



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: 20. 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
2.1.1 View Management	5
2.2 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	13
3.2.5 Kamera in 3D-Computergrafik	13
3.3 Computer Vision	14
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 Anforderungen	20
4.2 Design der Objekternennung	20
4.3 Design der Labels	21
4.4 Architektur	21
4.5 Interaktion	24
4.6 Implementierung der Objekterkennung	25
4.7 Ein Foto aufnehmen	25
4.7.1 Object Detection	25
4.7.2 Von dem Foto zum 3D-Raum	27
4.7.3 Raycast	29
4.7.4 LabelCreator	30
4.7.5 Azure Custom Vision	32
4.8 Entwicklung der Foto-Repräsentation	36
5 Auswertung	40
5.1 Laufzeitanalyse	40
5.1.1 Netzwerk	40
5.1.2 Objekterkennung	40

5.2	Evaluierung der Objekterkennung durch Azure Objekt Detection	42
5.3	Evaluierung der Objekterkennung durch Azure Custom Vision	42
5.4	Objekte in 3D-Szene lokalisieren	42
6	Zusammenfassung	45
6.1	Konzepte und implementierte Funktionen	45
6.2	Vergleich mit Related Works	45
6.3	Ausblick	46
7	Literaturverzeichnis	48
8	Anhang	51
8.1	Markierte Objekte aus unterschiedlichen Blickwinkeln	51
8.2	Ausschnitt Trainingsbilder der Custom Vision Iterationen 4 und 6	52
8.3	Beispielbilder von Objekten mit Labels	53

Abbildungsverzeichnis

1	Objekterkennung von Huynh et al. (2019)	4
2	Context Aware Shooter von Chen et al. (2018)	7
3	Spatial Mapping	9
4	Magic Leap One AR-Brille.(MagicLeap 2018a)	10
5	Unity GameObject mit Components	12
6	View Frustum	13
7	Trainingsbild für Azrue Custom Detection	17
8	Evaluierung eines Azure Custom Detection Modells	18
9	Diagramm der Architektur	22
10	Klassendiagramm der Skripts	23
11	UI Ausgabe in der Szene	24
12	Klassendiagramm von <i>AzureObjectDetection</i>	25
13	Klassendiagramm für Json-Umwandlung	26
14	Klassendiagramm von <i>PixelToWorld</i>	27
15	Klassendiagramm von Raycast	29
16	Labels in der Szene	30
17	Haarbürste zwei mal erkannt	31
18	Haarbürste drei mal erkannt	31
19	Iteration 1 Analysebeispiel	33
20	Iteration 4 Analysebeispiel	34
21	Iteration 4 zweites Analysebeispiel	35
22	Iteration 6 Analysebeispiel	36
23	Ränder der Near Clipping Plane in Unity finden	37
24	Ränder des Foto-Anzeige-Elements finden	38
25	Tiefe des Foto-Anzeige-Elements finden	39
26	Laufzeit einer Objekterkennung	40
27	Laufzeitanalyse über 13 Bild-Analysen	41
28	Diagramm Laufzeitanalyse	41
29	Erkennt Objekte auf RGB-Bilde markiert	43
30	Erkannte Objekte in der Szene markiert	43
31	Label setzen, bei fehlerhafter Spatial Map	44
32	Spatial Mapping transparenter Objekte	44
33	Objekte mit Label. Blickwinkel 1	51
34	Objekte mit Label. Blickwinkel 2	51
35	Objekte mit Label. Blickwinkel 3	52
36	Ausschnitt der Trainingsbilder. Iteration 4	52
37	Ausschnitt der Trainingsbilder. Iteration 6	53
38	Beispielbild mit Iteration 4.	53
39	Beispielbild mit Iteration 6.	54

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. Letzteres ermöglicht komplexere Interaktionen zwischen digitalen und virtuellen Elementen. Semantische Informationen werden durch die Gegenstände erschlossen, die sich in der Umgebung befinden.

Eine Möglichkeit, um Objekte in der Umgebung zu erkennen, verwendet Markierungen der realen Welt. Dabei handelt es sich um statische Bilder, beispielsweise um ein Foto oder einen QR-Code. Die Markierungen werden von einer Kamera eingescannt. Der Marker ist einzigartig für jeden Gegenstand. So können die Objekte eindeutig voneinander unterschieden werden. Der Nachteil bei diesem Vorgehen ist ein hoher Arbeitsaufwand. Dieser entsteht durch die Notwendigkeit, jeden Gegenstand der realen Welt einzeln zu markieren.

Markierungen in der realen Welt können umgangen werden, indem der Nutzer per Geste auf Objekte der realen Welt weist, die vom System erkannt werden sollen. Für jedes dieser Objekte muss der Nutzer angeben, um welche Art von Gegenstand es sich handelt. Auf diese Weise kann die Applikation unterschiedliche Objekte unterscheiden. Auch hier ist der Arbeitsaufwand hoch. Dieser Nachteil ergibt sich aus der hohen Anzahl erforderlicher Gesten.

Aufgrund ihres hohen Arbeits- und Zeitaufwandes sind beide beschriebene Verfahren ungeeignet für den Zweck, Objekte in großen oder dynamischen Umgebungen zu erkennen. Nur eine vollautomatische Objekterkennung lässt sich gut skalieren. Mit dieser Methode können semantische Informationen einer reale Umgebung mit deutlich weniger Aufwand erfasst werden. So lassen sich komplexe Anwendungsbereiche für Augmented Reality erschließen.

Um eine automatisierte Objekterkennung zu erreichen, kann 'Object Detection' aus dem Bereich der Computer Vision verwendet werden. 'Object Detection' ist darauf ausgelegt, Objekte in RGB-Bildern zu erkennen.(O'Shea and Nash 2015)

1.1 Zielsetzung

Thema dieser Arbeit ist die automatische Erkennung von Objekten in einer AR-Umgebung mithilfe von 'Object Detection'. Die AR-Brille 'Magic Leap One Lightwear' wird als Benutzerinterface und als 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. Alle Fotos werden durch ein trainiertes neuronales Netzwerk nach Objekten durchsucht. Die neuronalen Netze geben für die gefundenen Fotos sowohl eine Klasse als auch eine Position auf dem Foto an. Die jeweiligen 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, um spezifische Objekte zu erkennen. Auf diese Weise, sollen die semantischen Informationen erweitert werden, die von der Applikation erkannt werden können.

Als Maximalziel werden die Labels der erkannten Objekte als virtuelle Elemente mit der AR-Brille 'Magic Leap One' angezeigt.

2 Related Work

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

Huynh et al. (2019) beschreiben ein Framework, mit dem die Lernmethode 'loci' in Augmented Reality umgesetzt werden kann. Die Lernmethode beruht darauf, Gegenstände der realen Welt mit Notizen zu beschriften. Für die Umsetzung mit AR wird eine automatische Echtzeit-Objekterkennung entwickelt. Dabei werden die Objekte auf RGB-Fotos der AR-Umgebung erkannt und in eine 3D-Rekonstruktion der Umgebung übernommen.

Die Lernmethode soll auf der AR-Brille 'Hololens' ausgeführt werden. Diese hat zu wenig Rechenleistung, um Object Detection durchzuführen. Daher wird eine Server-Client Architektur aufgesetzt. Die Kamera der AR-Brille nimmt ein Video der Umgebung auf. Die einzelnen Frames des Videos werden an den Server geschickt. Dieser nutzt ein neuronales Netzwerk, um alle erkennbaren Objekte in den Frames zu finden und mit Bounding-Boxen zu lokalisieren.

Für die Objekterkennung wird das neuronale Netzwerk von TensorFlow verwendet. Im Rahmen einer Analyse findet das Netzwerk mehrere Objekte in einem Foto. Die Objekterkennung soll in Echtzeit durchgeführt werden. Um die Laufzeit der Erkennung möglichst gering zu halten, wird die niedrigste Kameraauflösung mit 896x504 Pixel verwendet. Zusätzlich werden die Fotos als JPEG komprimiert. Auf diese Weise braucht die Analyse lediglich 30 ms pro Foto. So wird eine Echtzeit-Erkennung mit 30 Frames pro Sekunde möglich. Dennoch wird die Erkennung in der Applikation durch eine Netzwerk-Verzögerung von 150 ms zwischen der AR-Brille und dem Server verspätet.

Die AR-Brille 'Hololens' nummeriert die Frames, welche an den Server geschickt werden. Zusätzlich wird für jedes Frame die Kameraposition gespeichert, mit der es aufgenommen wurde. Aus dieser Weise können Frames asynchron analysiert werden. Anschließend dienen die Bounding-Boxen der Objekte und die Kameraposition dazu, die Objekte in der 3D-Umgebung zu lokalisieren. Dafür wird der Mittelpunkt jeder Bounding-Box mithilfe eines Raycastes in die 3D-Szene projiziert.

Um Fehlern bei der Objekterkennung entgegenzuwirken, erfolgt die Markierung erst, wenn der Gegenstand auf mehreren Fotos gefunden wird. Durch die Analyse des ersten Fotos wird eine ungefähre Position in der AR-Umgebung für das Objekt ermittelt. Diese Position wird anschließend mit den Objekten verglichen, die während der nächsten 60 Frames erkannt werden. Falls Objekte gefunden werden, welche dieselbe semantische Bedeutung und eine ähnliche Position haben, wird davon ausgegangen, dass es sich um einen einzigen Gegenstand handelt, der in den 60 Frames mehrmals aufgenommen wurde. Siehe Abbildung 1.

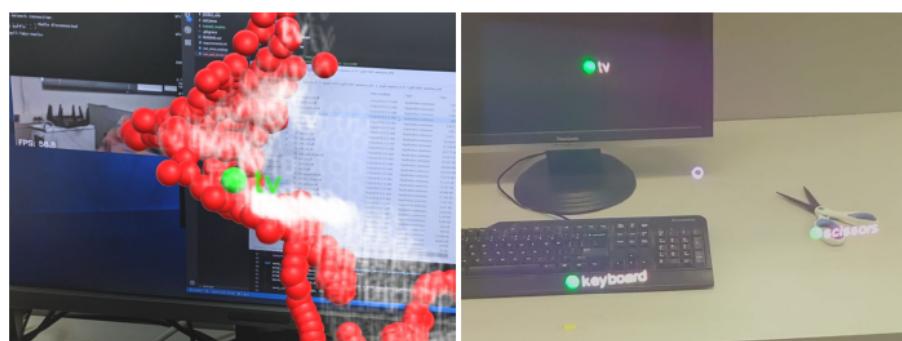


Abbildung 1: Ungefährre Positionen des Objektes über 60 Frames aufgenommen in rot. Finale Position in grün.(Huynh et al. 2019)

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 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 Echtzeit-Detektion nicht relevant. Daher darf 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 bereits in der Umgebung markiert, wenn sie nur ein einziges Mal erkannt wurden. Bei erneuter Aufnahme des Objektes, weist die zweite Analyse dem Objekt erneut die gleiche Klasse und eine ähnliche Position der AR-Umgebung zu. Daran lässt sich erkennen, das es sich um dasselbe Objekt handelt.

Die erste Position, die für das Objekt berechnet wurde, und die weitere Position, die aus der zweiten Aufnahme hervorgeht, unterscheiden sich mit hoher Wahrscheinlichkeit ein wenig voneinander. Daher wird der Mittelwert beider Positionen gebildet, um eine akkurate Position für das Label des Objektes zu finden. Mit jedem neuem Bild wird so eine immer akkurate Position für das Label des Objektes ermittelt. (Huynh et al. 2019)

2.1.1 View Management

In Augmented Reality Anwendungen werden häufig Objekte der realen Welt durch Labels mit Informationen annotiert. Die Labels können entweder 3D oder 2D-Elemente sein. Letztere befinden sich auf der Bildebene. Als Text-Label wird ein Schriftzug bezeichnet, der über eine Leader-Line mit einem Anchor verbunden ist. Der Anchor gibt die Position des Objektes in der Szene an, welches durch das Label beschriftet wird.

View Management ist das Erstellen eines Layouts für die Labels. Das Ziel besteht darin, die Labels gut lesbar auf der Bildebene zu positionieren und darzustellen. Dazu gehört, dass Labels sich gegenseitig nicht verdecken und auf eine verständliche Weise mit ihren Anchors verbunden sind. Einige View Management-Vorgehen legen auch Wert darauf, relevante Regionen der realen Welt zu bestimmen, die reich an Informationen oder für eine gegebene AR-Anwendung relevant sind. Diese Bereiche der realen Welt sollen nicht durch Labels verdeckt werden.

Es gibt viele Vorgehensweisen, um View Management umzusetzen. Sie unterscheiden sich je nachdem, ob sie ein Layout für 3D oder 2D-Labels erzeugen. Weiterhin unterscheiden sich die Vorgehensweisen darin, ob sie Wissen über die Geometrie der Szene nutzen (Geometry-Based) oder Fotos der Bildebene verwenden (Image-Based).

(Grasset et al. 2012) stellen ein Image-Based View Management-Vorgehen für 2D-Labels vor. Das Verfahren wird eingesetzt um Gebäude und Plätze der realen Welt mit 2D-Labels zu beschriften. Mit einer Aufnahme der Bildebene wird eine Saliency-Map erzeugt. Diese gibt die Einzigartigkeit jedes Pixels an. Die Map wird mit einer Kantendetektion kombiniert, um eine Importance-Map zu erstellen. Letztere gibt an, wie informationsreich die Pixel sind. Mit der Importance-Map als Grundlage wird entscheiden, in welchen Bildbereichen Labels gesetzt werden können und welche Bildbereiche zu vermeiden sind.

Es wird eine Menge an Funktionen aufgestellt, mit denen die optimale Position eines Labels gefunden werden kann. Jede Funktion berechnet eine Wertung für ein gegebenes Label und eine Position der Bildebene im Hinblick auf die unterschiedlichen Ziele, welche ein Layout erfüllen soll. Dazu gehört, dass gewisse Bereiche der Importance-Map und der Kantendetektion-Map vermieden werden und dass Labels sich gegenseitig nicht verdecken sollen. Darüber hinaus soll die Distanz zwischen einem Label und seinem Anchor minimiert werden. Zudem kann eine favorisierte Richtung für die Leader Line angegeben werden, die den Look des Layouts beeinflusst. Mit einem Greedy Algorithmus wird die

Position des Displays gefunden, an dem die Layout-Ziele, laut den Funktionen am besten erfüllt sind.

Zusätzlich zu dem Layout, welches die Positionen der Labels bestimmt, wird das Aussehen der Labels an ihren Hintergrund adaptiert, damit die Labels lesbar sind. Beispielweise erhalten die Leader-Lines eine unterschiedliche Farbe, abhängig von den Farben der Bildregionen, auf denen sie liegen. Grasset et al. (2012)

Unsere Applikation würde durch View Management profitieren. Nach der Erkennung eines Gegenstandes in der Umgebung erfolgt eine Markierung in dem 3D-Raum durch ein 3D-Label. Die Lesbarkeit der Labels kann durch View Management verbessert werden, indem ihre Positionen über die Zeit verändert werden.

Das Layout der Labels müsste auf Veränderungen des Betrachtungswinkels und auf Hinzukommen von neuen Labels reagieren. Idealerweise würden keine Informationen verdeckt werden. Labels sollten weder andere Labels noch Gegenstände der realen Welt verdecken. Besonders sollte darauf geachtet werden, dass Objekte, welche durch ein Label annotiert werden, nicht verdeckt sind.

View Management geht jedoch über den Rahmen dieser Arbeit hinaus und kann daher nicht umgesetzt werden.

2.2 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 wird ein First-Person-Shooter vorgestellt, bei dem das Aussehen von Einschusslöchern davon abhängt, auf welches Material geschossen wird. Siehe Abbildung 2.

Für die Erkennung der Materialien werden mehrere Frames der AR-Brille 'Hololens' segmentiert. Die dadurch entstehenden semantischen Informationen werden abgespeichert, um bei späteren Interaktionen der AR-Applikation abrufbar zu sein. Die Erkennung der Materialien ist nicht echtzeitfähig. Die Interaktionen können dennoch in Echtzeit ablaufen, da sie auf bereits gespeicherte Materialien-Informationen zugreifen.

RGB-Bilder der Umgebung werden aufgenommen und mit einem neuronalen Netzwerk analysiert, um semantische Informationen zu erheben. Das neuronale Netz wird von Chen et al. (2018) für den First-Person-Shooter aufgesetzt. Das Netz wurde darauf trainiert, 23 unterschiedliche Materialien - beispielsweise Holz, Glas, Stoff etc. - zu erkennen und Bilder nach ihnen zu segmentieren. Dabei wird für jedes Pixel ein Material angegeben.

Mithilfe der Kamera-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 semantischen Informationen wiedergibt, die in dem Frame erkannt wurde.

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

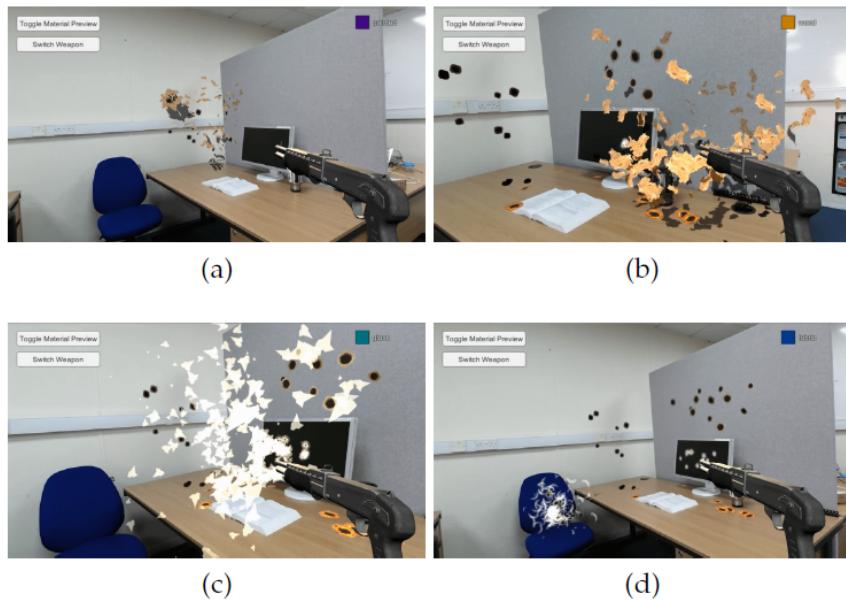


Abbildung 2: Es gibt unterschiedliche Interaktionen, wenn auf eine Wand (a), einen Tisch (b), einen Bildschirm (c) und einen Stuhl (d) geschossen wird.(Chen et al. 2018)

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 jedes Pixel semantische Informationen, sondern nur solche Pixel, welche 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, welcher in Augmented Reality abgebildet und erweitert werden soll. Umgebung kann beispielsweise ein Zimmer sein, in dem eine AR-Anwendung ausgeführt wird. Die AR-Umgebung umfasst sowohl die reale Umgebung als auch die virtuellen Elemente.

