Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Marcell Klepper

Objekterkennung im Kontext einer Augmented-Reality-Anwendung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Marcell Klepper

Objekterkennung im Kontext einer Augmented-Reality-Anwendung

Bachelorarbeit eingereicht im Rahmen des Studiums

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai v. Luck
Zweitgutachter : Prof. Dr. Andreas Meisel
Fachbetreuer : Paul Assendorp

Abgegeben am 6. Mai 2019

Marcell Klepper

Thema der Bachelorarbeit

Objekterkennung im Kontext einer Augmented-Reality-Anwendung

Stichworte

Objekterkennung, Augmented Reality, Machine Learning, Klassifikation, Punktwolken, Sensordaten

Kurzzusammenfassung

Augmented Reality etabliert sich zunehmend als neues Werkzeug zur Bewältigung von Problemen und Verbesserung von Prozessen in Industrieanwendungen. Auch bei der Durchführung von Testprozessen in der Fertigungsindustrie birgt Augmented Reality großes Potential. Deshalb wird in dieser Arbeit ein realer Anwendungsfall untersucht und dazu ein Verfahren zur Erkennung und Ortung von Objekten im Kontext von Augmented Reality entwickelt.

Marcell Klepper

Title of the paper

Object detection in context of an augmented reality application

Keywords

Object recognition, object detection, augmented reality, machine learning, classification, point clouds, sensor data

Abstract

Augmented reality is emerging as a new tool to solve problems and improve processes in industry applications. For instance applying augmented reality has great potential to assist test processes in the manufacturing industry. Therefore, this thesis evaluates a real-world use case and develops a method to detect and locate objects in context of augmented reality.

Danksagung

Ich möchte mich an dieser Stelle zunächst bei Prof. Kai v. Luck und Prof. Andreas Meisel für die Betreuung dieser Arbeit bedanken und für Ihre Ratschlägen, wann immer ich Probleme hatte oder eine andere Perspektive auf ein Problem brauchte.
Ebenfalls möchte ich mich beim 3DSpace der HAW Hamburg bedanken, wo ich das Modell des Motorblocks, das ich in dieser Arbeit verwendet habe, ausdrucken durfte, obwohl der Druck an die vier Tage dauerte.
Großer Dank gilt der Werum Software & Systems AG und im Speziellen Hendrik Bohlen und Manfred Warzawa, die mir diese Arbeit ermöglichten und mir während der Umsetzung immer wieder gutes Feedback gegeben haben.
Ich möchte mich auch besonders bei Paul Assendorp von Werum bedanken, der immer ein offenes Ohr hatte, wenn ich jemanden brauchte, um Ideen auszutauschen, immer gutes Feedback und hilfreiche Denkanstöße gegeben hat und mit seinem Fachwissen zum Thema Machine Learning eine sehr große Hilfe war.
Und nicht zuletzt möchte ich mich bei meinen Freunden und meiner Familie bedanken, die mir während des gesamten Studiums immer zur Seite standen und mir bei der sprachlichen Korrektur der Arbeit eine große Hilfe waren.

Inhaltsverzeichnis

Abkürzungsverzeichnis

AR ..... *Augmented Reality*
ASIC ..... *Application-Specific Integrated Circuits*
AV ..... *Augmented Virtuality*
CNN ..... *Convolutional Neural Network*
CPU ..... *Central Processing Unit*
CV ..... *Computer Vision*
DL ..... *Deep Learning*
DNN ..... *Deep Neural Network*
GPU ..... *Graphical Processing Unit*
HMD ..... *Head Mounted Display*
HPU ..... *Holographic Processing Unit*
ICP ..... *Iterative Closest Point*
ILSVRC ..... *ImageNet Large-Scale Visual Recognition Challenge*
IMU ..... *Inertial Measurement Unit*
ML ..... *Maschine Learning*
MR ..... *Mixed Reality*
MRTK ..... *Mixed Reality Toolkit*
PCD ..... *Point Cloud Data*
PCL ..... *Point Cloud Library*
PLA ..... *Polyactid*
PVA ..... *Polyvinylalkohol*
RANSAC ..... *Random Sample Consensus*
SDKs ..... *Software Development Kit*
SIFT ..... *Scale Invariant Feature Transform*
SLAM ..... *Simultanous Localization and Mapping*
TPU ..... *Tensor Processing Unit*
TUI ..... *Tangible User Interface*
UX ..... *User Experience*
VIO ..... *Visual Inertial Odometry*
VR ..... *Virtual Reality*

1 Einleitung

1.1 Motivation

In der heutigen Industrie stehen Entwicklungsabteilungen zunehmend vor der Herausforderung, in Anbetracht von stetig komplexer werdenden Produkten eine effiziente Entwicklung und Validierung zu gewährleisten. Durch die steigenden Anforderungen in der Produktentwicklung werden im Zuge der Digitalisierung neue Werkzeuge benötigt, die Ingenieure in ihrer Arbeit unterstützen. Nur so kann auch in Zukunft eine zuverlässige Entwicklung ermöglicht werden.

