acer E5-571G-795A*.

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

Tabelle 4: Parametrisierung der uEye Kamera für das Tracking.

Die Auflösung des Displays beträgt insgesamt  $2160 \times 1200$  Pixel, was  $1080 \times 1200$  Pixeln pro Auge entspricht. Die Brille bietet ein Sichtfeld von bis zu  $110^\circ$  bei einer Bildwiederholrate von  $90\text{ Hz}$  [1]. 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  $1280 \times 1024$  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  $37\mu\text{s}$  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 mit den Parametern aus Tabelle 4 initialisiert und steuert.

### 3.1.5 Leap Motion

Paul

Bei der Leap Motion [13] 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 5 erahnt werden kann, besteht das Gerät im wesentlichen aus zwei integrierten weitwinkel Kameras und drei einfachen Infrarot LEDs. Die LEDs haben jeweils eine Wellenlänge von  $850\text{ nm}$ . Der durch die beiden Kameras aufgespannte Interaktionsraum der Leap Motion ähnelt einer umgedrehten Pyramide, mit einem Flächeninhalt von knapp  $243\text{ cm}^2$ .

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  $60\text{ cm}$  auf  $80\text{ cm}$ . 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.

### 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 [8]. Die Marker bestehen ähnlich wie QR-Codes aus einer zweidimensionalen Matrix, mit schwarzen oder weißen Feldern, welche die kodierten Daten binär, wie in Abbildung 2, darstellen. Weiterführend kann die Anzahl der Bits variabel gewählt werden (siehe Abbildung 3), 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 [9]. Diese Dictionarys bestehen aus Markern mit gleicher Bit-Anzahl und sind zusätzlich auf eine maximalen Anzahl von IDs begrenzt um eine möglichst hohe Performanz zu gewährleisten.

---

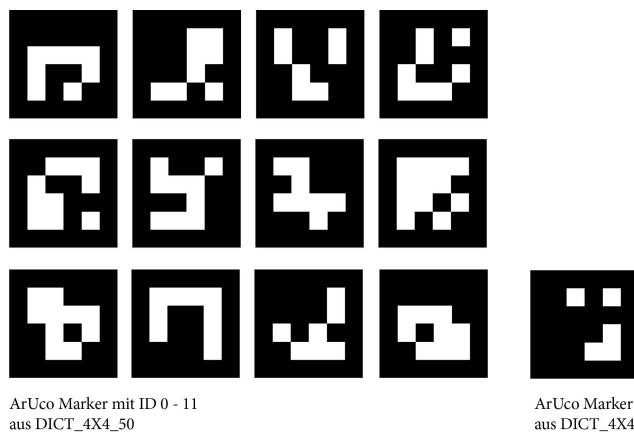


Abbildung 2: Alle genutzten 16 bit ArUco Marker des Prototypen. Links die zwölf IDs der Würfel Marker und rechts die ID, welche zur Kalibrierung benötigt wird. Die maximale Anzahl der Marker ist auf 50 begrenzt.

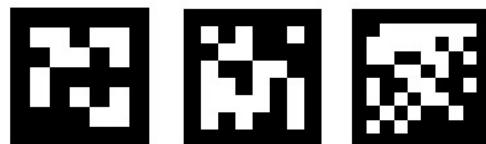


Abbildung 3: ArUco Marker mit unterschiedlicher Bitgröße. Von Links nach Rechts:  $n = 5$ ,  $n = 6$  und  $n = 8$ . Quelle: [8]

### 3.1.8 Würfel Marker

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 designet. 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 4.8) aus den Bildern der uEye Kamera ermittelt. Alle zwölf Marker stimmen mit der Form und Farbe, sowie Material und Oberflächenbeschaffenheit aus Abbildung 4 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 [21] generiert.

