



Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fachbereich Informatik
Department of Computer Science

Bachelorarbeit

Automatisches Labeln von Objekten in einer Augmented Reality Umgebung

von
Janelle Pfeifer

Erstprüfer: Prof. Dr. Ernst Kruijff
Zweitprüfer: Prof. Dr. André Hinkenjann
Eingereicht am: 15. Oktober 2020

Erklärung

Janelle Pfeifer
Delpstraße 28
53359 Rheinbach

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum Unterschrift

Inhaltsverzeichnis

Abbildungsverzeichnis	1
1 Einleitung	2
1.1 Zielsetzung	2
2 Related Work	4
2.1 Huynh et al. (2019): In-Situ Labeling for Augmented Reality Language Learning	4
2.2 Bell et al. (2001): View Management for Virtual and Augmented Reality	5
2.3 Chen et al. (2018): Context-Aware Mixed Reality: A Framework for Ubiquitous Interaction	6
3 Grundlagen	8
3.1 Grundlagen zu Augmented Reality	8
3.1.1 Augmented Reality	8
3.1.2 AR System	8
3.1.3 Grundlagen zu 3D Szenen für AR	8
3.1.4 Spatial Mapping	9
3.2 Magic Leap AR Brille	9
3.2.1 Hardware	9
3.2.2 Betriebssystem	10
3.2.3 Unity Applikationen für Magic Leap One	11
3.2.4 Lokale und globale Koordinatensysteme in 3D Szenen	12
3.2.5 Kamera in 3D-Computergrafik	13
3.3 Computer Vision	13
3.3.1 Object Detection	14
3.3.2 Artificial Neural Networks	14
3.3.3 Convolutional Neural Networks	14
3.3.4 Azure Computer Vision Services	15
3.3.5 Azure Object Detection	16
3.3.6 Azure Custom Vision	16
4 Umsetzung	20
4.1 Design der Objekternennung	20
4.2 Architektur	20
4.3 Interaktion	23
4.4 Implementierung der Objekterkennung	24
4.5 Ein Foto aufnehmen	24
4.5.1 Object Detection	24
4.5.2 Von dem Foto zum 3D Raum	26
4.5.3 Raycast	28
4.5.4 LabelCreator	29
4.5.5 Azure Custom Vision	30
4.6 Entwicklung der Foto-Repräsentation	34
5 Auswertung	38
5.1 Laufzeitanalyse	38
5.2 Evaluierung der Objekterkennung durch Azure Objekt Detection	40
5.3 Evaluierung der Objekterkennung durch Azure Custom Vision	40
5.4 Objekte in 3D Szene lokalisieren	40

6 Zusammenfassung	44
6.1 Konzepte und Implementierte Funktionen	44
6.2 Ausblick	44
7 Literaturverzeichnis	45
8 Anhang	48
8.1 Markierte Objekte aus unterschiedlichen Blickwinkeln	48
8.2 Ausschnitt Trainingsbilder der Custom Vision Iterationen 4 und 6	49
8.3 Beispielbilder von Objekten mit Labels	50

Abbildungsverzeichnis

1	Objekterkennung von Huynh et al. (2019)	4
2	View Management von Bell et al. (2001)	5
3	Context Aware Shooter von Chen et al. (2018)	6
4	Spatial Mapping	9
5	Magic Leap One AR Brille.(MagicLeap)	10
6	Unity GameObject mit Components	12
7	View Frustum	13
8	Trainingsbild für Azrue Custom Detection	17
9	Evaluierung eines Azure Custom Detection Modells	18
10	Diagramm der Architektur	21
11	Klassendiagramm der Skripts	22
12	UI Ausgabe in der Szene	23
13	Klassendiagramm von <i>AzureObjectDetection</i>	24
14	Klassendiagramm für Json umwandlung	25
15	Klassendiagramm von <i>PixelToWorld</i>	26
16	Klassendiagramm von Raycast	28
17	Labels in der Szene	29
18	Haarbürste zwei mal erkannt	30
19	Haarbürste drei mal erkannt	30
20	Iteration 1 Analysebeispiel	31
21	Iteration 4 Analysebeispiel	33
22	Iteration 4 zweites Analysebeispiel	33
23	Iteration 6 Analysebeispiel	34
24	Ränder der Near Clipping Plane in Unity finden	35
25	Ränder des Foto-Anzeige-Elements finden	36
26	Tiefe des Foto-Anzeige-Elements finden	37
27	Laufzeit einer Objekterkennung	38
28	Laufzeitanalyse über 13 Bild-Analysen	39
29	Diagramm Laufzeitanalyse	39
30	Erkennt Objekte auf RGB-Bilde markiert	41
31	Erkannte Objekte in der Szene markiert	41
32	Label setzen, bei fehlerhafter Spatial Map	42
33	Spatial Mapping transparenter Objekte	43
34	Objekte mit Label. Blickwinkel 1	48
35	Objekte mit Label. Blickwinkel 2	48
36	Objekte mit Label. Blickwinkel 3	49
37	Ausschnitt der Trainingsbilder. Iteration 4	49
38	Ausschnitt der Trainingsbilder. Iteration 6	50
39	Beispielbild mit Iteration 4.	50
40	Beispielbild mit Iteration 6.	51

1 Einleitung

Augmented Reality (AR) ist eine Vermischung der realen Welt mit virtuellen Elementen. Es wird durch Anzeigegeräte, wie Handys, Tablets oder Augmented Reality Brillen präsentiert und bietet ein intuitives Benutzerinterface, um Objekte der realen Welt mit Informationen anzureichern. Eine AR Umgebung bietet dem Nutzer eine erweiterte Wahrnehmung der realen Welt, indem sie diese anzeigt und gleichzeitig 3D Objekte, 2D Overlays oder Audioelemente hinzufügt.

Die Interaktion der virtuellen Elemente miteinander, mit dem Nutzer und der realen Umgebung ist ein Grundbestandteil von AR Anwendungen. Um die Interaktion mit der Umgebung zu ermöglichen, müssen Informationen über die Geometrie der Umgebung vorliegen. Es gibt somit ein grobes Verständnis davon, wie ein Mesh der Umgebung aussieht. Dieses Geometrische Verständnis kann durch ein semantisches Verständnis der Umgebung erweitert werden. Dieses ermöglicht komplexere, Interaktionen zwischen digitalen und virtuellen Elementen. Semantische Informationen über die Umgebung können durch die Gegenstände erschlossen werden, die sich darin befinden.

Es gibt mehrere Möglichkeiten Gegenstände zu erkennen. Zum einen können Markierungen in der realen Welt verwendet werden. Dabei handelt es sich um statische Bilder, beispielsweise ein Foto, oder ein QR Code, die von einer Kamera eingescannt werden. Der Marker ist einzigartig für jedes Objekt, damit sie voneinander unterschieden werden können. Der Nachteil bei diesem Vorgehen ist der Arbeitsaufwand, der damit verbunden ist, jeden Gegenstand einzeln zu markieren.

Wenn man Markierungen in der realen Welt umgehen möchte, kann man den Nutzer der Applikation bitten, beispielsweise per Geste auf Objekte der realen Welt zu weisen, die erkannt werden sollen. Für jedes der Objekte muss der Nutzer angeben, um welche Art von Gegenstand es sich handeln, damit die Applikation unterschiedliche Objekte auseinander halten kann und die korrekten Informationen mit ihnen assoziiert. Auch hier ist der Arbeitsaufwand hoch.

Beide der Verfahren lassen sich schlecht skalieren um große AR Umgebungen abzudecken. Nur eine vollautomatische Objekterkennung ist skalierbar. Damit könnte man mit deutlich weniger aufwand Semantische Informationen über eine reale Umgebung erfahren und somit komplexere Anwendungsbiete für AR erschließen.

Um diese Automatisierung zu erreichen, kann Image Based Object Detection aus dem Bereich der Computer Vision verwendet werden. Dieses Verfahren ist darauf ausgelegt Objekte in Bildern zu erkennen. Die Objekterkennung mithilfe von 2D Abbildungen ist sehr performant. Und Besser als versuche 3D Wolken zu interpretieren.(O'Shea und Nash 2015)

1.1 Zielsetzung

In dieser Thesis wird das Erkennen und Labeln von Objekten in einer AR Umgebung, mithilfe von Image Based Object Detection, automatisiert. Die AR-Brille Magic Leap One Lightwear wird als Benutzerinterface und Plattform verwendet.

Das Minimalziel besteht darin Fotos der Umgebung zu analysieren und gefundene Objekte in einer 3D Szene mit Labels zu versehen. Mithilfe der Kamera des AR Gerätes werden Fotos von der Umgebung aufgenommen. Diese Fotos werden, durch ein trainiertes neuronales Netzwerk, nach Objekten durchsucht. Diese Positionen werden in einer digitalen Abbildung der Umgebung lokalisiert und mit Labels markiert.

Das erweiterte Ziel besteht darin, ein zweites neuronales Netzwerk in die Objekterkennung einzubinden. Dieses kann trainiert werden, spezifischen Objekten zu erkennen und damit die semantischen Informationen zu erweitern, die erkannt werden können.

Als Maximalziel werden die Labels der erkannten Objekte als virtuelle Elemente mit der Magic Leap One dargestellt.

2 Related Work

2.1 Huynh et al. (2019): In-Situ Labeling for Augmented Reality Language Learning

Huynh et al. (2019) schaffen ein Framework, mit dem die Lernmethode 'loci' in Augmented Reality umgesetzt und erweitert werden kann. Die Lernmethode beruht darauf, Gegenstände der Welt mit Notizen zu beschriften.

Dafür wurde folgende automatische Real-Time Objekt Erkennung entwickelt:

Mithilfe von Image Based Object Detection werden Objekte auf RGB-Fotos der AR Umgebung erkannt. Diese Objekte werden dann in die 3D Szene der Umgebung übernommen.

Die AR Brille hat zu wenig Rechenleistung, um Image Based Object Detection durchzuführen, daher wurde eine Server-Client Architektur aufgesetzt. Die Videokamera der AR Brille nimmt ein Video der Umgebung auf, das dann frameweise an den Server geschickt werden. Die einzelnen Frames werden an den Server geschickt. Dieser nutzt ein neuronales Netzwerk, um alle erkennbaren Objekte in dem Bild zu finden und mit Bounding Boxen zu lokalisieren.

Die ObjectDetection API von TensorFlow wird verwendet. Es findet mehrere Objekte in einem Foto in einer Analyse und gibt Bounding Boxen an. Die Objekterkennung soll in Real-Time durchgeführt werden. Um die Laufzeit dieser möglichst gering zu halten, wird die niedrigste Kameraauflösung mit 896x504 Pixel verwendet und die Fotos werden als JPEG komprimiert. Damit braucht die Analyse 30 ms pro Foto. Dies ermöglicht eine Real-Time Erkennung mit 30 Frames pro Sekunde.

Trotzdem ist die Erkennung in der Applikation durch einen Netzwerk Delay von 150 ms zwischen der Hololens und dem Server verspätet.

Die Hololens nummeriert die Frames, die an den Server geschickt werden. Zusätzlich wird für jedes Frame die Kameraposition gespeichert, mit der es aufgenommen wurde. So können Frames asynchron analysiert werden.

Ist die Analyse durchgelaufen, werden die Bounding Boxen der Objekte und die Kameraposition genutzt, um die Objekte in der 3D Umgebung zu lokalisieren. Dafür wird der Mittelpunkt jeder Bounding Box mithilfe eines Raycastes in die 3D Szene projiziert.

Ein Objekt wird erst markiert, wenn es auf mehreren Fotos gefunden wurde, um Fehlern bei der Objekterkennung entgegenzuwirken. Durch die Analyse des ersten Fotos wird eine ungefähre Position in der AR Umgebung für das Objekt ermittelt. Dieses Objekt wird dann mit den Objekten verglichen, die während der nächsten 60 Frames erkannt wurden. Haben die Objekte dieselbe semantische Bedeutung und eine ähnliche Position in der AR Umgebung, wird davon ausgegangen, dass es sich um ein einziges Objekt handelt, welches in den 60 Frames mehrmals aufgenommen wurde. Siehe Abbildung 1.

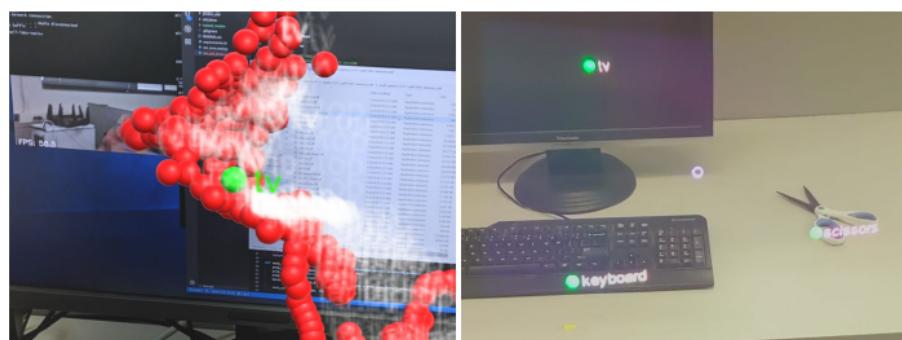


Abbildung 1: Ungefähre Positionen des Objektes über 60 Frames aufgenommen in rot. Finale Position in grün.(Huynh et al. 2019)

Unser Vorgehen zur Objekterkennung ähnelt dem Framework von Huynh et al. (2019). Auch hier soll die AR Umgebung durch das Hinzufügen von Labels semantisch angereichert werden. Es wird ebenfalls ein neuronales Netz verwendet, um die Object Detection auf RGB-Bildern durchzuführen. Wir haben dafür keinen eigenen Server aufgesetzt, sondern nutzen einen Cloud Service von Microsoft Azure. Durch Abspeichern von erhobenen semantischen Informationen ist eine Real-Time Detektion nicht relevant. Daher kann die Bild-Analyse länger dauern, was es erlaubt, mehrere neuronale Netze zu verwenden, die nach unterschiedlichen semantischen Informationen suchen.

In unserem Verfahren werden erkannte Objekte schon in der Umgebung markiert, wenn sie ein einziges Mal erkannt wurden. Wenn das Objekt erneut aufgenommen wird, wird die neue Analyse dem Objekt die gleiche Klasse und eine ähnliche Position der AR Umgebung zuweisen. Daran lässt sich erkennen, dass es sich um dasselbe Objekt handelt.

Die erste Position, die für das Objekt berechnet wurde und die Position, die aus der zweiten Aufnahme hervorgeht, unterscheiden sich mit hoher Wahrscheinlichkeit ein wenig voneinander. Daher werden die Positionen gemittelt, um eine akkurate Position für das Label des Objektes zu finden. (Huynh et al. 2019)

2.2 Bell et al. (2001): View Management for Virtual and Augmented Reality

Bell et al. (2001) beschreiben View Management für interaktive 3D Benutzeroberflächen. Als View Management wird die Positionierung von Labels bezeichnet. Die Labels können sich auf eine 2D Ebene beschränken oder im 3D Raum liegen. View Management zielt darauf ab die Labels so zu positionieren, dass sie einander und relevante reale Objekte nicht verdecken. Gleichzeitig sollen die Labels die Gegenstände der realen Welt auf eine verständliche Art annotieren. Labels sollen nahe bei den Objekten liegen, zu denen sie gehören. Siehe Abbildung 2.

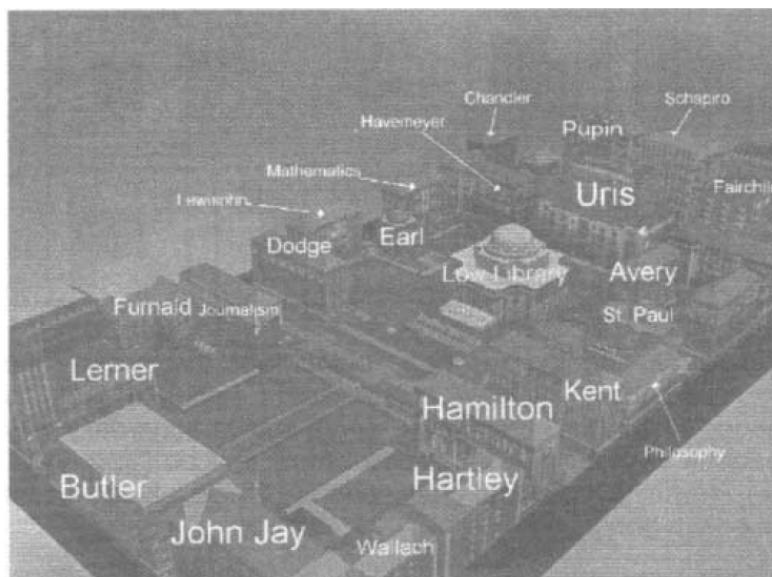


Abbildung 2: 3D Labels durch View Management positioniert.(Bell et al. 2001)

Unsere Applikation würde durch View Management profitieren. Nachdem Gegenstände in der Umgebung erkannt wurden, werden sie in dem 3D Raum durch Labels markiert. Die Lesbarkeit dieser Labels kann durch View Management verbessert werden, indem ihre Positionen über Zeit verändert werden.

Es müsste auf Veränderungen des Betrachtungswinkels und auf Hinzukommen von neuen Labels reagiert werden. Idealerweise würden die Labels einander und zusätzlich die Gegenstände der realen Welt nicht verdecken, die mit einem Label annotiert sind.

View Management geht jedoch über den Rahmen dieser Arbeit hinaus und konnte nicht implementiert werden. (Azuma und Furmanski 2003; Bell et al. 2001)

2.3 Chen et al. (2018): Context-Aware Mixed Reality: A Framework for Ubiquitous Interaction

Chen et al. (2018) stellen ein Framework vor, das einer AR Umgebung semantische Eigenschaften zuweist, um realistische Interaktionen zwischen virtuellen und realen Objekten zu erreichen. Insbesondere sollen physikalische Interaktionen an Realismus gewinnen.

Das Framework reichert die Umgebung mit Informationen über die Materialien an, aus denen reale Oberflächen und Gegenstände in der Umgebung bestehen. Die Umgebung wird in einer 3D Szene durch ein Mesh repräsentiert. Die einzelnen Voxel des Meshes werden mit Labels annotiert, um ihnen Semantik zuzuweisen.

Als beispielhafte Applikation wurde ein First-Person-Shooter vorgestellt, bei dem das Aussehen von Einschusslöchern davon abhängt auf welches Material geschossen wurde. Siehe Abbildung 3.

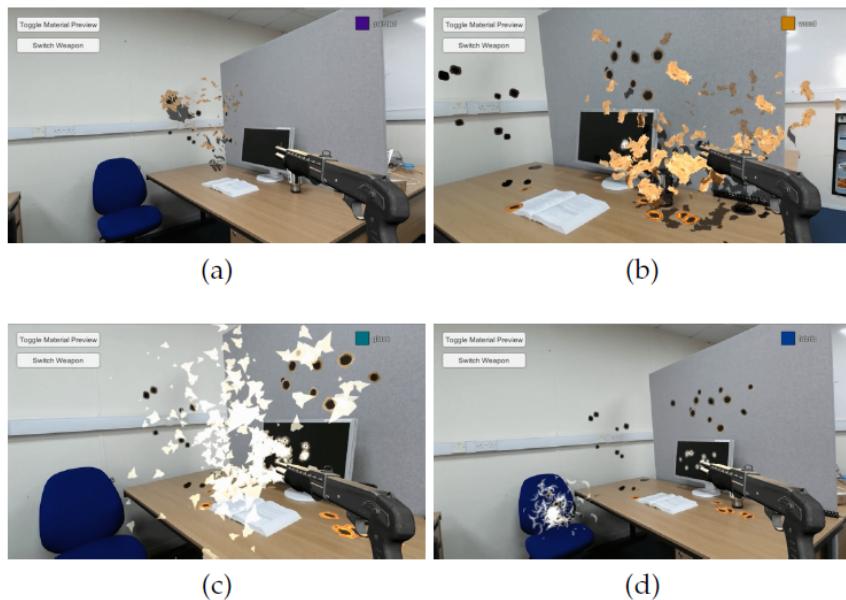


Abbildung 3: Es gibt unterschiedliche Interaktionen, wenn auf eine Wand (a), einen Tisch (b), einen Bildschirm (c) und einen Stuhl (d) geschossen wird.(Chen et al. 2018)

Für die Erkennung der Materialien werden mehrere Frames der Hololens segmentiert. Die dadurch entstehenden semantischen Informationen werden abgespeichert, um bei späteren Interaktionen der AR Applikation abgerufen werden zu können. Die Erkennung der Materialien ist nicht echtzeitfähig, aber durch das Speichern der Materialien im Raum können die Interaktionen in Echtzeit ablaufen. Die semantischen Informationen müssen nicht mit jedem Frame bestimmt werden.