Augmented Reality weist drei grundlegende Merkmale auf, die ein möglichst nahtloses Verschmelzen der realen Welt mit den virtuellen Elementen unterstützen:

- 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.

In einer AR-Umgebung kann navigiert werden, indem der Nutzer sich physikalisch durch die reale Umgebung bewegt. Die reale Welt und die virtuellen Elemente stehen dabei immer in demselben räumlichen Verhältnis zueinander. Für AR ist die einzige Möglichkeit der Navigation. Andernfalls würde die Verschmelzung der virtuellen Elemente mit der realen Welt schlechter gelingen.

Da die reale Welt immer zu sehen ist, gibt sie sowohl eine Referenz als auch 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 Hard- und Software, die benötigt wird, um eine AR-Umgebung zu erzeugen. Es muss die Vermischung der realen Welt mit virtuellen Elementen anzeigen. Darüber hinaus muss es sowohl die Interaktionen des Nutzers mit virtuellen Elementen als auch die Interaktionen von virtuellen Elementen mit der realen Welt simulieren.

Ein AR-System ist in der Regel nicht an einen bestimmten Ort gebunden, sondern kann in unterschiedlichen Umgebungen eingesetzt werden, die diverse reale Gegenstände aufweisen. AR-Applikationen müssen in der Lage sein, die unterschiedlichen Umgebungen zu unterstützen. (Dörner et al. 2019)

3.1.3 Grundlagen zu 3D-Szenen für AR

Die digitalen 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 keine Zeitverzögerung zwischen der virtuellen Welt und der realen Welt bemerken.

Um Interaktionen 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 mithilfe von Sensoren verfolgt und in die Szene einbezogen 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, welche Aussagen über die Geometrie der Umgebung treffen. Beispielsweise haben AR-Geräte eine Tiefenkamera, welche 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 Hilfe der entstehenden 'Spatial Map' können digitale Elemente mit der Umgebung interagieren. Zudem wird die 'Spatial Map' bei der Darstellung der digitalen Elemente verwendet, um zu entscheiden, wie digitale Elemente die Umgebung verdecken oder von ihr verdeckt werden.(Microsoft 2018b)

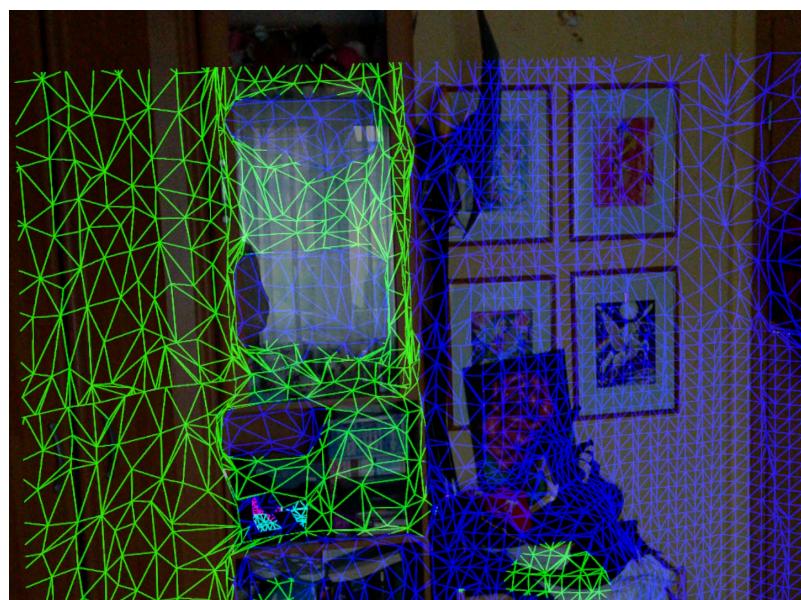


Abbildung 3: Spatial Map. Die Farben des Meshes zeigen die Entfernung des Nutzers zu den Objekten an.

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 eine Recheneinheit, welche über ein Kabel mit dem Display verbunden ist. Die transportable 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 sind einer der 'Denver' Kerne und zwei der 'ARM Cortex' Kerne für Applikationen nutzbar.

Die AR-Brille 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 2018b, 2020b)

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.

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

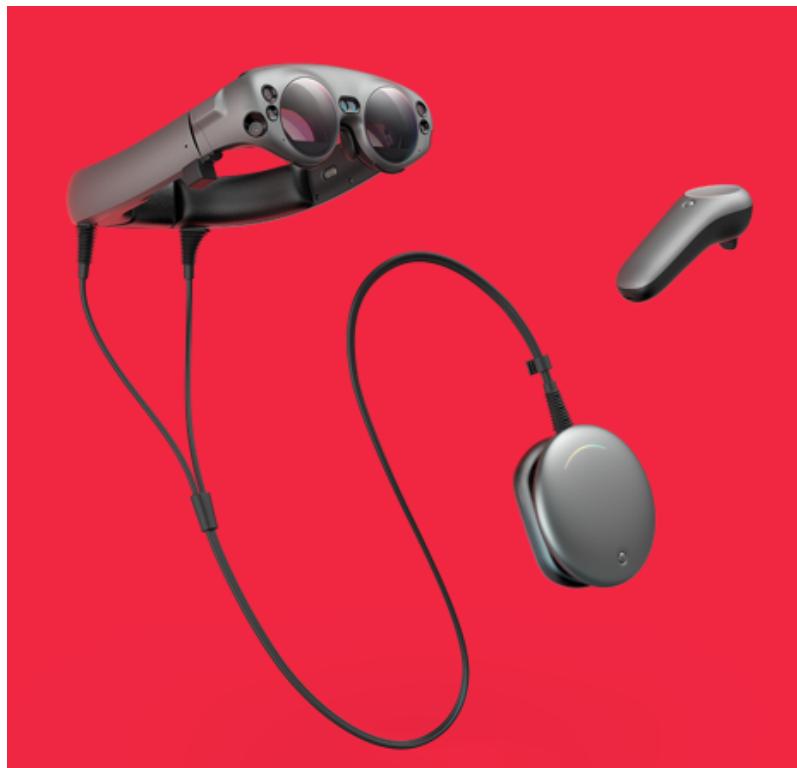


Abbildung 4: Magic Leap One AR-Brille.(MagicLeap 2018a)

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, 2020d, e)

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 die Verarbeitung des Controller-Inputs und die Verwaltung der Zugriffsrechte von Applikationen. Dazu gehört beispielsweise das Recht (die Permission), die Fotokamera und das Netzwerk zu nutzen.(Leap 2018; MagicLeap 2020d)

Selbst wenn mehrere Applikationen das Recht zur Kameranutzung haben, kann zu jedem Zeitpunkt nur eine Applikation auf die Fotokamera zugreifen. Die Applikationen müssen sich mit der Kamera-Ressource von Lumin OS verbinden. Diese reguliert, welche der Applikationen eine Fotoanfrage stellen darf und das aufgenommene Foto 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 oder Children weiterer GameObjects sind.(Unity 2017)

GameObjects haben weder eine Funktionalität noch ein Aussehen. Beides wird durch Components hinzugefügt. Einige Components sind bereits in Unity integriert. So erhält jedes GameObject eine Transform Component, die ihm eine Position in der Szene gibt. Darüber hinaus können Components mithilfe eines Mesh Renderers einem GameObject eine sichtbare 3D-Form geben. Components können auch komplexere Verhalten implementiert. Beispielsweise kann ein GameObject als Kamera der Szene fungieren. Um eine neue Funktionalität hinzuzufügen, kann ein C# Script geschrieben 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, welches das Set-up der Applikation erleichtert. Dazu gehören die Build-Optionen des Projektes und die Einstellung der Hauptkamera 'Unity Kamera'. Letztere rendert die Szene für das Display der AR-Brille 'Magic Leap One'. Die Kamera verfolgt die Bewegungen 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.(MagicLeap 2020c)
- *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.(MagicLeap 2020f)
- *MLRaycast* greift auf die Spatial Mapping Funktionalitäten von Lumin OS zu. Somit kann ein Raycast auf die Rekonstruktion der Umgebung beim Betriebssystem in Auftrag gegeben werden. Ein Raycast ist ein Strahl, welcher sich durch einen 3D-Raum bewegt. Der Ursprung und die Richtung des Strahles müssen angegeben werden. Objekte, die einen Schnittpunkt mit dem Strahl aufweisen, gelten als 'vom Raycast getroffen'.(MagicLeap 2020g; Unity 2020e)
- *MLSpatialMapper* zeigt die Rekonstruktion der Umgebung von Lumin OS als Mesh an. Dieses besteht aus GameObjects, welche von *MLSpatialMapper* erzeugt werden. Das Mesh wird ständig aktualisiert, um das geometrische Verständnis des Betriebssystems widerzuspiegeln.(MagicLeap 2019b, 2020e)

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. Folgende Methoden können unterschieden werden:

- *Awake* wird aufgerufen, wenn das GameObject initialisiert wird.
- *Update* wird zu jedem Frame der Anwendung aufgerufen.
- *OnDestroy* wird aufgerufen, wenn das GameObject aus der Szene gelöscht wird.

Monobehaviors können über globale Variablen verfügen. Wenn das Skript einem GameObject als Component angehört, kann der Inhalt der globalen Variable in Unity modifiziert werden. Dies eröffnet die Möglichkeit, einem Skript eine Referenz auf ein bestimmtes GameObject zu geben. Siehe Abbildung 5.

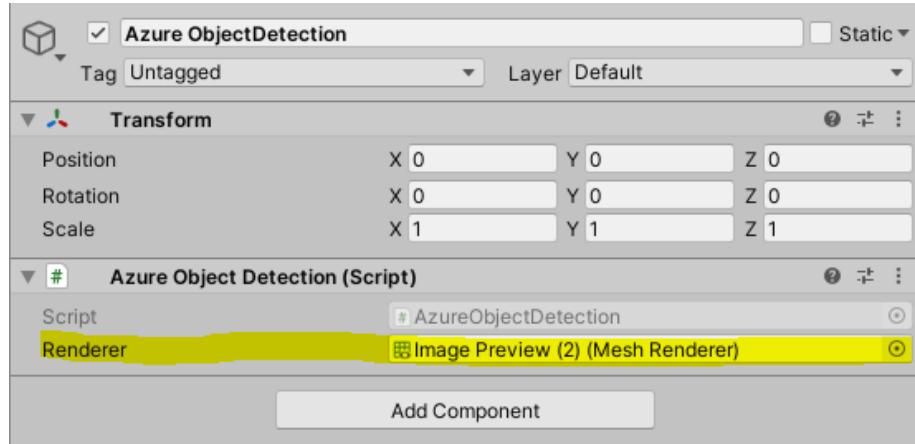


Abbildung 5: GameObject mit Components. Das Component 'Azure Object Detection' verfügt über eine globale Variable, welche auf eine Mesh Renderer Component eines anderen Objektes verweist.

Monobehaviors können als Singleton fungieren, indem sie eine globale statische Referenz auf sich selbst 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 welche das neue GameObject angefügt 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' genannt), in welchem 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. Dazu wird die Koordinate als Vektor angegeben und mit der *cameraToWorld* Matrix multipliziert. Das Resultat ist ein Vektor, welcher eine Koordinate im globalen Koordinatensystem angibt.(Unity 2020b, d)

3.2.5 Kamera in 3D-Computergrafik

View Frustum ist das Teilvolumen einer 3D-Szene, welche 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, welche 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 6.(Unity 2020c)

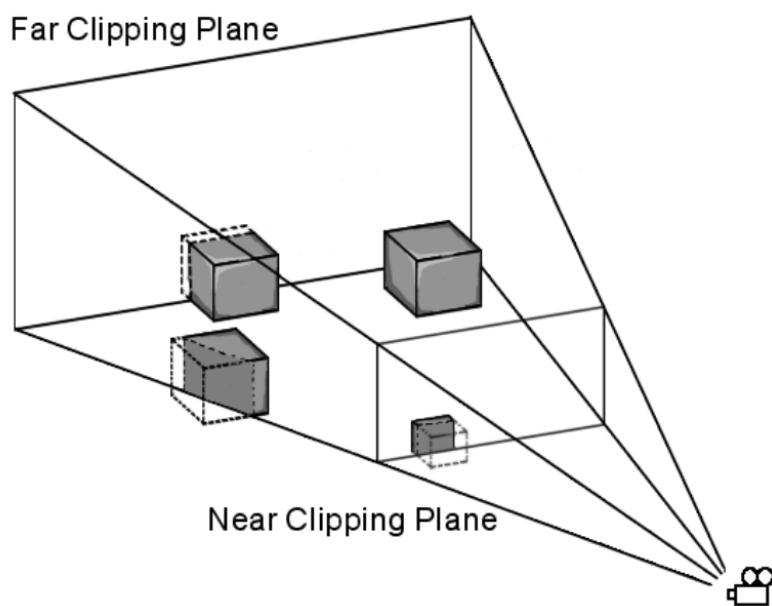


Abbildung 6: Diagramm des View Frustum mit Clipping Planes.

3.3 Computer Vision

Computer Vision hat zum Ziel, die Inhalte digitaler Bilder und Videos zu verstehen. Sowohl semantische als auch geometrische Inhalte werden dabei analysiert. Zu den Aufgaben von Computer Vision gehören Objekterkennung und Bildsegmentierung.

In diesem Bereich kommen Statistik, Bildverarbeitung und maschinelles Lernen zum Einsatz. Maschinelles Lernen wird eingesetzt, um neuronale Netze zu formen, welche in der Lage sind Bild- und Videoanalysen durchzuführen .(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 auch der Name 'Image Based Object Detection' verwendet. 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 werden sowohl eine Klasse als auch eine Bounding-Box bestimmt. Die Klasse gibt an, um welche Art von Objekt es sich handelt. So wird Beispielsweise angegeben, ob es sich um eine Tastatur oder einen Computerbildschirm handelt (Object Classification). Die Bounding-Box gibt ein Viereck in dem Bild an, in welchem sich das Objekt befindet(Object Localization). (Felzenszwalb et al. 2010; Zhao et al. 2019)

3.3.2 Artificial Neural Networks

'Artificial Neural Networks' sind Machine-Learning-Architekturen, welche beispielsweise Musik, Text oder Bilder nach Mustern durchsuchen. Artificial Neural Networks sind für keine genaue Aufgabe programmiert, sondern lernen indem sie mit Beispieldaten trainiert werden. Jedem Beispiel wird ein Label zugeordnet, welches angibt, ob das gesuchte Muster in dem Beispiel vorhanden ist. Die Struktur des Networks verfügt über Gewichte, welche 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 and Nash 2015)

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

Das Neural Network bekommt eine Menge an Daten als Input. Die Knoten arbeiten zusammen, um den Output zu erzeugen. Jeder einzelne Knoten führt eine Berechnung durch. Mithilfe von Gewichten wird entschieden, wie viel Einfluss das Rechenergebnis der einzelnen Knoten auf die nächste Ebene hat.(Jiao et al. 2019; O'Shea and 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 and 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, um Rechenzeit und Volumen der Trainingsdaten zu verringern.(Jiao et al. 2019; Jmour et al. 2018; O'Shea and Nash 2015)

Convolutional Neural Networks sind Machine-Learning-Architekturen, die darauf ausgelegt sind, Muster in Bildern zu erkennen. Sie müssen auf das zu erkennende Muster trainiert werden. Dazu wird ihnen eine Menge an Beispielbildern gegeben, die teilweise das Muster erhalten. Für jedes Beispiel wird der gewünschte Output angegeben, der erreicht werden soll. Die Struktur des Network verfügt über Gewichte, welche die Berechnung des Outputs beeinflussen. Mit jedem Trainingsbild passt das Network die Gewichte an, damit die Muster korrekt erkannt werden.(Jiao et al. 2019; O’Shea and Nash 2015)

Die Knoten in einer Ebene eines Convolutional Neural Networks 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 and Nash 2015)

3.3.4 Azure Computer Vision Services

‘Microsoft Azure’ ist eine Cloud-Computing-Plattform, welche im Jahr 2010 von Microsoft eingeführt wurde. Ihr Ziel ist es, Cloud-Services zur Verfügung zu stellen, die unter anderem Computer Vision Aufgaben durchführen. Diese sind über ‘REST-APIs’ erreichbar.

Eine ‘REST-API’ ist eine Programmierschnittstelle, die sich an dem Verhalten des World Wide Webs orientiert. ‘REST’ steht für ‘Representation State Transfer’ und bezeichnet einen Architekturansatz für die Kommunikation zwischen verteilten Systemen. REST verlangt unter anderem ein Client-Server-Modell und eine zustandslose Kommunikationen zwischen Client und Server. Jede Anfrage eines Clients beinhaltet alle Informationen, welche für die Antwort des Servers relevant sind.

