Bachelorarbeit

Automatisches Labeln von Objekten in einer Augmented Reality Umgebung

Janelle Pfeifer
Delpstraße 28
53359 Rheinbach
janelle.pfeifer@smail.inf.h-brs.de

Hochschule Bonn-Rhein-Sieg
Institute of Visual Computing
Fachbereich Informatik
Studiengang: Informatik (B.SC.)

Erstprüfer: Prof. Dr. Ernst Kruijff
Zweitprüfer: Prof. Dr. André Hinkenjann

Rheinbach, 1.10.2020

Inhaltsverzeichnis

Zusammenfassung

todo

1 Einleitung

Augmented Reality (AR) ist eine Vermischung der realen Welt mit digitalen Elementen. Es wird durch Anzeigegeräte, wie Handys, Tablets oder Augmented Reality Brillen, präsentiert und bietet ein intuitives Benutzerinterface, um Informationen über Objekte der realen Welt anzuzeigen. Dafür muss erfasst werden, welche Gegenstände sich in der Umgebung befinden.

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

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

Beide der Verfahren lassen sich schlecht skalieren um große AR Umgebungen abzudecken. Nur eine vollautomatische Objekterkennung ist skalierbar.

Um diese Automatisierung zu erreichen, kann Image based Object Detection aus dem Bereich der Computer Vision verwendet werden. Dieses Verfahren ist darauf ausgelegt Objekte in Bildern zu erkennen.[1]

In dieser Thesis wird das Erkennen und Labeln von Objekten in einer AR Umgebung, mithilfe von Image based Objekt Detection, automatisiert. Das AR Gerät Magic Leap Lightwear wird als Benutzerinterface und Plattform verwendet.

2 Grundlagen

2.1 Object Detection

Objekt Detection ist eine Aufgabe der Computer Vision. Es sollen Objekte in einem Bild erkannt werden. Für die Objekte wird eine Klasse und eine Bounding Box bestimmt. Die Klasse gibt an, um welche Art von Objekt es sich handeln. Beispielsweise ob es eine Tastatur oder ein Computerbildschirm ist. Die Bounding Box gibt ein Viereck auf dem Bild an, in dem sich das Objekt befindet.

2.2 Artificial Neural Networks

Artificial Neural Networks sind Machine Learning Architekturen. Sie können beispielsweise Musik, Text oder Bilder nach Mustern durchsuchen. Sie sind für keine

genaue Aufgabe programmiert, sondern lernen indem sie mit Beispieldaten trainiert werden.

Für jedes Beispiel gibt es ein Label, das angibt ob es das gesuchte Muster enthält oder nicht. Die Struktur des Networks verfügt über Gewichte, die Einfluss auf den Output haben. Mit jedem Trainingsbeispiel passt das Network die Gewichte an, sodass der Output dem Label des Beispiels entspricht.[1, 2]

2.3 Convolutional Neural Networks

Convolutional Neural Networks sind auf das Verarbeiten von Bildern spezialisiert. Sie nutzen aus, das Bilder viele Redundanzen und informationsarme Bereiche haben. Daher können mit jedem Verarbeitungsschritt des Networks Informationen weggelassen werden. So können Rechenzeit und Volumen der Trainingsdaten verringert werden.[1, 2, 3]

2.4 REST Anfragen

todo REST, was ist eine API, POST, Repsonse, ResponceCode

2.5 Azure Computer Vision

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

Dabei sendet der Anwender ein Bild an Microsoft, dort wird es verarbeitet und ein Ergebnis zurückgeschickt.[4, 5, 6, 7]

Die Object Detection basiert auf einem trainierten KI Modell. Dieses kann nur Objekte erkennen, für die es trainiert wurde. Zusätzlich können Objekte, die in dem Foto sehr klein sind oder nah bei anderen Objekten liegen, nicht erkannt werden.[8]

Der Service ist durch eine REST-API erreichbar. Mit einer Post Anforderung werden die Bilddaten übertragen und die Analyse angefragt. Die Response Nachricht beinhaltet eine Json-Datei, welche die gefundene Objekte und deren Positionen auf dem Foto beinhaltet.