RGB-Bilder der Umgebung werden aufgenommen und mit einem neuronalen Netzwerk analysiert, um semantische Informationen zu erheben. Das neuronale Netz wurde von Chen et al. (2018) für den First-Person-Shooter aufgesetzt. Das Netz wurde darauf tra-

niert 23 unterschiedliche Materialien zu erkennen und Bilder nach ihnen zu segmentieren. Dabei wird für jedes Pixel ein Material angegeben.

Mithilfe der Camera Position des Frames werden die Material-Informationen auf ein 3D Modell der Umgebung, ähnlich einer Textur, projiziert. Als Resultat wird an jedes Voxel des 3D Modells ein Label pro Frame gehängt, das die semantische Information wiedergibt, die in dem Frame erkannt wurde.

Bei der Segmentierung können Fehler auftreten, bei denen das auf dem Foto erkannte Material nicht dem realen Material entspricht. Um diesen Fehlern entgegenzuwirken werden mehrere Frames aufgenommen und analysiert.

Die Informationen aus den Frames bilden sich auf die Voxel ab. Durch die Kumulierung der Segmentierungen erhält jedes Voxel eine Menge an Labels. Voxel die in schwer einzuordnenden Bildbereichen liegen, erhalten Labels die unterschiedliche Informationen beinhalten.

Um sicherzustellen, dass jedes Voxel des 3D Modells nur eine semantische Bedeutung hat, werden die gesetzten Labels mit einer Bayesian Fusion und einem neuronalen Netz überarbeitet. Als Resultat hat jedes Voxel nur ein einziges Label.

Unser Verfahren versucht ebenfalls das Verständnis der AR Umgebung durch semantische Informationen zu erweitern. Auch wir nutzen ein neuronales Netzwerk, um RGB-Fotos der Umgebung zu analysieren. In unserem Verfahren erhält nicht jeder Pixel semantische Informationen, sondern nur Pixel, die die Position eines Gegenstandes markieren. Daher haben nur ausgewählte Voxel des 3D Meshes ein Label.(Chen et al. 2018)

3 Grundlagen

3.1 Grundlagen zu Augmented Reality

3.1.1 Augmented Reality

Augmented Reality vermischt die reale Welt mit digitalen (virtuellen) Elementen, um dem Nutzer eine erweiterte Wahrnehmung zu ermöglichen. Es können 3D Objekte, 2D Overlays oder Audioelemente verwendet werden, um eine reale Umgebung mit Informationen zu bereichern.

Die Umgebung beinhaltet den Teil der realen Welt, der in Augmented Reality abgebildet und erweitert werden soll. Beispielweise ein Zimmer, in dem eine AR Anwendung ausgeführt wird. Die AR Umgebung umfasst die reale Umgebung und die virtuellen Elemente. Augmented Reality weist drei grundlegende Merkmale auf:

- Die Realität wird mit dem Virtuellen kombiniert. Dazu werden reale Elemente mit Virtuellen überlagert.
- Interaktion mit virtuellen Elementen erfolgen in Echtzeit.
- Virtuelle Elemente haben einen festen räumlichen Platz in der AR Umgebung.

Die Merkmale unterstützen ein möglichst nahtloses Verschmelzen der realen Welt mit den virtuellen Elementen.

In einer AR Umgebung kann navigiert werden, indem der Nutzer sich physikalisch durch die reale Umgebung bewegt. Die reale Umgebung und die virtuellen Elemente stehen immer in dem gleichen räumlichen Verhältnis zueinander. Für AR ist keine andere Art der Navigation möglich, da sie die Verschmelzung der realen Welt mit den digitalen Elementen brechen würden.

Da die reale Welt immer zu sehen ist, gibt sie eine Referenz und einen Kontext für die virtuellen Objekte an. Beispielsweise steht die Größe von virtuellen Objekten immer in Relation zu der realen Umgebung.(Dörner et al. 2019)

3.1.2 AR System

Ein AR System besteht aus der Hardware und Software, die benötigt wird, um eine AR Umgebung zu erzeugen.

Das System muss die Vermischung der realen Welt mit virtuellen Elementen anzeigen und Interaktionen des Nutzers mit virtuellen Elementen, sowie Interaktion von virtuellen Elementen mit der realen Welt simulieren

Ein AR System ist in der Regel nicht an einen bestimmten Ort gebunden. Es kann in unterschiedlichen Umgebungen eingesetzt werden, die unterschiedliche reale Gegenstände aufweisen. AR Applikationen müssen diese unterstützen.(Dörner et al. 2019)

3.1.3 Grundlagen zu 3D Szenen für AR

Die virtuellen Inhalte der AR Anwendung werden in einer virtuellen Szene gespeichert. AR Anwendungen müssen in Echtzeit laufen. Daher muss die virtuelle Szene echtzeitfähig sein. Im besten Fall kann ein Nutzer keinen Unterschied zwischen der virtuellen Welt und der realen Welt, bezüglich auf zeitliche Verzögerungen, bemerken.

Um Interaktion mit digitalen Elementen zu ermöglichen, werden relevante Teile der realen Welt in der 3D Szene repräsentiert. So werden beispielsweise die Wände und der Boden eines Raumes in der Szene abgebildet. Auch die Position eines Controllers und die Blickrichtung des Nutzers kann mit Sensoren verfolgt und in die Szene miteinbezogen werden.

3.1.4 Spatial Mapping

Um virtuelle Objekte an eine Umgebung anzupassen und Interaktion zwischen virtuellen Objekten und der realen Welt zu ermöglichen, benötigt ein AR System Informationen über die Geometrie der Umgebung.

Mit den Sensoren der AR Hardware werden Informationen gesammelt, die Aussagen über die Geometrie der Umgebung treffen. Beispielsweise haben AR Geräte eine Tiefenkamera, die Entfernung messen kann. Die Daten der Sensoren werden gesammelt und in Relation zu der Bewegung des Gerätes gesetzt, um die Umgebung zu rekonstruieren. Dieser Vorgang nennt sich Spatial Mapping.

Mit der entstehenden Spatial Map können digitale Elemente mit der Umgebung interagieren, diese verdecken oder von ihr verdeckt werden.(Microsoft 2018b)

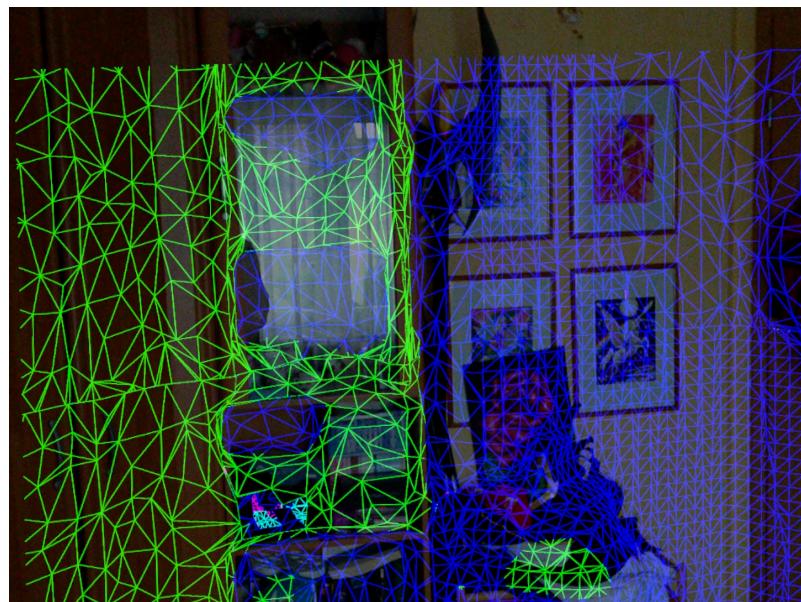


Abbildung 4: Spatial Map. Die Farben des Meshes zeigen die Entfernung des Nutzers zu den Objekten an.

3.2 Magic Leap AR Brille

Die Magic Leap One Lightwear ist eine Augmented Reality Brille, die von dem Unternehmen Magic Leap entwickelt wurde. Sie verfügt über ein Head-Mounted Display und einer Recheneinheit, die über ein Kabel mit dem Display verbunden ist. Die Recheneinheit kann an der Hüfte getragen werden, was die AR Brille mobil macht.

3.2.1 Hardware

Die Recheneinheit besitzt zwei Denver 2.0 64 Bit Prozessor-Kerne und vier ARM Cortex A57 46 bit Kerne. Davon ist einer der Denver Kerne und zwei der ARM Cortex Kerne für Applikationen nutzbar.

Sie besitzt neun Sensoren und mehrere Kameras. Dazu gehören:

- ein Infrarot Tiefen-Sensor,
- ein Eye Tracker,
- eine Foto- und Videokamera, die im Format 16:9 mit einer Auflösung von 1920x1080 Pixeln aufnimmt und

- mehrere Umgebungskameras die in unterschiedliche Richtungen ausgerichtet sind.
(MagicLeap 2018, 2020c)

Der Output geschieht über ein Display mit einem 50 Grad Field of View und einem Seitenverhältnis von 4:3. Das Display ist transparent. Daher kann die reale Welt immer betrachtet werden. Selbst wenn ein weißes Objekt angezeigt wird, schimmert die reale Welt noch durch. Das Display kann keine schwarzen Objekte anzeigen.

Eingaben erfolgen über einen 6 Degree of Freedom Controller. Er verfügt über 3 Knöpfe (Trigger, Bumper, Home Button) und ein Touchpad. (MagicLeap 2018, 2020c)

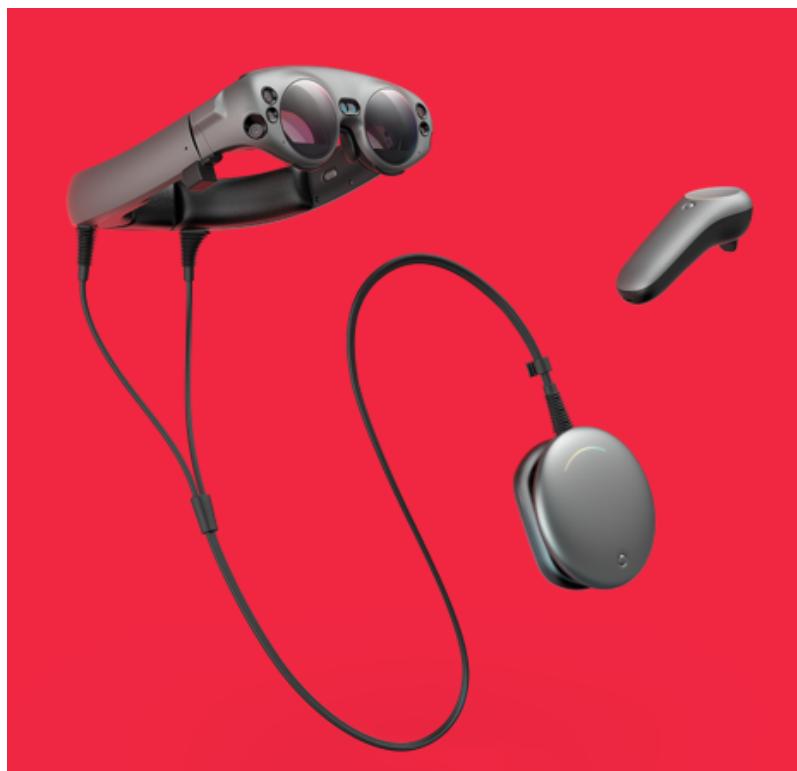


Abbildung 5: Magic Leap One AR Brille.(MagicLeap)

3.2.2 Betriebssystem

Die Magic Leap Brille läuft auf dem Betriebssystem Lumin OS. Dieses wurde für Augmented Reality entwickelt und bietet Applikationen entsprechende Funktionalitäten an. Beispielsweise führt das Betriebssystem Spatial Mapping durch.(MagicLeap 2019a, 2020d) Dabei werden mit den Sensoren und Kameras der Brille Daten aufgenommen und in einen zeitlichen Zusammenhang mit der Bewegung der Brille gesetzt, um eine Rekonstruktion des Raumes zu erhalten.(MagicLeap 2019a, b, 2020b, d)

Lumin OS bietet es Applikationen an,

- Raycasts auf die Umgebung durchzuführen und
- ein Mesh der Rekonstruktion zu erhalten.

Neben dem Spatial Mapping unterstützt Lumin OS das Verarbeiten vom Input des Controllers und verwaltet die Zugriffsrechte der Applikationen. Dazu gehört beispielsweise das Recht (Permission) die Fotokamera und das Netzwerk zu nutzen.(Leap 2018; MagicLeap 2020d)

Selbst wenn mehrere Applikationen das Recht dazu haben, kann nur eine Applikation zum gleichen Zeitpunkt auf die Fotokamera zugreifen. Die Applikationen müssen sich

mit der Kamera-Ressource von Lumin OS verbinden, diese reguliert, welche Applikation Fotoanfragen stellen kann und die aufgenommenen Fotos erhält.

3.2.3 Unity Applikationen für Magic Leap One

Unity ist eine Entwicklungsumgebung, mit der Applikationen für Lumin OS erstellt werden können. Ein Unity Projekt besteht aus mindestens einer Szene, in der mehrere Game-Objects existieren. 'GameObject' ist ein Oberbegriff für alle Elemente, die in der Szene existieren. Sie bilden eine hierarchische Struktur, in der GameObjects Parents und Children anderen GameObjects sind.(Unity 2017)

GameObjects haben selbst keine Funktionalität und kein Aussehen. Diese werden durch Components hinzugefügt. Einige Components sind bereits in Unity integriert. Sie geben einem GameObject eine Position in der Szene (Transform Component), machen sie sichtbar (Mesh Renderer) oder gibt dem GameObject die notwendige Logik um als Kamera zu fungieren.

Um eine neue Funktionalität zu implementieren, kann ein C# Script geschrieben werden und einem GameObject als Component zugewiesen werden.

Magic Leap unterstützt die Entwicklung von Applikationen für Lumin OS mit Unity. So wird ein Unity Project Template von Magic Leap angeboten, dass das Set-up der Applikation erleichtert. Dazu gehören Build Optionen und das Einstellen der Hauptkamera der Szene – der Unity Kamera. Die Unity Kamera rendert die Szene für das Display der Magic Leap One. Die Kamera verfolgt die Bewegung der Magic Leap Brille und kopiert diese in der Unity Szene. (MagicLeap 2020a)

Das Unity Template verfügt über vorgefertigte Skripts, die auf Funktionalitäten von Lumin OS zugreifen:

- *MLInput* stellt Informationen über den Controller zur Verfügung. Damit können die Knöpfe und Touchpad-Gesten überwacht werden.
- *MLCamera* greift auf die Fotokamera der AR Brille zu. Fotoanfragen und Resultat-Daten werden ebenfalls durch *MLCamera* behandelt.
- *MLPrivilegeRequestBehavior* ermöglicht es den Nutzer nach dem Recht zu fragen, auf bestimmte Teile des AR Systems zuzugreifen. Beispielsweise muss nach Permission gefragt werden, um die Fotokamera-Ressource zu verwenden.
- *MLRaycast* greift auf die Spatial Mapping Funktionalitäten von Lumin OS zu. Es gibt einen Raycast auf die Rekonstruktion der Umgebung beim Betriebssystem in Auftrag.
- *MLSpatialMapper* zeigt die Rekonstruktion der Umgebung von Lumin OS als Mesh an. Das Mesh besteht aus GameObjects, die von *MLSpatialMapper* erzeugt werden. Sie werden ständig aktualisiert, um das geometrische Verständnis des Betriebssystems wiederzuspiegeln.

Scripts können von der Klasse *Monobehavior* erben. *Monobehaviors* verfügen über Methoden, die von Unity zu unterschiedlichen Events der Laufzeit ausgeführt werden. Nur Skripts, die ein Component eines GameObjects sind, werden aufgerufen.

- *Awake* - diese Methode wird aufgerufen, wenn das GameObject initialisiert wird.
- *Update* - die Update Methoden der Scripts werden zu jedem Frame aufgerufen.
- *OnDestroy* - diese Methode wird aufgerufen, wenn das GameObject aus der Szene gelöscht wird.

Monobehaviors können über globale Variablen verfügen. Wenn das Skript einem GameObject als angehören, kann der Inhalt der globalen Variable in Unity modifiziert werden.

Dadurch kann einem Skript eine Referenz auf ein bestimmtes GameObject gegeben werden. Siehe Abbildung 6.

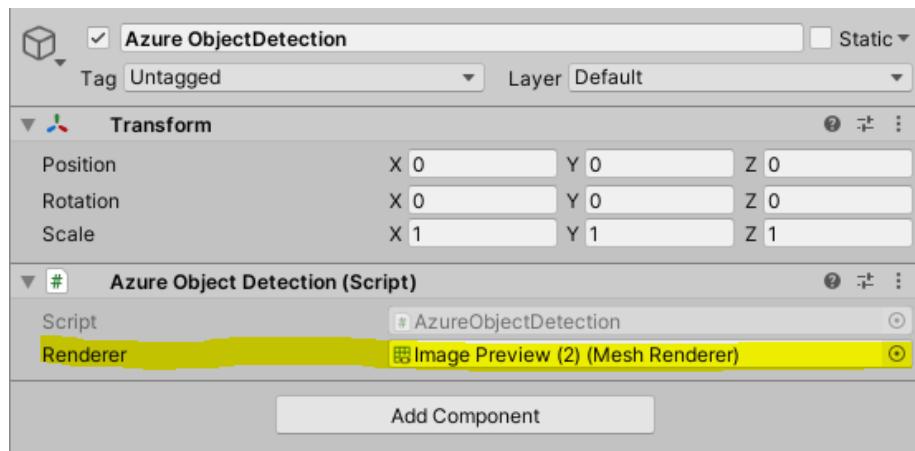


Abbildung 6: GameObject mit Components. Die Component Azure Object Detection verfügt über eine globale Variable (in gelb), die auf eine Mesh Renderer Component eines anderen Objektes verweist.

Monobehaviors können als Singleton fungieren, indem sie eine globale statische Referenz auf sich selber anlegen, wenn ihre *Awake* Methode aufgerufen wird.

```

1 public class MySingletonClass : MonoBehaviour
2 {
3     public static MySingletonClass instance = null;
4     private void Awake()
5     {
6         if (instance != null && instance != this)
7         {
8             Destroy(this.gameObject);
9         }
10        instance = this;
11    }
12    public void DoSomething(){...}
13 }
```

Jedes andere Script kann die Instanz des Singletons aufrufen, indem es die statische Referenz der Klasse des Singletons verwendet.

```
1 MySingletonClass.instance.DoSomething();
```

Skripts können während der Laufzeit GameObjects erzeugen. Um festzulegen, wie das GameObject aussehen soll, kann in dem Unity Editor ein Prefab erstellt werden. Prefabs sind vorgefertigte, abgespeicherte GameObjects, deren Child Objects, Components und globale Variablen festgelegt werden. Das Prefab fungiert als Vorlage für GameObjects die zur Laufzeit initialisiert werden.

Während der Initialisierung wird eine Position der Szene angegeben, an die das neue GameObject gesetzt wird. Nachdem das Objekt initialisiert wurde, kann auf seine Components und Scripts zugegriffen werden.(Unity 2018)

3.2.4 Lokale und globale Koordinatensysteme in 3D Szenen

In der 3D Szene werden die Positionen von Objekten als Matrizen in dreidimensionalen Koordinatensystemen verwaltet. Es gibt ein globales Koordinatensystem (auch Weltkoordinatensystem oder World Space), in dem alle Objekte relativ zu einem Ursprung liegen.

Jedes Objekt hat zusätzlich ein eigenes, lokales Koordinatensystem (Objektkoordinatensystem). Dessen Ursprung liegt in dem jeweiligen Objekt. Die Position und Rotation des Objektes in dem globalen Koordinatensystem bestimmt die Relation zwischen dem globalen und dem lokalen Koordinatensystem.

Das lokale Koordinatensystem einer Kamera wird auch Camera Space genannt. Die Relation zwischen dem Camera Space und dem globalen Koordinatensystem wird in Unity durch die cameraToWorld Matrix beschrieben. Mithilfe dieser Matrix kann eine Koordinate aus dem Camera Space in die entsprechende Koordinate des globalen Koordinatensystems transformiert werden.(Unity 2020a)

Dazu wird die Koordinate als Vektor angegeben und mit der cameraToWorld Matrix multipliziert. Das Resultat ist ein Vektor, der eine Koordinate im globalen Koordinatensystem angibt.(Unity 2020a, c)

3.2.5 Kamera in 3D-Computergrafik

View Frustum ist das Teilvolumen einer 3D Szene, die auf einen zweidimensionalen Bildschirm abgebildet wird. Alle Objekte, die von der Kamera gesehen werden, befinden sich in dem View Frustum.