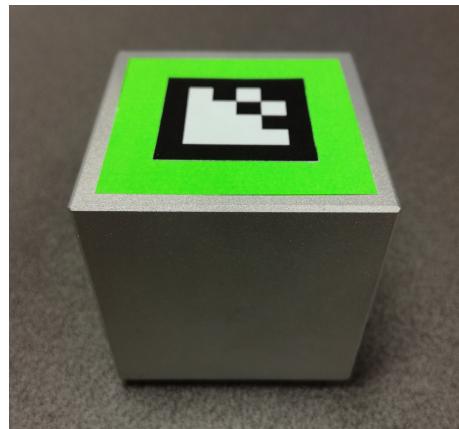


Abbildung 4: Würfel Marker mit grüner Fläche und einem ArUco Marker, die mittig auf den Aluminumwürfel aufgebracht sind.

### 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. [24]

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 [22]. Sie ist kompatibel mit Programmen wie Unity, welches für das Projekt benutzt wurde und in ?? beschrieben wird.

Die Bildmodi der Kamera sind in Tabelle 5 aufgeführt.



Abbildung 5: Explosionszeichnung der Leap Motion.[14]

### 3.2.2 Webcam

Vera

### 3.3 Software

Vera

#### 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 [10] und wird von Unity Technologies [26] 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 [28].

Unity bietet eine sehr breite Plattformunterstützung [27] und erlaubt ebenso die Entwicklung für Head-Mounted-Displays, wie etwa die Oculus Rift [19][29] oder auch die in diesem Projekt verwendete HTC Vive [29].

Die zu Beginn des Projekts verwendete Stereo-Kamera Ovrvision Pro stellt ein Software-Development-Kit (SDK) für Unity (Version 5) zur Verfügung [23]. 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 [17] 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 [6].

Für dieses System ist vor allem das Modul `calib3d` [20] und das extra Modul `aruco` [21] verwendet. Das erste Modul `calib3d` bietet alle notwendigen Funktionen zur

---

Erstellung, Verwendung und Weiterverarbeitung von intrinsischen und extrinsischen Kamerakalibrierungen an (siehe Kapitel 4.9). 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 Winsoc

Laura und Lukas

## 4 System

Lukas

Die Benutzung von *MArC* ist in dem Programm mitgelieferten ReadMe-Datei beschrieben. Darin wird erklärt, welche Hard- und Software komponenten erforderlich sind, wie das System gestartet und kalibriert wird. Des weiteren enthält die ReadMe eine Übersicht über die enthaltenen Quellcode-Dateien.

### 4.1 Aufbau

Lukas und Vera

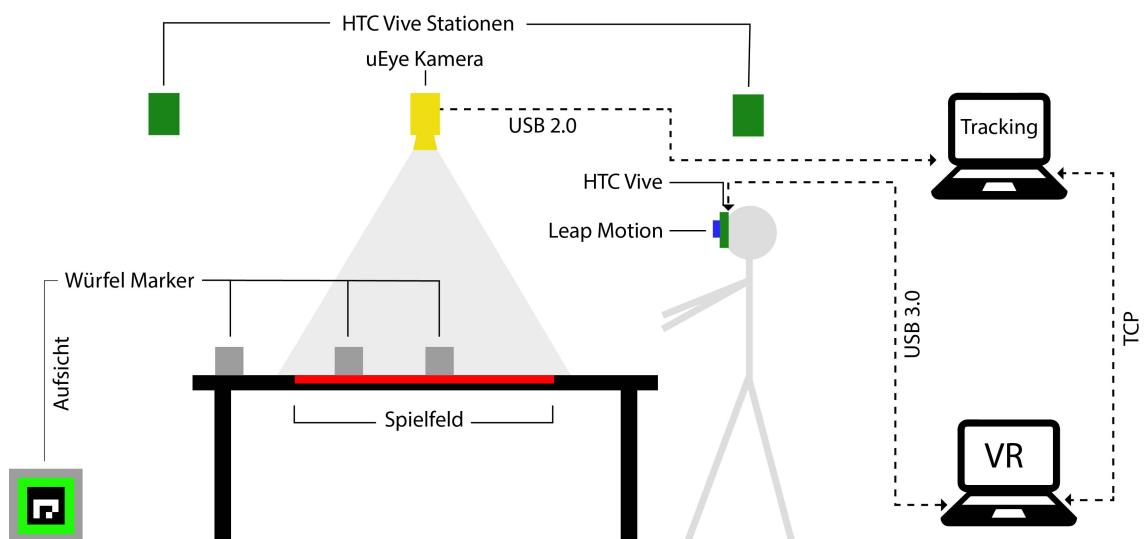


Abbildung 6: Aufbau des *MArC* System.