Eines dieser möglichen Werkzeuge ist die Augmented Reality (AR). AR ist schon seit langem ein Thema, zu dem geforscht wird. Das Verständnis, was genau AR ist, wurde 1994 von Paul Milgram [1] in seinem Reality-Virtuality Continuum geprägt. Dieses Konzept beschreibt alle Varianten der Zusammensetzung von realen und virtuellen Objekten. So gibt es die zwei Extreme: Die reale Welt und die virtuelle Realität (VR). Die reale Welt enthält keine virtuellen Objekte, während die VR eine komplett immersive, virtuelle Umgebung darstellt. Zwischen diesen beiden Extremen liegt das Konzept der Mixed Reality (MR), also die Kombination von realem und virtuellem. Hierbei wird wiederum zwischen AR und Augmented Virtuality (AV) unterschieden. AR bezeichnet die Anreicherung der realen Welt mit virtuellen Objekten, AV hingegen die Anreicherung der virtuellen Welt mit realen Objekten.

AR-Anwendungen werden immer häufiger verwendet, um Aufgaben zu erleichtern, die ansonsten viel Schulung und Erfahrung benötigen würden. So gibt es bereits heute Unternehmen, die durch AR-Anwendungen in Schulungen für höhere Effizienz in der Fertigung gesorgt haben. Ein solches Beispiel ist Boeing, das dadurch 35 % weniger Zeit für Schulungen benötigt und den Erfolg bei ersten Versuchen ohne Erfahrungen um 90 % erhöhen konnte [2].

Ein weiteres Werkzeug zur Unterstützung von Ingenieuren ist die HyperTest-Plattform[1] der Werum Software & Systems AG. Diese unterstützt die Forschungs- und Entwicklungsabteilungen von Kunden bei der Durchführung von Tests an Prüflingen auf spezialisierten Prüfständen. Als Prüflinge kommen z. B. Autos, Automotoren oder Flugzeug- und Raketenturbinen in Frage. An diesen müssen zu Testzwecken verschiedene Sensoren angebracht werden. Die Stellen, an denen diese installiert werden müssen, werden derzeit jedoch auf Strukturzeichnungen festgehalten. Ein Ingenieur, der vor solch einem Prüfling steht, muss entsprechend zur genauen Bestimmung der korrekten Position diese von der zweidimensionalen Zeichnung auf das dreidimensionale Objekt übersetzen und im Gedächtnis halten.

Der Unterschied zwischen den Informationen und ihrem Kontext wird als kognitive Distanz [2] bezeichnet. Die kognitive Distanz ist einer der Faktoren, die Menschen kognitiv belasten. Dies ist insofern ein Problem, als dass Menschen nur begrenzte kognitive Kapazitäten besitzen [2]. Wenn also eine Aufgabe diese Kapazitäten auslastet, können sie nicht mehr für andere Aufgaben verwendet werden. Daher wird versucht, solche Aufgaben zu vereinfachen.

AR bietet die Möglichkeit, diese Aufgaben zu vereinfachen. Eine AR-Anwendung kann die benötigten Informationen über die reale Welt legen und somit direkt im Kontext ebendieser arbeiten, wodurch die kognitive Distanz überwunden und die Kapazitäten des Menschen entlastet werden. Durch Ersetzen der beschriebenen Vorgehensweise mit einer AR-Anwendung ließe sich dieser Prozess vereinfachen und die Qualität steigern, indem Fehler bei der manuellen Ausführung verhindert bzw. erkannt werden können.

Aus diesen Gründen wird in dieser Arbeit eine Augmented-Reality-Anwendung zur Lösung eines Szenarios der Objekterkennung im Umfeld von Test- und Messdatenaufzeichnungen untersucht.

1.2 Aufbau der Arbeit

In Kapitel 2 wird zunächst die Problemstellung dieser Arbeit ausformuliert. Des Weiteren gibt das Kapitel einen Überblick der dazu relevanten Techniken und Verfahren zur Objekterkennung in AR-Anwendungen. Dies bietet die Grundlage für das dritte Kapitel, in dem die praktische Implementierung für Experimente zum Lösen der betrachteten Problemstellung dargestellt wird. Hierzu wird die Durchführung der verschiedenen zu evaluierenden Ansätze beschrieben. In Kapitel 4 werden die Ergebnisse diskutiert, die im Rahmen dieser Arbeit erzielt wurden. Dazu wird insbesondere ausgewertet, welche Vor- und Nachteile die betrachteten Ansätze bieten und welche Hindernisse sich mit den vorhandenen Technologien ergeben haben. In Kapitel 5 wird schließlich ein Fazit gezogen und ein Ausblick auf Möglichkeiten zu weiterführenden Arbeiten gegeben.