Eine Rest-API kann mit http/s realisiert werden. Eine HTTP-Anfrage (HTTP-Request) wird verwendet, um auf eine Funktion eines Servers zuzugreifen. Dabei wird der Server durch seine Web URL angesprochen. Diese wird auch ‘Endpoint’ genannt. HTTP-Methoden, wie ‘GET’ und ‘POST’ teilen dem Service mit, welche Funktion durchgeführt werden soll. Mit der Methode ‘GET’ werden Daten von dem Server angefordert. Die Methode ‘POST’ erlaubt die Übermittlung 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 und die Endpoint-URL an. In den ‘Headers’ werden weitere Informationen über die Anfrage an den Server angegeben. Für die Anwendung einer API, wird der Authentifizierungsschlüssel des Clients in einem ‘Header’ gespeichert. Zuletzt hat der Request einen ‘Body’. Abhängig von der HTTP-Methode ist der ‘Body’ entweder leer, oder er beinhaltet 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.

Jeder 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 Responce-Code und einen Statustext an. Der Responce-Code teilt dem Client mit, ob die Anfrage erfolgreich bearbeitet wurde. Wenn ein Error aufgetreten ist, gibt der Responce-Code Auskunft über die Art des Errors. Der Statustext gibt eine kurze Erklärung des Responce-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

Bei erfolgreicher Bearbeitung beinhaltet der Body die Antwort auf die HTTP-Anfrage.

3.3.5 Azure Object Detection

Microsoft Azure bietet einen Computer Vision Service an. Dabei handelt es sich um mehrere Künstliche-Intelligenz-Modelle, die für unterschiedliche Aufgaben trainiert wurden. Dazu gehört unter anderem ein Service für Object Detection. Dabei sendet der Anwender ein Bild an Microsoft. Der Service verarbeitet das Bild und schickt dem Anwender ein Ergebnis in Form einer Json-Datei zurück.(Microsoft 2010, 2018a, 2019b, 2020)

Die Object Detection basiert auf einem trainierten KI-Modell. Dieses weist unter anderem folgende Limitierungen auf: Zum einen kann es nur Objekte erkennen, für die es trainiert wurde. Zudem können Objekte nicht erkannt werden, die in dem Foto sehr klein sind oder nah bei anderen Objekten liegen.(Microsoft 2019a)

Der Object Detection Service ist durch eine REST-API erreichbar. Durch eine POST-Anforderung wird eine Analyse angefragt. In dem Body der HTTP-Anfrage werden dabei die Bilddaten übertragen. Die Response-Nachricht beinhaltet eine Json-Datei, welche die gefundenen Objekte und deren Positionen auf dem Foto beinhaltet. Um den Object Detection 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 Bildanalyse wird in der Response-Nachricht zurückgeschickt. Der Response-Code 200 gibt an, dass die Analyse erfolgreich durchgeführt wurde. In diesem Falle wird eine Json-Datei mitgeschickt. Beispiel:

```
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 Objekt einen 'confidence' Score. Dieser gibt die Wahrscheinlichkeit an, dass das Objekt korrekt erkannt wurde. Zusätzlich zu den Objekten werden eine requestId und Metadaten über das Bild zurückgeschickt.

3.3.6 Azure Custom Vision

Mit 'Azure Custom Vision' bietet Microsoft auch ein untrainiertes Object Detection KI-Modell an. Dieses muss vor dem Einsatz von einem Nutzer trainiert werden. Custom Vision kann als Ergänzung zu Azure Object Detection verwendet werden, da es für spezifische Objekte trainiert werden kann. (Micosoft 2018)

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. Jeder Tag repräsentiert eine Art von Gegenstand (Objekt-Klasse). Um einem Foto ein Tag anzufügen, muss eine rechteckige Region des Fotos markiert werden, in welchem der jeweilige Gegenstand zu sehen ist. Siehe Abbildung 7.

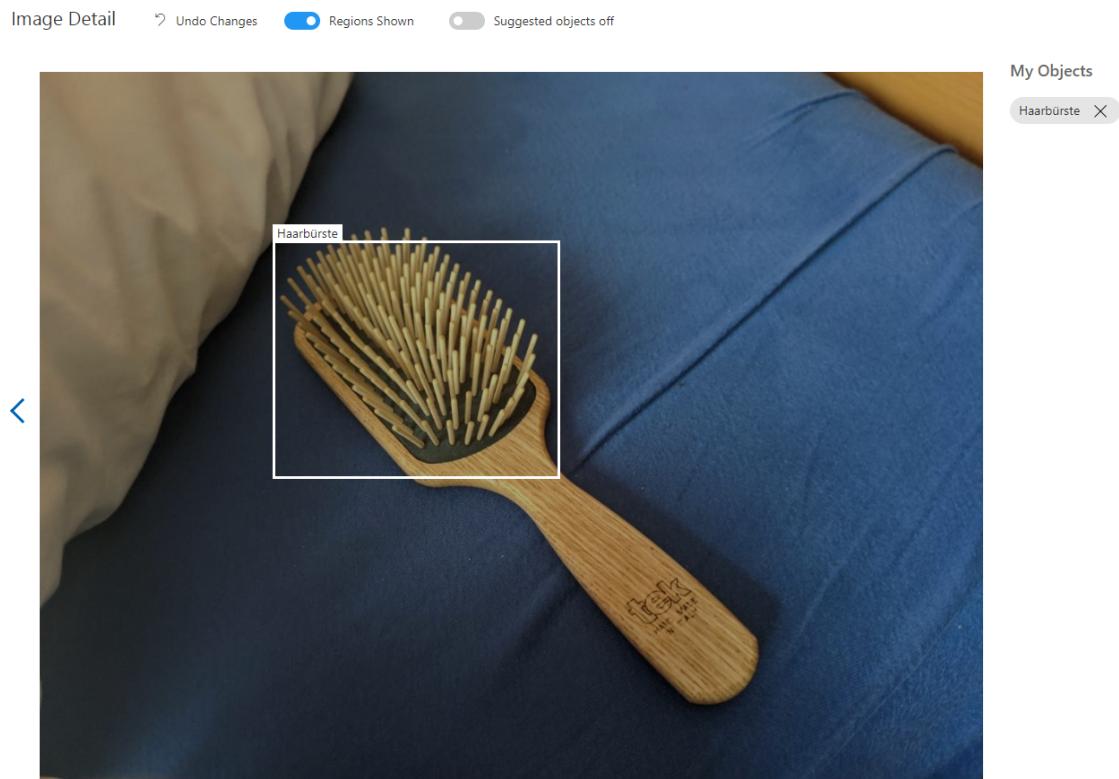


Abbildung 7: Trainingsbild für die Objekterkennung einer Haarbürste.

Für jede Objekt-Klasse müssen mindestens 15 Bilder hochgeladen werden, in welchen der entsprechende Tag vorkommt. Anschließend kann der Trainingsprozess gestartet werden. Eine größere Anzahl von Bildern in dem Trainingsset erhöht die Genauigkeit der Objekterkennung. Azure empfiehlt die Verwendung von mindestens 50 Bildern pro Tag. Bei einer geringen Anzahl von Trainingsbildern haben die einzelnen Bilder einen sehr großen Einfluss auf das Modell.

Durch das Training wird eine 'Iteration' des Modells erzeugt. Diese kann verwendet werden, um Object Detection durchzuführen. Bei jedem Start eines neuen Trainingsprozesses, wird eine weitere Iteration erstellt. Alle Iterationen werden unter einer eindeutigen Projekt-ID gespeichert und während einer Testphase evaluiert. Für jede Objekt-Klasse in einer Iteration werden die folgenden drei Metriken angesetzt:

- Precision - bezeichnet die Wahrscheinlichkeit, dass ein gefundenes Objekt, tatsächlich der angegeben Klasse angehört. (Die Wahrscheinlichkeit, dass es kein false positive ist.)
- Recall - greift aus einer Menge an Objekten, die einer Klasse angehören, den Prozentsatz an Objekten heraus, der von dem Modell korrekt lokalisiert und klassifiziert wurde.
- A.P (Average Precision) - bezeichnet 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 8.

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 8: Die Evaluierung des Modells

Mit einer erzeugten Iteration des Modells kann eine Object Detection durchgeführt werden. Der Custom Vision Service ist durch eine REST-API erreichbar. Der Endpoint beinhaltet die ID des Projektes und die Nummer der Iteration, welche verwendet werden soll. In der POST-Nachricht werden sowohl ein Authentifizierungsschlüssel als auch ein Bild mitgeschickt, welches von der Iteration analysiert werden soll.

Das Resultat der Analyse ist eine Json-Datei. Der Aufbau der Datei unterscheidet sich ein wenig von dem Dateiaufbau der Azure Object Detection.