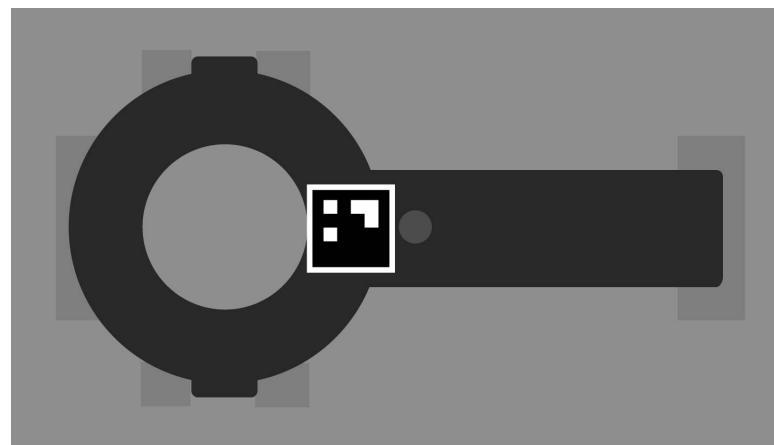


Abbildung 7: Kalibrierungscontroller des *MArC* System mit ArUco Marker.

## 4.2 Systemvoraussetzung

Lukas

## 4.3 Starten des Systems

Lukas

Nachdem sichergestellt wurde, dass alle in der ReadMe-Datei beschriebenen Voraussetzungen bestehen, kann das System gestartet werden, indem zunächst die Tracking-Anwendung (auf dem einen Computer) und anschließend die aus Unity heraus erstellte Anwendung (auf dem anderen Computer) gestartet wird. Auf letzterem Computer beginnt darauffolgend die Menüführung, welche in [4.4](#) beschrieben ist.

## 4.4 Menüführung

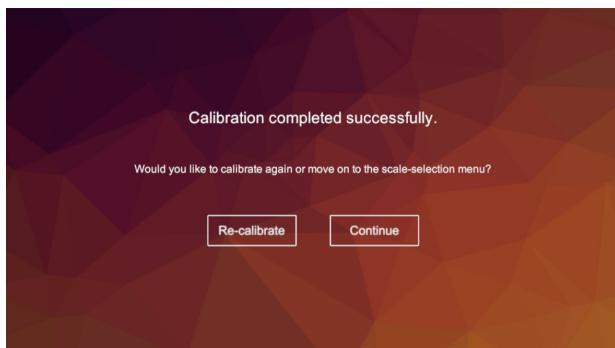
Lukas

Die Menüführung dient dazu, den Benutzer durch alle notwendigen Schritte zu leiten, die vor dem Starten der eigentlichen Simulation erforderlich sind. Im nachfolgenden Abschnitt [4.4.1](#) werden alle verfügbaren Menüs der Anwendung aufgelistet und kurz beschrieben, während im Abschnitt [4.4.2](#) der Ablauf der Menüführung erläutert wird.

### 4.4.1 Menüs

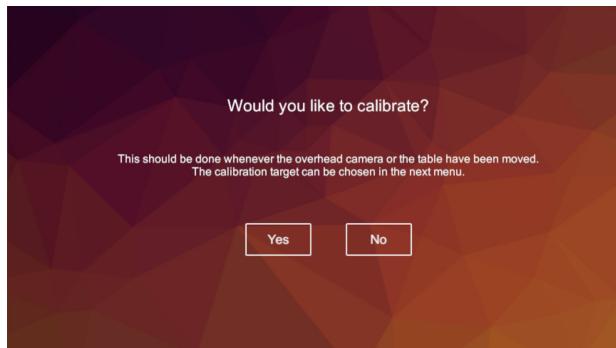
Laura

Die folgenden Menüs sind Bestandteil der Menüführung:

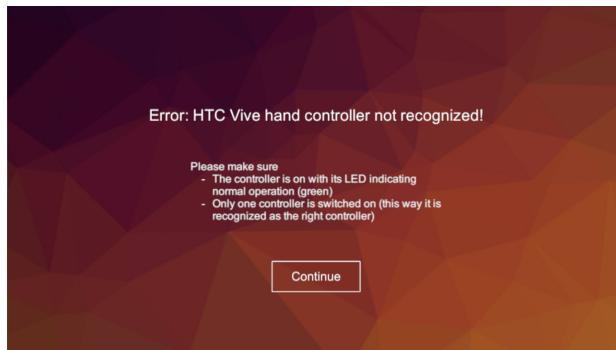


#### CalibDone:

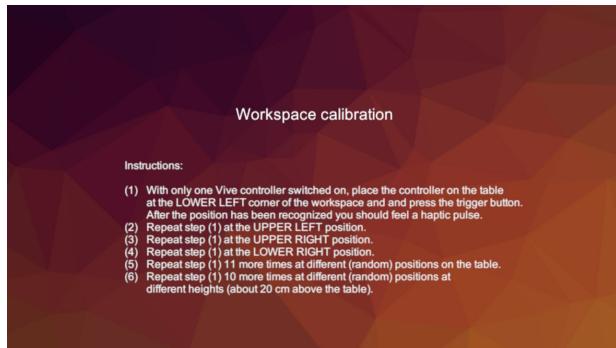
Wird aufgerufen, wenn die Kalibrierung des Arbeitsbereichs abgeschlossen ist. Es informiert den Benutzer, dass die Kalibrierung erfolgreich war und der Vorgang fortgesetzt werden kann.

**CalibrateOrNot:**

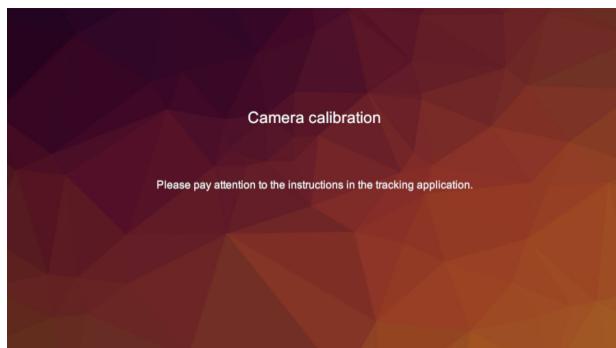
Erscheint nach dem Verlassen des Welcome-Menüs und erlaubt dem Benutzer eine Kalibrierung durchzuführen oder eine bereits durchgeführte Kalibrierung zu laden.

**ControllerNotFound:**

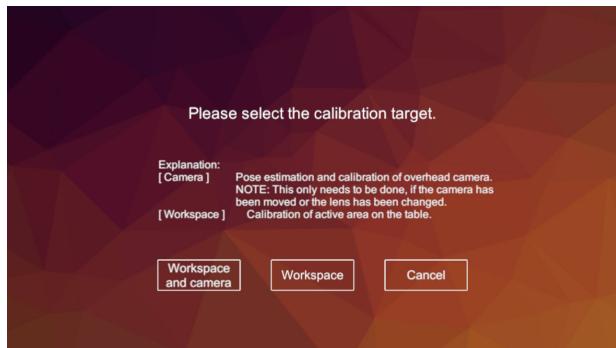
Warnt den Benutzer nach dem Starten der Kalibrierung, dass der HTC Vive Controller, welcher für die Kalibrierung benötigt wird, nicht eingeschaltet ist. Während das Menü angezeigt wird, kann der Benutzer den Controller einschalten und anschließend auf **Continue** klicken.