[1] Mehr Informationen zu HyperTest und Werum unter: https://www.werum.de

2 Analyse

2.1 Problemstellung

Im Kontext der Aufzeichnung von Test- und Messdaten von Tests oder Experimenten an Prüflingen müssen diese mit Sensoren versehen werden, damit die gewollten Informationen erfasst werden können. Diese Sensoren müssen an bestimmten Stellen am Prüfling angebracht werden, sodass z. B. die Temperatur an einer kritischen Stelle ausgelesen werden kann und nicht an einer Stelle, an der sie irrelevant ist.

In der Industrie wird auch heute noch mit Konstruktionszeichnungen dieser Prüflinge gearbeitet. Das bedeutet, dass ein Ingenieur beispielsweise anhand dieser Zeichnungen die korrekten Stellen am Objekt finden muss, um anschließend die Sensoren dort anzubringen. Dieses Problem lässt sich mithilfe einer AR-Anwendung vereinfachen. So können z. B. AR-Hologramme direkt am realen Prüfling an spezifische Stellen projiziert werden, an denen Sensoren benötigt werden. Dies sorgt für eine wesentlich intuitivere Arbeit und ermöglicht so eine effizientere Durchführung der Testprozesse.

Für diesen Anwendungsfall müssen verschiedene Probleme gelöst werden. Das Hauptproblem ist das Erkennen des Prüflings sowie dessen Position und Orientierung in Relation zur Kamera oder zum Nutzer. Weitere Aspekte sind die Lösung der Benutzerinteraktion mit der Anwendung und die Anzeige von Informationen für den Benutzer.

Es handelt sich also um ein Problem der Objekterkennung, Orientierungserkennung und Augmented Reality. Es muss ein System erstellt werden, das den Prüfling sowie dessen Position und Orientierung im Raum im Verhältnis zur verwendeten AR-Hardware bestimmt und anschließend die Anweisungen an diesem Prüfling korrekt darstellt.

Die Interaktion des Benutzers zum Starten der Objekterkennung kann durch verschiedene Ansätze gelöst werden. So kann durchgehend das Blickfeld des Nutzers nach dem zu findenden Objekt gescannt werden. Dies ist jedoch von der Rechenleistung der verwendeten Hardware abhängig. Andere mögliche Ansätze sind zum einen das Starten des Erkennens durch eine aktive Aktion des Nutzers, wie beispielsweise einen Sprachbefehls oder eine Geste. Ein anderer ist das Erfassen, dass der Nutzer für eine zuvor festgelegte Zeit seinen Kopf stillhält. Beide Ansätze setzen voraus, dass der Benutzer, der die Anwendung verwendet, weiß, welche Objekte erkannt werden können, oder welche nicht. Im Kontext

dieser Arbeit kann davon ausgegangen werden, dass dies der Fall ist, weshalb beide generell einsetzbar sind.

Für den betrachteten Anwendungsfall wird hier der Ansatz des aktiven Startens des Scans durch den Nutzer verwendet, da die Limitation durch die Hardware durch diesen am geringsten scheint. Die meisten AR-Geräte unterstützen sowohl Spracheingaben durch Mikrofone als auch Gestenerkennung. Beide können parallel zueinander unterstützt werden. So kann auch Benutzern die Interaktion ermöglicht werden, die beispielsweise in beiden Händen Gegenstände halten und nicht durch Gesten mit der Anwendung interagieren können.

Ein weiterer Aspekt des Anwendungsfalls ist das Anzeigen von Informationen am erkannten Objekt. Die Darstellung von Hinweisen in Form von Hologrammen ist ein großer Teil der AR. Daher sollte dies für das zu erstellende Szenario nur ein geringes Problem darstellen. Es gibt aber verschiedene Möglichkeiten dies so umzusetzen, dass diese an der korrekten Stelle an einem realen Gegenstand angezeigt werden. So könnte ein digitaler Klon des Objekts an der genauen Position des zu erkennenden platziert werden und dieses mit Hologrammen angereichert werden. Somit kann die gesamte Darstellung von Informationen unabhängig von der Erkennung verändert werden. Auch würde dies mit dem einmaligen Starten des Erkennens durch den Nutzer zusammen funktionieren.