```

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-afc4-4ff4-8cd-3
      ee8fe79cb8f","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}}]}

```

In der Antwort werden Informationen über die Iteration wiedergegeben, welche für die Analyse genutzt wurde. Die gefundenen Objekte werden in der Liste "predictions" aufgeführt.

zählt. Die Objektklasse wird in "tagName" gespeichert. Die "probability" gibt das Vertrauen des Modells darin an, dass das Objekt korrekt erkannt wurde. Die erkannten Objekte müssen nach ihrer Probability gefiltert werden, da auch Objekte mit einer sehr niedrigen Probability in der Json Datei enthalten sind.

4 Umsetzung

In dieser Arbeit wird angestrebt, Objekte einer AR-Umgebung zu erkennen und mit Labels zu versehen. RGB-Bilder der Umgebung werden aufgenommen und von neuronalen Netzen nach Objekten durchsucht. Gefundenen Objekte werden in einer 3D-Rekonstruktion der Umgebung lokalisiert und mit Labels markiert. Durch diesen Vorgehen können Objekte der Umgebung automatisch erkannt und mit Labels ausgestattet werden.

4.1 Anforderungen

Die Objekterkennung und das Setzen der Labels soll ohne Zutun eines menschlichen Nutzers ablaufen. Der Vorgang braucht nicht in Echtzeit zu geschehen. Er soll jedoch eine Laufzeit haben, welche das Labeln eines Raumes binnen 10 Minuten erlaubt.

Objekte werden anhand des Mittelpunktes ihrer Bounding-Box markiert. Die Foto-Position des Objektes soll möglichst genau auf die 3D-Szene übertragen werden. An die berechnete 3D-Position der Szene wird ein Label gesetzt, welches die Klasse des Objektes wiedergibt. Jedes Objekt wird durch maximal ein Label annotiert. Wenn ein Objekt mehrmals erkannt wird, soll kein neues Labels in der Szene erzeugt werden. Die Applikation soll erkennen, dass das Objekt bereits durch ein Label markiert ist.

Die Applikation soll über ein minimales View Management verfügen. Die Labels werden als 3D-Objekte dort in der Szene platziert, wo das entsprechende Objekt lokalisiert wurde. Dabei wird nicht darauf geachtet, ob sie andere Labels oder reale Gegenstände verdecken. Wenn der Nutzer sich in der Szene bewegt, sollen die Labels nachgeführt werden, damit der Schriftzug der Labels jederzeit dem Nutzer zugewandt ist.

4.2 Design der Objekternnung

Im Folgenden werden die Arbeitsschritte einer Objekterkennung 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 an Azure Object Detection und Azure Custom Vision geschickt. Die beiden Services untersuchen das Foto nach Objekten, um deren Klassen und Positionen auf dem Foto anzugeben.

Für jedes Objekt soll ein Label erstellt werden, welches 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. Diese Fotorepräsentation muss sowohl die korrekte Skalierung als auch die korrekte Position und Rotation haben, um das räumliche Verhältnis zwischen der realen Fotokamera und der Umgebung nachbilden zu können.

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 Position auf der Clipping Plane bestimmt.

Als Nächstes wird ein Raycast von der Kamera aus durch die Clipping Plane Position geschickt. Der Raycast schneidet sich mit einem Mesh, welches die reale Welt abbildet. Die getroffene Position wird mit einem Label markiert. Dort befindet sich das Objekt, welches 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.

4.3 Design der Labels

Die Labels der erkannten Objekte werden als 3D-Objekte in der Szene platziert. Sie bestehen aus einer Sphäre (Anchor) und einem Schriftzug. Die Sphäre liegt an der 3D-Position des Objektes. Der Schriftzug befindet sich über der Sphäre. Er gibt die Klasse des Objektes wieder, welches durch ein neuronales Netzwerk erkannt wurde.

Um sicherzustellen, dass Objekte durch maximal ein Label annotiert werden, wird jedes Mal, wenn ein Label in der Szene platziert werden soll, überprüft, welche Labels sich in der Nähe befinden. Gibt es bereits ein Label mit demselben Schriftzug, wird davon ausgegangen, dass es sich auf dasselbe Objekt der realen Welt bezieht. Ein Gegenstand wurde demnach zweimal erkannt. Aus den beiden Objekterkennungen gehen zwei 3D-Positionen hervor, die für das Objekt lokalisiert werden.

Beide 3D-Positionen werden jeweils mit einem 3D-Objekt markiert. Das Label des Objektes wird in den Mittelpunkt der Positionen gesetzt. Fall das Objekt öfter als zwei Mal erkannt wird, erfolgt die Markierung aller 3D-Positionen, welche durch die Objekterkennungen bestimmt wurden. Das Label sitzt in der Mitte aller lokalisierten Positionen.

Wenn der Nutzer sich in der Szene bewegt, rotieren die Labels sich um ihre Anchor, damit der Schriftzug dem Nutzer immer zugewandt bleibt.

Die Labels werden in einer Unity-Szene als GameObjects umgesetzt. Letzteren wird über Components die beschriebene 3D-Form der Labels gegeben, die von der Unity-Kamera gerendert wird. Die gerenderten Frames werden von dem Display der Magic Leap Brille angezeigt.

4.4 Architektur

Die AR-Brille 'Magic Leap One' übernimmt alle Berechnungen im 3D-Raum und führt Spatial Mapping durch. Da die Analyse von RGB-Fotos sehr speicher- und rechenintensiv ist, wird sie an eine REST-API delegiert. Die AR-Brille 'Magic Leap One' wird als Interaktionsmöglichkeiten für den Nutzer verwendet. Zudem nimmt sie 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 9

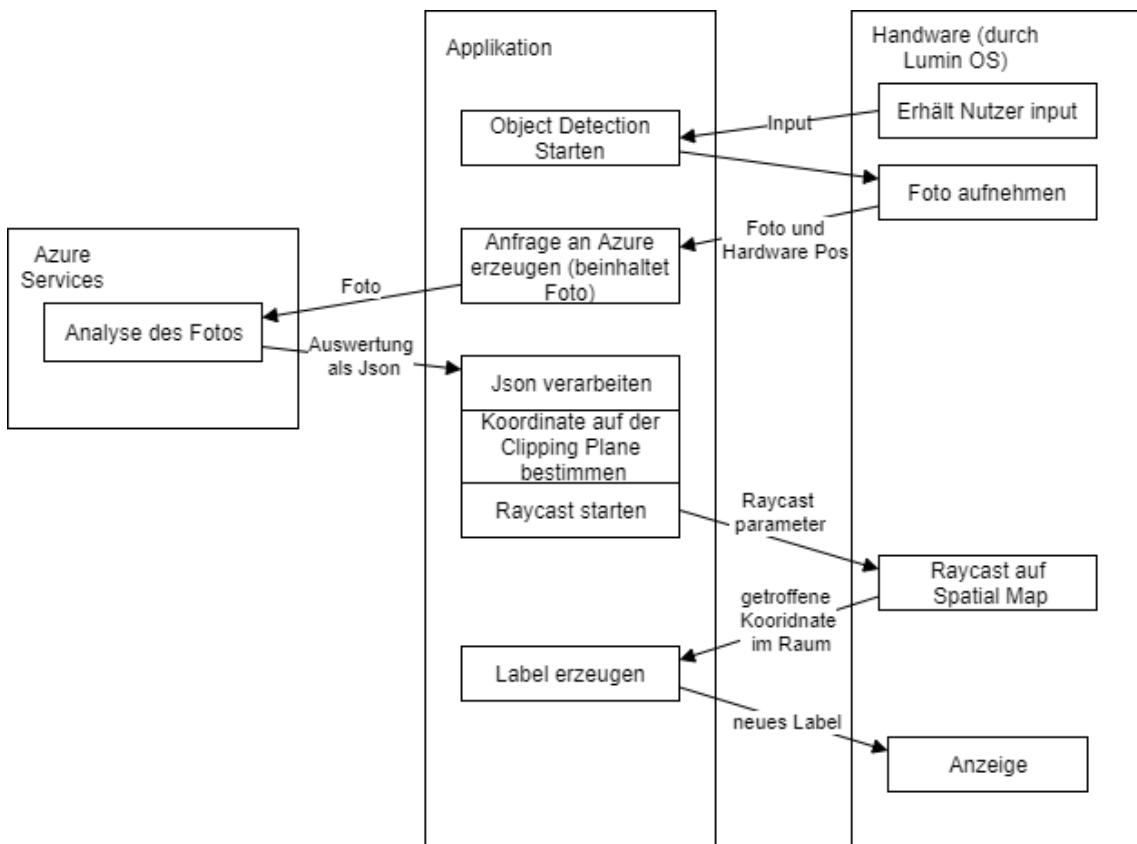


Abbildung 9: Diagramm der Architektur inklusive Bearbeitungsschritte und Informationsweitergabe.

Die Arbeit wird mit der Entwicklungsumgebung Unity erzeugt. Als AR-Brille wird das Gerät 'Magic Leap One' verwendet. Für das Set-up des Unity-Projektes wird das Unity-Template von Magic Leap genutzt. Zusätzlich werden einige vorgefertigte Klassen von Magic Leap verwendet. Dazu gehören: *MLInput*, *MLCamera*, *MLRaycast*, *MLPrivilegeRequestBehavior* und *MLSpatialMapper*. Diese Klassen greifen auf Funktionalitäten des Betriebssystems Lumin OS zu.(MagicLeap 2020a)

Die benötigten Funktionalitäten für die Integration der Object Detection mit der REST-API werden in mehreren Script-Klassen umgesetzt. Ein Großteil der Scripts verhält sich wie Singletons. Sie existieren nur einmalig in der Szene. Das Klassendiagramm auf Abbildung 10 zeigt die Scripts und ihre Relationen zueinander.

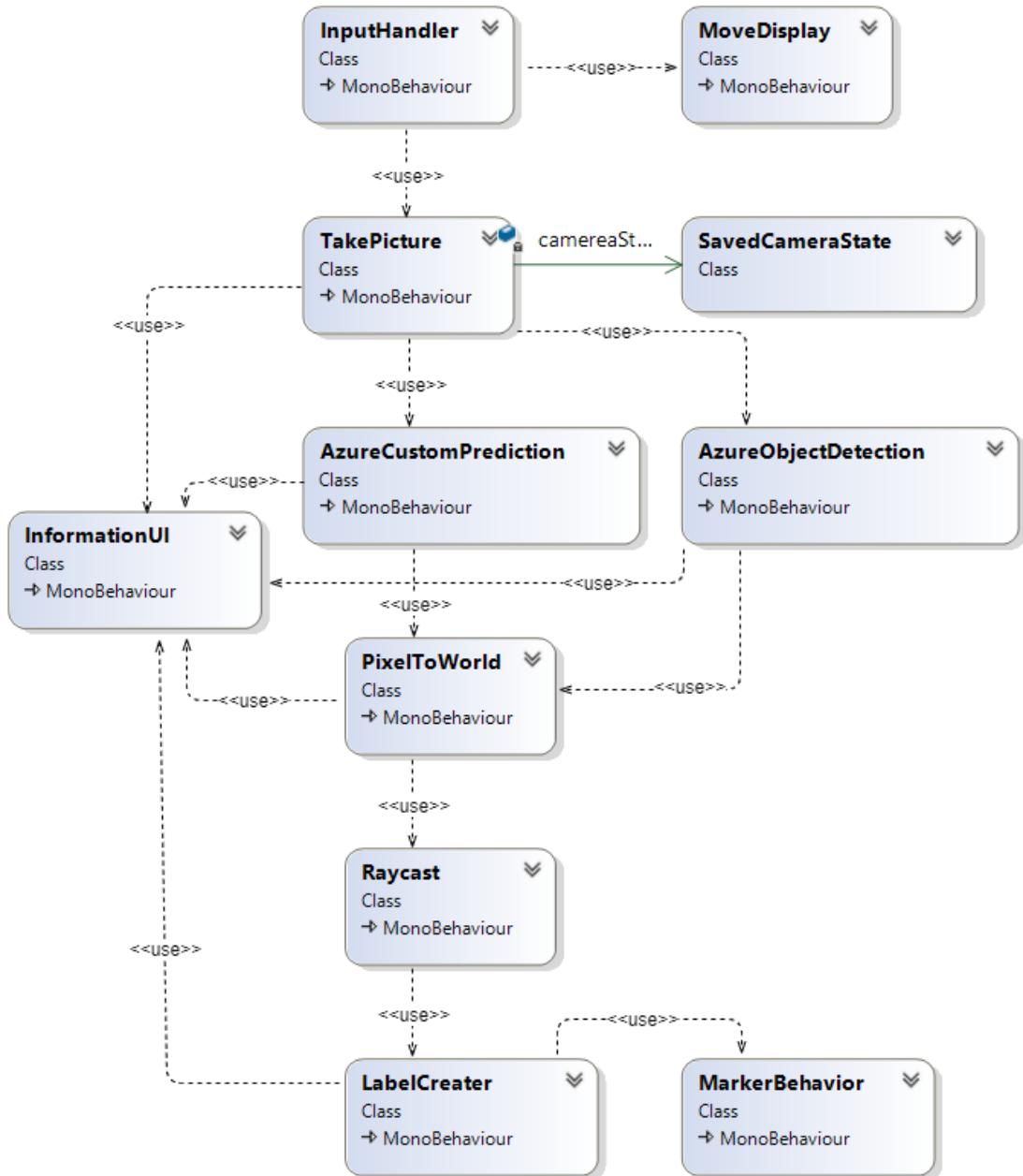


Abbildung 10: Klassendiagramme der Scripts

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

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

Sobald ein Objekt erkannt und eine Position auf dem Mesh der Umgebung bestimmt wird, erfolgt der Aufruf von *LabelCreator*. Diese Klasse erzeugt das Label des erkannten Objektes. Jedes Label wird als GameObject in der Unity-Szene erzeugt. Die Labels verfügen jeweils über ein *MarkerBehavior* Component. Mithilfe von *MarkerBehavior* kann der Schriftzug des jeweiligen Labels angepasst werden.

4.5 Interaktion

Die Klasse *InputHandler* verarbeitet den Input des Nutzers und startet entsprechende Aktionen durch die Klassen *MoveDisplay* und *TakePicture*. Der Input ist durch *MLInput* abfragbar. Diese Klasse stellt Informationen über den Zustand des Controllers zur Verfügung. *InputHandler* nutzt *MLInput*, um den Controller zu überwachen. Wenn Tasten des Controllers gedrückt werden, startet *InputHandler* folgende Aktionen:

- Trigger: Start der Objekterkennung mit *TakePicture*
- Home Button: Mittige Platzierung des UI-Element vor dem Display
- Bumper: Verstecken der Labels
- Bumper halten: Entfernen des zuletzt erzeugten Labels

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. Der *LabelCreator* zeigt eine Liste aller Labels an, die in der Szene existieren. Siehe Abbildung 11.

Der *LabelCreator* ist für die Erstellung der Labels verantwortlich und sorgt dafür, dass die Labels für den Nutzer lesbar sind. Zu diesem Zweck 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, welches die Spatial Map der Umgebung wiedergibt. Das Spatial Mapping wird von dem Betriebssystem Lumin OS durchgeführt. Das Mesh wird durch die Klasse *MLSpatialMapper* von Magic Leap erzeugt und angezeigt. Siehe Abbildung 11.

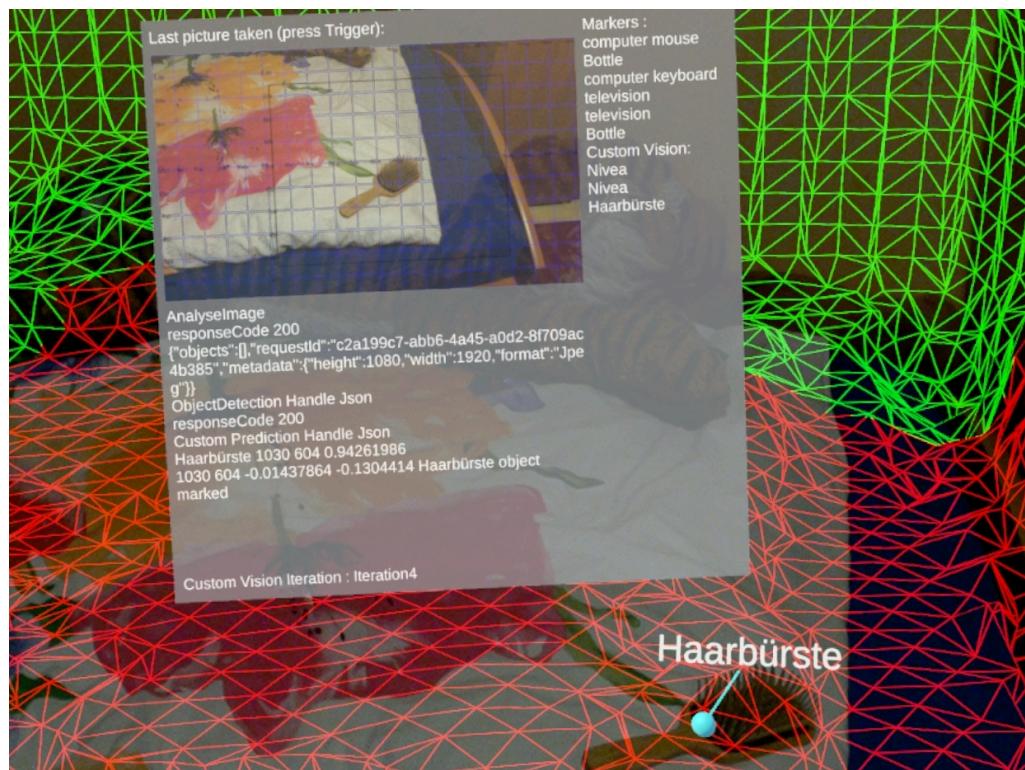


Abbildung 11: Ausgabe

4.6 Implementierung der Objekterkennung

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

4.7 Ein Foto aufnehmen

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

Wenn die Methode *TakeImage* aufgerufen wird, startet der Objekterkennung-Prozess. Die Foto-Aufnahme 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, sobald das Foto fertig ist. Die Daten des Bildes und der *SavedCameraState* werden, an die beiden Klassen *AzureObjectDetection* und *AzureCustomPrediction* weitergegeben. Diese führen Anfragen an die beiden Objekt Detection Services 'Azure Object Detection' und 'Azure Custom Vision' durch.

4.7.1 Object Detection

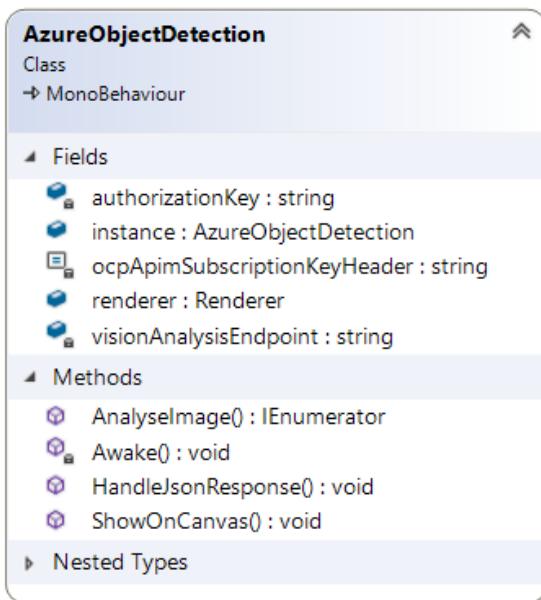


Abbildung 12: Klassendiagramm von *AzureObjectDetection*

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. Sobald die Antwort eintrifft, wird anhand des Response Codes geprüft, ob bei einem Fehler bei dem Request auftrat. Beispielsweise kann die Internetverbindung gestört sein oder die Authentifizierung abgelehnt werden. Wenn es

keinen Fehler gab, wird eine Json-Datei bei der Antwort mitgeschickt. Darin wird für jedes gefundene Objekt auf dem Foto eine Bezeichnung und eine Bounding-Box angegeben.

Die Json-Datei wird in *HandleJsonResponse* verarbeitet. Für den erwarteten Aufbau der Datei gibt es drei Klassen: 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 13. (Unity 2020a)

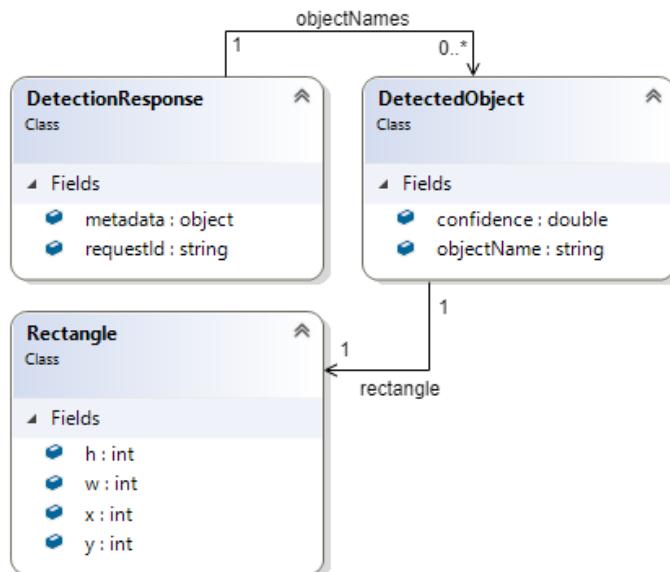


Abbildung 13: Klassendiagramm für die Umwandlung der Json-Datei in Objekte.

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. Zu diesem Zweck wird der *Cast* Methode der Mittelpunkt der Bounding-Box als u,v Foto-Koordinaten für das *DetectedObject* übergeben.

```

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

4.7.2 Von dem Foto zum 3D-Raum

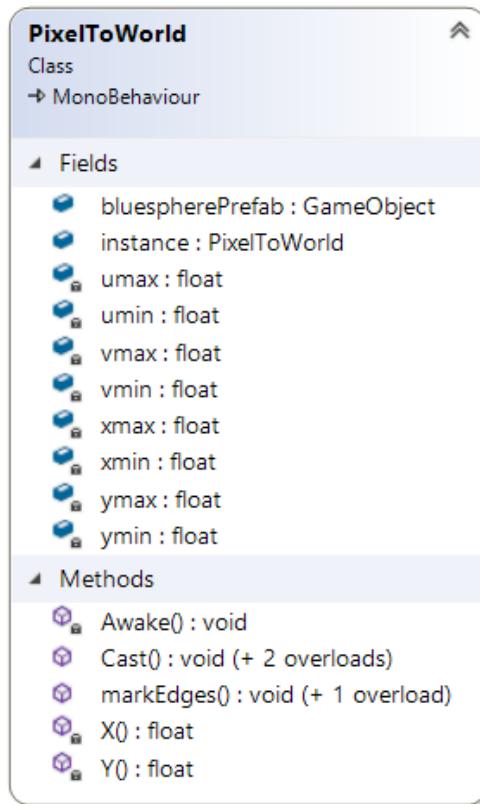


Abbildung 14: Klassendiagramm von *PixelToWorld*

Ein gefundenes Foto-Objekt soll in der 3D-Abbildung der realen Welt lokalisiert werden. Dafür nutzt die Methode *Cast* sowohl die u,v Foto-Koordinaten des Objekts als auch einen *SavedCameraState*. Der *SavedCameraState* beschreibt die Position der Unity-Kamera zum Aufnahmepunkt des Fotos. *SavedCameraState* beinhaltet zum einen die *cameraToWorldMatrix* und zum anderen den Ursprung der Kamera.

Das Foto kann mit dem Display und somit mit der Clipping Plane der Unity-Kamera approximiert werden. Die u,v Foto-Koordinaten werden in x,y,z Koordinaten des Camera Spaces umgewandelt. Die z-Achse verläuft durch den Ursprung der Kamera und folgt deren Blickrichtung. Punkte, welche sich auf der Clipping Plane befinden, sind 0.4 Einheiten von dem Ursprung der Kamera in Richtung der z-Achse entfernt. In dem Camera Space wird diese Entfernung mit $z = -0.4$ angegeben.

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

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

Mit den ausprobierten x und y-Werten ergeben sich Intervalle für die beiden Achsen x und y. Mithilfe einer Kombination der beiden Intervalle, können alle möglichen Foto-Koordinaten auf die Clipping Plane abgebildet werden. Die Intervalle lauten: [-0.2949, 0.2295] für x und [0.1546, -0.1507] für y. Durch die gewählten Intervallgrenzen wird

die Position und Skalierung des Fotos in Relation zu dem Display berücksichtigt. Siehe Kapitel 4.8 für die Entwicklung der *Cast* Methode und die Ermittlung der Intervallgrenzen.