**doPlaneCalibInVS:**

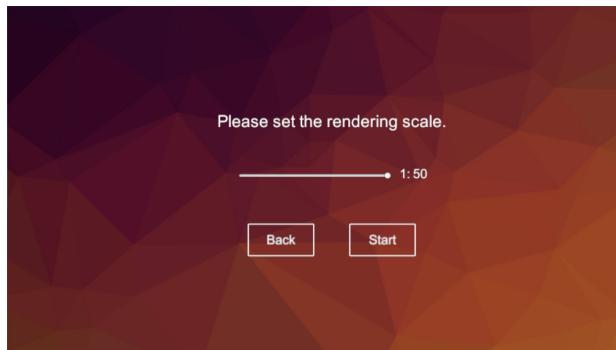
Dient dem Benutzer als Anleitung für die Durchführung der Arbeitsbereich-Kalibrierung. Diese wird in [4.9.2](#) genauer beschrieben.

**doPoseCalibInVS:**

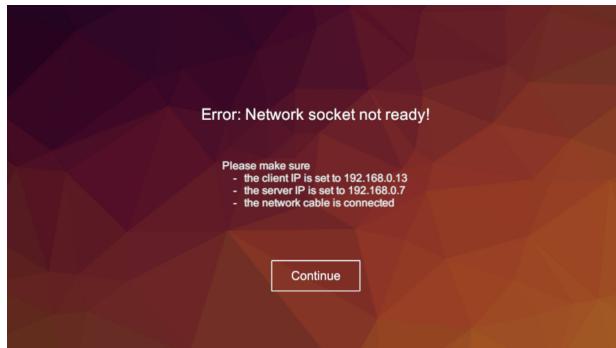
Dient dem Benutzer als Anleitung für die Durchführung der Kamera-Kalibrierung. Diese wird in [4.9.1](#) genauer beschrieben.

**SelectCalibrationTarget:**

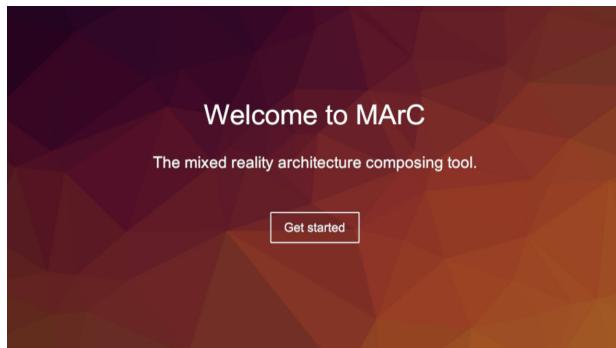
Erlaubt die Auswahl der Art der Kalibrierung. Es kann hier entweder nur der Arbeitsbereich oder sowohl der Arbeitsbereich, als auch die Kamera kalibriert werden. Die Kalibrierung ist näher in [4.9](#) beschrieben.

**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.

#### 4.4.2 Ablauf der Menüführung

Lukas

Der Ablauf der Menüführung von *MArC* ist in Abbildung 8 dargestellt. Die einzelnen Menüs sind bereits in 4.4.1 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 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 *Workspace* 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.