Clipping Plane bezeichnet eine Ebene, die den View Frustum quer zur Blickrichtung begrenzt. Es gibt eine vordere und eine hintere Clipping Plane. Die vordere Clipping Plane liegt nah an der Kamera. Alle Objekte die zwischen der Kamera und der vorderen Clipping Plane liegen, werden nicht angezeigt.

Die hintere Clipping Plane limitiert wie weit Objekte entfernt sein können, bevor sie nicht mehr zu sehen sind. Siehe Abbildung 7.(Unity 2020d)

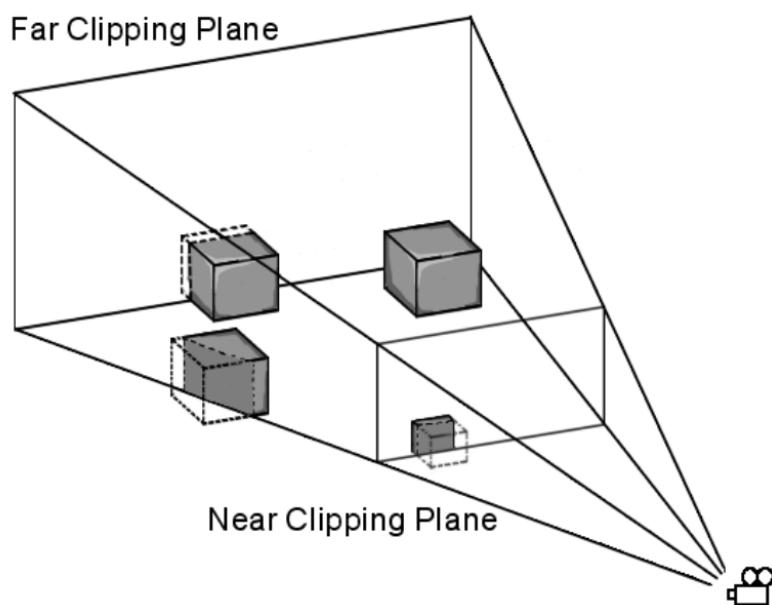


Abbildung 7: Diagramm des View Frustum mit Clipping Planes.

3.3 Computer Vision

Computer Vision setzt das Ziel die Inhalte digitaler Bilder und Videos zu verstehen. Sowohl semantische als auch geometrische Inhalte werden dabei analysiert. Zu den Auf-

gaben von Computer Vision gehören: Objekterkennung, Bild-Segmentierung, Menschen erkennen und Bewegungen in Videos nachverfolgen.

In diesem Feld kommen Statistik, Bilderverarbeitung und Maschinelles Lernen zum Einsatz. Maschinelles Lernen wird eingesetzt, um neuronale Netze zu formen, die Bild- und Videoanalysen durchführen können.(Maier et al. 2019)

3.3.1 Object Detection

Objekt Detection ist eine Aufgabe der Computer Vision. Es sollen mehrere Objekte in einem Bild erkannt werden. Daher wird es auch Image Based Object Detection genannt. Spezialisierungen von Object Detection sind beispielsweise Gesichtserkennung oder das Erkennen von Fußgängern. Durch eine Objekterkennung werden semantische Inhalte eines Bildes automatisch erhoben.

Für die Objekte wird eine Klasse und eine Bounding Box bestimmt. Die Klasse gibt an, um welche Art von Objekt es sich handelt. Beispielsweise ob es eine Tastatur oder ein Computerbildschirm ist. (Object classification) Die Bounding Box gibt ein Viereck in dem Bild an, in dem sich das Objekt befindet.(object localization)

3.3.2 Artificial Neural Networks

Artificial Neural Networks sind Machine Learning Architekturen. Sie können beispielsweise Musik, Text oder Bilder nach Mustern durchsuchen. Sie sind für keine genaue Aufgabe programmiert, sondern lernen indem sie mit Beispieldaten trainiert werden.

Für jedes Beispiel gibt es ein Label, das angibt, ob es das gesuchte Muster enthält oder nicht. Die Struktur des Networks verfügt über Gewichte, die Einfluss auf den Output haben. Mit jedem Trainingsbeispiel passt das Network die Gewichte an, sodass der Output dem Label des Beispiels entspricht.(Jiao et al. 2019; O’Shea und Nash 2015)

Artificial Neural Networks bestehen aus einer Menge an verbundenen Knoten, die jeweils eine Berechnung durchführen. Diese Knoten sind in Ebenen aufgeteilt, den Input Layer, den Output Layer, und mehrere Hidden Layer dazwischen. Die Knoten einer Ebene sind mit allen Knoten der vorherigen Ebene verbunden.(Jiao et al. 2019; O’Shea und Nash 2015)

Das Neural Network bekommt eine Menge an Daten als Input. Die Knoten arbeiten zusammen, um den Output zu erzeugen. Dabei wird über Gewichte entschieden, wie viel Einfluss das Ergebnis der einzelnen Knoten auf die nächste Ebene hat.(Jiao et al. 2019; O’Shea und Nash 2015)

Um ein Neural Network zu trainieren, wird der Output von einem Mensch bewertet. Das Neural Network nutzt diese Bewertung, um die Gewichte der einzelnen Knoten zu verändern. So passt sich das Neural Network an. (Jiao et al. 2019; O’Shea und Nash 2015)

3.3.3 Convolutional Neural Networks

Convolutional Neural Networks sind auf das Verarbeiten von Bildern spezialisiert. Sie nutzen aus, dass Bilder viele Redundanzen und informationsarme Bereiche haben. Daher können mit jedem Verarbeitungsschritt des Networks Informationen weggelassen werden. So können Rechenzeit und Volumen der Trainingsdaten verringert werden.(Jiao et al. 2019; Jmour et al. 2018; O’Shea und Nash 2015)

Convolutional Neural Networks sind Machine Learning Architekturen, die darauf ausgelegt sind, Muster in Bildern zu erkennen. Sie müssen auf das Muster trainiert werden. Dazu wird ihnen eine Menge an Bildern, die Teilweise das Muster erhalten, und der gewünschte Output, der erreicht werden soll, gegeben. Die Struktur des Network verfügt über Gewichte, die die Berechnung des Outputs beeinflussen. Mit jedem Trainingsbild

passt das Network die Gewichte an, damit es die Muster korrekt erkennen kann.(Jiao et al. 2019; O'Shea und Nash 2015)

Convolutional Neural Networks werden hauptsächlich eingesetzt, um Muster in Bildern zu erkennen. Daher ist ihre Struktur und ihre Arbeitsweise auf Bilder spezialisiert. Sie brauchen weniger Rechenzeit und weniger Trainingsdaten als ein generelles Artificial Neural Network für dieselbe Aufgabe brauchen würde.(Jiao et al. 2019; Jmour et al. 2018; O'Shea und Nash 2015)

Die Knoten in einer Ebene eines Convolutional Neural Network sind nur mit wenigen Knoten der vorherigen Ebene verbunden. So sinkt die Menge an Informationen mit jeder Ebene. Das CNN wird gezwungen sich auf wesentliche Teile des Bildes zu konzentrieren, mit denen beispielsweise ein Objekt oder Muster erkannt werden kann. (Jiao et al. 2019; O'Shea und Nash 2015)

3.3.4 Azure Computer Vision Services

Microsoft Azure ist eine Cloud Computing Plattform von Microsoft. Seit 2010 stellt sie Cloud Services zur Verfügung, die unter andrem Computer Vision Aufgaben durchführen. Diese sind über REST APIs erreichbar.

Rest steht für Representational State Transfer. Eine Rest API ist eine Programmierschnittstelle, die sich an dem Verhalten des World Wide Web orientiert. Die Kommunikation zwischen einem Client und dem Server orientiert sich an REST. Rest steht für Representation State Transfer und bezeichnet einen Architekturansatz wie verteilte Systeme miteinander kommunizieren können. REST verlangt unter anderem ein Client-Server Modell und zustandslose Kommunikationen zwischen ihnen. Jede Anfrage eines Clients beinhaltet alle Informationen, die der Server benötigt um sie zu beantworten.

Eine Rest-API kann mit http/s realisiert werden. Der Server / Service kann durch eine URL angesprochen werden. Diese wird auch Endpoint genannt. HTTP-Methoden, wie GET und POST kommunizieren an den Service welche Operation durchgeführt werden soll. Diese werden auch Http-Anfragen oder HTTP Request genannt. Mit Get werden Daten von dem Server angefordert. Die Post Methode erlaubt das Übermitteln von zusätzlichen Daten an der Server.

Ein HTTP Request besteht aus einer Start Line, einer Menge an Headers und einem Body. Die Start Line gibt die HTTP-Methode (GET, POST, ...) und eine Endpoint-URL an. In den Headers werden weitere Informationen über die Anfrage an den Server angegeben. Für die Anwendung einer API, wird Authentifizierungsschlüssel des Clients in einem Header gespeichert. Zuletzt hat der Request einen Body. Abhängig von der HTTP-Methode beinhaltet der HTTP-Request Daten für den Server. So speichert eine POST Nachricht ihre Daten, beispielsweise ein Foto, in dem Body. Die Länge und Art der Daten wird mit Headern angegeben.

Auf jede HTTP-Anfrage wird eine HTTP-Response als Antwort zurückgeschickt. Die HTTP-Response besteht ebenfalls aus einer Start Line, Headern und einem Body. Die Start Line gibt einen Response Code und einen Statustext an. Der Response Code teilt dem Client mit, ob die Anfrage erfolgreich bearbeitet wurde. Wenn ein Error aufgetreten ist, gibt der Statuscode Auskunft über die Art des Errors. Der Statustext gibt eine kurze Erklärung des Response Codes.

Beispiel Response Codes:

- 404 - Url nicht gefunden
- 401 - Zugriff verweigert wegen ungültiger Authentifizierung
- 400 - fehlerhafte Anfrage. Der Service konnte die Anfrage nicht verstehen
- 200 - Anfrage wurde erfolgreich bearbeitet

Eine API schickt, bei einer erfolgreichen Bearbeitung der Anfrage, in dem Body der HTTP-Response die erfragten Daten mit.

3.3.5 Azure Object Detection

Microsoft Azure bietet einen Computer Vision Service an. Dabei handelt es sich um mehrere KIs, die für unterschiedliche Aufgaben trainiert wurden. Dazu gehört unter anderem ein Service für Object Detection.

Dabei sendet der Anwender ein Bild an Microsoft, dort wird es verarbeitet und ein Ergebnis in Form einer Json Datei zurückgeschickt.(Microsoft, 2018a, 2019a, 2020)

Die Object Detection basiert auf einem trainierten KI-Modell. Dieses kann nur Objekte erkennen, für die es trainiert wurde. Zusätzlich können Objekte, die in dem Foto sehr klein sind oder nah bei anderen Objekten liegen, nicht erkannt werden.(Microsoft 2019b)

Der Service ist durch eine REST-API erreichbar. Mit einer Post Anforderung werden die Bilddaten übertragen und die Analyse angefragt. Die Response Nachricht beinhaltet eine Json-Datei, welche die gefundene Objekte und deren Positionen auf dem Foto beinhaltet. Um den Service zu nutzen wird eine Http Post Anforderung an die Webadresse geschickt. Als Endpoint wird die Webadresse der REST-API angegeben. Die Nachricht muss folgenden Inhalt haben:

- Ein Authentifizierungsschlüssel, der mit einem Azure Account verbunden ist
- und Bilddaten

Das Resultat der Analyse wird in der Response-Nachricht zurückgeschickt. Der Response Code 200 gibt an, das die Analyse durchgeführt wurde. In diesem Falle wird eine Json Datei mitgeschickt. Beispielhaft:

```
1 {"objects": [{"rectangle": {"x": 1377, "y": 900, "w": 157, "h": 138}, "object": "computer mouse", "confidence": 0.681},  
2 {"rectangle": {"x": 1336, "y": 0, "w": 584, "h": 841}, "object": "display", "confidence": 0.873},  
3 {"rectangle": {"x": 315, "y": 25, "w": 906, "h": 622}, "object": "display", "confidence": 0.839},  
4 {"rectangle": {"x": 447, "y": 800, "w": 862, "h": 166}, "object": "computer keyboard", "confidence": 0.71}],  
5 "requestId": "a77e261a-7d40-4159-bf78-19d8fb61ad92",  
6 "metadata": {"height": 1080, "width": 1920, "format": "Jpeg"}}
```

Die gefundenen Objekte befinden sich in der Liste mit dem Key 'objects'. Jedes Element der Liste hat ein Bounding Box 'rectangle' und eine Klassenbezeichnung 'object'. Zusätzlich hat jedes Object einen 'confidence' Score. Dieser gibt die Wahrscheinlichkeit an, das das Objekt korrekt erkannt wurde. Neben den Objekten wird noch eine requestId und Metadaten über das Bild zurückgeschickt.

3.3.6 Azure Custom Vision

Azure bietet zusätzlich einen Computer Vision Service an, den der Nutzer trainieren kann. Das verwendete KI-Modell ist für Objekt Detection entwickelt und ist untrainiert. Custom Vision kann als Ergänzung zu Azure Object Detection verwendet werden.(Micosoft 2018)

Das KI-Modell lässt sich für Image Klassifizierung oder Objekt Detection trainieren. Über eine Webseite lässt sich der Trainingsprozess des Modells steuern und auswerten. Der Nutzer lädt Fotos hoch und fügt jedem Foto händisch einen oder mehrere Tags hinzu. Die Tags geben die Objekt-Klasse an, die sich in dem Bild befindet. Um das Modell für Object Detection zu trainieren, kann eine rechteckige Region in dem Foto markiert werden, um das Objekt zu lokalisieren. Siehe Abbildung 8. In einem Bild können mehrere Regionen markiert werden. Jede Region wird mit einem Tag annotiert.

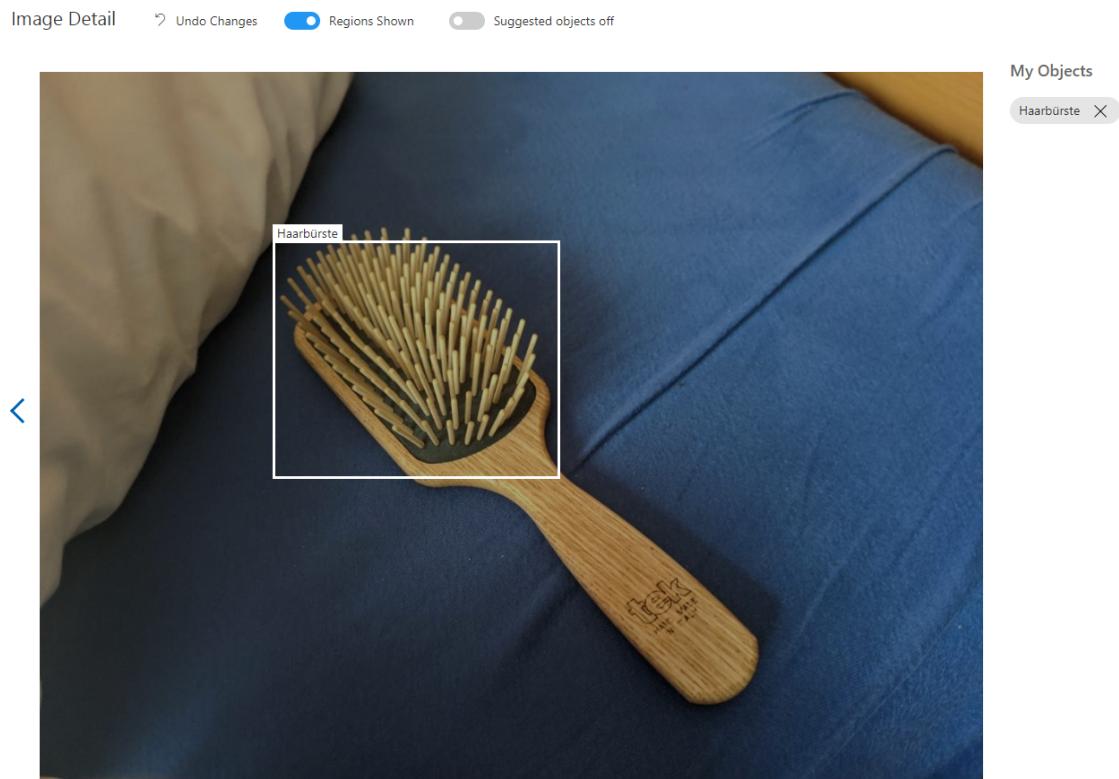


Abbildung 8: Trainingsbild für die Objekterkennung einer Haarbürste.

Für jede Objekt-Klasse müssen mindestens 15 Bilder hochgeladen werden, in denen der entsprechende Tag vorkommt. Nur dann kann der Trainingsprozess gestartet werden. Mit mehr Bildern in dem Trainingsset wird die Objekterkennung des Modells jedoch robuster. Azure empfiehlt es mindestens 50 Bilder pro Tag zu verwenden. Wenn mit nur wenigen Bildern trainiert wird, haben die einzelnen gewählten Bilder einen sehr großen Einfluss auf das Modell.

Durch das Trainieren wird eine Iteration des Modells erzeugt. Diese Iteration kann verwendet werden, um Object Detection durchzuführen. Wird erneut der Trainingsprozess gestartet, wird eine weitere Iteration erstellt. Die Iterationen werden alle unter einer Projekt ID gespeichert.

Die Iterationen des Modells werden in einer Testphase evaluiert. Für jede Objekt-Klasse werden drei Metriken angesetzt.

- Precision - die Wahrscheinlichkeit dass ein gefundenes Objekt, tatsächlich der angegeben Klasse angehört. (Die Wahrscheinlichkeit dass es kein false positive ist.)
- Recall - aus einer Menge an Objekten die einer Klasse angehören, der Prozentsatz an Objekten, die das Modell korrekt lokalisieren und klassifizieren konnte.
- a.p (Average Precision) - eine Gesamtwertung für die Evaluierung basierend auf Precision und Recall.

Die Evaluierungen der einzelnen Objekt-Klassen werden in Precision, Recall und mean Average Precision gemittelt. Siehe Abbildung 9.

Iteration 4

Finished training on **28.8.2020, 15:39:51** using **General** domain
 Iteration id: **2254989e-9263-499a-888b-f91cc88304a4**
 Published as: **Iteration4**



Performance Per Tag

Tag	Precision	▲	Recall	A.P.	Image count
Haarbürste	100.0%		100.0%	100.0%	16
Nivea	100.0%		100.0%	100.0%	25

Abbildung 9: Die Evaluierung des Modells

Mit einer erzeugten Iteration des Modells können Fotos auf Objekte untersucht werden. Genauso wie Azure Object Detection ist das Custom Modell durch eine REST-API erreichbar. Der Endpoint beinhaltet die ID des Projektes, und die Nummer der Iteration, die verwendet werden soll. In der Post Nachricht wird ein Authentifizierungsschlüssel und ein zu analysierendes Bild mitgeschickt.

Das Resultat der Analyse ist eine Json Datei. Der Aufbau der Datei unterscheidet sich vom Dateiaufbau von Azure Object Detection leicht. Informationen über die Iteration, die für die Analyse verwendet wurde, werden wiedergegeben.

```

1 {"id": "9cb0cc50-1dca-4b4a-b4d1-95d6bd25c352",
2 "project": "ac915246-5268-461f-bd11-cf0c1826d509",
3 "iteration": "2254989e-9263-499a-888b-f91cc88304a4",
4 "created": "2020-10-10T02:40:40.107Z",
5 "predictions": [{"probability": 0.997380137, "tagId": "d390d34e-afc4-4ff4-8
   dcd-3ee8fe79cb8f", "tagName": "Haarbürste", "boundingBox": {"left"
   : 0.258805573, "top": 0.226171583, "width": 0.303168833, "height"
   : 0.329167157}}, {
6 {"probability": 0.0141124222, "tagId": "d390d34e-afc4-4ff4-8dc
   d-3ee8fe79cb8f", "tagName": "Haarbürste", "boundingBox": {"left"
   : 0.542580664, "top": 0.507969, "width": 0.2195471, "height": 0.3491398}}, {
7 {"probability": 0.0116642443, "tagId": "263b3042-8958-4775-9a19-
   e2602d19c9b7", "tagName": "Nivea", "boundingBox": {"left": 0.542580664,
   "top": 0.507969, "width": 0.2195471, "height": 0.3491398}}]}

```

Die gefundenen Objekte werden in der Liste "predictions" aufgezählt. Die Objektklassen werden in "tagName" gespeichert. Die "probability" gibt das Vertrauen des Modells darin an, dass das Objekt korrekt erkannt wurde.

Die erkannten Objekte müssen bei nach ihrer Probability gefiltert werden, da auch Objekte mit einer sehr niedrigen Probability in der Json Datei enthalten sind.

4 Umsetzung

Das Ziel ist das Erkennen und Labeln von Objekten in einer AR Umgebung, durch Image Based Object Detection.