2.6 Azure Custom Vision

Azure bietet zusätzlich einen Computer Vision Service an, den der Nutzer trainieren kann. Custom Vision kann dann verwendet werden, wenn Azure Object Detection für Objekte nicht trainiert ist.[9]

dass es sich bei einem erkannten Objekt tatsächlich um eine Nivea Dose handelt

todo: was ist die Prediction? Erkälre die Iterations Erkläre was die Genauigkeit der Predation aussagt.

2.7 Spatial Mapping

Durch Spatial Mapping wird eine 3D Abbildung der realen Welt erschaffen. So können Hologramme mit der Umgebung interagieren, diese verdecken oder von ihr verdeckt werden.[10]

2.8 Magic Leap AR Brille

Die Magic Leap Lightwear ist eine Augmented Reality Brille. Sie besitzt neun Sensoren und mehrere Kameras. Dazu gehören:

- ein Infrarot Tiefen-Sensor,
- ein Eye Tracker,
- eine Foto und Video Kamera, die im Format 16:9 mit einer Auflösung von 1920 x 1080 Pixeln aufnehmen,
- Umgebungskameras die in unterschiedliche Richtungen ausgerichtet sind. [11, 12]

Der Output geschieht über ein Display mit einem 50 Grad Field of View und einem Seitenverhältnis von 4:3. Eingaben erfolgen über einen 6 Degree of Freedom Controller.[11, 12]

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 Betriebsamstem Spatial Mapping durch.[13, 14]

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.[13, 14, 15, 16]

Lumin OS bietet es Applikationen an,

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

Neben dem Spatial Mapping unterstützt Lumin OS das Verarbeiten vom Input des Controllers und verwaltet die Zugriffsrechte der Applikationen. Dazu gehört beispielsweise der Zugriff auf die Fotokamera und das Netzwerk.[14, 17]

3D-Computergrafik

2.9 Lokale und globale Koordinatensysteme in 3D Szenen

In einer 3D Szenen werden die Positionen von Objekten als Matrizen in dreidimensionalen Koordinatensystemen verwaltet. Es gibt ein globales Koordinatensystem (auch Weltkoordinatensystem oder World Space), in dem alle Objekte relativ zu einem gewählten 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.[18]

Dazu wird die Koordinate als Vektor angegeben und mit der cameraToWorld Matrix multipliziert. Das Resultat ist ein Vektor, der eine Koordinate im globalen Koordinatensystem angibt.[18, 19]

2.10 Kamera in 3D-Computergrafik

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

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

Die hintere Clipping Plane limitiert wie weit Objekte entfernt sein können, bevor sie nicht mehr zu sehen sind.

3 Design

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

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

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

Da die Foto-Kamera 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 Skalierungs-Unterschiede ausgeglichen. Für jedes Objekte wird so eine Koordinate auf der Clipping Plane bestimmt.

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

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

4 Implementierung

Das Projekt wurde in Unity umgesetzt und für die Magic Leap AR Brille entwickelt. Es wurde ein Unity Projekt Template von Magic Leap verwendet.

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

MLSpatialMapper setzt Spatial Mapping um und erzeugt ein Mesh, das die Umgebung abbildet.

4.1 Ein Foto aufnehmen

Das Script TakePicture implementiert das Aufnehmen von einem Foto. Die Methode StartCapture wird aufgerufen, um den Prozess zu starten. MLCamera wird genutzt, um die Kamera der Magic Leap Brille anzusteuern.

Wenn das Foto aufgenommen wurde, wird OnCaptureRawImageComplete aufgerufen. Die Daten des Bildes werden an die Klassen AzureObjectDetection und AzureCustomPrediction weitergegeben. Dort wird die Analyse der Bilder gestartet.

4.2 Object Detection

In der Methode AnalyseImage von AzureObjectDetection wird ein Post Web Request zusammengestellt, um die Azure REST API anzufragen. Der Request erhält einen Authentifizierung für die API und das zu analysierende Foto.