## 4.5 Toogle

Paul

---

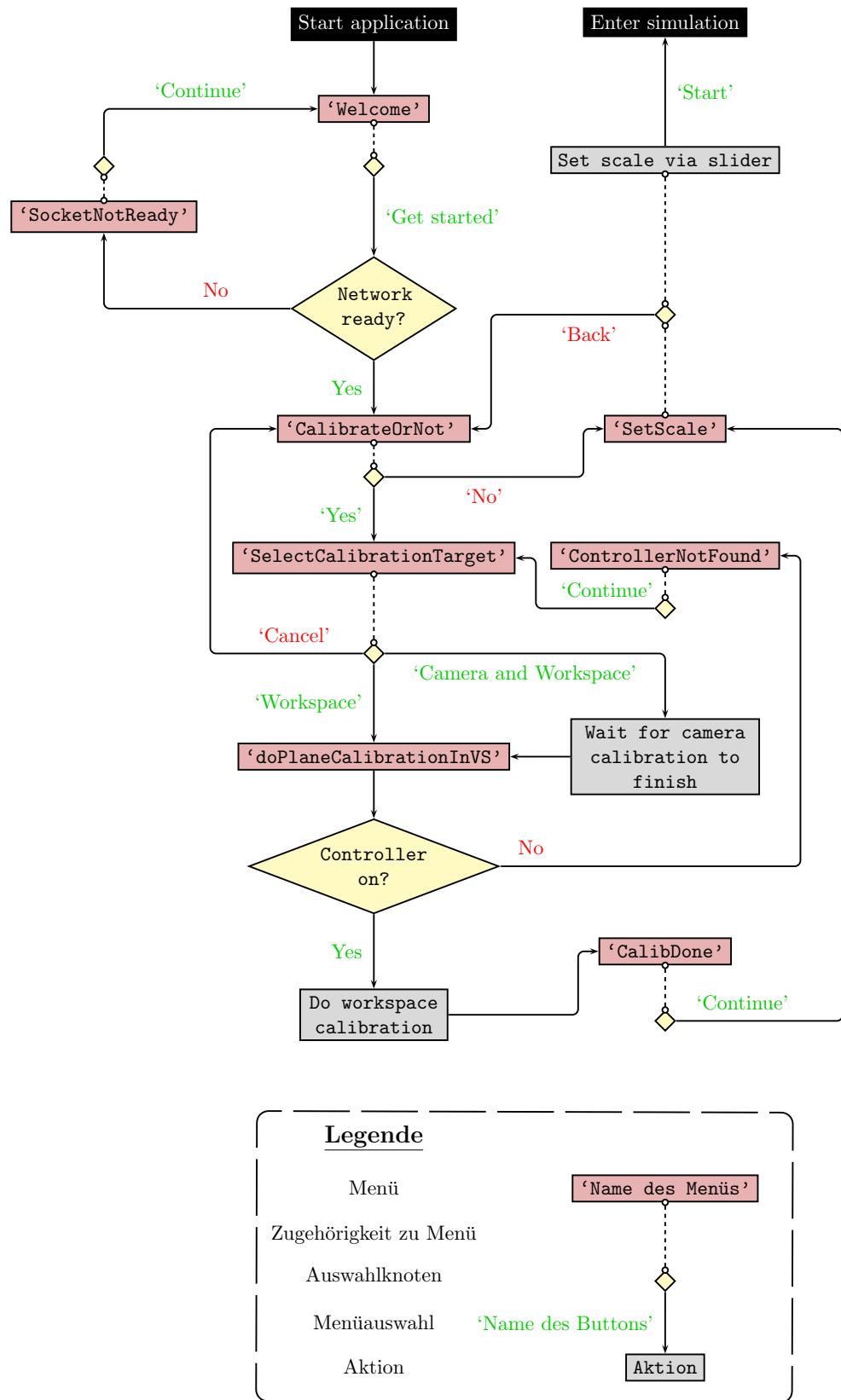


Abbildung 8: Flussdiagramm der Menüführung.

## 4.6 Kontex Menü

Laura und Lukas

### 4.6.1 Menüführung

Laura

### 4.6.2 Architektur Berechnung

Laura

## 4.7 Table Menü

Paul

### 4.7.1 Szenen Management

Paul

### 4.7.2 Match Modus

Paul

## 4.8 Tracking Algorithmus

Vera

### 4.8.1 Green Keying

Vera

### 4.8.2 ArUco Marker Tracking

Vera

---

#### 4.8.3 Marker Objekt

Vera

#### 4.8.4 Zuordnung der Identitäten

Vera

### 4.9 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 [4.9.1](#) 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 [4.9.2](#) 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 [4.9.3](#) und in [4.9.4](#) beschrieben.

#### 4.9.1 Kamerakalibrierung

Laura

Es gibt viele verschiedene wissenschaftliche Ausführungen über die Durchführung einer Kamerakalibrierung, wie z.B. [25], [34] und [7]. Dabei unterscheidet man häufig zwischen automatischen und manuellen Kalibrierungen.

In Abbildung 9 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.