4.1 Design der Objekterkennung

Im Folgenden werden die Arbeitsschritte einer Detection beschrieben.

Wenn der Nutzer das Signal gibt, beginnt die Detection. Als Erstes wird ein Foto mit der Kamera der AR Brille aufgenommen. Dieses Foto wird dann an Azure Object Detection und Azure Custom Vision geschickt. Die Services untersuchen das Foto nach Objekten, geben deren Klasse und Position auf dem Foto an.

Für jedes Objekt soll ein Label erstellt werden, das zeigt wo sich das Objekt in der realen Welt befindet. Dafür wird in der 3D Szene der AR Umgebung eine virtuelle Repräsentation des Fotos erschaffen. Die Fotorepräsentation muss die richtige Skalierung, Position und Rotation haben, um das räumliche Verhältnis zwischen der realen Fotokamera und der Umgebung nachzubilden.

Da die Fotokamera und das Display nahe beieinander liegen und den gleichen Blickwinkel haben, kann die Position des Displays als Repräsentation des Fotos genutzt werden. In der 3D Szene ist das Display mit der Hauptkamera gleichgesetzt. Die Clipping Plane der Kamera hat somit die gleiche Rotation und eine zumindest ähnliche Position und Skalierung wie das Foto.

Daher werden die Foto-Positionen auf Koordinaten der Clipping Plane abgebildet. Dabei werden verbleibende Positions- und Skalierungsunterschiede ausgeglichen. Für jedes Objekt wird so eine Koordinate auf der Clipping Plane bestimmt.

Als Nächstes wird ein Raycast, von der Kamera aus, durch die Clipping Plane Koordinate geschickt. Der Raycast schneidet sich mit einem Mesh, das die reale Welt abbildet. Die getroffene Position wird mit einem Schriftzug markiert. Dort befindet sich das Objekt, das auf dem Foto gefunden wurde.

Alle Objekte, die Azure Object Detection und Azure Custom Vision gefunden haben, werden so für den Nutzer in der AR Umgebung markiert.

4.2 Architektur

Die Magic Leap One übernimmt alle Berechnungen im 3D Raum und führt Spatial Mapping durch. Da die Analyse von 2D Fotos sehr speicher- und rechenintensiv ist, wird sie an eine REST-API delegiert. Die Magic Leap wird als Interaktionsmöglichkeiten für den Nutzer verwendet, nimmt die RGB-Fotos der Umgebung auf, liefert diese an die REST-API und zeigt die Ergebnisse und Zwischenstände der Objekterkennung mit UI Elementen in der Szene an. Siehe Abbildung 10

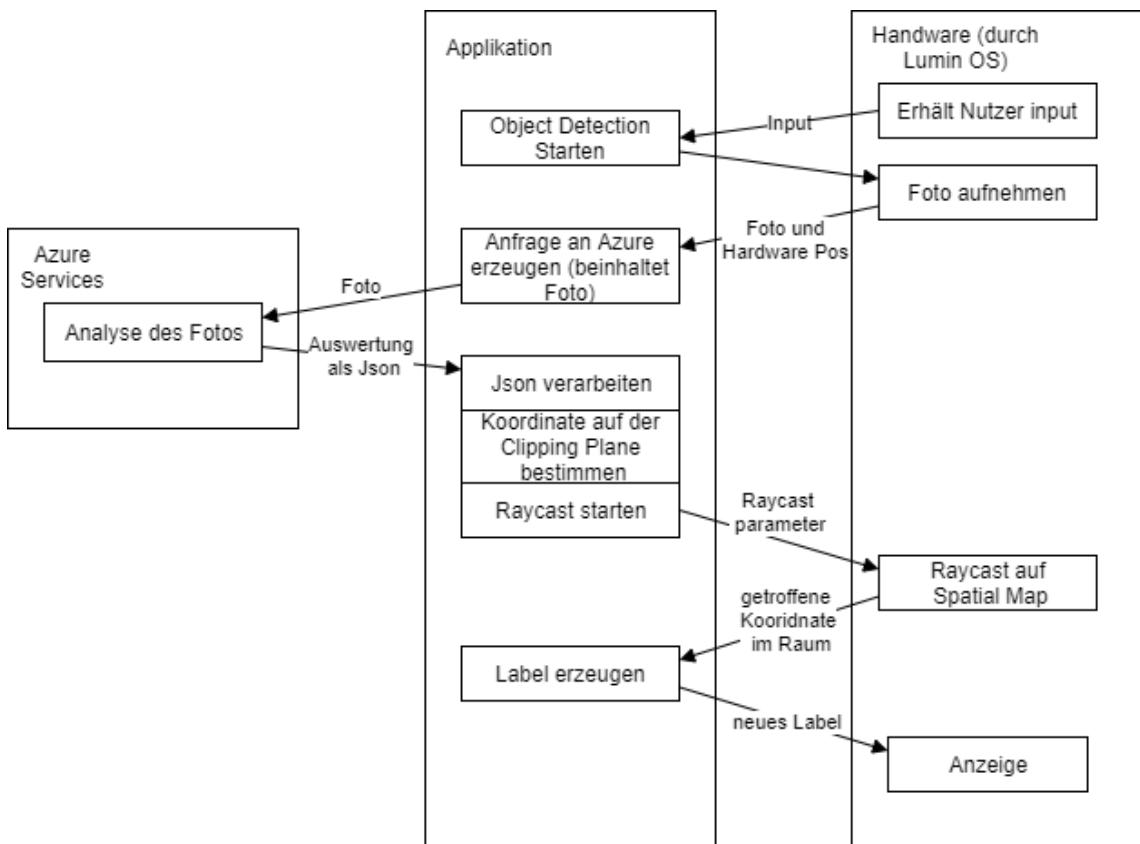


Abbildung 10: Diagramm der Architektur inklusive Bearbeitungsschritte und Informationsweitergabe.

Das Projekt wurde in Unity umgesetzt und für die Magic Leap AR Brille entwickelt. Es wurde das Unity Project Template von Magic Leap verwendet.(MagicLeap 2020a)

Zusätzlich werden einige vorgefertigte Klassen von Magic Leap verwendet. Dazu gehören *MLInput*, *MLCamera*, *MLRaycast*, *MLPrivilegeRequestBehavior* und *MLSpatialMapper*. Diese Klassen greifen auf Funktionalitäten des Lumin OS zu.

Die benötigten Funktionalitäten für die Integration der Object Detection mit der REST-API wurde in mehreren Script Klassen umgesetzt. Der Großteil der Scripts verhält sich wie Singletons. Sie existieren nur einmalig in der Szene.

Das Klassendiagramm auf Abbildung 11 zeigt die Scripts und ihre Relationen zueinander.

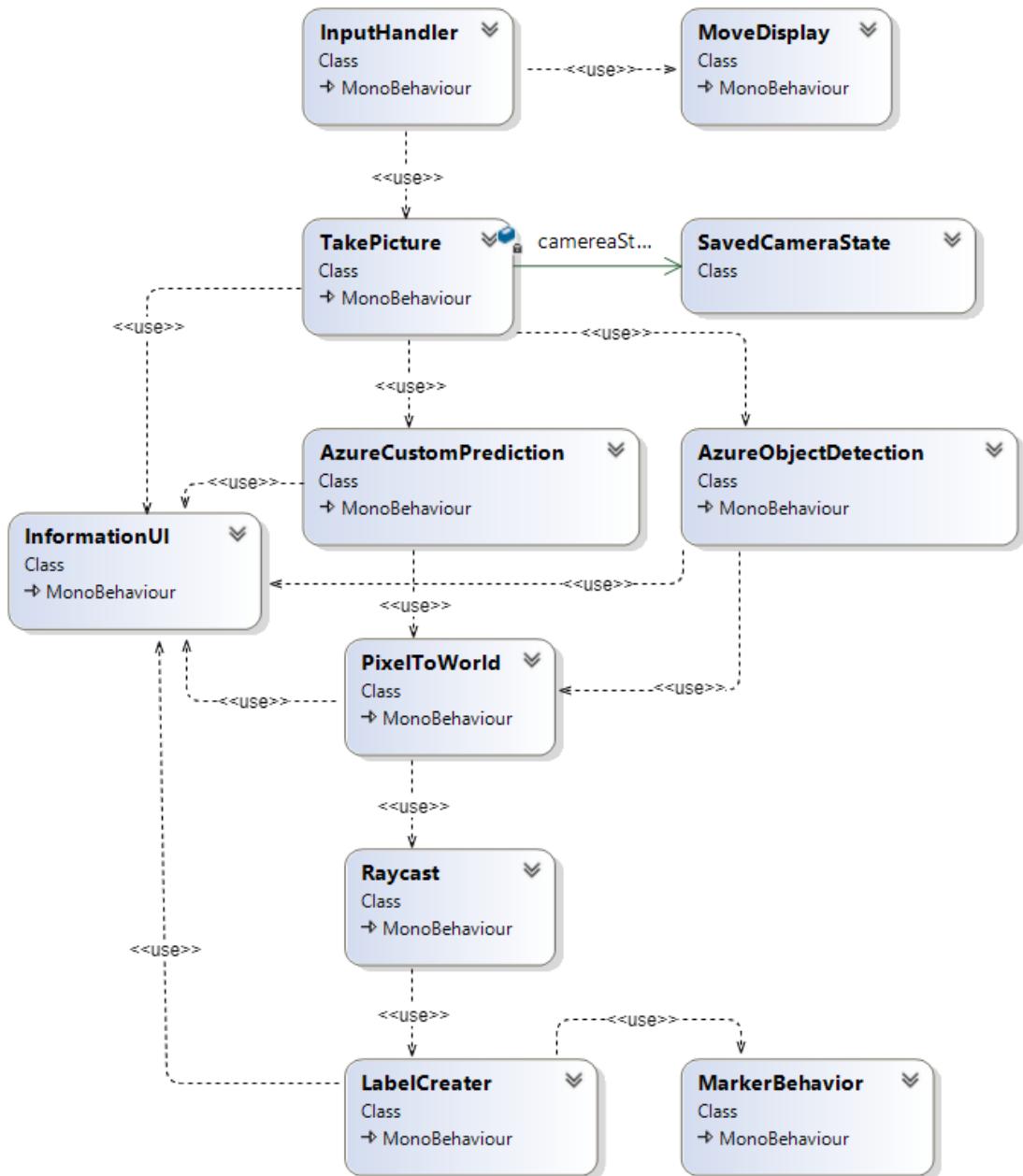


Abbildung 11: Klassendiagramme der Scripts

Die Klassen *InputHandler*, *MoveDisplay*, *LabelCreator*, *InformationUI* und *MarkerBehavior* sind für die Interaktion mit dem Nutzer zuständig.

TakePicture, *SavedCameraState*, *AzureCustomPrediction*, *AzureObjectDetection*, *PixelToWorld* und *Raycast* führen das Erkennen von Objekten anhand eines aufgenommenen Fotos durch und bestimmt eine Position für das Objekt in der 3D Umgebung. Der Prozess wird durch den *InputHandler* gestartet.

Wurde ein Objekt erkannt und eine Position auf dem Mesh der Umgebung bestimmt, wird der *LabelCreator* aufgerufen. Er erzeugt das Label mit dem entsprechenden Text.

MarkerBehavior ist eine Script, das jedes Label-GameObject hat, das erzeugt wird. Über das *MarkerBehavior* kann der Schriftzug des Labels angepasst werden.

4.3 Interaktion

InputHandler verarbeitet den Input des Nutzers und startet entsprechende Aktionen durch die *MoveDisplay* und *TakePicture* Klassen. *MLInput* ist eine vorgefertigte Klasse von Magic Leap. Sie stellt Informationen über den Zustand des Controllers zur Verfügung. *InputHandler* überwacht den Controller und startet die Aktionen, wenn die entsprechende Taste gedrückt wurde.

- Trigger: Objekt Erkennung starten mit TakePicture
- Home Button: UI Element mittig vor das Display setzen.
- Bumper: Labels verstecken
- Bumper halten: zuletzt erzeugten Label entfernen

Das UI Element wird von *InformationUI* gesteuert. *TakePicture* nutzt *InformationUI*, um das zuletzt aufgenommene Foto anzuzeigen. *AzureCustomPrediction*, *AzureObjectDetection* und *PixelToWorld* dokumentieren ihre Arbeitsschritte mit dem UI Element und der *LabelCreator* lässt eine Liste aller Labels anzeigen, die in der Szene existieren. Siehe Abbildung 12.

Der *LabelCreator* ist für das Erstellen der Labels verantwortlich und sorgt dafür, dass die Labels für den Nutzer lesbar sind. Dafür werden die Labels in Richtung der Kamera ausgerichtet und mitgeführt. Des Weiteren kann der *LabelCreator* Labels verstecken und entfernen.

Neben dem UI Element und den Labels wird auch ein Mesh angezeigt, das die Spatial Map der Umgebung wiedergibt. Das Spatial Mapping wird von Lumin OS durchgeführt und das Mesh wird durch die Klasse *MLSpatialMapper* von MagicLeap erzeugt. Siehe Abbildung 12.

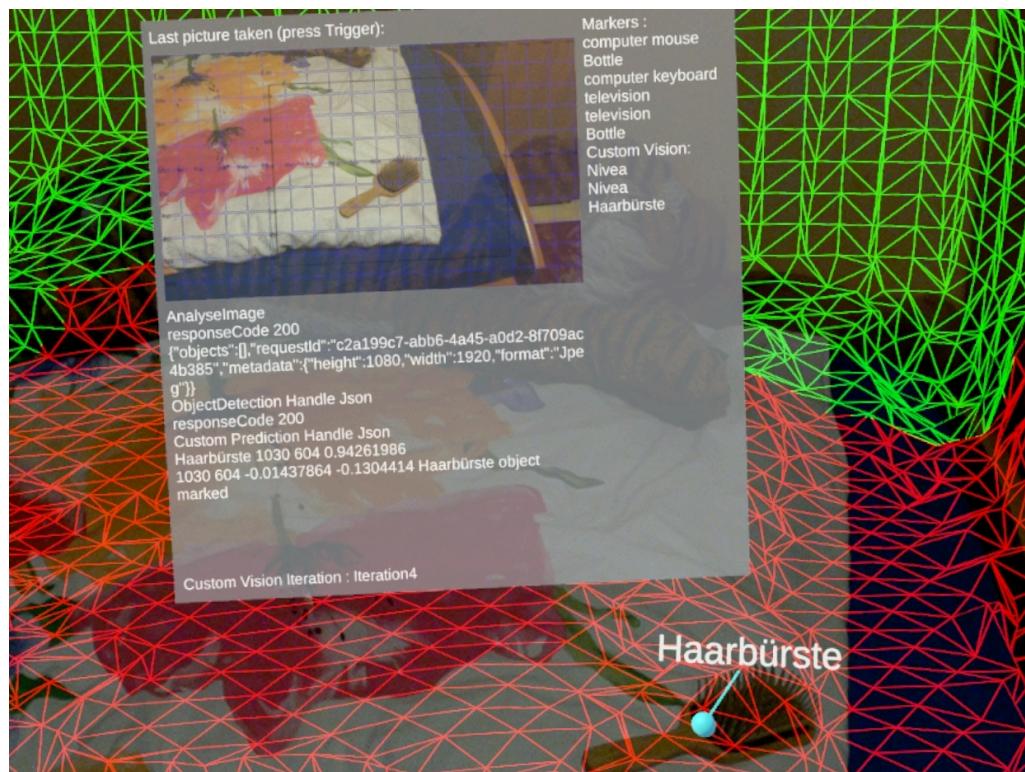


Abbildung 12: Ausgabe

4.4 Implementierung der Objekterkennung

Im Folgenden werden die Scripts besprochen, welche für die Objekterkennung zuständig sind.

4.5 Ein Foto aufnehmen

Das Script *TakePicture* implementiert das Aufnehmen eines Fotos. Dabei wird *MLCamera* von Magic Leap genutzt, um die Kamera der AR Brille anzusteuern. Wenn die Applikation gestartet wird, stellt dieses Script sicher, dass die Applikation die benötigte Permission hat die Kamera zu nutzen. Danach verbindet sich das Script über *MLKamera* mit der Kamera-Ressource. Die Kamera-Ressource wird wieder abgegeben, wenn die Applikation terminiert oder pausiert wird.

Wenn die Methode *TakeImage* aufgerufen wird, startet der Prozess der Objekterkennung. Das Aufnehmen der Fotos geschieht asynchron. Für jedes Foto wird ein Thread erzeugt, in dem *MLCamera* ein Foto aufnimmt. In diesem Thread wird zusätzlich die aktuelle Position der Unity Kamera als *SavedCameraState* gespeichert.

Die Methode *OnCaptureRawImageComplete* wird von *MLCamera* aufgerufen, wenn das Foto fertig ist. Die Daten des Bildes und der *SavedCameraState* werden, an die Scripts *AzureObjectDetection* und *AzureCustomPrediction*, weitergegeben. Dort wird die Analyse der Bilder gestartet.

4.5.1 Object Detection

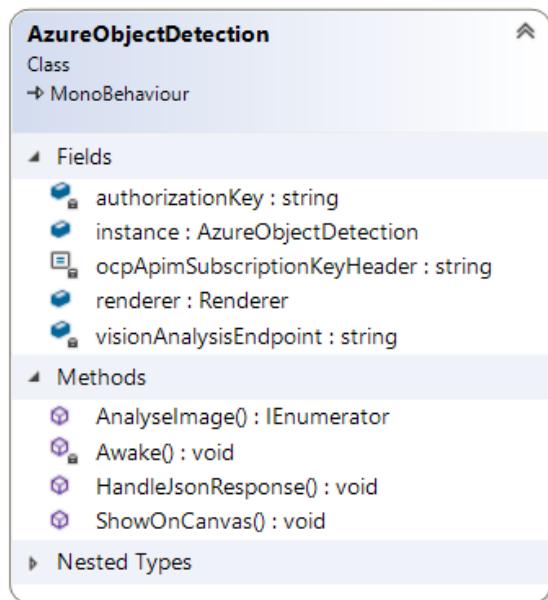


Abbildung 13: Klassendiagramm von *AzureObjectDetection*

In der Methode *AnalyseImage* von *AzureObjectDetection* wird ein Web Request zusammengestellt, um die Azure REST-API anzufragen. Der Request enthält eine Authentifizierung für die API und das zu analysierende Foto.

Der Webrequest wird verschickt und auf die Antwort gewartet. Wenn die Antwort eintrifft, wird anhand des ResponseCodes geprüft, ob es bei dem Request einen Fehler gab. Beispielsweise kann die Internetverbindung gestört sein oder die Authentifizierung abgelehnt werden. Wenn es keinen Fehler gab, wurde eine Json-Datei bei der Antwort mitgeschickt.

Darin wird für jedes gefundene Objekt auf dem Foto eine Bezeichnung (Klasse) und eine Bounding Box angegeben.

Die Json-Datei wird in *HandleJsonResponse* verarbeitet. Für den erwarteten Aufbau der Datei wurden drei Klassen geschrieben. Der Json String wird mit *JsonUtility* in ein *DetectionResponse* Object umgewandelt. Dabei werden alle gefundenen Foto-Objekte in einer Liste von *DetectedObjects* abgelegt. Siehe Abbildung 14. (Unity 2020b)

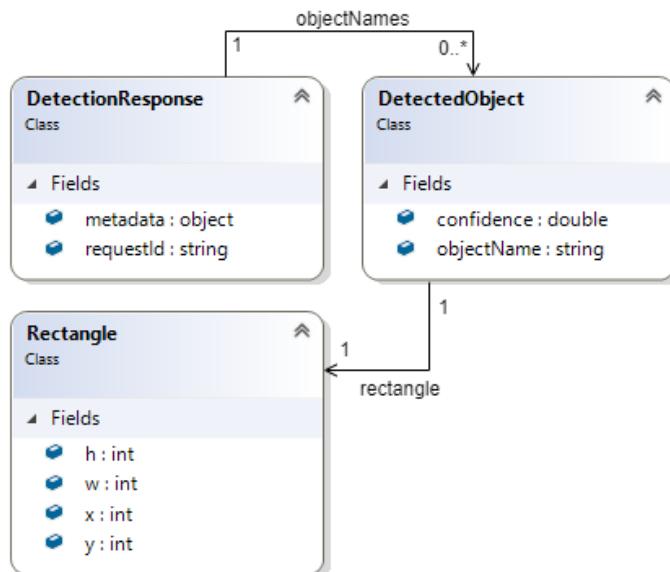


Abbildung 14: Klassendiagramm für die umwandeln der Json Datei in Objekte.