Diese Arbeit betrachtet die Erkennung eines Objektes und dessen Orientierung. Das Anzeigen von Informationen und die Benutzerinteraktion wird im Rahmen dieser Arbeit nicht umgesetzt und ist Teil von weiterführenden Arbeiten. Basierend auf dem hier beschriebenen Anwendungsfall sowie der weiteren Analyse in den folgenden Abschnitten wird in 2.5 ein Szenario mit Annahmen und Einschränkungen erstellt. Die Bearbeitung dieses Szenarios wird im Detail im dritten Kapitel beschrieben.

2.2 Augmented Reality

AR beschreibt die Kombination von virtuellen Elementen mit dem realen Umfeld [3]. Im Unterschied zur VR, in der die Nutzer mit einer komplett immersiven virtuellen Welt interagieren und somit von der realen Welt abgeschottet sind, wird in der AR die reale Welt mit virtuellen Objekten angereichert. Somit können Benutzer immer noch ihre Umgebung wahrnehmen und es scheint als würden die realen und virtuellen Objekte zusammen in der realen Welt existieren.

2.2.1 Tracking von Objekten

Zur korrekten kontinuierlichen Darstellung von virtuellen Objekten müssen der Standort und die Lage des Nutzers bzw. der verwendeten Kamera bekannt sein. Die Berechnung oder genaue Schätzung dieser Position bezeichnet man als Tracking [3].

Für das Tracking werden verschiedene Ansätze und Techniken verwendet. Zum einen gibt es das mobile Positions-Tracking, das besonders in AR-Anwendungen für Handheld-Geräte wie Smartphones zum Einsatz kommt. Hierfür wird GPS zur Bestimmung der Position des Nutzers verwendet. Ein Beispiel für eine Anwendung, hiervon Gebrauch macht, ist die Videospiel-Applikation *Pokémon Go* vom Entwickler Niantic, das während des Sommers 2016 global sehr populär war und beispielsweise fünf verschiedene Guinness Weltrekorde innerhalb von zwei Monaten nach dessen Veröffentlichung brach[2].

Für den Kontext dieser Arbeit ist es jedoch nicht relevant, wo auf der Welt sich der Nutzer befindet, um ein Objekt zu erkennen. Daher wird das mobile Positions-Tracking in dieser Arbeit nicht verwendet. In anderen Anwendungsfällen, wie etwa der Erkennung von Sehenswürdigkeiten, könnte es aber zum Beispiel sinnvoll sein.

Eine weitere Art des Trackings ist das sensorbasierte Orientierungs-Tracking. Dies bedeutet, dass die Lage eines Gerätes mithilfe von verschiedenen Sensoren bestimmt wird. Hierfür werden z. B. Gyrosensoren, Magnetometer oder Inertialsensoren verwendet. Diese Kombination ist heutzutage in so gut wie allen Smartphones und Tablets zu finden.

Magnetometer können das Erdmagnetfeld um das Gerät messen. Sie sind aber auch anfällig für Störungen durch andere Magnetfelder im direkten Umfeld und sind daher vor allem eine Ergänzung als die Grundlage der Rotationsbestimmung. Zur Messung der Beschleunigung des Gerätes werden Inertialsensoren benutzt, die abhängig von ihrer Art entweder die lineare Beschleunigung oder die Rotationsgeschwindigkeit messen.

Des Weiteren kann eine Kamera zum Tracking verwendet werden. Hierzu werden Marken oder bestimmte geometrische Eigenschaften des aufgenommenen Bildes erkannt. Meist werden schwarz-weiße Marken mit geometrischen Figuren verwendet, da diese robuster gegenüber verschiedenen Beleuchtungsbedingungen sind als farbige. Sie haben oft einfache Formen wie Quadrate oder Kreise und haben eindeutigen, einfarbigen Rand.

Je nach verwendeter Methode zum Tracking der Marken können diese unterschiedlich aussehen. Manche Verfahren geben bestimmte Muster für den inneren Teil der Marke, während andere wiederum keine Vorgaben zum Aussehen der Muster machen. Dies hat

[2] Informationen zu den gebrochenen Rekorden sind zu finden unter:
http://www.guinnessworldrecords.com/news/2016/8/pokemon-go-catches-five-world-records-439327

verschiedene Vor- und Nachteile, so ist man z. B. bei festgelegten Marken eingeschränkter in der Auswahl einer passenden Marke, kann dadurch aber auch an Leistung gewinnen, wenn die Anwendung auf ebendiese vorgegebenen Marken optimiert wurde. Im Gegensatz dazu kann man bei frei gestaltbaren Marken diese an das gesuchte Objekt anpassen, muss jedoch in den meisten Fällen immer die gesamte Marke in der Aufnahme haben, damit das Tracking funktioniert.