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 für *u,v* berechnet. Diese Position wird, mithilfe der *cameraToWorldMatrix* des *SavedCameraState*, in eine Position *p* des globalen Koordinatensystems umgewandelt. Auf diese Weise werden sowohl Position als auch 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 }
```

4.7.3 Raycast

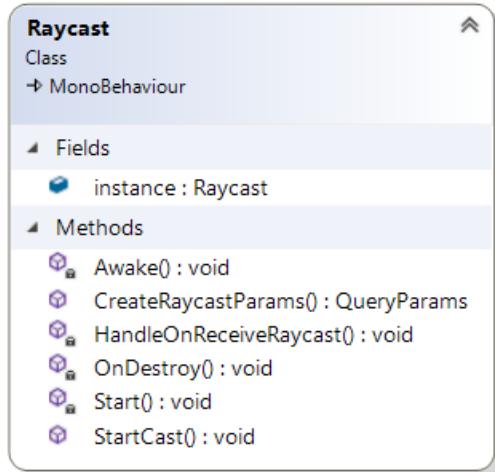


Abbildung 15: Klassendiagramm von Raycast

Als Nächstes wird ein Raycast durch den Ursprung der Kamera und die Position p gesendet. *MLRaycast* wird genutzt, um einen Schnittpunkt mit der Rekonstruktion der Welt von Lumin OS zu bestimmen. Die Stelle, welche von dem Raycast getroffen wird, 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 von dem Raycast getroffene Stelle der AR-Umgebung. Diese wird an die Methode *CreateMarker* von der Klasse *LabelCreator* weitergegeben.

4.7.4 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 Label für das *DetectedObject* in der 3D-Umgebung dient. Das Prefab besteht aus einer Sphäre und einem Schriftzug. Dem neu instanzierten GameObject wird die Bezeichnung des *DetectedObject* als Schriftzug zugewiesen. Siehe Abbildung 16.

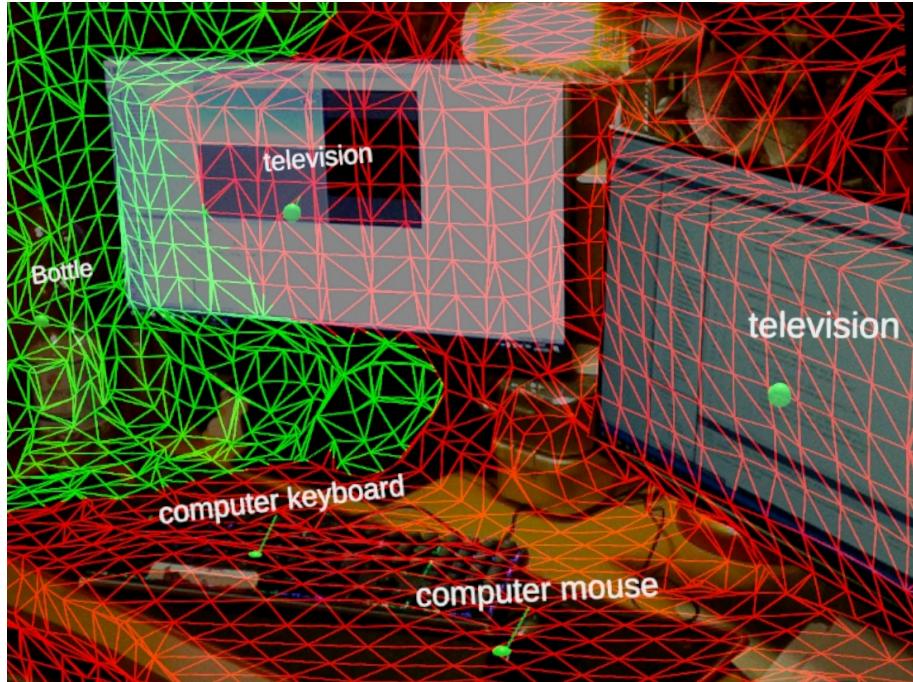


Abbildung 16: Labels in der Szene

Bevor ein neues Label erzeugt werden kann, muss überprüft werden, ob das Objekt, welches dadurch annotiert werden soll, bereits ein Label hat.

Zu diesem Zweck werden alle bereits existierenden Labels durchgegangen. Dabei wird nach Labels gesucht, die in der Nähe des zukünftigen Labels liegen und denselben Schriftzug aufweisen. Bei Auffinden eines solchen Labels wird davon ausgegangen, dass es sich auf dasselbe Objekt bezieht wie das zukünftige Label. In diesem Falle wird kein neues Label erzeugt, sondern das Alte modifiziert. Diese Modifikation ist in der Methode *UpdateLocation* des *MarkerBehaviors* implementiert, welches jedes Label besitzt. Der Methode *UpdateLocation* wird die 3D-Position übergeben, an welcher ursprünglich das neue Label erzeugt werden sollte.

MarkerBehavior speichert alle 3D-Positionen in einer Liste ab, die jemals für das dazugehörige Label angegeben wurden. Dazu zählen sowohl die Position an welcher das Label initialisiert wurde, als auch alle Positionen, die durch *UpdateLocation* übergeben wurden. Eine neu übergebene 3D-Position wird durch eine kleine Sphäre markiert. Das Label wird in den Mittelpunkt aller gespeicherten Positionen gesetzt.

Als Resultat werden alle Positionen, an welchen das Objekt in der Szene lokalisiert wurde, jeweils mit einer kleinen Sphäre markiert. Das Label befindet sich im Mittelpunkt aller Positionen. Siehe Abbildungen 17 und 18.

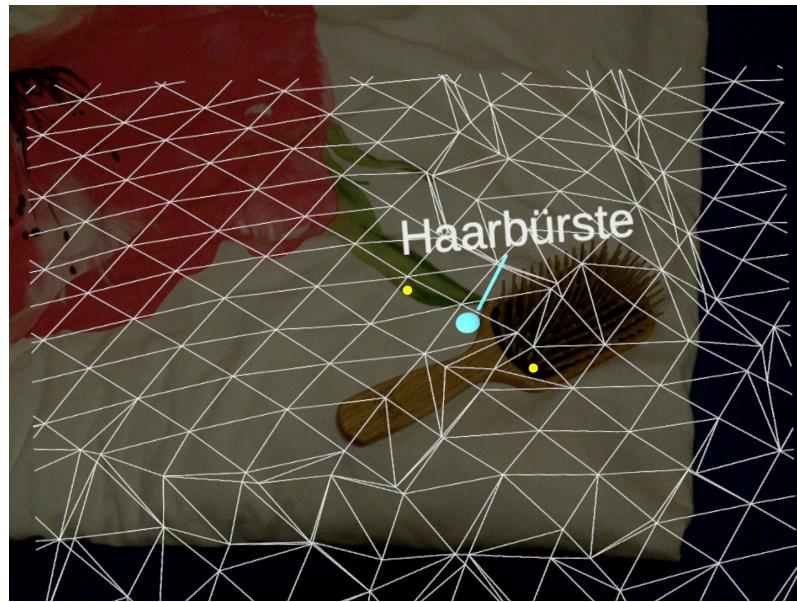


Abbildung 17: Haarbürste zweimal erkannt.

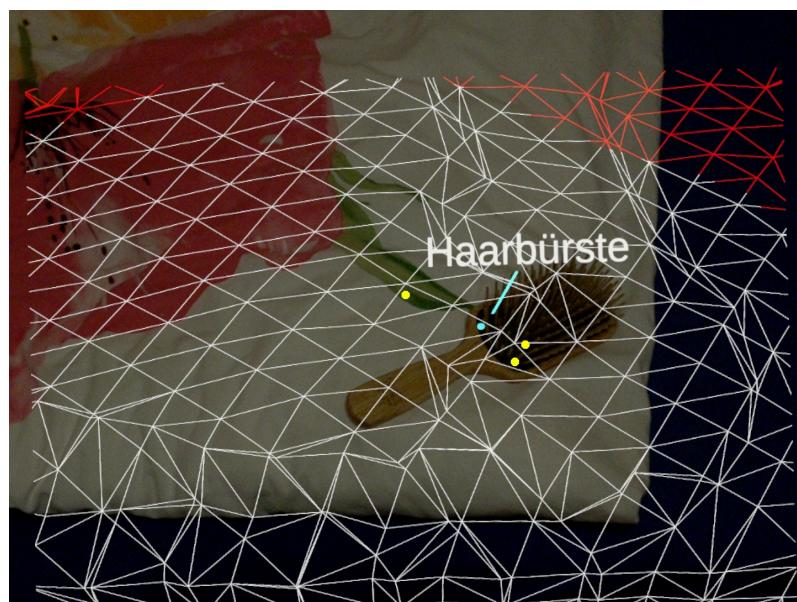


Abbildung 18: Haarbürste dreimal erkannt.

Bei jedem Frame der Applikation werden alle Labels zur Kamera hingedreht, sodass eine rechtwinklige Ausrichtung zur Kamera erfolgt. Dieses Verhalten ist in *MarkerBehavior* implementiert.