Die gefundenen Objekte sollen im 3D Raum mit einem Label gekennzeichnet werden. Dafür wird für jedes *DetectedObject* die Methode *Cast* von der Klasse *PixelToWorld* aufgerufen. Der Methode wird der Mittelpunkt der BoundingBox als u,v Foto-Koordinaten für das *DetectedObject* übergeben. Siehe Abbildung ??.

```

1 public void HandleJsonResponse(System.String jsonResponse,
2     SavedCameraState cpos)
3 {
4     float starttime = Time.time;
5     jsonResponse = jsonResponse.Replace("object", "objectName");
6     //c# doesn't like "public string object"
7     DetectionResponse det = new DetectionResponse();
8     det = JsonUtility.FromJson<DetectionResponse>(jsonResponse);
9     foreach (DetectedObject obj in det.objectNames)
10    {
11        Debug.Log(obj.objectName);
12        int x = obj.rectangle.x + (obj.rectangle.w / 2);
13        int y = obj.rectangle.y + (obj.rectangle.h / 2);
14        PixelToWorld.instance.Cast(x, y, cpos, obj.objectName);
15    }
  
```

4.5.2 Von dem Foto zum 3D Raum

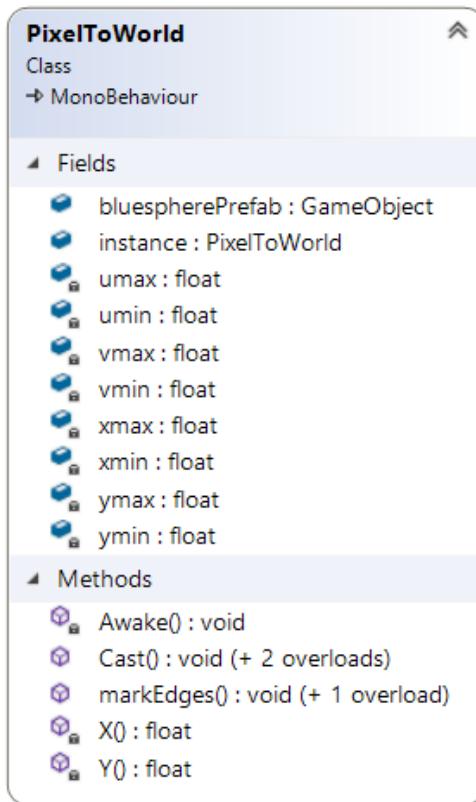


Abbildung 15: Klassendiagramm von *PixelToWorld*

Ein gefundenes Foto-Objekt soll in der 3D Abbildung der realen Welt lokalisiert werden. Dafür nutzt die Methode `Cast` die u,v Foto-Koordinaten des Objekts und einen `SavedCameraState`. Der `SavedCameraState` beschreibt die Position der Unity Kamera zu dem Zeitpunkt als das Foto aufgenommen wurde. `SavedCameraState` beinhaltete die `cameraToWorldMatrix` und den Ursprung der Kamera.

Das Foto kann mit dem Display und somit mit der Clipping Plane der Hauptkamera approximiert werden. Die u,v Foto-Koordinaten werden zunächst in x,y,z Koordinaten in dem Camera Space umgewandelt. Der z Anteil gibt die Entfernung von dem Ursprung der Kamera in Blickrichtung an. Dabei befinden sich Punkte mit einer Entfernung von 0.4 Einheiten auf der Clipping Plane. In dem Camera Space mit $z = -0.4$ angegeben.

Die x und y Dimensionen beschreiben die Achsen, die horizontal und vertikal zur Clipping Plane verlaufen. Mit dem festgelegten $z = -0.4$, kann jeder Punkt auf der Clipping Plane durch x und y angegeben werden. Dazu gehören auch Punkte die außerhalb des View Frustum liegen.

Es wurden Werte für x und y ausprobiert, mit denen die Ränder des Fotos auf der Clipping Plane angegeben werden können. Dabei wurde auf die unterschiedlichen Seitenverhältnisse des Fotos und des Displays geachtet. Darüber hinaus ist der Bildausschnitt des Displays kleiner. Daher liegen die Ränder des Fotos außerhalb des View Frustum.

Sind diese x und y Werte bekannt, ergibt sich für die Achsen jeweils ein Intervall, die kombiniert alle Foto-Koordinaten auf die Clipping Plane abbilden können. Die Intervalle lauten: $[-0.2949, 0.2295]$ für x und $[0.1546, -0.1507]$ für y. Mit den Intervallen wird die Position und Skalierung des Fotos in Relation zu dem Display – und der Hauptkamera –

berücksichtigt. Siehe Kapitel 4.6 für die Entwicklung der Cast Methode und die Ermittlung der Intervallwerte.

Es werden zwei lineare Funktionen aufgestellt:

- Die Funktion X bildet das Intervall für u [0,1920] auf das Intervall für x [-0.2949,0.2295] ab.
- Die Funktion Y bildet das Intervall für v [0,1080] auf das Intervall für y [0.1546,-0.1507] ab.

Die Funktionen sind folgendermaßen umgesetzt:

```

1 //Picture u and v ranges
2 private float umin = 0;//left
3 private float umax = 1920;//right
4 private float vmin = 0;//up
5 private float vmax = 1080;//down
6 //Offset Vektor x and y ranges
7 private float xmin = -0.2949F;//left
8 private float xmax = 0.230F;//right
9 private float ymin = 0.1546F;//up
10 private float ymax = -0.1507F;//down
11
12 private float X(float u)
13 {
14     float slope = ((xmax - xmin) / (umax - umin));
15     float b = xmin - slope * umin;
16     return slope * u + b;
17 }
18 private float Y(float v)
19 {
20     float slope = ((ymax - ymin) / (vmax - vmin));
21     float b = ymin - slope * vmin;
22     return slope * v + b;
23 }
```

Mit den Funktionen wird eine Position im Camera Space, auf der Clipping Plane, für u,v berechnet. Diese Position wird dann, mithilfe der *cameraToWorldMatrix* des *SavedCameraState*, in eine Position p des globalen Koordinatensystems umgewandelt. Damit wird die Position und Rotation der Kamera – und somit des Fotos – in der 3D Szene berücksichtigt.

```

1 public void Cast(float u, float v, SavedCameraState cpos, GameObject
    clippingPlaneMarker, string objectName, bool showClippingPlane, int
    material)
2 {
3     //scale v,v to x,y range
4     Vector3 offset = new Vector3(X(u), Y(v), -0.4F);
5     Vector3 p = cpos.cameratoWorldMatrix.MultiplyPoint(offset);
6     Raycast.instance.StartCast(Raycast.instance.CreateRaycastParams
        (cpos.ctransform, p), objectName, material);
7     if (showClippingPlane)// show point on clipping plane
8     {
9         GameObject sphere2 = Instantiate(clippingPlaneMarker, p
            , Quaternion.identity);
10    }
11 }
```

4.5.3 Raycast

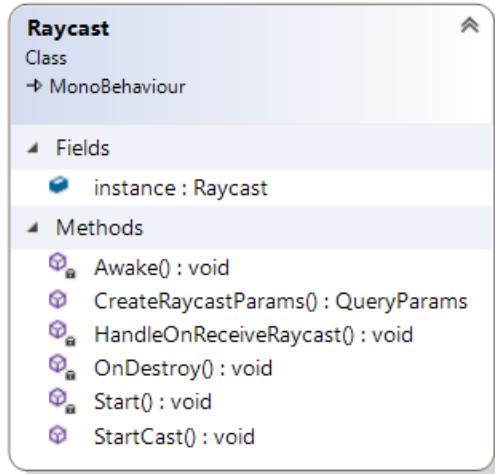


Abbildung 16: Klassendiagramm von Raycast

Als Nächstes wird ein Raycast durch den Ursprung der Kamera und die Position p gesendet. *MLRaycast* wird genutzt, um einen Schnittpunkt mit der Rekonstitution der Welt von Lumin OS zu bestimmen. Die Stelle, die der Raycast trifft, beschreibt die Position des *DetectedObject* im 3D Raum.

Für den *MLRaycast* werden zwei Parameter benötigt:

- Ein *QueryParams* Objekt, das Ursprung und Richtung für den Raycast beinhaltet.
 - Ursprung: Kamera-Ursprung aus *SavedCameraState*
 - Richtung: Richtungsvektor von dem Kamera-Ursprung zu der Position p
- Eine Methode die aufgerufen wird, wenn der Raycast fertig ist.
 - Callback Methode: *HandleOnRecieveRaycast*

```

1 public MLRaycast.QueryParams CreateRaycastParams(Transform ctransform,
2     Vector3 target)
3 {
4     MLRaycast.QueryParams _raycastParams = new MLRaycast.
5         QueryParams
6     {
7         // Update the parameters with our Camera's transform
8         Position = ctransform.position,
9         Direction = target - ctransform.position,
10        UpVector = ctransform.up,
11        // Provide a size of our raycasting array (1x1)
12        Width = 1,
13        Height = 1
14    };
15    return _raycastParams;
16 }
  
```

Wenn der Raycast fertig ist, wird die Methode *HandleOnRecieveRaycast* aufgerufen. Der Parameter *point* beinhaltet dabei die getroffene Stelle der AR Umgebung. Diese wird an die Methode *CreateMarker* von der Klasse *LabelCreator* weitergegeben.

4.5.4 LabelCreator

`CreateMarker` erhält den Punkt *point*, der getroffen wurde und die Bezeichnung für das *DetectedObject*. An der Position von *point* wird ein Prefab GameObject instanziert, das als Markierung für das *DetectedObject* in der 3D Umgebung dient.

Das Prefab besteht aus einer Kugel und einem Schriftzug, der den Namen des *DetectedObject* anzeigen soll. Dem neu instanzierten GameObject wird die Bezeichnung des *DetectedObject* als Schriftzug zugewiesen. Siehe Abbildung 17.

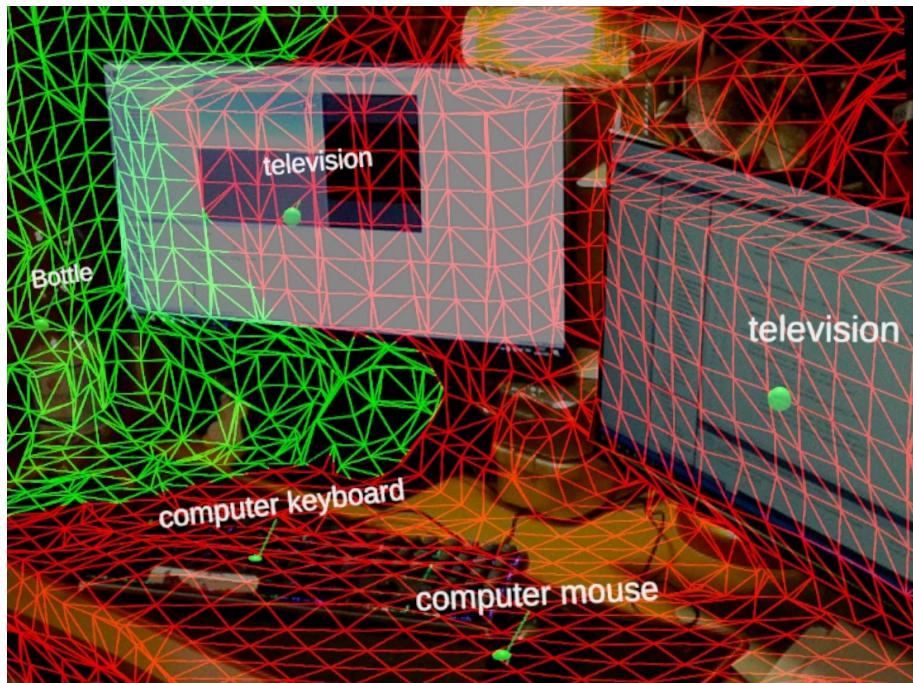


Abbildung 17: Labels in der Szene

Wenn ein Label nahe eines anderen Labels erzeugt werden soll, das denselben Schriftzug hat, wird davon ausgegangen, dass ein Objekt der realen Welt erneut erkannt wurde. Daher wird kein neues Label erstellt, sondern das alte Label modifiziert. Die Positionen, an denen das Objekt in der Szene lokalisiert wurde, werden mit jeweils einer grauen Sphäre markiert und das Label wird in den Mittelpunkt der grauen Sphären gesetzt. So wird die Position des Objektes genauer, wenn es häufiger erkannt wurde. Siehe Abbildungen 18 und 19.

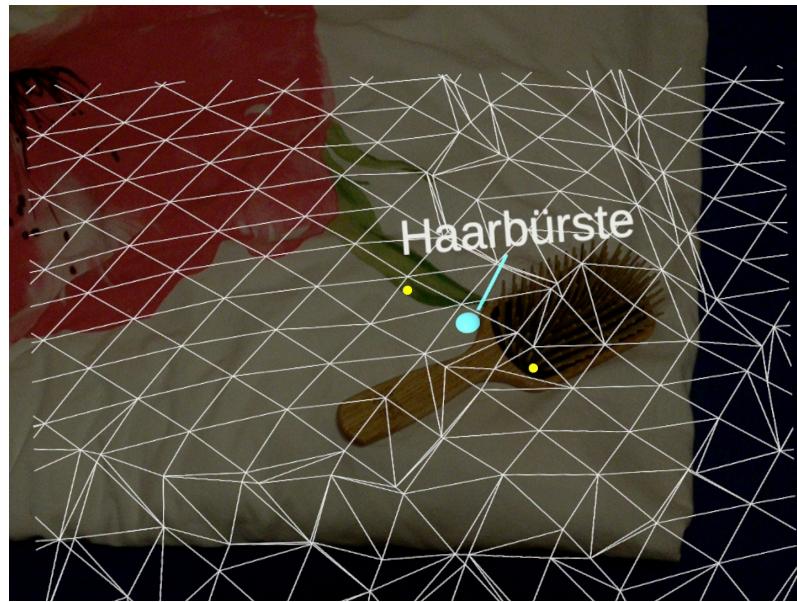


Abbildung 18: Haarbürste zwei mal erkannt. Graue Sphären hier in Gelb markiert.

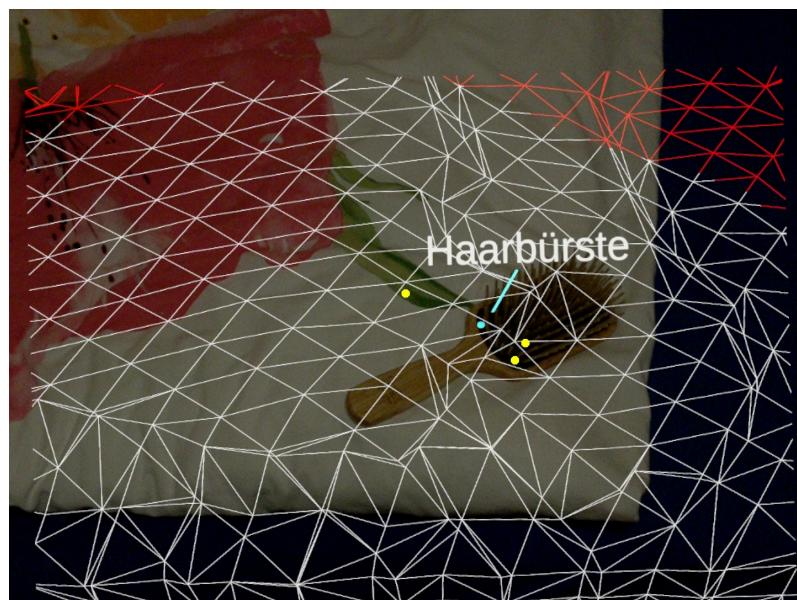


Abbildung 19: Haarbürste drei mal erkannt. Graue Sphären hier in Gelb markiert.

Die erzeugten Labels werden in den Listen *MarkerCustomPrediction* und *MarkerObjectDetection* gespeichert, je nachdem welcher Azure Service verwendet wurde. Zu jedem Frame der Applikation werden die Labels in den Listen zur Kamera hingedreht, sodass sie rechtwinklig zur Kamera ausgerichtet sind. Als Resultat werden die Labels der Bewegung der Kamera nachgeführt. Solange das Label in dem View Frustum liegt, kann der Nutzer es lesen. Siehe Anhang 8.1.

4.5.5 Azure Custom Vision

Neben der Bildanalyse mit Azure Object Detection wird auch Azure Custom Vision verwendet. Die AI wurde über die Webseite trainiert.

Die Anfrage an den Service geschieht in der Klasse *AzureCustomPrediction*. Ähnlich wie bei *AzureObjectDetection* wird ein Webrequest erstellt mit einem Authorization Key und

einem Foto. Die Anfragen an die beiden REST-APIs werden parallel in unterschiedlichen Threads erstellt und bearbeitet.

In der Antwort wird eine Json Datei zurückgeschickt, die die gefundenen Objekte angibt. Da die Json Datei ein etwas anderes Format hat, wurde eine eigene *HandleJsonResponse* Methode dafür geschrieben.

Azure Custom Vision gibt für jedes *DetectedObject* eine Probability an. Diese gibt das Vertrauen des Modells darin an, dass das Objekt korrekt erkannt wurde.

Es wurde ein Schwellwert definiert, der entscheidet wie hoch die Probability mindestens sein muss, um das Objekt zu akzeptieren und in der Szene zu markieren.

Für jedes akzeptierte Objekt wird die Methode *Cast* von *PixelToWorld* aufgerufen, um das Objekt in der realen Welt zu lokalisieren und zu markieren.

Das Trainieren Es wurde probiert das Custom Vision Modell auf drei unterschiedliche Objekte zu trainieren. Dabei wurden vier Iterationen erstellt.

Iteration 1:

Zunächst wurde probiert Tuben von Acrylfarbe zu erkennen. In der Nutzung dieser Iteration traten viele fehlerhaften Objekterkennungen auf. Es wurden Acrylfarben an Stellen erkannt, an denen es keine gab. Siehe Abbildung 20.

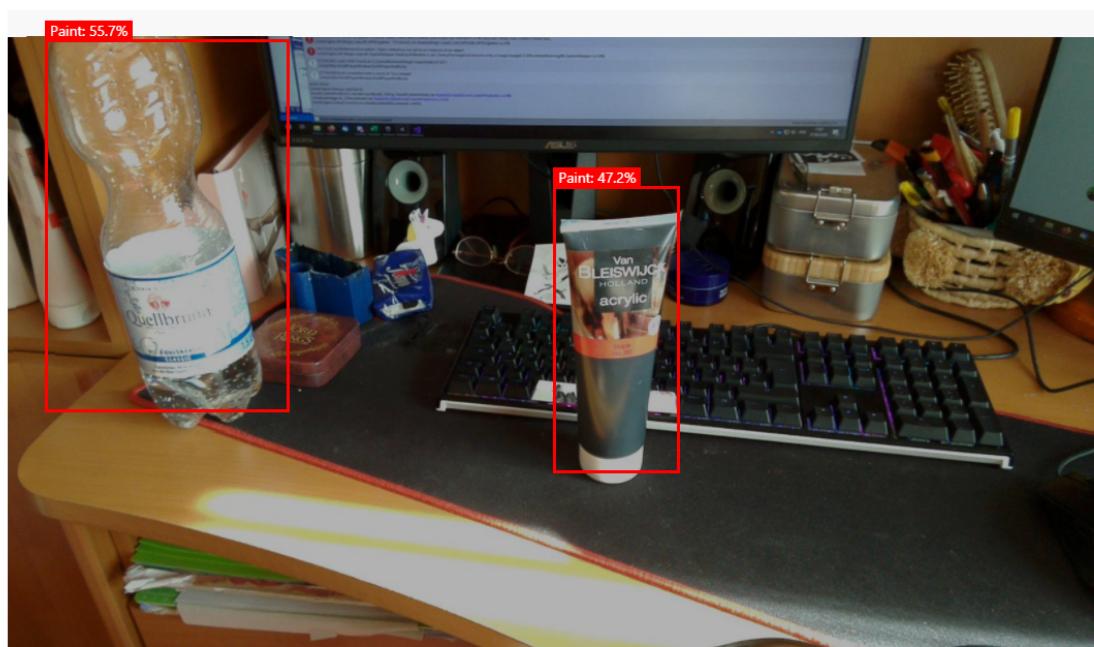


Abbildung 20: Iteration 1 Analysebeispiel. Eine Wasserflasche und eine Farbtube wurden beide als Farbtuben erkannt. Die Wasserflasche wurde mit einer Probability von 55.7 Prozent markiert, während die tatsächliche Farbtube eine Probability von 47.5 Prozent hat.

Die Evaluierung des Modelles lautet: 66.7% Precision, 66.7% Recall, 89.7% mean Average Precision. In einer Testphase konnte das Modell 66,7% der Nivea Dosen auf den Bildern erkennen (Recall) und 66,7% der Objekte die als Nivea Dosen markiert wurden, waren tatsächlich Nivea Dosen (Precision).

- Precision - die Wahrscheinlichkeit das ein gefundenes Objekt, tatsächlich der angegeben Klasse angehört. (Die Wahrscheinlichkeit das es kein false positive ist.)