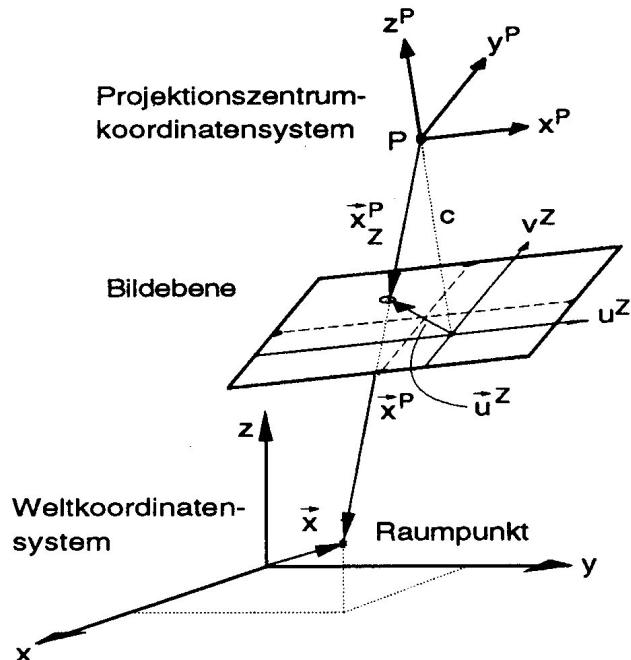


Abbildung 9: Lage des Kamerakoordinatensystems in Bezug auf Projektionsebene und Weltkoordinatensystem. [16]

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 6 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 6 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  $\varphi$  den Drehwinkel um die  $x^P$ -Achse und  $\vartheta$  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 6: Parameter und Berechnungsschritte der Kamera Kalibrierung.[\[16\]](#)

$$R_v = \begin{pmatrix} r_{v11} & r_{v12} & r_{v13} \\ r_{v21} & r_{v22} & r_{v23} \\ r_{v31} & r_{v32} & r_{v33} \end{pmatrix} = \begin{pmatrix} \cos\vartheta & \sin\vartheta \sin\varphi & -\sin\vartheta \cos\varphi \\ 0 & \cos\varphi & \sin\varphi \\ \sin\vartheta & -\cos\vartheta \sin\varphi & \cos\vartheta \cos\varphi \end{pmatrix} \quad R_v \in \mathbb{R}^{3x3} \quad (1)$$

Die negative verkippte Kamerakonstante  $-c^V$ , die in Tabelle 6 aufgeführt ist, berechnet sich wie in [\[16\]](#) beschrieben nach der Formel:

$$-c^V = -\frac{c + u^V \cdot r_{v13} + v^V \cdot r_{v23}}{r_{v33}} \quad (2)$$

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. **Begründung??**

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  $M_{intrinsisch}$  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.

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

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 4 beschrieben werden kann, reicht für die Translation ein Vektor, wie  $t$  aus Formel 5 aus.

$$R = R_\gamma \ R_\beta \ R_\alpha = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \quad R \in \mathbb{R}^{3x3} \quad (4)$$

$$t = (t_x \ t_y \ t_z)^T \quad t \in \mathbb{R}^{3x1} \quad (5)$$

Diese beiden Transformationen können wie in Formel 6 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  $R \in SO(3)$ .

$$M_{extrinsisch} = (R \ | \ t) \quad M_{extrinsisch} \in \mathbb{R}^{3x4} \quad (6)$$

Schritt 1 und 2 lassen sich vereinfachen, indem man die intrinsische Kameramatrix  $M_{intrinsisch}$  und die extrinsische Kameramatrix  $M_{extrinsisch}$  nach der Formel 7 zu einer Matrix  $M$  zusammenfasst.

$$M = M_{intrinsisch} \cdot M_{extrinsisch} \quad M \in \mathbb{R}^{3x4} \quad (7)$$

Die Umrechnung von Kamerakoordinaten in Weltkoordinaten kann dann mit der invertierten Matrix  $M$ , wie in Formel 8 berechnet werden.

$$\vec{x} = M^{-1} \cdot \vec{u} \quad (8)$$

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 Transformation wird in Kapitel 4.9.2 erläutert.

#### 4.9.2 Kalibrierung des Arbeitsbereichs

Laura

#### 4.9.3 Kalibrierungfehler

Laura

#### 4.9.4 Mögliche Fehlerquellen

Laura

### 5 Ausblick

#### 5.1 Trackingmethoden

Vera

#### 5.2 AR Erweiterung mit der Webcam

???