Des Weiteren ist die Größe der eingesetzten Marke wichtig. Falls sie zu groß ist, kann es passieren, dass sie u. U. nicht mehr vollständig auf dem Bild zu sehen ist. Sollte sie andererseits zu klein sein, wird sie nicht oder nur fehlerhaft ausgemacht werden. Ein weiterer Einflussfaktor ist hierbei die Auflösung der verwendeten Kamera. Sollte diese zu gering sein, werden auch schlechtere Ergebnisse erzielt, da die Marken nicht gut erkannt werden können.

Ein großer Vorteil der Marken ist die einfache Integrierbarkeit, dadurch, dass sie lediglich ausgedruckt und an einem beliebigen Ort angebracht werden können. So kann man sie beispielsweise auch in Zeitschriften drucken.

Ein genereller Nachteil des markerbasierten Trackings ist, dass eine Marke oft direkt auf oder an einem zu erweiternden Gegenstand angebracht werden muss. Wenn eine Person also zu nah an diesen herantritt, kann die Marke nicht mehr erkannt werden und das Tracking schlägt fehl. Ähnlich ist es auch mit Marken, die sich im Hintergrund befinden und dadurch Probleme hervorrufen, da sie z. B. zu klein sind, um sie zuverlässig zu verfolgen. Dies führt dazu, dass die Marken oftmals im Blickfeld des Nutzers als störend empfunden werden.

Eine Alternative zu diesem Verfahren ist das geometriebasierte Tracking. Dieses ist genau genommen eine verallgemeinerte Form des markerbasierten Trackings, da bei ersterem nicht mehr Marken erkannt werden, sondern beliebige zuvor bekannte 2D- oder 3D-Modelle. Hierbei wird anhand der Unterschiede zwischen den aus zwei Bildern extrahierten Ecken und Kanten die Veränderung des Blickwinkels berechnet. Dieses Verfahren wird oft verwendet, wenn die betrachteten Bildbereiche wenige andere Merkmale abgesehen von der geometrischen Form enthalten.

Andere, schlechter mit bloßem Auge erkennbare, Merkmale lassen sich mit Merkmalsdetektoren identifizieren. Wenn genügend Merkmale erkannt werden, kann auf Basis dessen - durch Vergleich der Beschreibungen (Deskriptoren) der Merkmale - die Position und Rotation der Kamera berechnet werden. Dieses Verfahren wird als merkmalsbasiertes Tracking bezeichnet und wird in vielen AR-Anwendungen verwendet. Ein weitverbreitetes Beispiel ist Simultanous Localization and Mapping (SLAM) [4] [5].

2.2.2 Registrierung von Objekten

Registrierung bezeichnet die passende Darstellung von digitalen Inhalten in der realen Welt [3]. Hierbei muss beachtet werden, dass sie sowohl aus jeder Perspektive korrekt dargestellt werden müssen (geometrische Registrierung), als auch die Beleuchtung angemessen sein sollte (fotometrische Registrierung).

Die geometrische Registrierung basiert auf dem beschriebenen Tracking. Auf Grundlage der berechneten oder geschätzten Transformation zwischen der Kamera und dem verfolgten Objekt wird dieses korrekt im Blickfeld des Benutzers dargestellt. Das heißt, dass auch wenn sich der Anwender und damit die Kamera bewegt, die Position des Objektes durch Anpassung der Transformation relativ zu diesem nicht verändert.

Die Qualität der Registrierung hängt daher insbesondere von der Qualität und der Geschwindigkeit des Tracking-Verfahrens ab. Wenn das angewandte Verfahren zu ungenau ist, werden Objekte an falschen Positionen registriert, z. B. in einer realen Wand und nicht vor ihr. Doch auch die Geschwindigkeit ist wichtig. Probleme treten z. B. auf, wenn die Tracking-Rate wesentlich geringer ist als die Bildwiederholrate des verwendeten Displays. Dadurch kann es passieren, dass ein Objekt für eine Weile mit der Bewegung der Kamera mitzieht, bevor es an seine eigentliche Position zurückspringt.

Diesem Problem lässt sich bei Video-See-Through-Displays (siehe 2.2.3) vorbeugen, indem die Bildwiederholrate angepasst wird, sodass sie mit der Tracking-Rate übereinstimmt. Wenn jedoch die Wiederholrate zu gering gesetzt wird, kann dies zu ruckelnden oder abrupten Bildwechseln führen. Dies kann als ebenso störend empfunden werden wie die springenden Objekte. Für optisches See-Through-AR, bei dem der Nutzer jederzeit die Realität um sich herum wahrnimmt, sollte dies entsprechend nicht angewendet werden.