- Recall - aus einer Menge an Objekten die einer Klasse angehören, der Prozentsatz an Objekten, die das Modell korrekt lokalisieren und klassifizieren konnte.
- m.a.p (mean Average Precision) - eine Gesamtwertung für die Evaluierung basierend auf Precision und Recall.

Iteration 2 In der zweiten Iteration wurde probiert, das Modell darauf zu trainieren, eine blaue Dose von Nivea Hautcreme zu erkennen. Form und Farbe der Dose sind sehr simpel, daher wurde davon ausgegangen, dass dieser Gegenstand leichter zu erkennen ist. Die berechnete Prediction des Modells lag bei 80 Prozent. Wenn eine Nivea Dose erkannt wurde, ist das Modell sich zu 80 Prozent sicher, dass es sich tatsächlich um eine Nivea Dose handelt.

Die Evaluierung des Modells lautet: 80% Precision, 100% Recall, 100% mean Average Precision. In einer Testphase konnte das Modell alle auf den Bildern erhaltenen Nivea Dosen zu erkennen (100% Recall), es hat jedoch auch Objekte fälschlicherweise als Dosen markiert (80% Precision).

Trotzdem wurden in vielen Fotos fälschlicherweise Nivea Dosen erkannt.

Iteration 3 In der dritten Iteration wurde versucht die vorherige Iteration zu verbessern. Es wurden ausgewählte Trainingsfotos entfernt, die die Dose von einem seitlichen Winkel zeigten. Die Erwartung war, dass die Detektion der Dose aus dem Blickwinkel von oben konsistenter wird. Zusätzlich wurden mehr Fotos von der Dose auf unterschiedlich gefärbten und gemusterten Untergründen hinzugefügt.

Die Precision sank auf 75 Prozent.

Die Evaluierung des Modells lautet: 75% Precision, 100% Recall, 100% mean Average Precision. Dieses Modell hat in der Testphase häufiger Objekte fälschlicherweise als Dosen markiert.

Iteration 4 In der vierten Iteration wurden zwei Fotos von der Nivea Dose entfernt, was die Precision auf 100 Prozent steigen ließ. In der Umsetzung mit der Magic Leap Anwendung wurden trotzdem häufig Objekte fälschlicherweise als Nivea Dose markiert.

Neben der Dose wurde diese Iteration darauf trainiert eine bestimmte Holzhaarbürste zu erkennen. Aufgrund von dem komplexeren, und markanten Aussehen der Bürste wurde davon ausgegangen, dass die Bürste besser von anderen Objekten zu unterscheiden ist. Die Bürste wurde nur mit den Borsten nach oben fotografiert.

Die Evaluierung des Modells lautet: 100% Precision, 100% Recall, 100% mean Average Precision.

Objekte, die mit einer Wahrscheinlichkeit von über 80 Prozent als Haarbürsten markiert wurden, waren in der Anwendung tatsächlich Haarbürsten. Durch das Setzen dieses Schwellenwertes wurden keine Objekte fälschlicherweise als Haarbürste erkannt. Siehe Abbildung 22.



Abbildung 21: Iteration 4 Analysebeispiel. Die tatsächliche Haarbürste wurde mit einer Probability von 92 Prozent erkannt. Eine Blume auf einer Decke wurde zu 71,3 Prozent als Haarbürste erkannt.

Wenn die Haarbürste in dem Foto kleiner abgebildet ist, wird sie mit einer geringeren Probability erkannt. Dadurch kann die Bürste nicht mehr durch einen Schwellwert von anderen Objekten unterscheiden werden.



Abbildung 22: Iteration 4 Analysebeispiel. Die tatsächliche Haarbürste wurde mit einer Probability von 40,4 Prozent erkannt.

Iteration 5 Iteration 5 ist eine Variation von Iteration 4. Die Objekterkennung der Nivea Dose entfernt. Die Iteration ist nur noch darauf trainiert die Haarbürste zu erkennen. Die Evaluierung des Modells lautet: 100% Precision, 100% Recall, 100% mean Average Precision.

Iteration 6 In der Iteration 6 wurde die Erkennung der Haarbürste verbessert. Den Empfehlungen von Azure folgend, wurde die Menge an Trainingsbildern auf 51 erhöht. Auf den Trainingsbildern ist die Haarbürste teilweise recht klein zu sehen. Siehe Anhang 8.2. Dieser Iteration wurde eine Stunde Zeit gelassen, um das Training durchzuführen. Vorherige Iteration hatten ca. 10 Minuten.

Die Evaluierung des Modells lautet: 100% Precision, 100% Recall, 100% mean Average Precision.

Haarbürsten, die kleiner in einem Foto sind, konnten von dieser Iteration erkannt werden. Siehe Abbildung 23. Mit den vorherigen Iterationen war das nicht möglich. Die Haarbürste musste immer recht groß im Bild sein.

In den Bildern, die von Iteration 6 analysiert wurden, waren alle Objekte, die das Model mit einer Wahrscheinlichkeit von über 50 Prozent als Haarbürste markiert hatte, tatsächlich Haarbürsten.

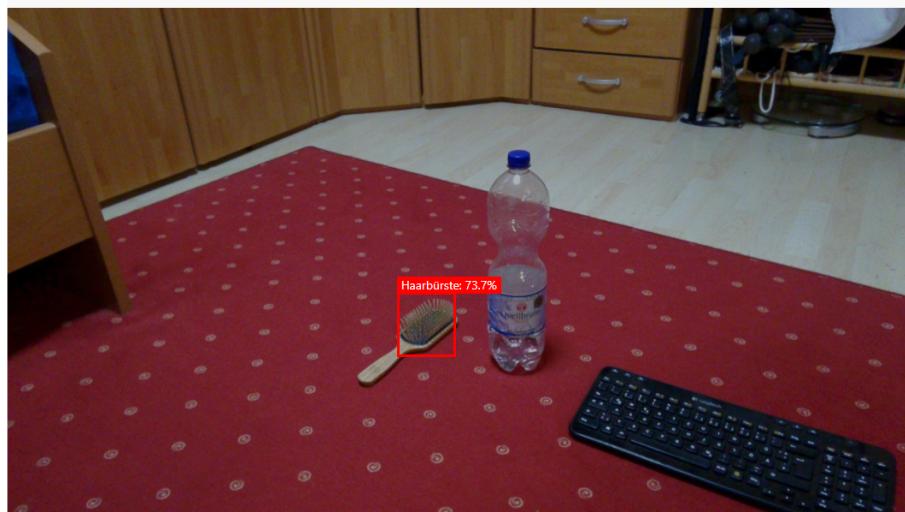


Abbildung 23: Iteration 6 Analysebeispiel. Haarbürste mit einer Probability von 73,7 Prozent erkannt.

4.6 Entwicklung der Foto-Repräsentation

Um die u,v Foto-Koordinate eines gefundenen Objektes auf der Clipping Plane der Kamera zu lokalisieren, wurden ein paar Herangehensweisen ausprobiert.

Das Ziel ist das Setzen einer Markierung in den 3D Raum, basierend auf den Foto-Koordinaten. Das Foto beinhaltet keine Information über die Entfernung zu dem Objekt. Dafür muss ein Raycast durchgeführt werden.

Mit einer Repräsentation des Fotos in dem 3D Raum ist es möglich diesen Raycast durchzuführen. Dazu muss das Foto nicht tatsächlich in dem 3D Raum vorhanden sein. Es muss jedoch mit dem Input der Foto-Koordinaten ein Output der x,y,z-Koordinaten in dem 3D Raum erzeugt werden, mit dem der Raycast durchgeführt werden kann.

Die Position des Fotos hängt mit der Kamera zusammen, daher kann das Foto durch den Camera Space simuliert werden. Als Erstes wurde probiert ein Sphären-Objekt an eine gezielte Koordinate des Camera Space zu bewegen.

Wenn die Kamera am Ursprung des globalen Koordinatensystems liegt und eine neutrale Rotation hat, stimmt der Camera Space mit dem globalen Koordinatensystem überein. Die Sphäre wurde in der Szene per Hand bewegt, um markante Koordinaten des Camera Space abzulesen. Siehe Abbildung 24.

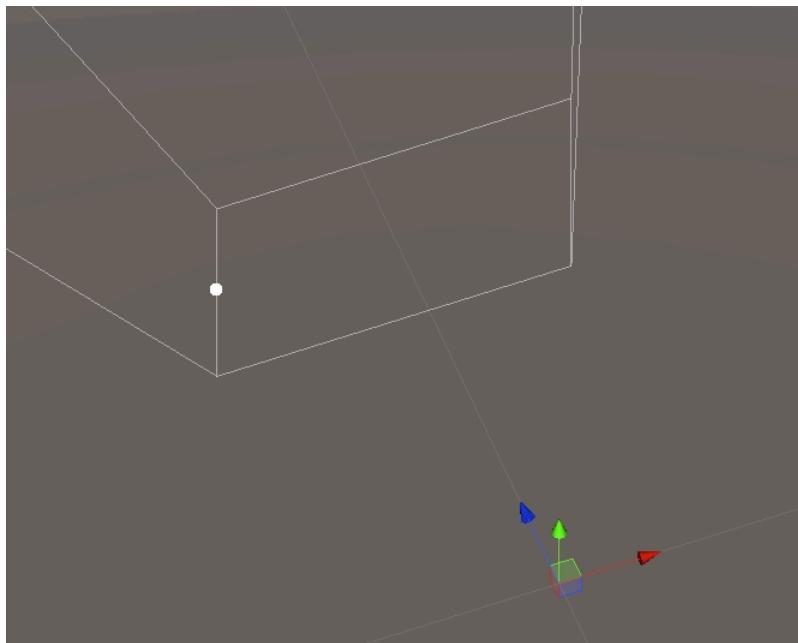


Abbildung 24: Die weiße Sphäre liegt auf dem linken Rand der Clipping Plane.

Dabei wurden folgende Camera Space Koordinaten gefunden:

- Near Clipping Plane bei $z = -0.37$
- linker Rand bei $x = -0.153$
- rechter Rand bei $x = 0.153$
- oberer Rand bei $y = 0.1147$
- unterer Rand bei $y = -0.1147$

Die x und y Koordinaten hängen von den u,v Koordinaten des Fotos ab. Es wurden lineare Funktionen aufgestellt um u,v auf x,y abzubilden. Diese Abbildung dient als Repräsentation des Fotos im 3D Raum, unter Berücksichtigung der Position und Skalierung des Fotos im Verhältnis zu der Kamera.

Dann wurde getestet wie genau *DetectedObjects* in der AR Umgebung lokalisiert werden. Es wurden testweise Fotos aufgenommen, analysiert und die *DetectedObjects* markiert. Die entstandenen Markierungen lagen in Sichtfeld, jedoch nicht an den erwarteten Stellen.

Um dem Problem auf den Grund zu gehen, wurde ein UI Objekt erstellt, das ein aufgenommenes Foto während der Laufzeit anzeigt. Das Foto wurde dann mit dem Display verglichen. Dabei fiel auf, dass sie ein unterschiedliches Seitenverhältnis haben und das Display einen kleinen Bildausschnitt zeigt.

Es gibt zwei Möglichkeiten die Unterschiede zwischen Foto und Display auszugleichen. Entweder wird das Foto auf das Display zugeschnitten oder das gesamte Foto wird verwendet. Im zweiten Fall würden auch Objekte erkannt, die außerhalb des Sichtfeldes liegen. Es wurde die Entscheidung getroffen das Foto zuzuschneiden. Damit gibt es ein besseres Feedback für den Nutzer, wenn ein Objekt gefunden wurde.

Das Zuschneiden wurde realisiert, indem die Intervalle für u und v der Abbildungsfunktionen stärker eingegrenzt wurden. Alle Objekte, die außerhalb der Intervalle liegen, werden ignoriert. Um die Intervalle zu bestimmen wurde dem Fotoanzeige-UI-Element ein Gitter hinzugefügt. Mit dem Gitter kann die u,v Position von beliebigen Stellen des Fotos abgelesen werden. Durch Aufnehmen von Fotos und Vergleichen mit dem Sichtfeld des Displays wurde abgelesen, bei welcher u,v Position des Fotos die Ecken des Displays zu finden sind. Die Intervalle wurden dem entsprechend eingegrenzt.

Mit den durchführten Veränderungen der Intervalle konnten *DetectedObjects* korrekt in der Umgebung lokalisiert werden. Jedoch wurden sehr häufig Objekte nicht markiert, obwohl sie im Sichtfeld des Nutzers lagen, weil deren Mittelpunkt außerhalb eines Intervalls lag.

Daher wurde entschieden die zweite Möglichkeit zu implementieren und das gesamte Foto zu verwenden und Objekte auch zu markieren, wenn sie komplett außerhalb des Sichtfeldes liegen. Dafür wurden die Intervalle für u und v wieder auf die ursprünglichen Werte – [0,1920] und [0,1080] – gesetzt. Die Intervalle für x und y mussten vergrößert werden.

Um die x und y Intervalle bestimmen zu können, wurde das Fotoanzeige-UI-Element Parallel zu der Clipping Plane gelegt. Das Element folgt den Bewegungen der Kamera und liegt möglichst nah an der Near Clipping Plane. Das Display der Magic Leap Brille zeigt selbst solide Objekte leicht durchsichtig an. Das wurde genutzt, um Fotos aufzunehmen, mit dem UI Element anzuzeigen und mit der realen Welt zu vergleichen. Durch Ausprobieren wurde das UI Element so skaliert und verschoben, dass das angezeigte Foto mit der realen Welt so weit wie möglich übereinstimmt. Siehe Abbildungen 25 und 26.

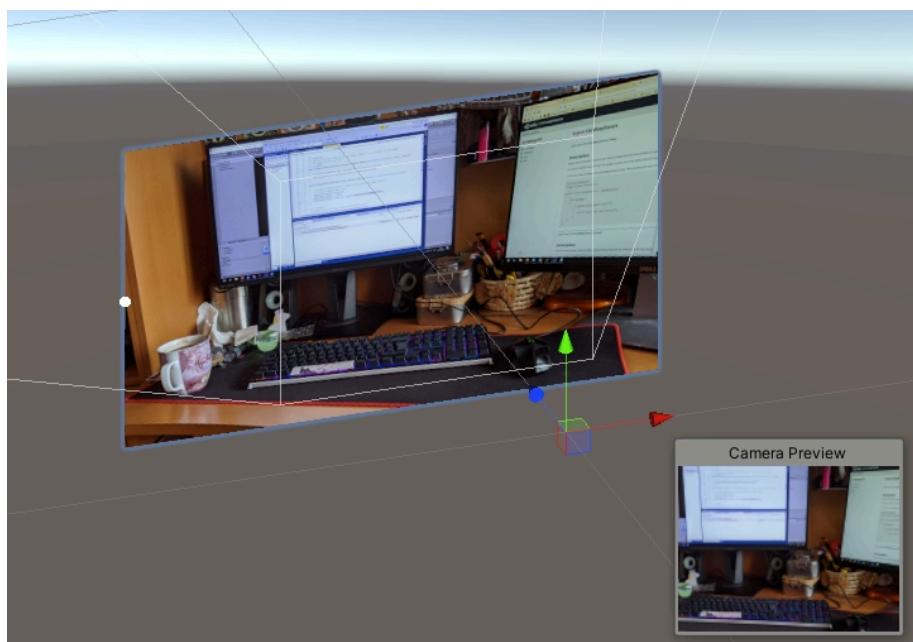


Abbildung 25: Das aufgenommene Foto füllt das gesamte Display aus, wenn es angezeigt wird. Die weiße Sphäre liegt auf dem linken Rand des Foto-Anzeige-Elementes.

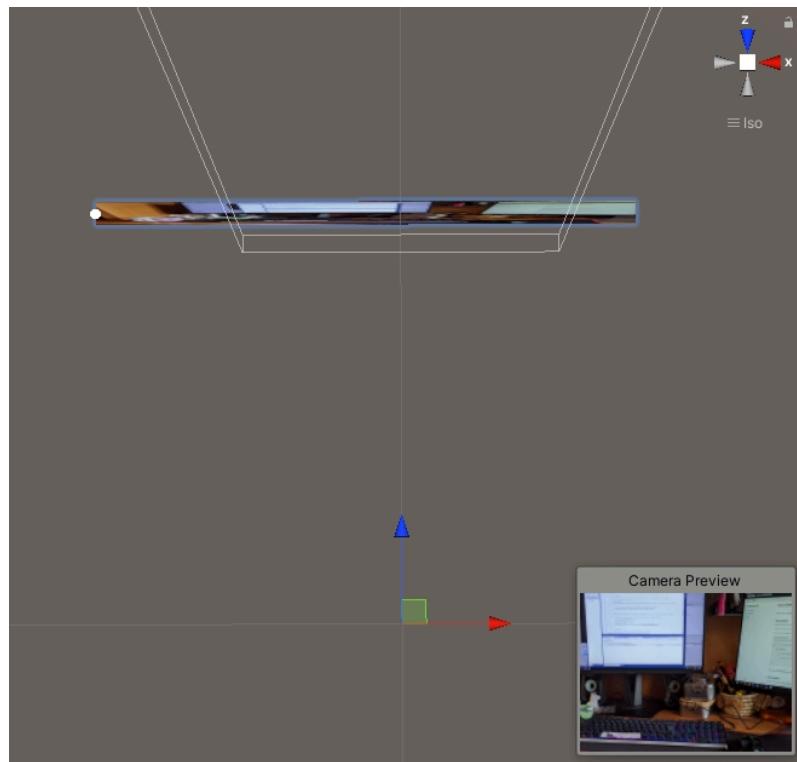


Abbildung 26: Die weiße Sphäre befindet sich nicht mehr in dem View Frustum und das Foto-Anzeige-Element befindet sich ein wenig hinter der Near Clipping Plane.

Die Ränder des UI Elementes wurden genutzt, um die Intervalle für x und y zu bestimmen.

- für x: [-0.2949, 0.2295]
- für y: [0.1546, -0.1507]
- Zusätzlich wurde z = -0.4 gesetzt. Das UI Element musste ein wenig weiter von der Clipping Plane entfernt sein, um angezeigt zu werden.

Mit diesen Intervallen konnten *DetectedObjects* gut lokalisiert werden und es wurden keine Objekte mehr weggelassen, von denen der Nutzer erwarten würden, dass sie markiert werden.

5 Auswertung

In diesem Kapitel geht es um die Evaluierung und Auswertung der vorgestellten Anwendung.

5.1 Laufzeitanalyse

Die Laufzeit der Objekterkennung wurde aufgezeichnet. Die Erkennung beginnt damit ein Foto aufzunehmen und endet mit der Erzeugung der Labels.

Die genutzte Netzwerkverbindung hatte eine Dowloadgeschwindigkeit von 180 Mbps und ein Uploadgeschwindigkeit von 18 Mbps.

Bilder haben Auflösung von 1090x1820 Pixel. Wenn sie in den HTTP-Anfragen verschickt haben, sie die Größe von 5 Mb.

Der Netzwerk Delay zwischen den REST-APIs und der Applikation wurde getestet, indem HTTP-Anfragen mit einem inkorrekt Authentifizierungsschlüssel gestellt wurden. In dem Body der Anfrage wurde ein Bild mitgeschickt, dieses wurde jedoch nicht analysiert.

Im Durchschnitt beträgt die Round Trip Time 0,18 Sekunden in 14 Durchführungen dieses Tests (minimal 0,13 Sekunden und maximal 0,28 Sekunden).