Der Webrequest wird verschickt und auf die Antwort gewartet. Wenn die Antwort eintrifft, wird anhand des ResponseCodes nachgeschaut, ob es bei dem Request einen Fehler gab. Beispielsweise kann die Internetverbindung gestört sein oder die Authentifizierung abgelehnt werden. Wenn es keinen Fehler gab, wurde eine Json-Datei bei der Antwort mitgeschickt. Darin wird für jedes gefundene Objekt auf dem Foto eine Bezeichnung (Klasse) und eine Bounding Box angegeben.

Die Json-Datei wird in HandleJsonResponse verarbeitet. Für den erwarteten Aufbau der Datei wurden drei Klassen geschrieben. Der Json String wird mit JsonUtility in ein DetectionResponse Object umgewandelt. Dabei werden alle gefundenen Foto-Objekte in einer Liste von DetectedObject abgelegt.[20]

Die gefundenen Objekte sollen im 3D Raum mit einer Markierung gekennzeichnet werden. Dafür wird für jedes DetectedObject die Methode Cast von der Klasse PixelToWorld aufgerufen. Der Methode wird der Mittelpunkt der BoundingBox als u,v Foto-Koordinate für das DetectedObject übergeben.

4.3 Von dem Foto zum 3D Raum

Das gefundene Foto-Objekt soll in der 3D Abbildung der realen Welt lokalisiert werden. Dafür nutzt die Methode Cast die u,v Foto-Koordinate des Objekts und die Kameraausrichtung.

Das Foto kann mit dem Display und somit mit der Clipping Plane der Hauptkamera approximiert werden. Die u,v Foto-Koordinate wird zunächst in eine x,y,z Koordinate in dem Camera Space umgewandelt. Der z Anteil gibt die Entfernung von dem

Ursprung der Kamera an in Blickrichtung an. Dabei befinden sich Punkte mit einer Entfernung von 0.4 Einheiten auf der Clipping Plane. In dem Camera Space mit z = -0.4 angegeben.

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

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

Sind diese x und y Werte bekannt, ergibt sich für die Achsen jeweils ein Intervall, die kombiniert alle Foto-Koordinaten auf die Clipping Plane abbilden können. Die Intervall lauten: [-0.295,0.2281] für x und [0.1546,-0.1507] für y. Mit den Intervallen wird die Position und Skalierung des Fotos in Relation zu dem Display - und der Hauptkamera - berücksichtigt. Siehe Kapitel 4.7 für die Entwickelung der Cast Methode und die Ermittlung der Intervallwerte.

Es werden zwei lineare Funktionen aufgestellt:

- Die Funktion X bildet das Intervall für u [0,1920] auf das Intervall für x [-0.295,0.2281] ab.
- Die Funktion Y bildet das Intervall für v [0,1080] auf das Intervall für y [0.1546,-0.1507] ab.

Siehe Abbildung 1.

```
40  //Picture u and v ranges
41  private float umin = 0;//left
42  private float umax = 1920;//right
43  private float vmin = 0;//up
44  private float vmax = 1080;//down
45  //Offset Vektor x and y ranges
46  private float xmin = -0.295F;//left
47  private float xmax = 0.2281F;//right
48  private float ymin = 0.1546F;//up
49  private float ymax = -0.1507F;//down
50  private float X(float u)
51  {
52      float slope = ((xmax - xmin) / (umax - umin));
53      float b = xmin - slope * umin;
54      return slope * u + b;
55  }
56  private float Y(float v)
57  {
58      float slope = ((ymax - ymin) / (vmax - vmin));
59      float b = ymin - slope * vmin;
60      return slope * v + b;
61  }
```

Abb. 1. Funktionen X und Y

Mit den Funktionen wird eine Koordinate im Camera Space für u,v berechnet. Diese Koordinate wird dann, mithilfe der cameraToWorldMatrix, in die Koordinate p des

globale Koordinatensystem umgewandelt. Damit wird die Position und Rotation der Kamera - und somit des Fotos - in der 3D Szene berücksichtigt. Siehe Abbildung 2.

```
28      public void Cast(float u, float v, SavedCameraState cpos,GameObject clippingPlaneMarker,
29          string objectName, bool showClippingPlane, int material)
30      {
31          //scale v,v to x,y range
32          Vector3 offset = new Vector3(X(u), Y(v), -0.4F);
33          Vector3 p = cpos.cameratoWorldMatrix.MultiplyPoint(offset);
34          Raycast.instance.StartCast(Raycast.instance.CreateRaycastParams(cpos.ctransform, p), objectName, material);
35          if (showClippingPlane)
36          {
37              GameObject sphere2 = Instantiate(clippingPlaneMarker, p, Quaternion.identity);// show point on clipping plane
38          }
39          ResultAsText.instance.Add(u + " " + v + " " + X(u) + " " + X(v) + " " + objectName + " object marked");
40      }
```