Ein weiterer wichtiger Aspekt für eine gute geometrische Registrierungsqualität ist die Latenz des Trackings. Diese bezeichnet die Zeit zwischen der Messung der neuen Position der Kamera nach einer Bewegung und dem Zeitpunkt, an dem diese für die Objekttransformation verwendet werden kann. Eine hohe Latenz wirkt sich ähnlich aus wie eine zu niedrige Tracking-Rate. Objekte folgen zunächst der Bewegung der Kamera und springen dann zurück, wenn die neue Transformation erhalten wurde.

Generell führt eine fehlerhafte geometrische Registrierung zu einer Zerstörung der Immersion und der entsprechenden User Experience (UX) und sollte daher vermieden werden.

Fotometrische Registrierung wird im Vergleich zur geometrischen Registrierung, die zwingend notwendig für AR-Anwendungen ist, nur selten eingesetzt. Für die fotometrische Registrierung von AR-Elementen benötigt man gemessene oder geschätzte Daten zur

Beleuchtungssituation der Position, an welcher sie registriert werden soll. Auf Basis dieser wird das anzuzeigende Objekt dann visuell angepasst [3]. Der Schwerpunkt dieser Arbeit ist jedoch nicht die fotometrische Registrierung, weshalb diese hier nicht weitergehend beleuchtet wird. Allerdings lässt sich allgemein sagen, dass durch eine korrekte fotometrische Registrierung die Immersion des Nutzers wesentlich gesteigert werden kann [3].

2.2.3 Visuelle Ausgabe

Zur visuellen Ausgabe von AR-Anwendungen werden verschiedene Technologien verwendet [3]. Sehr verbreitet sind Handheld-Geräte wie Smartphones oder Tablets, die oftmals eine Kamera auf der Rückseite und Sensoren zur Lagebestimmung verbaut haben. Hierdurch ist hybrides Tracking auf Basis von sensorbasierten und optischen Daten möglich ist. Eine weitere Möglichkeit ist ein sogenanntes Video-See-Through-Display. In den meisten Fällen sind dies Datenbrillen oder Head-Mounted-Displays (HMDs). In der VR wird auf diesen ein Video mit einer virtuellen Welt dargestellt. In der AR aber wird stattdessen ein um digitale Objekte angereichertes Video der Umgebung des Benutzers auf das Display projiziert. Hierfür sind eine oder mehrere Videokameras an dem HMD angebracht, welche die Umgebung aufnehmen.

Eine Alternative hierzu sind die Optical-See-Through-Displays, die ebenfalls als Datenbrillen bezeichnet werden. Diese zeigen jedoch kein Video der Umgebung überlagert mit digitalen Elementen. Stattdessen werden diese nur auf das Display projiziert. Ein Nachteil dieser Technologie ist jedoch, dass es durch die verschiedenen Techniken zur Überlagerung zu geringerem einfallendem Licht kommt. Dadurch scheint es für den Benutzer, als sei die Realität verdunkelt, ähnlich dem Effekt einer Sonnenbrille. Auch sind diese Brillen durch die geringe Lichtstärke oft nicht geeignet zum Einsatz im Sonnenlicht [3].

2.2.4 Interaktion mit Objekten

Die grundlegende Interaktion mit AR liegt in der Navigation durch die reale Welt, da AR lediglich eine Erweiterung dieser ist [3]. Dadurch können die Bewegung des Anwenders und die virtuelle Bewegung nicht entkoppelt werden, wie dies in der VR möglich ist. Die Benutzerinteraktion ist also auch gebunden an die physische Nähe zum virtuellen Objekt. So muss der Benutzer an einem bestimmten Ort sein und z. T. in Richtung des Objektes schauen.

Zur Selektion von Menüpunkten oder Objekten wird in AR-Anwendungen für Datenbrillen zumeist die Blickrichtung des Benutzers gemessen. Hierzu kann z. B. Eye-Tracking angewandt werden, das jedoch zusätzliche Kameras zur Aufnahme der Augen sowie deren Kalibrierung benötigt. Oftmals wird statt der tatsächlichen Blickrichtung die Orientierung des Kopfes verwendet. Dazu wird im Sichtfeld des Benutzers ein Cursor oder Fadenkreuz angezeigt, um

eine bessere Indikation zu bieten, was selektiert wird. Dies muss dann noch um eine Aktion zum Auslösen der Selektion ergänzt werden. Hierzu gibt es verschiedene Eingabemethoden, etwa Sprachbefehle oder Gestensteuerung.

Eine weitere Eingabemethode sind sogenannte Tangible User Interfaces (TUI) [3]. Mit diesen werden reale Gegenstände bezeichnet, die bestimmte Eigenschaften oder Zustände von digitalen Objekten beeinflussen. So kann beispielsweise die Rotation eines virtuellen Würfels durch einen realen Würfel dargestellt werden.