```
1 public void Update()
2 {
3     mainMarker.transform.LookAt(Camera.main.transform.position);
4 }
```

Auf diese Weise werden die Labels der Kamera nachgeführt. Solange ein Label in dem View Frustum liegt, kann der Nutzer es lesen. Siehe Anhang 8.1.

4.7.5 Azure Custom Vision

Neben 'Azure Object Detection' wird auch der Service 'Azure Custom Vision' zur Bildanalyse verwendet. Dieser Service bietet ein Object Detection Modell an, welches mithilfe einer Webseite trainiert wird. Als 'Iteration' wird ein trainiertes KI-Modell bezeichnet. Dieses kann zur Objekterkennung verwendet werden. Die Anfragen an eine Iteration geschieht in der Klasse *AzureCustomPrediction*. Ähnlich wie bei *AzureObjectDetection* wird ein Webrequest erstellt. Dieser beinhaltet einen Authentifizierungs-Schlüssel und ein Foto. Die Anfragen an die beiden Services werden parallel in unterschiedlichen Threads erstellt und bearbeitet.

In der Antwort wird eine Json-Datei zurückgeschickt, welche die gefundenen Objekte angibt. Da die Json-Datei ein etwas anderes Format als das Analyseresultat von Azure Object Detection hat, verfügt *AzureCustomVision* über eine eigene *HandleJsonResponse* Methode. Dabei gibt Azure Custom Vision für jedes *DetectedObject* eine Probability an. Diese sagt aus, wie groß das Vertrauen des Modells darin ist, dass das Objekt korrekt erkannt wurde. Mithilfe eines definierten Schwellenwerts wird entschieden, 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. Diese Methode behandelt alle Objekte gleich, unabhängig davon, von welchem Object Detection Service sie erkannt wurden - entweder von Azure Custom Vision oder von Azure Object Detection. Die Position des Objektes wird in der Szene mithilfe von *Cast* und einem Raycast lokalisiert. Anschließend erzeugt *LabelCreator* ein Label an der entsprechenden Position.

Labels für Objekte von Azure Custom Vision unterscheiden sich lediglich durch eine unterschiedliche Farbgebung von Labels, welche durch Azure Object Detection erzeugt werden.

Iterationen von Azure Custom Vision Das Custom Vision Modell wird auf drei unterschiedliche Objekte trainiert: eine Farbtube, eine blaue Dose und eine Haarbürste. Dabei werden sechs Iterationen erstellt.

Iteration 1 Die Iteration 1 ist darauf trainiert, eine Farbtube zu erkennen. Bei der Nutzung dieser Iteration tritt das Problem vieler fehlerhafter Objekterkennungen auf. Es werden Acrylfarben an Stellen erkannt, an denen es keine gibt. Siehe Abbildung 19.

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

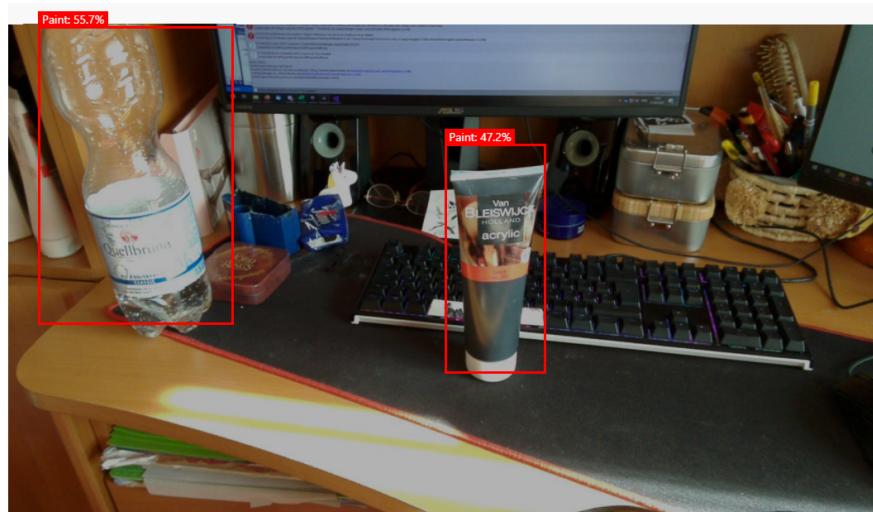


Abbildung 19: Iteration 1 Analysebeispiel. Eine Wasserflasche und eine Farbtube werden beide als Farbtuben erkannt. Die Wasserflasche wird mit einer Probability von 55.7 Prozent markiert, während die tatsächliche Farbtube eine Probability von 47.5 Prozent hat.

Iteration 2 Die Iteration 2 wird darauf trainiert, eine blaue Nivea-Dose zu erkennen. Form und Farbe der Dose sind sehr simpel. Dennoch ist auch dieser Gegenstand nicht leicht zu erkennen.

Die Evaluierung des Modells lautet: 80% Precision, 100% Recall, 100% mean Average Precision. Das bedeutet: In der Testphase kann das Modell alle auf den Bildern erhaltenen Nivea-Dosen erkennen (100% Recall). Es markiert jedoch auch Objekte als Dosen, die keine Dosen sind (80% Precision).

Auch bei der Nutzung der Iteration außerhalb der Testphase werden viele Gegenstände fälschlicherweise als Nivea-Dosen erkannt (false positives).

Iteration 3 Mit der Iteration 3 wird eine Verbesserung von Iteration 2 beabsichtigt. Für diesen Zweck wird die Menge an Trainingsdaten um 10 zusätzliche Fotos erweitert. Die Datenmenge steigt damit von ursprünglich 18 auf 28 Fotos. In den Trainingsdaten wird die Dose auf unterschiedlichen Hintergründen abgebildet, welche sich durch Farbe und Muster unterscheiden.

Die Evaluierung des Modells lautet: 75% Precision, 100% Recall, 100% mean Average Precision. Auch diese Iteration markiert in der Testphase häufig Objekte fälschlicherweise als Dosen (75% Precision). Bei der Verwendung außerhalb der Testphase werden ebenfalls Objekte fälschlicherweise als Dosen markiert. Die beabsichtigte Verbesserung ist damit nicht gelungen.

Iteration 4 Mit der Iteration 4 wird ein weiterer Verbesserungsversuch durchgeführt. Zu diesem Zweck werden solche Trainingsfotos aus der Datenmenge entfernt, welche die Dose aus einem seitlichen Blickwinkel zeigen. Die Dose soll nur erkannt werden, wenn sie auf dem Foto von oben zu sehen ist. Erwartet wird, dass während der Nutzung der Iteration weniger Gegenstände fälschlicherweise als Nivea-Dose erkannt werden.

In der Testphase wird diese Erwartung erfüllt: Precision und Recall der Nivea-Dose steigen auf 100 Prozent. Das Modell kann in dem Test alle Nivea-Dosen fehlerfrei erkennen. Bei der Verwendung des Modells außerhalb der Testphase zeigt sich allerdings, ein an-

deres Ergebnis: Häufig werden Objekte fälschlicherweise als Nivea-Dose markiert. Dies liegt die Vermutung nahe, dass das Objekt 'Nivea Dose' grundsätzlich problematisch für die Object Detection ist. Dies könnte an der Homogenität des Objektes liegen.

Diese Vermutung wird anhand eines Objektes mit markanterer Struktur überprüft. Die Wahl fällt auf eine Holzhaarbürste. Wird dieses Objekt - mit den Borsten nach oben zeigend - fotografiert, weist es ein markantes Muster auf. Aufgrund von ihres komplexeren Aussehens ist davon ausgegangen, dass die Bürste leichter von anderen Gegenständen zu unterscheiden ist, als dies bei der Dose der Fall war. Es werden 16 Trainingsbilder für die Erkennung der Bürste aufgenommen. Wie beschrieben, wird darauf geachtet, dass die Borsten nach oben zeigen. Siehe Anhang 8.2.

Die Evaluierung der Haarbürsten-Objekterkennung lautet: 100% Precision, 100% Recall, 100% Average Precision. Das bedeutet: In der Trainingsphase werden alle Haarbürsten fehlerfrei erkannt.

Bei Verwendung der Iteration 4 außerhalb der Trainingsphase zeigt sich, dass die Haarbürste tatsächlich leichter von anderen Objekten unterschieden werden kann, als dies bei der Dose der Fall war.

Es lässt sich beobachten, dass Objekte, welche mit einer Probability von 80 Prozent als Haarbürste markiert werden, tatsächlich Haarbürsten sind. Durch das Setzen dieses Schwellenwertes können korrekte Objekterkennungen von false positives unterschieden werden. Siehe Abbildung 21.



Abbildung 20: 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 relativ wenig Platz einnimmt, wird sie mit einer geringeren Probability erkannt. Dadurch ist es nicht mehr möglich, die Bürste mithilfe eines Schwellenwertes von anderen Objekten zu unterscheiden.



Abbildung 21: 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. In dem Training wird die Objekterkennung der Nivea-Dose entfernt. Diese Iteration ist nur noch darauf trainiert, das Objekt Haarbürste zu erkennen. Die Trainingsbilder für das Objekt Haarbürste bleiben unverändert. Die Evaluierung des Modells lautet: 100% Precision, 100% Recall, 100% mean Average Precicion.

Die Genauigkeit der Haarbürsten-Erkennung bleibt unverändert.

Iteration 6 In der Iteration 6 wird die Erkennung der Haarbürste verbessert. Ziel ist es, auch kleine Abbildungen der Haarbürste fehlerfrei zu erkennen. Den Empfehlungen von Azure folgend, wird die Menge an Trainingsbildern auf 51 erhöht. Weiterhin zeigt die Haarbürste in allen Bildern mit den Borsten nach oben. Auf den neuen Trainingsbildern ist die Haarbürste teilweise recht klein zu sehen. Siehe Anhang 8.2. Der Iteration 6 wird eine Stunde Zeit gelassen, um das Training durchzuführen. Der Trainingsprozess der vorherigen Iterationen 5 und 4 dauerte lediglich ca. 10 Minuten.

Die Evaluierung von Iteration 6 in der Testphase lautet: 100% Precision, 100% Recall, 100% mean Average Precision. Das auch diesmal die Testphase fehlerfrei durchlaufen wird.

Bei der Verwendung der Iteration außerhalb der Testphase können diesmal auch Haarbürsten erkannt werden, die in einem Foto klein abgebildet sind. Siehe Abbildung 22.

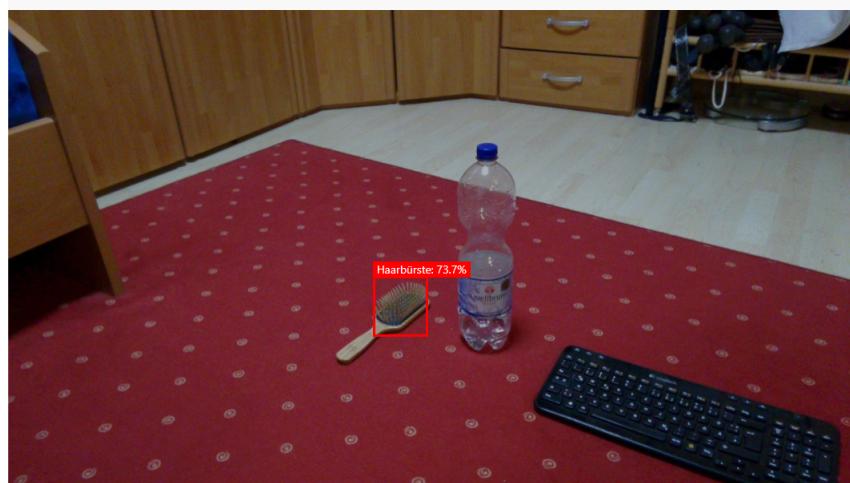


Abbildung 22: Iteration 6 Analysebeispiel. Die Haarbürste wurde mit einer Probability von 73,7 Prozent erkannt.

Somit ist das Ziel der Iteration 6 erfüllt. In den Bildern, die von Iteration 6 analysiert werden, sind alle Objekte die mit einer Probability von über 50 Prozent als Haarbürste markiert werden, tatsächlich Haarbürsten. Korrekte Objekterkennungen lassen sich gut von false positives unterscheiden.

4.8 Entwicklung der Foto-Repräsentation

In diesem Abschnitt wird die Entwicklung der Cast Methode beschrieben.

Ziel dieser Methode ist das Setzen einer Markierung in der 3D-Szene, basierend auf gegebenen Foto-Koordinaten. Das Foto beinhaltet keine Information über die Entfernung eines Objektes zur Kamera. Um die Entfernung zu ermitteln, ist ein Raycast erforderlich. Mit einer Repräsentation des Fotos in der 3D-Szene ist es möglich, diesen Raycast durchzuführen. Dazu muss das Foto nicht tatsächlich in dem 3D-Raum vorhanden sein. Es wird lediglich mit Koordinaten gearbeitet. Es muss die Möglichkeit existieren, die Foto-Position eines Objektes in eine Position der 3D-Szene umzuwandeln. Letztere muss das Verhältnis der Foto-Position zu der Umgebung widerspiegeln. Wenn diese Funktionalität erfüllt ist, kann die Position der 3D-Szene dafür verwendet werden, einen Raycast auf die Umgebung durchzuführen und somit die korrekte Position für den Gegenstand zu finden.

Die Position des Fotos hängt mit der Unity-Kamera zusammen. Daher kann das Foto durch den Camera Space simuliert werden. Als Erstes wird probiert ein Sphären-Objekt an eine gezielte Koordinate des Camera Spaces zu setzen. Unter den Voraussetzungen, dass die Kamera sowohl am Ursprung des globalen Koordinatensystems liegt als auch eine neutrale Rotation aufweist, stimmt der Camera Space mit dem globalen Koordinatensystem überein. Die Sphäre wird in der Szene per Hand bewegt, um markante Koordinaten des Camera Spaces abzulesen. Siehe Abbildung 23.

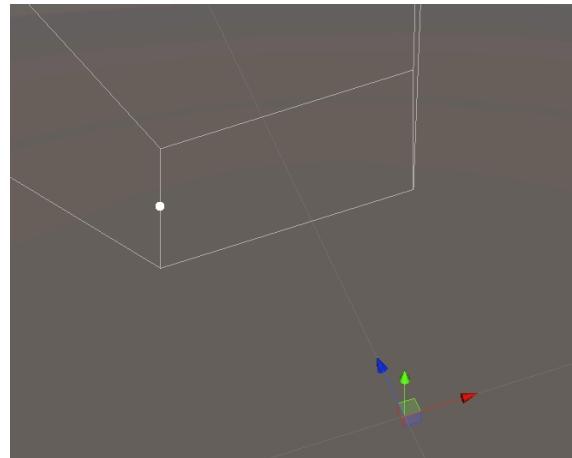


Abbildung 23: Die weiße Sphäre liegt auf dem linken Rand der Clipping Plane.

Dabei werden 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. Daher werden lineare Funktionen aufgestellt, um u,v auf x,y abzubilden. Diese Abbildungen dienen als Repräsentation des Fotos im 3D-Raum. Sie berücksichtigen sowohl die Position als auch die Skalierung des Fotos im Verhältnis zu der Unity-Kamera.

Diese Version der Foto-Repräsentation wird getestet. Geprüft wird, wie gut die Lokalisierung von *DetectedObjects* in der AR-Umgebung gelingt. Es zeigt sich, dass die entstehenden Markierungen zwar in dem View Frustum liegen, jedoch nicht die korrekten Positionen der *DetectedObjects* angeben.

Zur Problemanalyse wird ein UI-Objekt erstellt, welches während der Laufzeit die aufgenommenen Fotos anzeigt. Mithilfe des UI-Objektes werden die Fotos mit dem Sichtfeld des Displays verglichen. Dabei ist zu beobachten, dass Foto und Display ein unterschiedliches Seitenverhältnis haben. Darüber hinaus zeigt das Display einen kleinen Bildausschnitt an.

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. Letztere Möglichkeit würde dazu führen, dass auch Objekte erkannt werden, welche außerhalb des Sichtfeldes liegen. Dies würde zu einem inkonsistenten Feedback für den Nutzer führen, da manche Labels außerhalb seines Sichtfeldes entstünden. Um diesen Nachteil zu umgehen, wird die Entscheidung getroffen, das Foto auf das Display zuzuschneiden.