Abb. 2. Cast Methode

4.4 Raycast

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

Für den MLRaycast werden zwei Parameter benötigt:

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

Siehe Abbildung 3.

```
19      public MLRaycast.QueryParams CreateRaycastParams(Transform ctransform, Vector3 target)
20      {
21          MLRaycast.QueryParams _raycastParams = new MLRaycast.QueryParams
22          {
23              // Update the parameters with our Camera's transform
24              Position = ctransform.position,
25              Direction = target - ctransform.position,
26              UpVector = ctransform.up,
27              // Provide a size of our raycasting array (1x1)
28              Width = 1,
29              Height = 1
30          };
31          return _raycastParams;
32      }
```

Abb. 3. Cast Methode

Wenn der Raycast fertig ist, wird die Methode HandleOnRecieveRaycast aufgerufen. Der Parameter point beinhaltet dabei die getroffene Koordinate. Diese wird an die Methode CreateMarker von der Klasse LabelCreater weitergegeben.

4.5 LabelCreater

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

Das Prefab besteht aus einer Kugel und einem Schriftzug, der den Namen des DetectedObject anzeigen soll. Dem neu instanziierten GameObject wird die Bezeichnung des DetectedObject als Schriftzug zugewiesen. Siehe Abbildung 4.

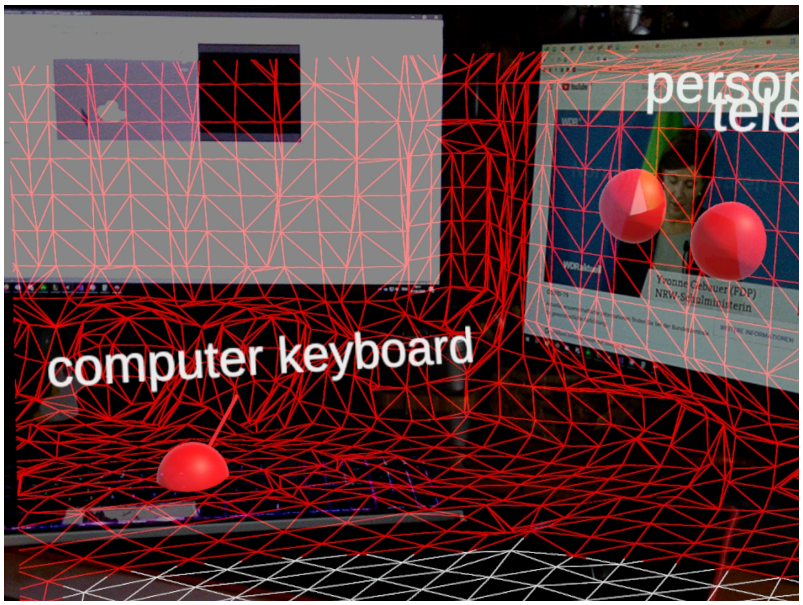

Abb. 4. Markierungen in der Welt

4.6 Azure Custom Vision

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

Die Anfrage an den Service passiert in der Klasse AzureCustomPrediction. Ähnlich wie bei AzureObjectDetection wird ein Webrequest erstellt mit einem Authorization Key für den Service und einem Foto als Payload.

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

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

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

Iteration 1:

Zunächst wurde probiert Tuben von Acrylfarbe zu erkennen. Die Genauigkeit davon war nicht sehr hoch. Es wurden in Fotos Acrylfarben an Stellen erkannt, an denen es keine gab. Siehe Abbildung 5.