2.3 Objekterkennung und Ausrichtung

Objekterkennung bezeichnet das Erkennen eines Objektes innerhalb eines bestimmten Kontextes, wie z. B. eines Bildes. Sie lässt sich unterscheiden in das reine Klassifizieren von Objekten sowie das Orten desselben. Diese Arbeit beschäftigt sich mit beiden Problemen. Daher werden zunächst Möglichkeiten der Klassifizierung und anschließend zur Ortung beschrieben.

Die Objekterkennung in Bildern ist seit vielen Jahren ein Forschungsgebiet, das sehr aktiv untersucht wird und entsprechend auch viele Neuerungen und Durchbrüche verzeichnet hat. So wurde 2012 ein großer Durchbruch mit dem AlexNet [6] geschaffen, das eine weit höhere Genauigkeit erreicht hat als zuvor möglich. Computer Vision (CV) ist aber bereits seit fast 60 Jahren ein Forschungsgebiet. Eines der ersten Papers [7], die großen Einfluss auf CV hatten, war genau genommen im Bereich der Neurophysiologie angesiedelt. Hubel und Wiesel haben in diesem herausgefunden, dass es einfache und komplexe Neuronen im primären visuellen Kortex gibt und dass die Verarbeitung von Gesehenem immer mit einfachen Strukturen wie Kanten startet. Dies ist eines der Kernprinzipien von Deep Learning (DL).

Ein weiterer Grundstein aktueller CV wurde bereits 1963 von Lawrence Roberts [8] gelegt. Dieser hat aus zweidimensionalen Bildern dreidimensionale Repräsentationen von Objekten erstellt. David Marr kam 1982 [9], basierend auf den Ideen von Hubel und Wiesel, zu dem Ergebnis, dass das Sehen und dadurch das Erkennen von Dingen hierarchisch stattfinden. Er hat damit eine Rahmenstruktur eingeführt, die in Low-Level-Algorithmen für die Erkennung von Kanten, Ecken oder Kurven als Sprungbrett für ein höheres Verständnis von visuellen Daten verwendet wird. Ungefähr zur gleichen Zeit hat Kunihiko Fukushima, ebenfalls inspiriert von Hubel und Weisel, ein künstliches Netzwerk von einfachen und komplexen Zellen zur Erkennung von wiederkehrenden Mustern erstellt [10]. Dieses Netzwerk beinhaltete bereits Convolutional Layers, Receptive Fields und Filter (mehr zu diesen in 2.3.2). Damit ist es eines der Vorgänger moderner Convolutional Neural Networks (CNNs).

Ende der 90er Jahre hat sich der Fokus der CV-Forschung verschoben. Viele Wissenschaftler haben nicht mehr versucht, den Ansatz von Marr umzusetzen, sondern haben stattdessen zu merkmalsbasierter Objekterkennung geforscht. Eine der bezeichnenden Arbeiten hierzu kam

von David Lowe [11], welcher ein System zur Erkennung von Objekten beschreibt, das lokale Merkmale verwendet. In 2001 wurde dann das erste Echtzeit-Gesichtserkennungsframework von Paul Viola und Michael Jones beschrieben [12], das sogenannte Haar-Merkmale verwendet. Erst seit 2012 sind die CNNs wieder in den Fokus gerückt. Dies hat mehrere Gründe. Zum einen haben aktuelle Computer eine weit höhere Rechenleistung als noch in den 80ern oder 90ern. Dies hängt nicht zuletzt auch mit effizient parallelisierbaren Grafikkarten zu tun. Zum anderen gibt es heutzutage einfachen Zugang zu großen, beschrifteten Datensätzen wie ImageNet [13] oder Coco [14], die das Erforschen und Trainieren von neuronalen Netzen vereinfachen.

In den folgenden Teilabschnitten werden zunächst die zwei Ansätze für die Klassifizierung von Objekten genauer beschrieben. Erst wird die Objekterkennung durch Algorithmen ausgeführt. Anschließend werden neuronale Netze und insbesondere die für die Bildauswertung relevanten CNNs vorgestellt sowie eine Entscheidung getroffen, welcher der Ansätze verfolgt wird. Daraufhin werden zwei Möglichkeiten zur Ortung von Objekten mithilfe von Tiefendaten erläutert.

2.3.1 Merkmalsextraktion aus Bildern

Grundlegende Objekterkennung setzt oft auf Methoden wie Kanten- oder Farberkennung, um Objekte zu erkennen. Es muss jedoch auch immer auf die Art der Anwendung geachtet werden, um das bestmögliche Verfahren zu verwenden. So reichen für manche Programme die Erkennung einer bestimmten Farbe, um einen Gegenstand zu erkennen, wenn z. B. jedes der zu erkennenden Objekte unterschiedlich koloriert wurde. Jedoch stellt dies einen idealisierten Fall dar und ist entsprechend bei vielen Realanwendungen nicht möglich.