Das Zuschneiden wird realisiert, indem die Intervalle für u und v der Abbildungsfunktionen stärker eingegrenzt werden. Ignoriert werden alle Objekte, die außerhalb der Intervalle liegen. Um die neuen Intervallgrenzen zu bestimmen, wird dem Fotoanzeige-UI-Element ein Gitter-Element hinzugefügt. Mit dem Gitter kann die u,v Position von beliebigen Stellen des Fotos abgelesen werden. Durch Fotoaufnahmen und Vergleiche mit dem Sichtfeld des Displays wird abgelesen, bei welcher u,v Position des Fotos die Ecken des Displays zu finden sind. Die Intervalle werden dementsprechend eingegrenzt.

Mit den durchgeföhrten Veränderungen der Intervallgrenzen können *DetectedObjects* zwar prinzipiell korrekt in der Umgebung lokalisiert werden. Als problematisch erweisen sich allerdings solche Objekte, die nur teilweise im Sichtfeld des Nutzers liegen. Der Mittelpunkt dieser Objekte liegt nämlich knapp außerhalb der Intervalle. Daher werden die Objekte ignoriert, obwohl der Nutzer von einer Markierung ausgeht.

Die Zugschneidung des Fotos föhrt somit aus der Sicht des Nutzers zu einem inkonsistenten Verhalten, bezöglich der Objekterkennung. Darüber hinaus dauert es bei dem Verfahren mit den zugeschnittenen Fotos insgesamt länger einen Raum mit Labels zu versehen, da bei jeder Bildanalyse Daten verworfen werden.

Die Zugschneidung der Fotos bringt daher zu viele Nachteile mit sich. Daher wird darauf verzichtet. Stattdessen werden die unbeschnittenen Fotos verwendet.

Um den Zuschnitt rückgängig zu machen, werden die Intervalle für u und v wieder auf die ursprünglichen Werte – [0,1920] und [0,1080] – gesetzt. Folgerichtig müssen auch die Intervalle für x und y vergrößert werden. Um die x und y-Intervallgrenzen zu bestimmen, wird das Fotoanzeige-UI-Element parallel zu der Clipping Plane positioniert. Das Element folgt den Bewegungen der Kamera und ist möglichst nah an der Near Clipping Plane platziert. Siehe Abbildungen 24 und 25.

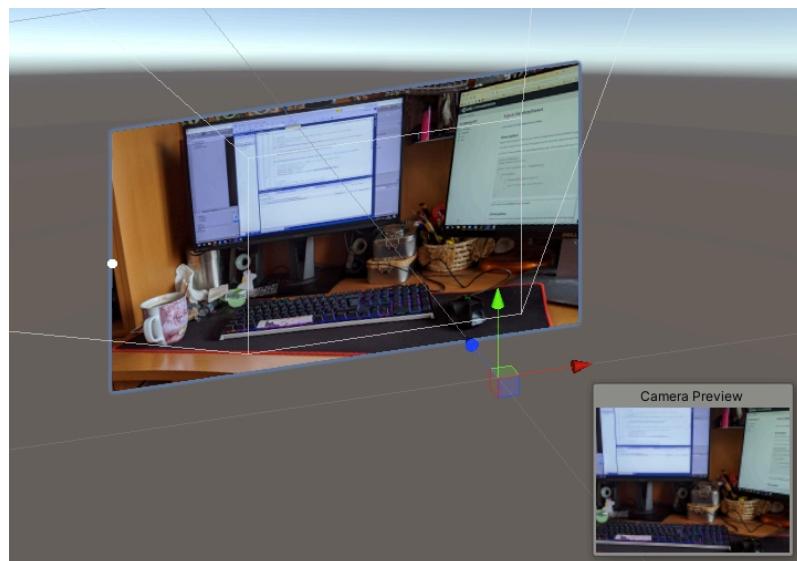


Abbildung 24: Bei der Anzeige füllt das aufgenommene Foto das gesamte Display aus. Die weiße Sphäre befindet sich auf dem linken Rand des Foto-Anzeige-Elementes.

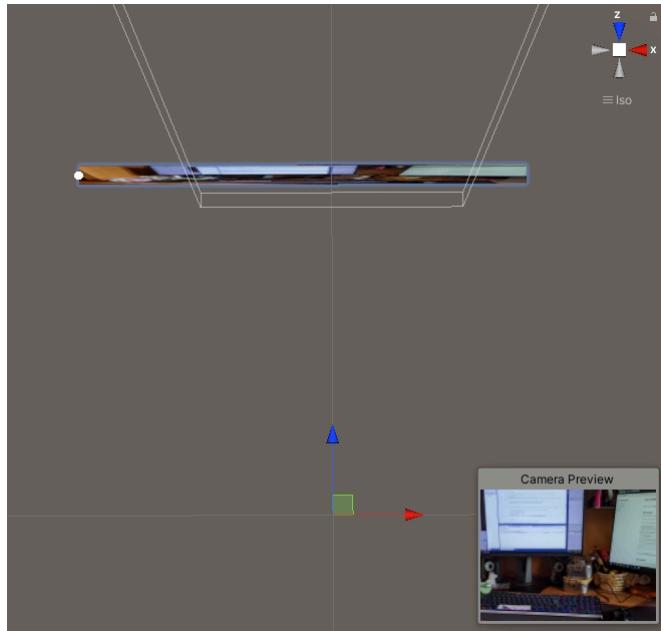


Abbildung 25: Die weiße Sphäre befindet sich nicht mehr in dem View Frustum und das Foto-Anzeige-Element befindet sich ein wenig hinter der Near Clipping Plane.

Das Display der Magic Leap Brille zeigt sogar solide Objekte leicht transparent an. Diese Eigenart des AR-Displays wird genutzt, um aufgenommene Fotos mit der realen Welt zu vergleichen. Durch Ausprobieren wird das UI-Element so skaliert und verschoben, dass das angezeigte Foto mit der realen Welt so weit wie möglich übereinstimmt. Das bedeutet, dass der Nutzer möglichst wenig Unterschiede zwischen dem angezeigten Foto und der durchschimmernden realen Welt sieht.

Die Ränder des skalierten UI-Elementes werden 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 Near Clipping Plane entfernt sein, um angezeigt zu werden.

Mit diesen Intervallen können *DetectedObjects* gut lokalisiert werden. Zudem werden keine Objekte weggelassen, von denen Markierung der Nutzer ausgeht.

5 Auswertung

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

5.1 Laufzeitanalyse

5.1.1 Netzwerk

Die genutzte Netzwerkverbindung hat eine Download-Geschwindigkeit von 180 Mbps und ein Upload-Geschwindigkeit von 18 Mbps. Mit einer Bildauflösung von 1090x1820 Pixeln, haben die Foto ein Größe von 5 Mb.

Getestet wird der Netzwerk-Delay zwischen den REST-APIs der Objekterkennungsservices und der AR-Brille. Zu diesem Zweck wird eine HTTP-Anfrage mit einem inkorrekt Authentifizierungsschlüssel an die Services geschickt. In dem Body der Anfrage wird ein Bild mitgeschickt. Es ergibt sich Folgendes: Die Analyse wird abgelehnt, da die Authentifizierung fehlschlägt. Daraufhin wird umgehend eine Response Nachricht zurückgeschickt. Die Länge des Round-Trip-Times wird mit dem Netzwerk-Delay gleichgesetzt, da keine Analyse stattfindet. Die durchschnittliche Round-Trip-Time beträgt 0,18 Sekunden (minimal 0,13 Sekunden und maximal 0,28 Sekunden). Dieser Durchschnitt ergibt sich aus der Auswertung von 14 Tests.

5.1.2 Objekterkennung

Die Laufzeit der automatischen Objekterkennung für AR wird aufgezeichnet. Die Erkennung beginnt mit der Fotoaufnahme und endet mit der Erzeugung der Labels. Siehe Abbildung 26

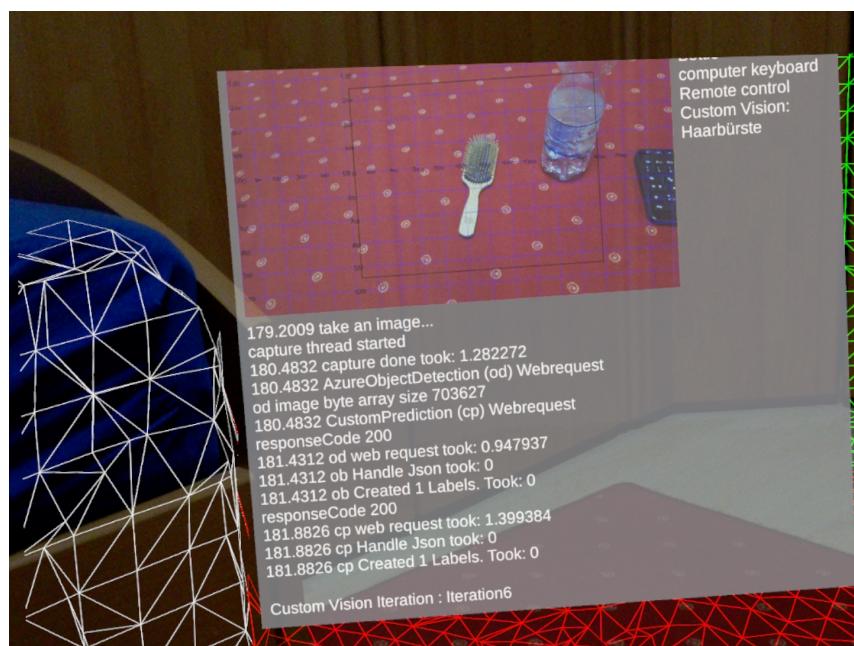


Abbildung 26: Durchlauf mit Laufzeit Aufzeichnung

Bei der Auswertung von 13 aufgenommenen Fotos wird festgestellt, dass Fotoaufnahmen durchschnittlich 1,11 Sekunden dauern. Inklusive Netzwerk Delay dauern Anfragen an Azure Object Detection durchschnittlich 0,96 Sekunden. Abzüglich des durchschnittlichen

Netzwerk-Delays von 0,18 Sekunden ergibt sich eine durchschnittliche Analysezeit von 0,78 Sekunden.

Inklusive Netzwerk Delay dauern bei Iteration 6 Anfragen an Azure Custom Vision durchschnittlich 1,84 Sekunden. Abzüglich des durchschnittlichen Netzwerk-Delays von 0,18 Sekunden ergibt sich somit eine durchschnittliche Analysezeit von 1,66 Sekunden.

Objekterkennungen mithilfe von Azure Object Detection und Azure Custom Vision werden parallel zueinander in unterschiedlichen Threads ausgeführt. Daher addieren sich ihre Laufzeiten nicht. Dementsprechend liegt die durchschnittliche Gesamtaufzeit einer automatischen AR-Objekterkennung 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 27 und 28.

	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 27: Laufzeitanalyse über 13 Bild-Analysen.

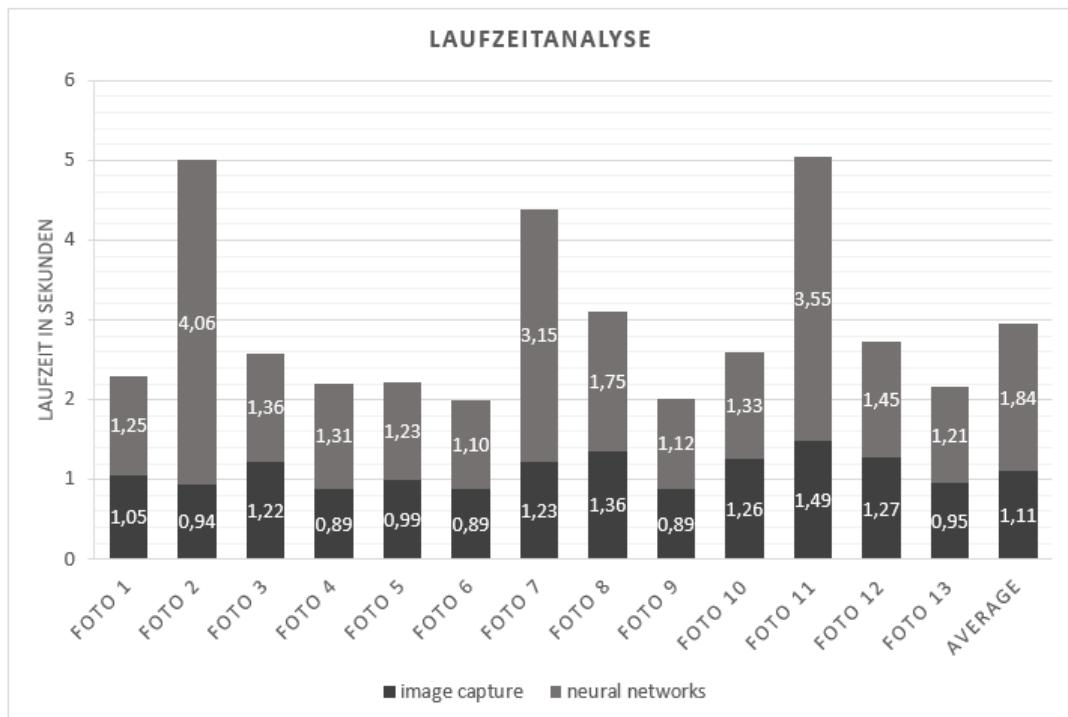


Abbildung 28: Diagramm Laufzeitanalyse

Da die Fotoaufnahme und die Anfragen an die Objekterkennungsservices zu lange dauern, ist es nicht möglich, die automatische Objekterkennung in Echtzeit durchzuführen. Die Verwendung eines einzigen neuronalen Netzwerkes würde die Laufzeit dieses Arbeitsschrittes verringern. Allerdings würde damit die Möglichkeit verloren gehen, mehrere neuronale Netze zu verwenden, um unterschiedliche Computer Vision Aufgaben erledi-

gen zu können. Aus einem RGB-Bild können beispielsweise mehr semantische Informationen extrahiert werden, wenn Image Segmentation und Object Detection kombiniert werden. Daher kommt die Verwendung eines einzigen neuronalen Netzwerkes nicht in Frage. Stattdessen wird dem Problem, dass die AR-Objekterkennung nicht echtzeitfähig ist, folgendermaßen begegnet: Die Resultate der Objekterkennungen werden in der Szene abgespeichert. Auf diese Weise können die semantischen Informationen in Echtzeit abgerufen werden, sobald sie benötigt sind.

Neben den Objekterkennung-Prozessen braucht das Aufnehmen der Fotos mit durchschnittlich 0.9 Sekunden einen signifikanten Teil der Gesamtaufzeit. Durch einen Umstieg von Fotos auf Frames eines Videostreams ist es prinzipiell möglich, die Aufnahmezeit zu reduzieren. Dies kann in einer weiterführenden Arbeit umgesetzt werden.

5.2 Evaluierung der Objekterkennung durch Azure Objekt Detection

Die Anwendung wird in einem Schlaf- und Arbeitszimmer getestet. Durch Azure Object Detection werden bei dem Test folgende Object-Klassen erkannt: Television, Person, Bottle, Keyboard, Computermouse, Cat, Bed, Luggage, Chair und Laptop.

Es stellt sich folgendes heraus: Azure Object Detection erkennt die Objekte mit unterschiedlicher Verlässlichkeit. Tastaturen und Personen werden in den meisten Aufnahmen korrekt erkannt. Das Lightwear-Gerät der 'Magic Leap One' wird gelegentlich als Computermaus interpretiert. Computerbildschirme werden durchweg als Television oder als Laptop markiert.

5.3 Evaluierung der Objekterkennung durch Azure Custom Vision

Die Genauigkeit der Objekterkennungen durch Azure Custom Vision hängt von dem Training des Modells ab. Azure Custom Vision erweist sich als effektive Ergänzung zur Azure Object Detection. Mithilfe der Iteration 6, wird das Objekt 'Haarbürste' zuverlässig erkannt. Dies ist auf das markante Aussehen des Objektes, die lange Testzeit von einer Stunde und die Verwendung von 51 Bildern zurückzuführen.

Für Bild-Regionen, in denen sich keine Haarbürste befindet, berechnet das Modell lediglich eine Probability von weniger als 5 Prozent dafür, dass sich dort eine Haarbürste befinden könnte. Regionen, die mit einer Probability von über 50 Prozent markiert werden, beinhalteten in der Anwendung tatsächlich zuverlässig eine Haarbürste.

Durch die niedrige Rate an false positives kann die Akzeptanzschwelle auf 60 Prozent gesetzt werden. Auf diese Weise wird die Rate der false negatives verringert. Dadurch wird die Haarbürste fast immer korrekt erkannt, wenn sie sich auf einem Bild befindet.

5.4 Objekte in 3D-Szene lokalisieren

In diesem Teil wird evaluiert, wie akkurat Foto-Objekte in der 3D-Umgebung lokalisiert und markiert werden. Um die Lokalisierung zu evaluieren, werden RGB-Bilder zwischen gespeichert, welche zur Objekterkennung genutzt werden. Auf den RGB-Bildern wird der Mittelpunkt der Bounding-Boxen markiert. Siehe Abbildung 29. Die Labels, welche in der Szene gesetzt werden, sollen mit diesen Positionen korrespondieren. Es lässt sich beobachten, dass die Positionen der Labels um weniger als 2 cm von der idealen Position abweichen. Siehe Abbildung 30.

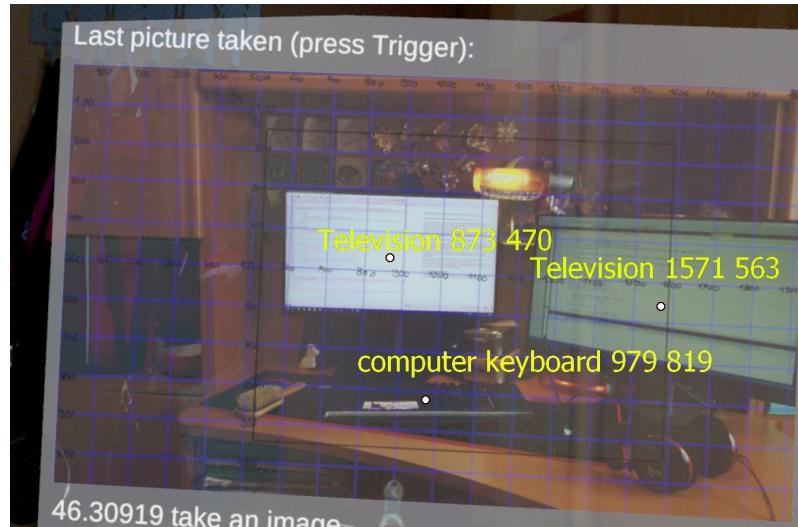


Abbildung 29: Die Mittelpunkte der erkannten Objekte werden, sind auf dem Foto per Hand markiert. Die Klassen und die u,v Foto-Koordinaten der Objekte sind angegeben.

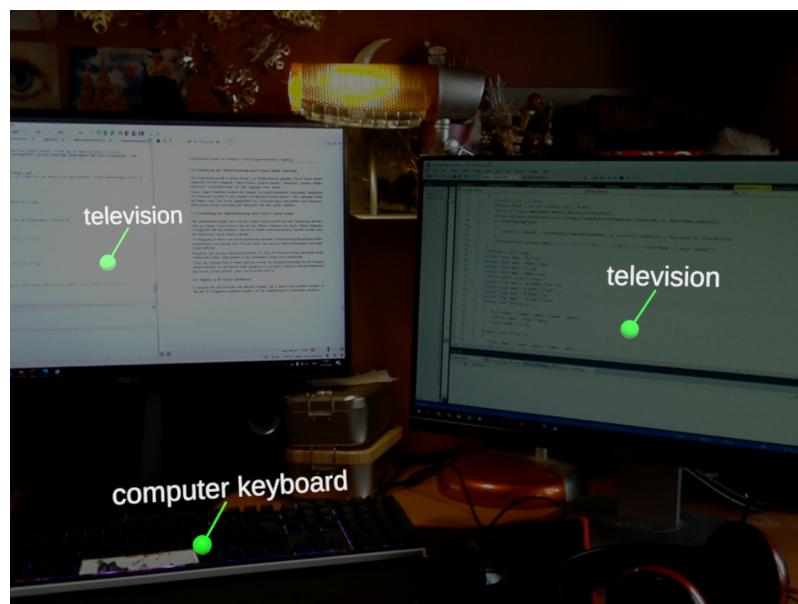


Abbildung 30: In der 3D-Szene markierte Objekte.

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

Wenn ein Objekt auf einem Foto erkannt wird und die Spatial Map an der Position des Gegenstandes fehlerhaft oder veraltet ist, wird das Objekt nicht korrekt markiert. Im vorliegenden Beispiel wird ein Stuhl auf einem Foto erkannt. Da der Stuhl kürzlich bewegt wurde, ist er jedoch nur bruchstückhaft in der Spatial Map vorhanden. Dies hat zur Folge, dass der Raycast bei Lokalisierung des Objektes durch den tatsächlichen Stuhl durchschießt. Der Raycast trifft stattdessen den Boden hinter dem Objekt und markiert diesen. Somit wird der Stuhl zwar korrekt durch die Objekterkennung erfasst, jedoch nicht korrekt in der Szene markiert. Siehe Abbildung 31.

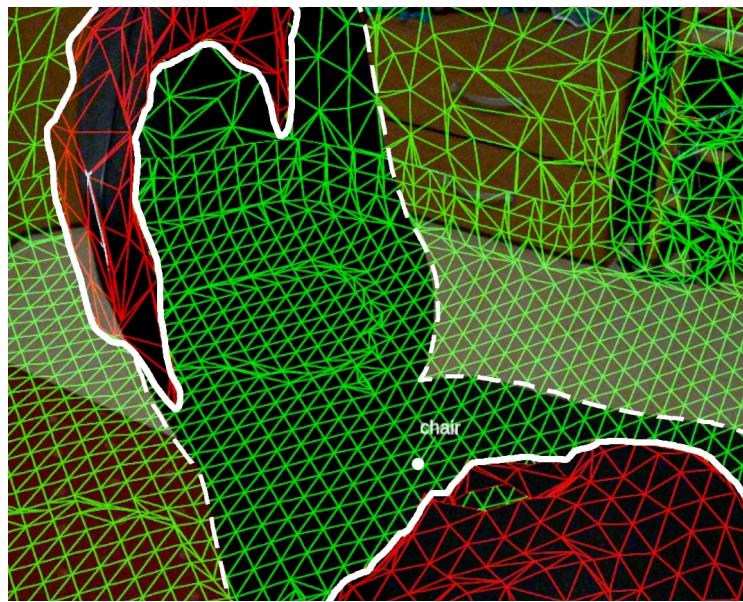


Abbildung 31: Spatial Map des Stuhles mit breiter Linie umrandet. Tatsächlicher Stuhl mit durchgebrochener Linie umrandet. Label des Stuhls markiert. Das Label befindet sich in der Szene hinter dem Stuhl auf dem Boden.

Halb transparente Objekte werden ebenfalls nicht korrekt in die Spatial Map eingefügt. In folgendem Beispiel wird das Label eines Flaschen-Objekts nicht korrekt positioniert:

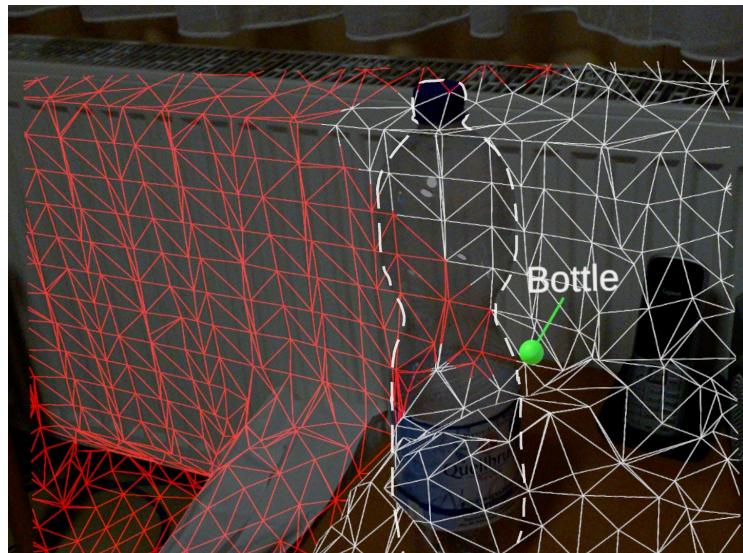


Abbildung 32: Das Label der Flasche liegt hinter dem tatsächlichen Objekt, da die Flasche nicht korrekt gemapped wurde.

6 Zusammenfassung

In diesem Kapitel werden die bearbeiteten Themen kurz zusammengefasst.

6.1 Konzepte und implementierte Funktionen

In dieser Arbeit eine automatische Objekterkennung und Markierung für AR-Umgebungen mithilfe von Image Based Object Detection umgesetzt.

Die Umgebung der AR-Brille wird mit RGB-Fotos aufgenommen. Diese Fotos werden mit Machine Learning Modellen nach Objekten durchsucht. Die Positionen der gefundene Objekte werden in der AR-Umgebung lokalisiert und mit einem Label markiert. Objekte werden durch nur maximal ein Label annotiert.

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.

6.2 Vergleich mit Related Works

Objekterkennung und Labeling in Echtzeit

Anders als das Framework von Huynh et al. (2019) läuft die vorgestellte Objekterkennung nicht in Echtzeit. Huynh et al. (2019) verwendet eine niedrige Kameraauflösung und Bildqualität, damit die Bildanalyse in 30 ms durchgeführt werden kann.