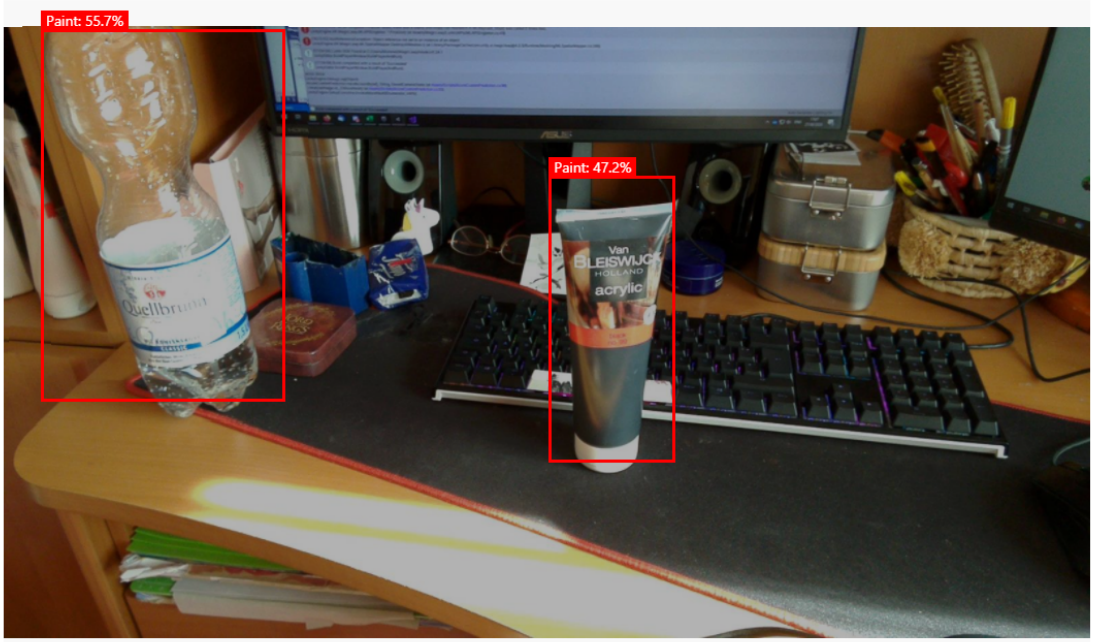

Abb. 5. Beispielfoto Iteration 1. Die Wasserflasche wurde als Farbe markiert mit 55.7 Prozent statistischer Konfidenz. Die Tatsächliche Farbtube wurde mit einer Konfidenz von 47.5 Prozent erkannt.

Iteration 2:

In der zweiten Iteration wurde probiert das Modell darauf zu trainieren, eine blaue Dose von Nivea Hautcreme to erkennen. Die Form und Farbe der Dose ist sehr simpel, daher wurde davon ausgegangen, dass sie leichter zu erkennen ist. Die berechnete Prediction Wahrscheinlichkeit während des Trainings lag bei 80 Prozent.

Trotzdem wurden in vielen Fotos fälschlicherweise Nivea Dosen erkannt.

Iteration 3:

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

Die Genauigkeit der Prediction sank auf 75 Prozent.

Iteration 4:

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

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

Die Genauigkeit für die Bürste lag bei 100 Prozent. In der Umsetzung mit der Magic Leap Anwendung wird die Bürste häufig nicht erkannt, obwohl sie im Bild ist und mit den Borsten nach oben liegt. Es werden jedoch keine Objekte fälschlicherweise als Haarbürste erkannt.

4.7 Entwicklung der Foto-Repräsentation

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

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

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

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

Wenn die Kamera am Ursprung des globalen Koordinatensystem liegt und eine neutrale Rotation hat, stimmt der Camera Space mit dem globalen Koordinatensystem überein. Die Sphäre wurde in der Szene per Hand bewegt um markante Koordinaten des Camera Space abzulesen. Dabei wurden folgende Camera Space Koordinaten gefunden:

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

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

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

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

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

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

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

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

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

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

- für x: [-0.295, 0.2281]
- für y: [0.1546, -0.1507]
- Zusätzlich wurde z = -0.4 gesetzt. Das UI Element musste ein wenig weiter von der Clipping Plane entfernt sein um angezeigt zu werden.