#### 5.3 Daten als Excel Tabelle

???

#### 5.4 Validierung

???

### 6 Projektmanagement

#### 6.1 Zeitplanung

Paul

#### 6.2 Gant Chart

Paul

---

### 6.3 Planungsmethoden

Paul

### 6.4 Reflexion

#### 6.4.1 Kommuniktion im Team und nach Außen

Paul

#### 6.4.2 Probleme

Paul

## 7 Zusammenfassung

Lukas

## 8 Steckbrief

Vera

---

## Literatur

- [1] Unity. [https://www.1stvision.com/cameras/IDS/dataman/uEyeLE\\_Brochure\\_english\\_ND.pdf](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] 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.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] O. Faugeras. *Three-dimensional Computer Vision: A Geometric Viewpoint*. Artificial intelligence. MIT Press, 1993.
- [8] 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.
- [9] 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.
- [10] John Haas. *A History of the Unity Game Engine*. PhD thesis, Worcester Polytechnic Institute, 2014.
- [11] HTC. HTC Vive. <https://www.vive.com/>. Aufgerufen: 30. November 2016.
- [12] HTC. HTC Vive – Für Vive geeignete Computer. <https://www.vive.com/de/ready/>. Aufgerufen: 18. März 2017.
- [13] Leap Motion. Leap Motion. <https://www.leapmotion.com/>. Aufgerufen: 30. November 2016.
- [14] Leap Motion. Leap Motion Blog. <http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/>. Aufgerufen: 2. Januar 2017.

- [15] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.
  - [16] 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.
  - [17] 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.
  - [18] Mobile World Congress. Mobile World Congress. <https://www.mobileworldcongress.com/>. Aufgerufen: 30. November 2016.
  - [19] Oculus. Oculus – Unity Intro. <https://developer3.oculus.com/documentation/game-engines/latest/concepts/unity-intro/>. Aufgerufen: 14. März 2017.
  - [20] OpenCV. Camera Calibration and 3D Reconstruction. [http://docs.opencv.org/3.1.0/d9/d0c/group\\_\\_calib3d.html](http://docs.opencv.org/3.1.0/d9/d0c/group__calib3d.html). Aufgerufen: 18. März 2017.
  - [21] OpenCV. Detection of ArUco Markers. [http://docs.opencv.org/3.1.0/d5/dae/tutorial\\_aruco\\_detection.html](http://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html). Aufgerufen: 18. März 2017.
  - [22] Ovrvision Pro. Ovrvision Pro. <http://ovrvision.com/>. Aufgerufen: 30. November 2016.
  - [23] Ovrvision Pro. Ovrvision Pro – Informationen für Entwickler. <http://ovrvision.com/setup-en/>. Aufgerufen: 14. März 2017.
  - [24] Ovrvision Pro. Ovrvision Pro – Produktdetails. <http://ovrvision.com/product-en/>. Aufgerufen: 9. März 2017.
  - [25] T. Rahman and N. Krouglicof. 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.
  - [26] Unity Technologies. Unity. <https://unity3d.com/de>. Aufgerufen: 8. März 2017.
  - [27] Unity Technologies. Unity – Multiplatform. <https://unity3d.com/unity/multiplatform>. Aufgerufen: 14. März 2017.
  - [28] Unity Technologies. Unity – Public Relations. <https://unity3d.com/public-relations>. Aufgerufen: 14. März 2017.
  - [29] Unity Technologies. Unity – VR Overview. <https://unity3d.com/de/learn/tutorials/topics/virtual-reality/vr-overview>. Aufgerufen: 14. März 2017.
-

- [30] 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.
- [31] Valve. Valve Software. <http://www.valvesoftware.com/>. Aufgerufen: 30. November 2016.
- [32] 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.
- [33] 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.
- [34] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, Nov 2000.
- [35] 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.
- [36] Éric Marchand and François Chaumette. Feature tracking for visual servoing purposes. *Robotics and Autonomous Systems*, 52(1):53 – 70, 2005. Advances in Robot Vision.