Unser Ansatz zielt nicht auf Echtzeit ab, sondern auf ein breiteres semantisches Verständnis. Daher werden, anders als bei Huynh et al. (2019), mehrere neuronale Netzwerke verwendet. In unserer Applikation wird Azure Objekt Detection für den Großteil der Objekterkennung verwendet. Dieses Netz wird durch Azure Custom Prediction unterstützt. Durch die Kombination können insgesamt mehr Objekte erkannt werden. Der Nachteil in diesem Vorgenommen besteht darin, dass jedes der neuronalen Netze Zeit braucht um eine Analyse durchzuführen, unabhängig davon wie viele Objekte sie finden. Die Objekterkennung dauert damit insgesamt länger.

In unserem Verfahren wird die höchste Kameraauflösung für die RGB-Bilder verwendet. Die Bilder beinhalten mehr Informationen, um den neuronalen Netzwerken die beste Möglichkeit zu geben Objekte zu erkennen. Die größeren Bilddateien brauchen mehr Zeit um sie an den Server zu verschicken und analysiert zu werden.(Huynh et al. 2019)

Image Segmentation für kontextbezogene Interaktionen

Das Framework von Chen et al. (2018) nutzt Image Segmentation, um jedem Voxel der Umgebung eine semantische Bedeutung zuzuweisen. Die Segmentierung findet, so wie Objekterkennung, semantische Informationen einer AR Umgebung. Die Informationen unterscheiden sich jedoch darin, auf welcher Ebene sie angesetzt sind. Die Informationen, die Chen et al. (2018) erheben, sind low-level im Vergleich mit den Informationen durch Objekterkennung. In dem Beispiel der Shooter-Anwendung wird den Voxeln jeweils ein Material zugewiesen. Diese semantische Information ist vergleichsweise low-level und lässt sich sehr gut mit dem Szeneverständnis durch Objekterkennung kombinieren. So kann beispielsweise für jedes Objekt angegeben werden aus welchem Material es besteht.(Chen et al. 2018)

View Management

Das View Management in unserer Applikation ist sehr minimal. Die Labels sind als 3D Objekte in der Szene umgesetzt. Sie rotieren, damit ihre Schriftzug immer dem Nutzer zugewandt ist. Die Position der Schriftzüge, relativ zu dem Objekt das sie annotieren, wird nicht verändert. Es wird nicht sichergestellt, dass die Labels keine relevanten virtuellen oder realen Informationen verdecken.

Die Labels werden nur mit einer neuen Position aktualisiert, wenn ihr Anker versetzt wird.

6.3 Ausblick

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

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 and Karaman 2017)

Zusätzlich kann die vorgestellte Applikation durch das Einsetzen von View Management verbessert werden. Dadurch kann sichergestellt werden, dass die gesetzten Labels immer lesbar sind. Durch die Objekterkennung werden Bereiche der realen Welt ausgewiesen, die relevant sind, da sie semantische Informationen enthalten. Mithilfe von View Management Methoden kann sichergestellt werden, dass diese relevanten Bereich nicht verdeckt werden.

Je nachdem welche Anwendung mit der Objekterkennung unterstützt werden soll, ist es wichtig, dass die Informationen für einen Menschen lesbar sind. Die Applikationen von Huynh et al. (2019) beruht beispielsweise darauf, Objekte gut sichtbar mit Labels zu versehen, um das Lernen einer Fremdsprache zu unterstützen.

Die Objekte, die von der Applikation erkannt werden, hängen von den verwendeten Machine Learning Modellen ab. In Zukunft könnte man weitere Modelle verwenden, um die Fotos der Umgebung zu analysieren. Durch Image Segmentation und Image Klassifikation können die semantischen Informationen einer Szene erweitert werden. Beispielsweise können Kontextinformationen verwendet werden, aus denen hervorgeht, in welchem Raumes eines Gebäudes die AR Brille eingesetzt wird (Küche, Arbeitszimmer oder Schlafzimmer). Und durch Image Segmentation kann beispielsweise festgestellt werden aus welchen Materialien die Objekte der Umgebung bestehen. Insgesamt kann das semantische Verständnis der Szene erweitert werden durch das einsetzen von mehreren unterschiedlichen Bildanalysen.

Durch das automatische erkennen von Objekten in einer Umgebung, wird es möglich AR Anwendungen zu realisieren, die darauf beruhen die Objekte in dynamischen und großen Umgebung zu kennen. Beispielsweise kann eine Blindenführung entwickelt werden. Das semantische Verständnis der Umgebung kann genutzt werden, um einen Menschen durch einen Raum zu führen. Sowohl geometrische als auch semantische Informationen werden dabei verwendet, um eine Umgebung zu beschreiben und Anweisungen des Nutzers zu interpretieren.

Eine andere Anwendungsmöglichkeit wäre ein künstliches fotografisches Gedächtnis. Wenn semantische Informationen und insbesondere Objekte der Umgebung über einen längeren Zeitraum gespeichert werden, kann eine AR Anwendung wiedergeben, wann ein bestimmtes Objekt zuletzt gesehen wurde und in welchem Kontext es sich befunden hat.

Erweitert können periodisch Fotos der Umgebung aufgenommen werden und angezeigt werden, wenn der Nutzer nach Objekten fragt, die dort abgebildet sind.

7 Literaturverzeichnis

- L. Chen, W. Tang, W. N. John, R. T. Wan, and J. J. Zhang. Context-aware mixed reality: A framework for ubiquitous interaction. *arXiv: Computer Vision and Pattern Recognition*, 2018.
- R. Dörner, W. Broll, B. Jung, P. Grimm, and M. Göbel. *Einführung in Virtual und Augmented Reality*, pages 1–42. Springer Berlin Heidelberg, Berlin, Heidelberg, 2019. ISBN 978-3-662-58861-1. doi: 10.1007/978-3-662-58861-1_1. URL https://doi.org/10.1007/978-3-662-58861-1_1.
- P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1627–1645, 2010.
- R. Grasset, T. Langlotz, D. Kalkofen, M. Tatzgern, and D. Schmalstieg. Image-driven view management for augmented reality browsers. *ISMAR 2012 - 11th IEEE International Symposium on Mixed and Augmented Reality 2012, Science and Technology Papers*, pages 177–186, 11 2012. doi: 10.1109/ISMAR.2012.6402555.
- B. Huynh, J. Orlosky, and T. Höllerer. In-situ labeling for augmented reality language learning. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 1606–1611, 2019.
- L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu. A survey of deep learning-based object detection. *IEEE Access*, 7:128837–128868, 2019.
- N. Jmour, S. Zayen, and A. Abdelkrim. Convolutional neural networks for image classification. In *2018 International Conference on Advanced Systems and Electric Technologies (IC ASET)*, pages 397–402, 2018.
- M. Leap. App security, 2018. URL <https://developer.magicleap.com/en-us/learn/guides/application-security-overview>. (besucht am 18.09.2020).
- F. Ma and S. Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. *CoRR*, abs/1709.07492, 2017. URL <http://arxiv.org/abs/1709.07492>.
- MagicLeap. Jsonutility.fromjson, 2018a. URL <https://www.magicleap.care/hc/en-us>. (besucht am 1.10.2020).
- MagicLeap. magic leap 1, 2018b. URL <https://www.magicleap.com/en-us/magic-leap-1>. (besucht am 18.09.2020).
- MagicLeap. Lumin os overview, 2019a. URL <https://developer.magicleap.com/en-us/learn/guides/lumin-os-overview>. (besucht am 18.09.2020).
- MagicLeap. World rekonstruktion, 2019b. URL <https://developer.magicleap.com/en-us/learn/guides/world-reconstruction-overview-landing>. (besucht am 18.09.2020).
- MagicLeap. 1.1 unity setup, 2020a. URL <https://developer.magicleap.com/en-us/learn/guides/get-started-developing-in-unity>. (besucht am 12.10.2020).
- MagicLeap. Glossary and usage, 2020b. URL <https://developer.magicleap.com/en-us/learn/guides/glossary>. (besucht am 18.09.2020).

- MagicLeap. 2.3 control buttons - unity, 2020c. URL <https://developer.magicleap.com/en-us/learn/guides/control-buttons-unity>. (besucht am 18.09.2020).
- MagicLeap. Magic leap features, 2020d. URL <https://developer.magicleap.com/en-us/learn/guides/magic-leap-features>. (besucht am 18.09.2020).
- MagicLeap. 1.4 spatial meshing - unity, 2020e. URL <https://developer.magicleap.com/en-us/learn/guides/meshing-in-unity>. (besucht am 18.09.2020).
- MagicLeap. Privileges - unity, 2020f. URL <https://developer.magicleap.com/en-us/learn/guides/privileges-in-unity>. (besucht am 18.09.2020).
- MagicLeap. Raycast snippet - unity, 2020g. URL <https://developer.magicleap.com/en-us/learn/guides/raycast-snippet-unity>. (besucht am 12.10.2020).
- A. Maier, C. Syben, T. Lasser, and C. Riess. A gentle introduction to deep learning in medical image processing. *Zeitschrift für Medizinische Physik*, 29(2):86 – 101, 2019. ISSN 0939-3889. doi: <https://doi.org/10.1016/j.zemedi.2018.12.003>. URL <http://www.sciencedirect.com/science/article/pii/S093938891830120X>. Special Issue: Deep Learning in Medical Physics.
- Microsoft. Mr und azure 302b benutzerdefinierte vision, 2018. URL <https://docs.microsoft.com/de-de/windows/mixed-reality/mr-azure-302b>. (besucht am 17.09.2020).
- Microsoft. Microsoft azure computer vison, 2010. URL <https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/>. (besucht am 17.09.2020).
- Microsoft. Mr und azure 302 maschinelles sehen, 2018a. URL <https://docs.microsoft.com/de-de/windows/mixed-reality/mr-azure-302>. (besucht am 17.09.2020).
- Microsoft. Spatial mapping, 2018b. URL <https://docs.microsoft.com/de-de/windows/mixed-reality/spatial-mapping>. (besucht am 17. Septmber 2020).
- Microsoft. Erkennen von alltäglichen objekten in bildern, 2019a. URL <https://docs.microsoft.com/de-de/azure/cognitive-services/computer-vision/concept-object-detection>. (besucht am 24.09.2020).
- Microsoft. Detect common objects in images, 2019b. URL <https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-object-detection>. (besucht am 17.09.2020).
- Microsoft. What is computer vision, 2020. URL <https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home>. (besucht am 17.09.2020).
- K. O’Shea and R. Nash. An introduction to convolutional neural networks. *ArXiv*, abs/1511.08458, 2015.
- Unity. Gameobjects, 2017. URL <https://docs.unity3d.com/Manual/GameObjects.html>. (besucht am 12.10.2020).
- Unity. Prefabs, 2018. URL <https://docs.unity3d.com/Manual/Prefabs.html>. (besucht am 12.10.2020).
- Unity. Jsonutility.fromjson, 2020a. URL <https://docs.unity3d.com/ScriptReference/JsonUtility.FromJson.html>. (besucht am 24.09.2020).

Unity. Camera.cameratoworldmatrix, 2020b. URL <https://docs.unity3d.com/ScriptReference/Camera-cameraToWorldMatrix.html>. (besucht am 18.09.2020).

Unity. Understanding the view frustum, 2020c. URL <https://docs.unity3d.com/Manual/UnderstandingFrustum.html>. (besucht am 12.10.2020).

Unity. Matrix4x4.multiplypoint, 2020d. URL <https://docs.unity3d.com/ScriptReference/Matrix4x4.MultiplyPoint.html>. (besucht am 18.09.2020).

Unity. Physics.Raycast, 2020e. URL <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>. (besucht am 12.10.2020).

Z. Zhao, P. Zheng, S. Xu, and X. Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212–3232, 2019.

8 Anhang

8.1 Markierte Objekte aus unterschiedlichen Blickwinkeln

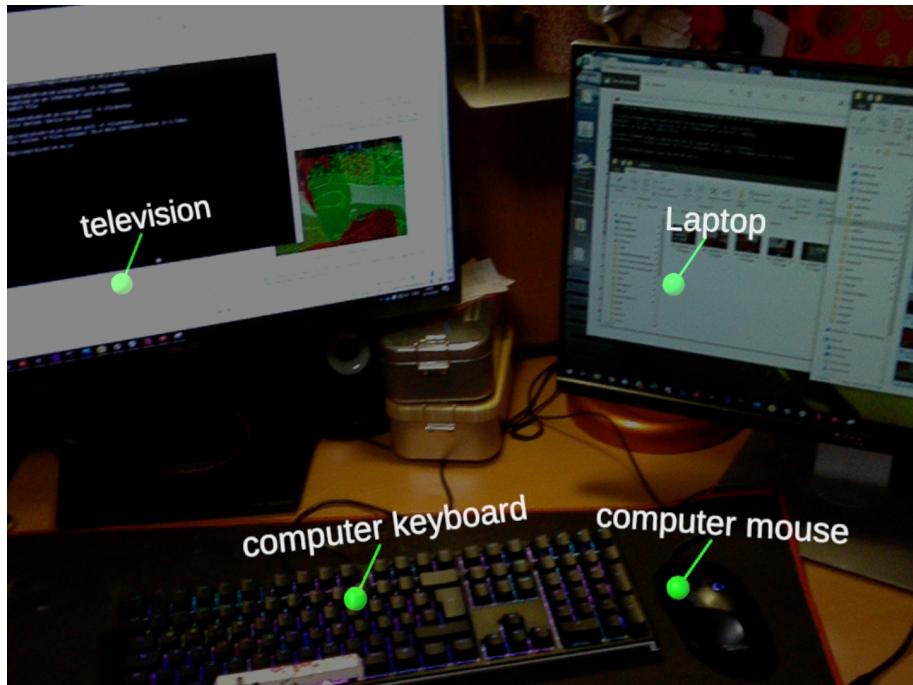


Abbildung 33: Objekte mit Label. Blickwinkel 1

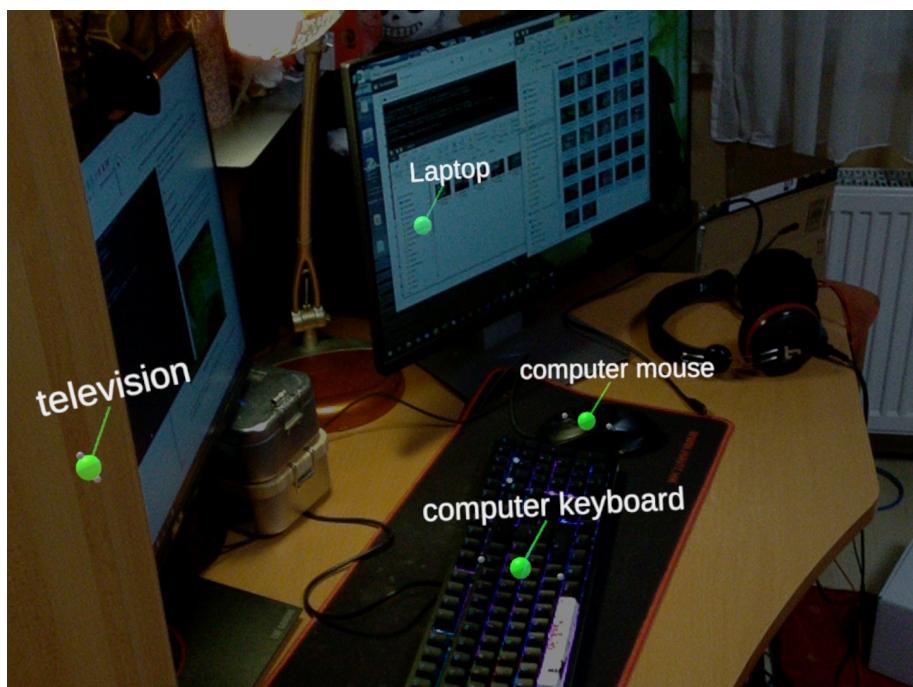


Abbildung 34: Blickwinkel 2

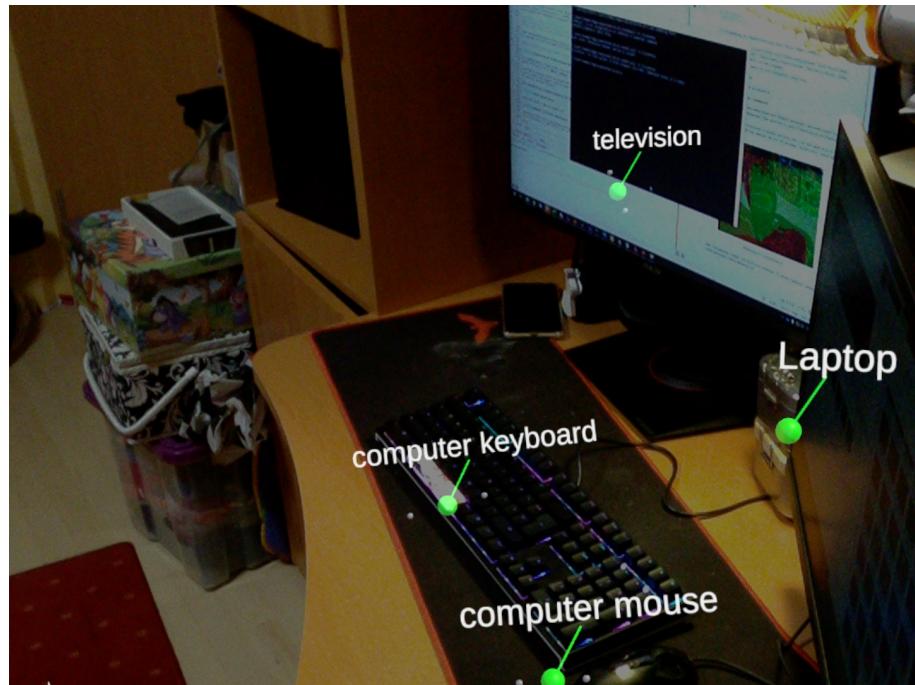


Abbildung 35: Blickwinkel 3

8.2 Ausschnitt Trainingsbilder der Custom Vision Iterationen 4 und 6



Abbildung 36: Ausschnitt der Trainigsbilder für Haarbürsten-Objekterkennung Iteration 4

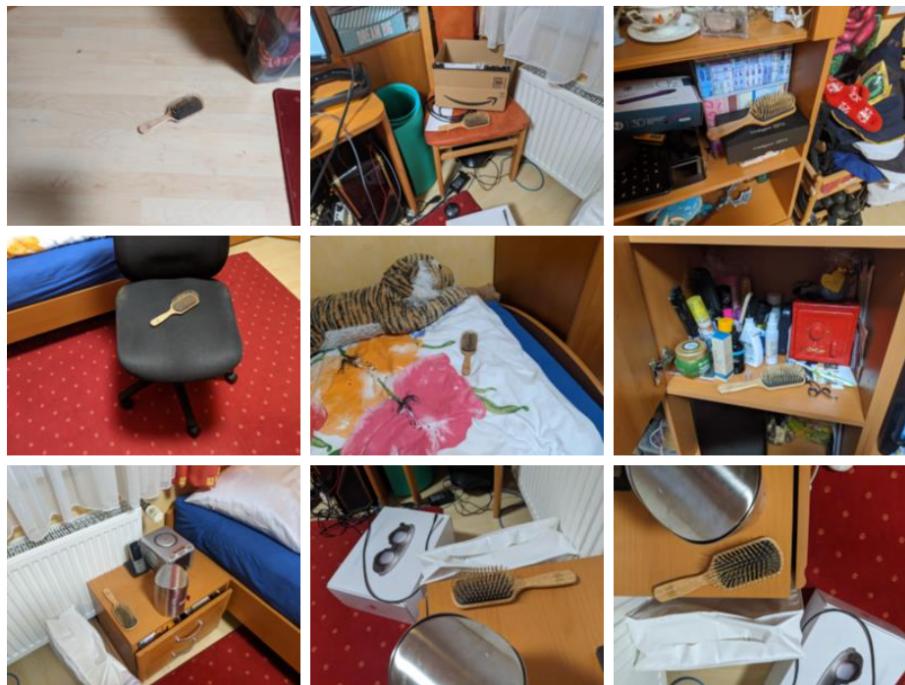


Abbildung 37: Ausschnitt der Trainingsbilder für Haarbürsten-Objekterkennung Iteration 4

8.3 Beispielbilder von Objekten mit Labels

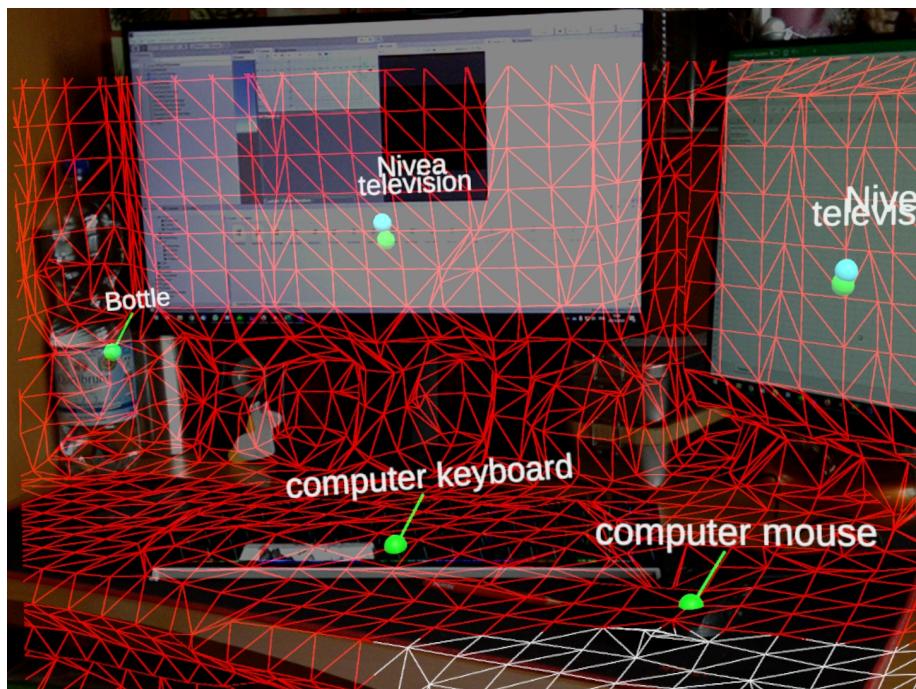


Abbildung 38: Beispielbild mit Iteration 4. Computerbildschirme werden als Nivea Dosen erkannt.

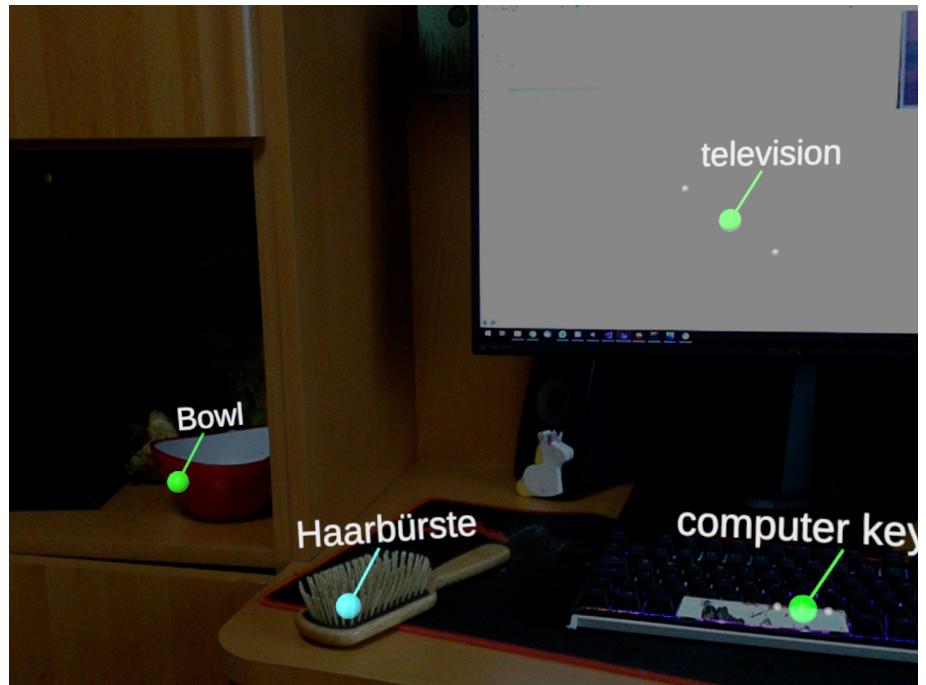


Abbildung 39: Beispielbild mit Iteration 6