Siehe Abbildung 27

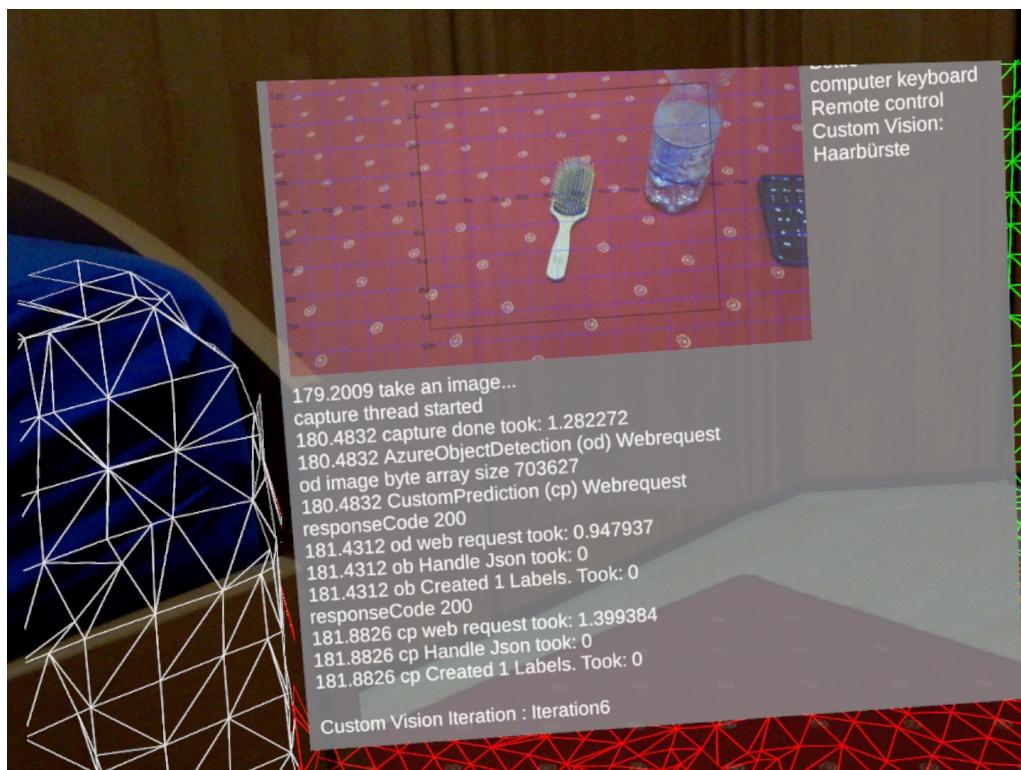


Abbildung 27: Durchlauf mit Laufzeit Aufzeichnung

Über 13 aufgenommene Fotos wurde festgestellt, dass das Aufnehmen des Fotos im Durchschnitt 1,11 Sekunden dauert. Die Anfragen an den Azure Object Detection Services, inklusive Netzwerk Response Time liegen Durchschnittlich bei 0,96 Sekunden. Die Anfragen an den Azure Custom Vision Service, inklusive Netzwerk Response Time, dauern im Durchschnitt 1,84 Sekunden. Für Azure Custom Vision wurde Iteration 6 verwendet.

Azure Object Detection und Azure Custom Vision werden parallel zueinander in unterschiedlichen Threads durchgeführt. Die durchschnittliche Gesamtaufzeit der Objekterkennung liegt bei 2,95 Sekunden.

Das Auslesen der Json Antworten, das Lokalisieren der Objekte in der 3D Szene und die Label-Erstellung, benötigt weniger als eine Mikrosekunde. Siehe Abbildung 28 und 29.

	Foto 1	Foto 2	Foto 3	Foto 4	Foto 5	Foto 6	Foto 7	Foto 8	Foto 9	Foto 10	Foto 11	Foto 12	Foto 13	Average
image capture	1,051	0,94	1,217	0,8866	0,99	0,891	1,2288	1,3553	0,894	1,255	1,4895	1,267	0,9518	1,11
obj det web request	0,918	0,874	0,904	0,973	1,085	0,768	0,881	0,919	0,956	1,04	1,0376	1,1211	1,0028	0,96
handle json	0	0	0	0	0	0	0	0	0	0	0	0	0	0,00
created labels	0	0	0	0	0	0	0	0	0	0	0	0	0	0,00
custom v web request	1,2522	4,06	1,364	1,306	1,2257	1,1016	3,149	1,748	1,123	1,3287	3,5534	1,4543	1,2123	1,84
handle json	0	0	0	0	0	0	0	0	0	0	0	0	0	0,00
created labels	0	0	0	0	0	0	0	0	0	0	0	0	0	0,00
total	2,3033	5,0029	2,6799	2,1931	2,2206	1,9934	4,3797	3,1049	2,0179	2,5839	5,0324	2,7227	2,1641	2,95

Abbildung 28: Laufzeitanalyse über 13 Bild-Analysen.

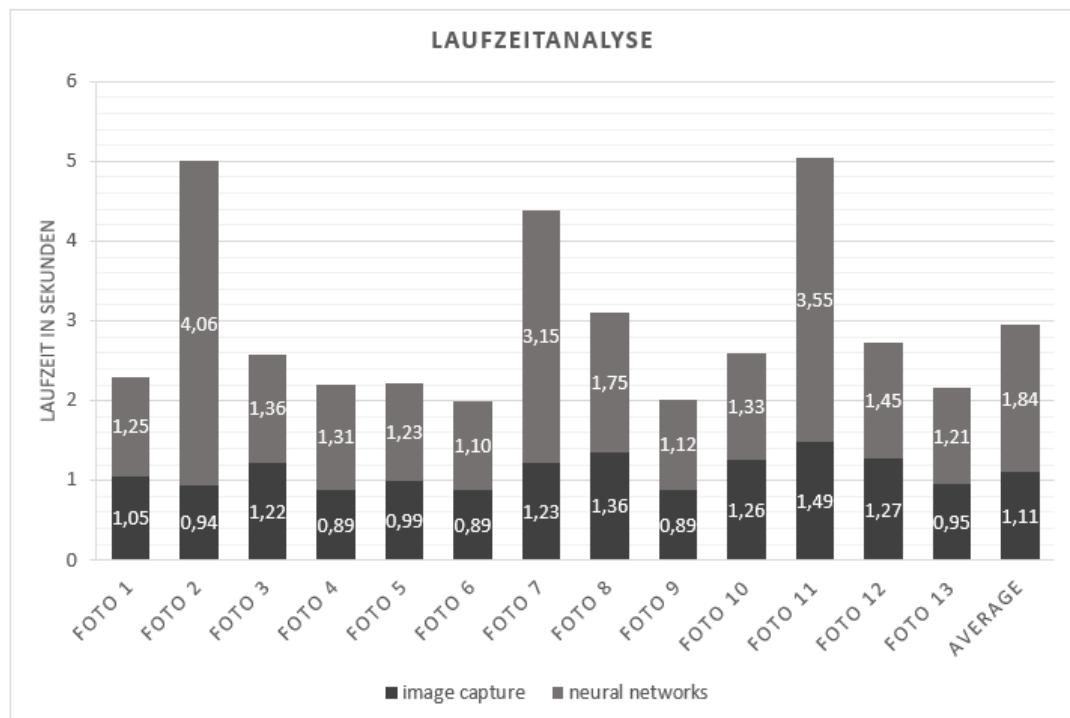


Abbildung 29: Diagramm Laufzeitanalyse

Der Prozess ist somit deutlich zu langsam um in Echtzeit ausgeführt zu werden.

Ein einziges neuronales Netzwerk für die Objekterkennung zu verwenden, würde die Laufzeit verringern. Allerdings geht dann die Möglichkeit verloren, mehrere neuronale Netze zu verwenden, die unterschiedliche Computer Vision Aufgaben erledigen. Aus einem RGB-Bild können beispielsweise mehr semantische Informationen extrahiert werden, wenn Image Segmentation und Object Detection kombiniert werden.

Neben den Objekterkennung-Prozessen braucht das Aufnehmen der Fotos mit durchschnittlich 0.9 Sekunden einen großen Teil der Laufzeit. Durch ein Umstieg von Fotos auf Frames eines Videostreams kann die Aufnahmezeit reduziert werden.

5.2 Evaluierung der Objekterkennung durch Azure Objekt Detection

Die Anwendung wurde in einem Schlaf- und Arbeitszimmer getestet. Durch Azure Object Detection konnten folgende Object-Klasse erkannt werden: Television, Person, Bottle, Keyboard, Computermouse, cat, bed, luggage, chair, laptop.

Azure Object Detection erkennt die Objekte mit unterschiedlicher Verlässlichkeit. Tastaturen und Personen wurden in den meisten Aufnahmen korrekt erkannt. Das Lightwear Gerät der Magic Leap One wurde gelegentlich als Computermaus interpretiert und Computerbildschirme wurden durchweg als Television oder als Laptop markiert.

5.3 Evaluierung der Objekterkennung durch Azure Custom Vision

Die Genauigkeit der Objekterkennungen durch Azure Custom Vision wird durch das Training des Modells bestimmt. Dieser Cloud Service hat sich als effektiv erweisen, um Azure Object Detection zu ergänzen. Mit der Iteration 6, die mit 51 Bildern eine Stunde lang trainiert wurde, kann die Haarbürste zuverlässig erkannt werden.

Für Regionen, in denen sich keine Haarbürste befindet, berechnet das Modell eine Probability von weniger als 5 Prozent dafür, dass sich dort fälschlicherweise eine Haarbürste befindet.

Regionen, die mit einer Probability von über 50 Prozent einer Haarbürste enthalten, beinhalteten in der Anwendung immer eine Haarbürste.

Durch die niedrige Rate an false positives konnte die Akzeptanzschwelle auf 60 Prozent gesetzt werden, um die Rate der false negatives zu verringern. Dadurch wird die Haarbüste fast immer korrekt erkannt, wenn sie auf einem Bild ist.

5.4 Objekte in 3D Szene lokalisieren

In diesem Teil wird evaluiert wie akkurat Objekte, die in einem Foto erkannt wurden, in der 3D Umgebung markiert werden. Um die Lokalisierung zu evaluieren, wurden die RGB-Bilder festgehalten, mit denen Objekterkennung durchgeführt wurde. Auf den RGB-Bildern wurde der Mittelpunkt der Bounding Boxen markiert. Siehe Abbildung 30. Das Label, das in der Szene gesetzt wird, soll mit dieser Position korrespondieren. Siehe Abbildung 31.

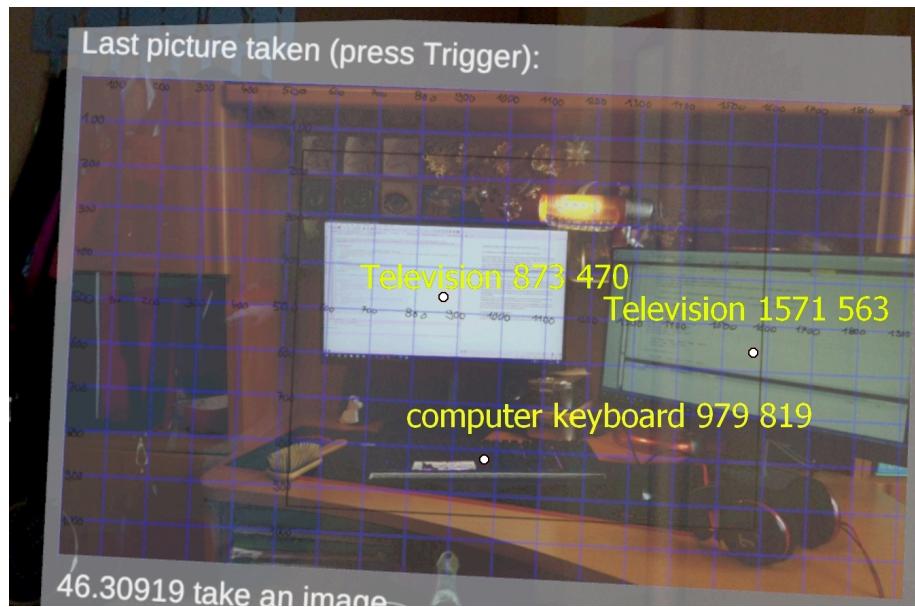


Abbildung 30: Die Mittelpunkte der erkannten Objekte wurden, auf dem Foto per Hand markiert. Die Klassen und die u,v Foto-Koordinaten der Objekte sind angegeben.

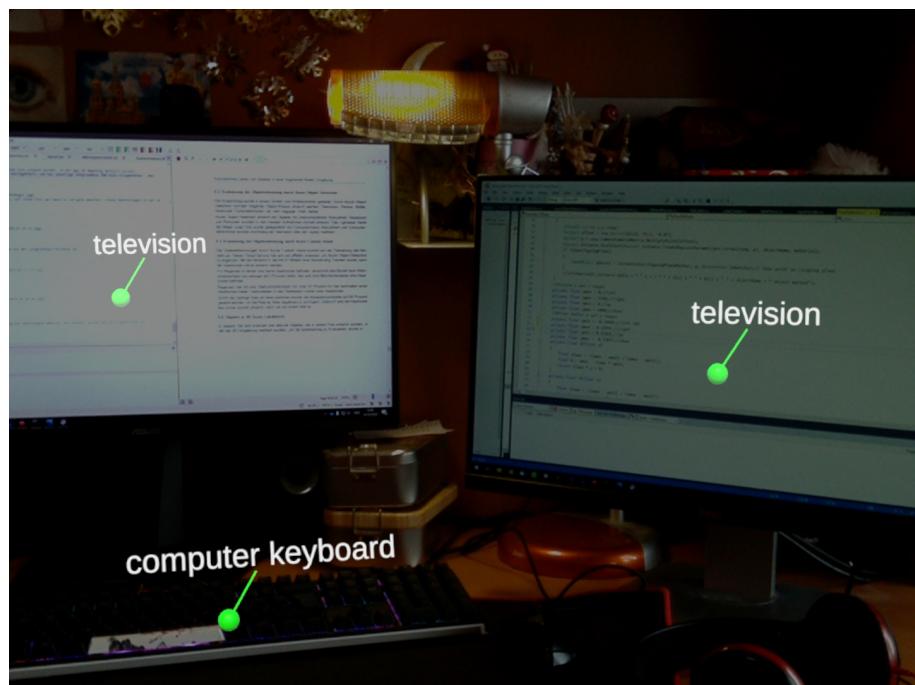


Abbildung 31: Objekte in der 3D Szene markiert.

Die Positionierungen der Labels weichen um weniger als 2 cm von der vorgesehenen Position ab.

Die Lokalisierung der Objekte hängt von dem geometrischen Verständnis der AR-Brille an. Objekte die häufig bewegt werden, wie beispielsweise ein Stuhl, werden erst in die Spatial Map eingefügt, wenn sie über längere Zeit nicht bewegt wurden. Das Einfügen des Objektes in die Map dauert ca. 2 Minuten.

Wenn der Objekte Detection Prozess auf ein Objekt angewandt wird, das noch nicht oder nicht komplett in der Spatial Map ist, wird es nicht korrekt markiert. Siehe Abbildung 32.

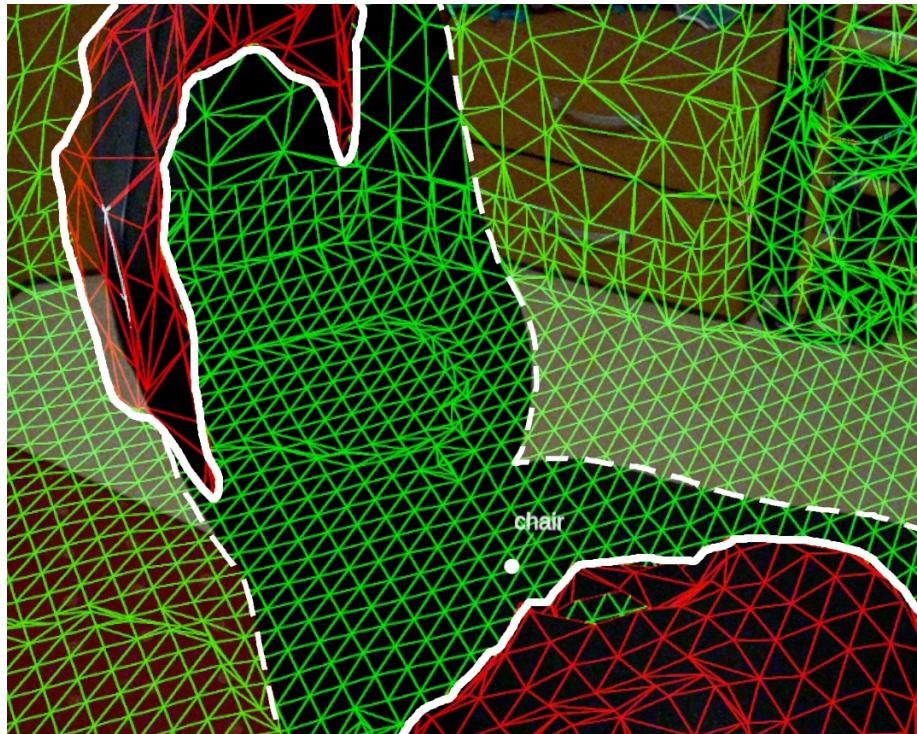


Abbildung 32: Spatial Map des Stuhles mit breiter Linie umrandet. Tatsächlicher Stuhl mit durchgebrochener Linie umrandet. Label des Stuhls markiert. Das Label befindet sich in der Szene hinter dem Stuhl auf dem Boden.

In diesem Beispiel ist ein Stuhl noch nicht in der Spatial Map. Wenn die Applikation einen Raycast auf die Umgebung durchführt, um das gefundene Objekt zu markieren, schießt der Raycast durch das tatsächliche Objekt hindurch, da es nicht in der Map ist. Der Raycast trifft den Boden hinter dem Objekt und markiert diesen. Der Stuhl wurde von der Objekterkennung korrekt erkannt, jedoch nicht korrekt in der Szene markiert.

Halb transparente Objekte werden ebenfalls nicht korrekt gemapped. Daher wird das Label eines Flaschen-Objekts nicht korrekt positioniert. Siehe Abbildung 33.

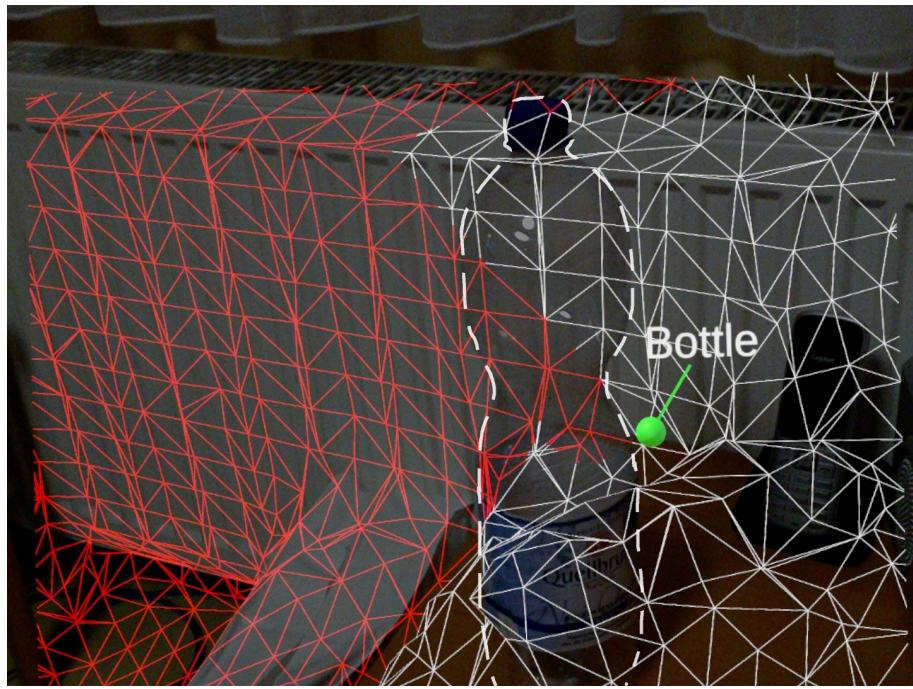


Abbildung 33: Das Label der Flasche liegt hinter dem tatsächlichen Objekt, da die Flasche nicht korrekt gemapped wurde.

6 Zusammenfassung

In diesem Kapitel werden die bearbeiteten Themen kurz zusammengefasst.

6.1 Konzepte und Implementierte Funktionen

In dieser Arbeit wird Image Based Objekt Detection in eine Augmented Reality Brille integriert, um eine automatische Objekterkennung und Markierung von Objekten zu ermöglichen.

Es werden RGB-Fotos von der Umgebung der AR Brille aufgenommen und mit Machine Learning Modellen analysiert. Das Modell ist darauf trainiert Objekte und Lebewesen in einem Bild zu erkennen. Durch die Analyse werden Objekte auf den RGB-Fotos erkannt. Diese werden dann in der AR Umgebung lokalisiert und mit einem Label markiert.

Das automatische Erkennen und Labeln von Objekten kann für große und dynamische Umgebungen verwendet werden. Es bietet somit eine Grundlage für AR Anwendungen die in einer solchen Umgebung arbeiten sollen oder eine ausgeprägtes semantisches Verständnis der Umgebung benötigen. Beispielsweise eine Blindenführung in unbekannten Umgebungen oder Applikationen aus den Gebieten Autonomous Driving und Robotics. (Chen et al. 2017)

6.2 Ausblick

Die automatische Erkennung von Objekten in einer AR Umgebung ist funktionsfähig und effektiv.

Die Objekte, die erkannt werden, hängen von dem verwendeten Machine Learning Modell ab. In Zukunft könnte man weitere Modelle verwenden, um die Fotos der Umgebung zu analysieren. Durch Image Segmentation und Image Klassifikation können weitere Informationen erhoben werden. Durch das Inkludieren von Kontextinformationen kann beispielsweise hervorgehen, in welchem Raum eines Hauses sich die AR Brille befindet (Küche, Arbeitszimmer, Schlafzimmer).