Mit diesen Intervallen für u,v,x und y konnten DetectedObjects gut lokalisiert werden und es wurden keine Objekte mehr weggelassen, von denen der Nutzer erwarten würden, das sie markiert werden.

5 Zusammenfassung

man kann zusätzlich zum Raycast auf die Spatial Map auch noch die Tiefentherapie nutzen. Spatial Mapping hat manchmal die Daten an der Stelle noch nicht, oder ist nicht aktualisiert. Besonders Probleme mit Wasserflaschen die nicht erkannt werden. Hinzuziehen von Tiefendaten kann das lokalisieren von Objekten im 3D Raum noch verbessern.

Literatur

[1] Keiron O'Shea und Ryan Nash. „An Introduction to Convolutional Neural Networks". In: *ArXiv e-prints* (Nov. 2015).

[2] Licheng Jiao u. a. „A Survey of Deep Learning based Object Detection". In: *IEEE Access* PP (Sep. 2019), S. 1–1. DOI: 10.1109/ACCESS.2019.2939201.

[3] N. Jmour, S. Zayen und A. Abdelkrim. „Convolutional neural networks for image classification". In: *2018 International Conference on Advanced Systems and Electric Technologies (IC ASET)*. 2018, S. 397–402.

[4] *Microsoft Azure Computer Vsion*. URL: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/ (besucht am 12. 06. 2020).

[5] *What is Computer Vision*. URL: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/home (besucht am 12. 06. 2020).

[6] *Detect common objects in images*. URL: https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-object-detection (besucht am 12. 06. 2020).

[7] *Mr und Azure 302 Maschinelles Sehen*. URL: https://docs.microsoft.com/de-de/windows/mixed-reality/mr-azure-302 (besucht am 12. 06. 2020).

[8] *Erkennen von alltäglichen Objekten in Bildern*. URL: https://docs.microsoft.com/de-de/azure/cognitive-services/computer-vision/concept-object-detection (besucht am 24. 09. 2020).

[9] *Mr und Azure 302b benutzerdefinierte Vision*. URL: https://docs.microsoft.com/de-de/windows/mixed-reality/mr-azure-302b (besucht am 12. 06. 2020).

[10] *Spatial Mapping*. URL: https://docs.microsoft.com/de-de/windows/mixed-reality/spatial-mapping (besucht am 17. 04. 2020).

[11] *magic leap 1*. URL: https://www.magicleap.com/en-us/magic-leap-1 (besucht am 18. 09. 2020).

[12] *Glossary and Usage*. URL: https://developer.magicleap.com/en-us/learn/guides/glossary (besucht am 18. 09. 2020).

[13] *Lumin OS Overview*. URL: https://developer.magicleap.com/en-us/learn/guides/lumin-os-overview (besucht am 18. 09. 2020).

[14] *Magic Leap Features*. URL: https://developer.magicleap.com/en-us/learn/guides/magic-leap-features (besucht am 18. 09. 2020).

[15] *World Reconstruktion*. URL: https://developer.magicleap.com/en-us/learn/guides/world-reconstruction-overview-landing (besucht am 18. 09. 2020).

[16] *1.4 Spatial Meshing - Unity*. URL: https://developer.magicleap.com/en-us/learn/guides/meshing-in-unity (besucht am 18. 09. 2020).

[17] *App Security*. URL: https://developer.magicleap.com/en-us/learn/guides/application-security-overview (besucht am 18. 09. 2020).

[18] *Camera.cameraToWorldMatrix*. URL: https://docs.unity3d.com/ScriptReference/Camera-cameraToWorldMatrix.html (besucht am 18. 09. 2020).

[19] *Matrix4x4.MultiplyPoint*. URL: https://docs.unity3d.com/ScriptReference/Matrix4x4.MultiplyPoint.html (besucht am 18. 09. 2020).

[20] *JsonUtility.FromJson*. URL: https://docs.unity3d.com/ScriptReference/JsonUtility.FromJson.html (besucht am 24. 09. 2020).