Um die Lokalisierung eines Objektes in der AR Umgebung zu verbessern, können die geometrischen Informationen einer Spatial Map durch rohe Daten einer Tiefenkamera ergänzt werden. Da es rechenintensiv ist, die Spatial Map zu erstellen, kann es dazu kommen, dass stellenweise die Map noch nicht aufgebaut ist oder in einem veralteten Zustand vorliegt, wenn ein Objekt lokalisiert werden soll. Die rohen Daten der Tiefenkamera könnten dazu verwendet werden, die Spatial Map zu ergänzen.(Ma und Karaman 2017)

7 Literaturverzeichnis

Azuma und Furmanski 2003

AZUMA, R. ; FURMANSKI, C.: Evaluating label placement for augmented reality view management. In: *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.*, 2003, S. 66–75

Bell et al. 2001

BELL, Blaine ; FEINER, Steven ; HÖLLERER, Tobias: View Management for Virtual and Augmented Reality. In: *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA : Association for Computing Machinery, 2001 (UIST '01). – ISBN 158113438X, 101–110

Chen et al. 2018

CHEN, Long ; TANG, Wen ; JOHN, Nigel ; WAN, Tao R. ; ZHANG, Jian J.: *Context-Aware Mixed Reality: A Framework for Ubiquitous Interaction*. 2018

Chen et al. 2017

CHEN, X. ; MA, H. ; WAN, J. ; LI, B. ; XIA, T.: Multi-view 3D Object Detection Network for Autonomous Driving. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, S. 6526–6534

Dörner et al. 2019

In: DÖRNER, Ralf ; BROLL, Wolfgang ; JUNG, Bernhard ; GRIMM, Paul ; GÖBEL, Martin: *Einführung in Virtual und Augmented Reality*. Berlin, Heidelberg : Springer Berlin Heidelberg, 2019. – ISBN 978–3–662–58861–1, 1–42

Huynh et al. 2019

HUYNH, B. ; ORLOSKY, J. ; HÖLLERER, T.: In-Situ Labeling for Augmented Reality Language Learning. In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2019, S. 1606–1611

Jiao et al. 2019

JIAO, L. ; ZHANG, F. ; LIU, F. ; YANG, S. ; LI, L. ; FENG, Z. ; QU, R.: A Survey of Deep Learning-Based Object Detection. In: *IEEE Access* 7 (2019), S. 128837–128868

Jmour et al. 2018

JMOUR, N. ; ZAYEN, S. ; ABDELKRIM, A.: Convolutional neural networks for image classification. In: *2018 International Conference on Advanced Systems and Electric Technologies (IC ASET)*, 2018, S. 397–402

Leap 2018

LEAP, Magic: *App Security*. <https://developer.magicleap.com/en-us/learn/guides/application-security-overview>. Version: 2018.– [Online; Stand 18. September 2020]

Ma und Karaman 2017

MA, Fangchang ; KARAMAN, Sertac: Sparse-to-Dense: Depth Prediction from Sparse Depth Samples and a Single Image. In: *CoRR* abs/1709.07492 (2017). <http://arxiv.org/abs/1709.07492>

MagicLeap

MAGICLEAP: *JsonUtility.FromJson*. <https://www.magicleap.care/hc/en-us>. – [Online; Stand 1. Oktober 2020]

MagicLeap 2018

MAGICLEAP: *magic leap 1.* <https://www.magicleap.com/en-us/magic-leap-1>. Version: 2018. – [Online; Stand 18. September 2020]

MagicLeap 2019a

MAGICLEAP: *Lumin OS Overview.* <https://developer.magicleap.com/en-us/learn/guides/lumin-os-overview>. Version: 2019. – [Online; Stand 18. September 2020]

MagicLeap 2019b

MAGICLEAP: *World Reconstruction.* <https://developer.magicleap.com/en-us/learn/guides/world-reconstruction-overview-landing>. Version: 2019. – [Online; Stand 18. September 2020]

MagicLeap 2020a

MAGICLEAP: *1.1 Unity Setup.* <https://developer.magicleap.com/en-us/learn/guides/get-started-developing-in-unity>. Version: 2020. – [Online; Stand 12. October 2020]

MagicLeap 2020b

MAGICLEAP: *1.4 Spatial Meshing - Unity.* <https://developer.magicleap.com/en-us/learn/guides/meshing-in-unity>. Version: 2020. – [Online; Stand 18. September 2020]

MagicLeap 2020c

MAGICLEAP: *Glossary and Usage.* <https://developer.magicleap.com/en-us/learn/guides/glossary>. Version: 2020. – [Online; Stand 18. September 2020]

MagicLeap 2020d

MAGICLEAP: *Magic Leap Features.* <https://developer.magicleap.com/en-us/learn/guides/magic-leap-features>. Version: 2020. – [Online; Stand 18. September 2020]

Maier et al. 2019

MAIER, Andreas ; SYBEN, Christopher ; LASSER, Tobias ; RIESS, Christian: A gentle introduction to deep learning in medical image processing. In: *Zeitschrift für Medizinische Physik* 29 (2019), Nr. 2, 86 - 101. <http://dx.doi.org/https://doi.org/10.1016/j.zemedi.2018.12.003>. – DOI <https://doi.org/10.1016/j.zemedi.2018.12.003>. – ISSN 0939–3889. – Special Issue: Deep Learning in Medical Physics

Micosoft 2018

MICOSOFT: *Mr und Azure 302b benutzerdefinierte Vision.* <https://docs.microsoft.com/de-de/windows/mixed-reality/mr-azure-302b>. Version: 2018. – [Online; Stand 17. September 2020]

Microsoft

MICROSOFT: *Microsoft Azure Computer VSION.* <https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/>. – [Online; Stand 17. September 2020]

Microsoft 2018a

MICROSOFT: *Mr und Azure 302 Maschinelles Sehen.* <https://docs.microsoft.com/de-de/windows/mixed-reality/mr-azure-302>. Version: 2018. – [Online; Stand 17. September 2020]

Microsoft 2018b

MICROSOFT: *Spatial Mapping*. <https://docs.microsoft.com/de-de/windows/mixed-reality/spatial-mapping>. Version: 2018. – [Online; Stand 17. September 2020]

Microsoft 2019a

MICROSOFT: *Detect common objects in images*. <https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-object-detection>. Version: 2019. – [Online; Stand 17. September 2020]

Microsoft 2019b

MICROSOFT: *Erkennen von alltäglichen Objekten in Bildern*. <https://docs.microsoft.com/de-de/azure/cognitive-services/computer-vision/concept-object-detection>. Version: 2019. – [Online; Stand 24. September 2020]

Microsoft 2020

MICROSOFT: *What is Computer Vision*. <https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home>. Version: 2020. – [Online; Stand 17. September 2020]

O'Shea und Nash 2015

O'SHEA, Keiron ; NASH, Ryan: An Introduction to Convolutional Neural Networks. In: *ArXiv e-prints* (2015), 11

Unity 2017

UNITY: *GameObjects*. <https://docs.unity3d.com/Manual/GameObject.html>. Version: 2017. – [Online; Stand 12. October 2020]

Unity 2018

UNITY: *Prefabs*. <https://docs.unity3d.com/Manual/Prefabs.html>. Version: 2018. – [Online; Stand 12. October 2020]

Unity 2020a

UNITY: *Camera.cameraToWorldMatrix*. <https://docs.unity3d.com/ScriptReference/Camera-cameraToWorldMatrix.html>. Version: 2020. – [Online; Stand 18. September 2020]

Unity 2020b

UNITY: *JsonUtility.FromJson*. <https://docs.unity3d.com/ScriptReference/JsonUtility.FromJson.html>. Version: 2020. – [Online; Stand 24. September 2020]

Unity 2020c

UNITY: *Matrix4x4.MultiplyPoint*. <https://docs.unity3d.com/ScriptReference/Matrix4x4.MultiplyPoint.html>. Version: 2020. – [Online; Stand 18. September 2020]

Unity 2020d

UNITY: *Understanding the View Frustum*. <https://docs.unity3d.com/Manual/UnderstandingFrustum.html>. Version: 2020. – [Online; Stand 12. October 2020]

8 Anhang

8.1 Markierte Objekte aus unterschiedlichen Blickwinkeln

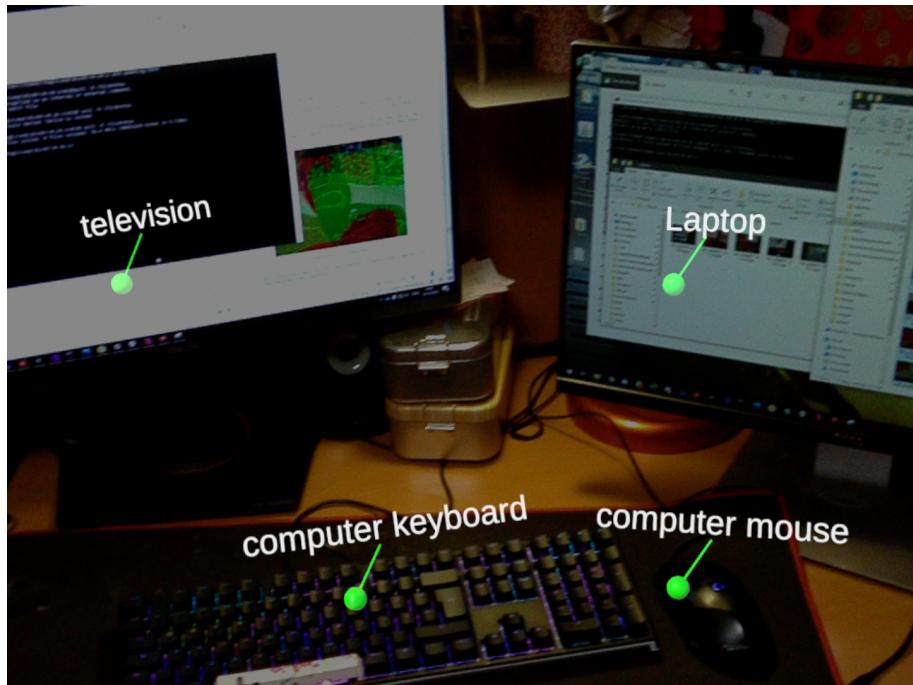


Abbildung 34: Objekte mit Label. Blickwinkel 1

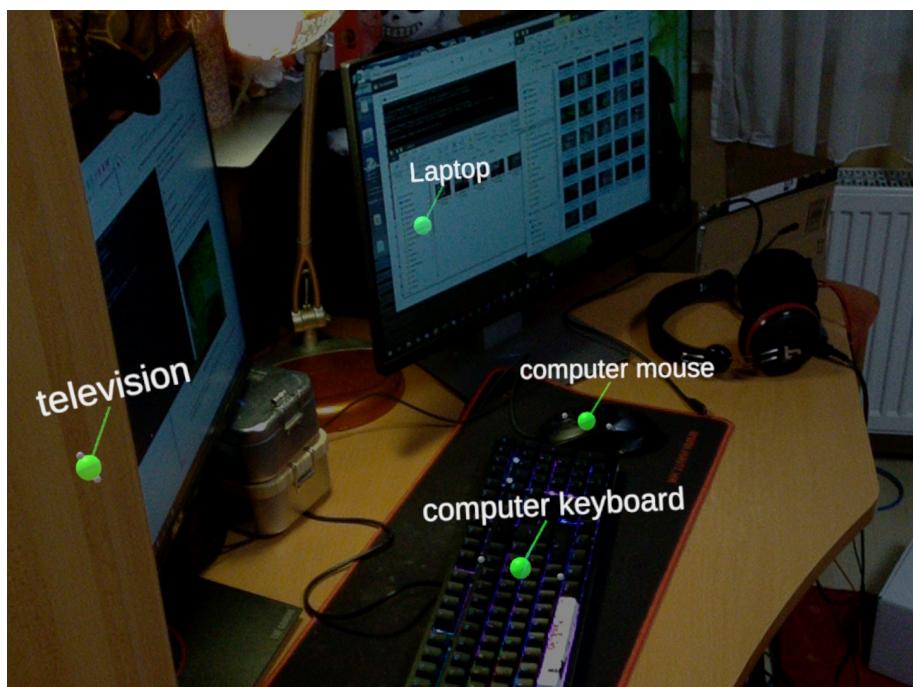


Abbildung 35: Blickwinkel 2

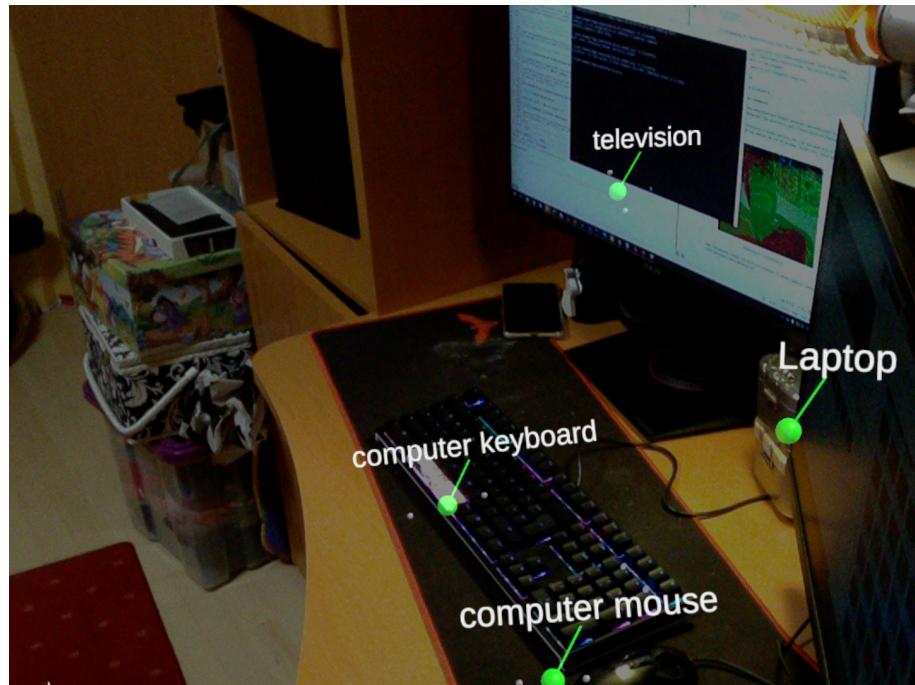


Abbildung 36: Blickwinkel 3

8.2 Ausschnitt Trainingsbilder der Custom Vision Iterationen 4 und 6



Abbildung 37: Ausschnitt der Trainigsbilder für Haarbürsten-Objekterkennung Iteration 4

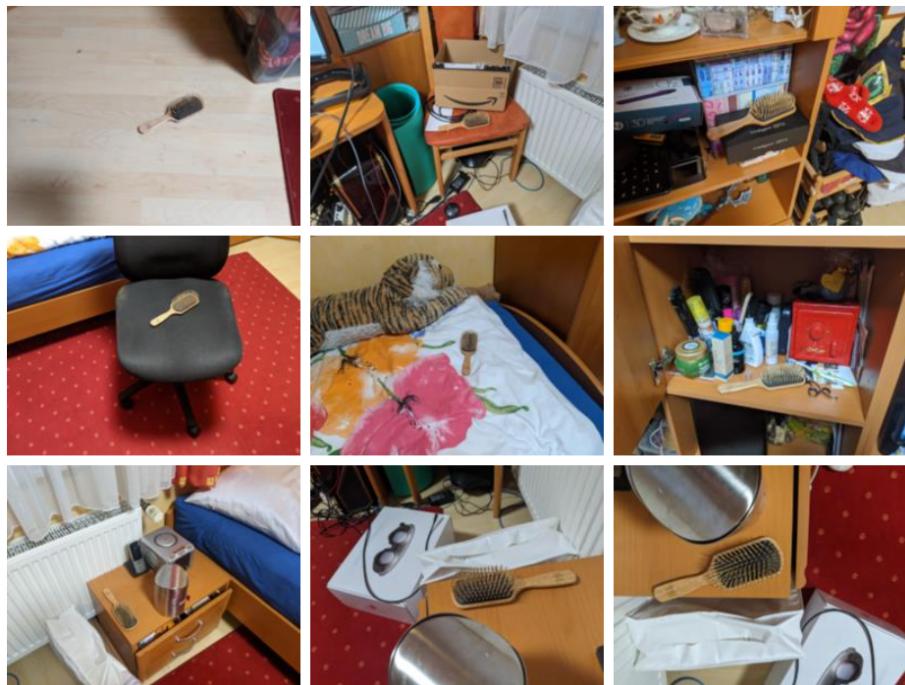


Abbildung 38: Ausschnitt der Trainingsbilder für Haarbürsten-Objekterkennung Iteration 4

8.3 Beispielbilder von Objekten mit Labels

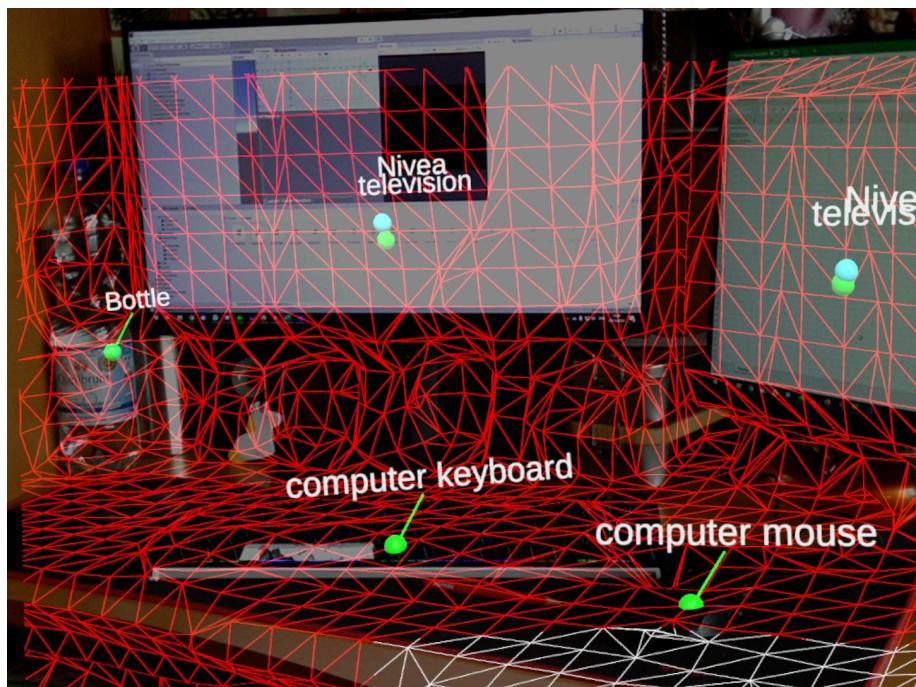


Abbildung 39: Beispielbild mit Iteration 4. Computerbildschirme werden als Nivea Dosen erkannt.

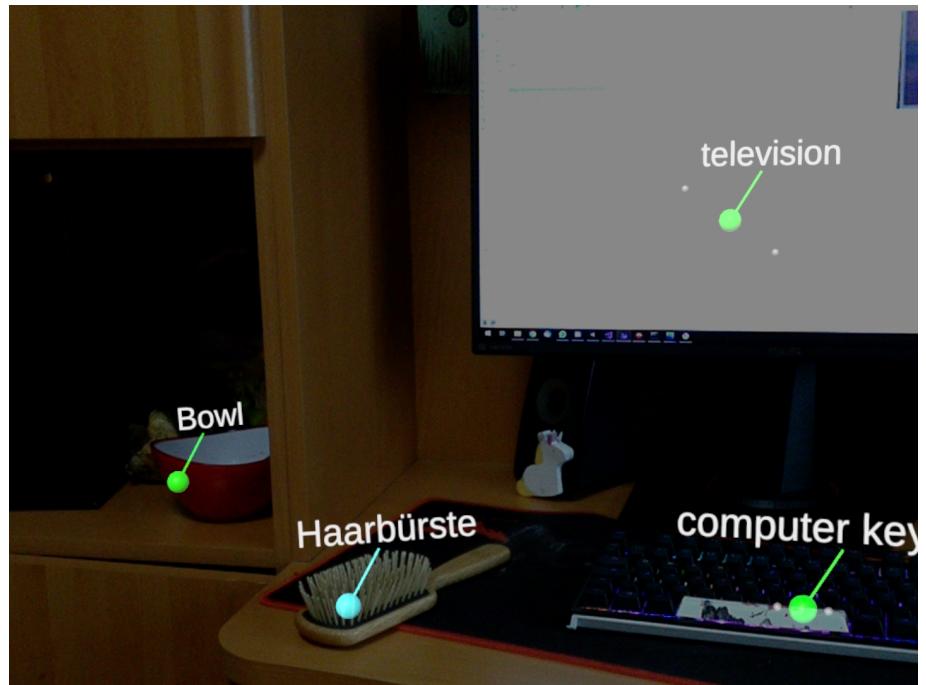


Abbildung 40: Beispielbild mit Iteration 6