Ein Verfahren, das oft verwendet wird, ist die Klassifizierung und der Vergleich von Merkmalen. So können Merkmale in zwei verschiedenen Bildern erfasst und anschließend verglichen werden, um zu erkennen, ob ein Objekt in beiden vorkommt. Beispiele für Algorithmen, die Merkmalserkennung mit oder ohne Deskriptoren implementieren, sind SIFT [15], SURF [16] oder HOG [17]. Es gibt auch neuronale Netze, die statt direkt auf den Bildern auf diesen Merkmalen arbeiten. Im Folgenden wird der Scale-Invariant Feature-Transform-Algorithmus (SIFT) nach David G. Lowe [15] beispielhaft beschrieben, um einen Eindruck zu vermitteln, wie diese Algorithmen arbeiten und weshalb sie funktionieren.

SIFT generiert Beschreibungen für Merkmalspunkte in Bildern. Diese Beschreibungen, auch Deskriptoren genannt, fassen die lokale Struktur um den Merkmalspunkt herum zusammen [18]. Die SIFT-Deskriptoren sind sehr robust und oft angewendet. Andere Varianten von Deskriptoren sind u. U. schneller berechenbar oder haben verschiedene Schwerpunkte. SIFT ist jedoch bis heute ein grundlegender Bestandteil der Objekterkennung und wird nach wie vor viel verwendet. In den vergangenen Jahren gibt es außerdem den Trend, Deskriptoren durch Machine Learning zu lernen. SIFT-Deskriptoren sind keine maschinell erlernten

Deskriptoren, sondern wurden manuell von Menschen entworfen, um eine Stelle und deren Umgebung bestmöglich zu beschreiben.

SIFT sucht nach lokalen ausgeprägten Punkten. Dies hat gegenüber einem Verfahren, das globale Punkte ermittelt, den Vorteil, dass beispielsweise ein Objekt, das in einem Bild an einer Stelle steht (z. B. ein Laptop auf einem Tisch), auch in einem anderen Kontext noch erkannt werden kann (z. B. wenn er dann auf dem Boden steht), da nach lokalen Punkten gesucht wird [18]. In einem globalen Kontext würde das Bild als Gesamtes angesehen und entsprechend das Objekt nicht mehr bestimmt werden. Im lokalen Kontext werden nur Teile des Bildes betrachtet und so können Anteile eines Bildes auch innerhalb eines anderen mit unterschiedlicher Umgebung erkannt werden.

Zum Finden dieser Punkte wird ein Verfahren eingesetzt, das sich aus mehreren Schritten zusammensetzt [18]. Der erste Schritt ist die Gauß-Glättung des Bildes. Anschließend werden die Extrempunkte durch das Berechnen von Differenzen zwischen verschiedenen Gauß-Glättungen gefunden. Der letzte Schritt ist dann die Unterdrückung von Maxima an den Kanten innerhalb des Bildes.

Durch das Glätten des Bildes und den anschließenden Vergleich mit verschiedenen Glättungen können Kanten oder Ecken und andere Details innerhalb des Bildes sehr gut erkannt werden [18]. Ein Beispiel ist der Vergleich eines ungeglätteten, einfarbigen Bildes, auf dem nur ein andersfarbiger Fleck zu sehen ist, mit einem geglätteten Bild. Die Differenz der Intensität der Pixelwerte in den einfarbigen Bereichen wird 0 betragen, da diese nicht geglättet werden. An den Rändern zwischen dem Fleck und dem Hintergrund jedoch wird die Differenz nicht 0 sein, da diese geglättet werden und die Intensitäten der Pixelwerte sich entsprechend unterscheiden. Dadurch werden die Kanten zwischen unterschiedlich gefärbten Objekten oder Teilobjekten hervorgehoben.

Um nun aber robuste Ergebnisse bei unterschiedlichen Entfernungen zu erhalten, wird das Bild in verschiedene Größen skaliert und für alle diese die Differenzen zwischen den geglätteten Bildern berechnet [18]. Dies führt dazu, dass beispielsweise ein Objekt aus einem oder zwei Metern Entfernung betrachtet immer noch als das gleiche Objekt erkannt wird.

Anschließend wird in diesen Glättungen der Skalierungen nach Extrema gesucht. Durch diese verschiedenen Level an Glättungen können unterschiedlich feingranulare Unterschiede erkannt werden. Wenn beispielsweise ein wenig geglättetes Bild mit einem geringfügig mehr geglätteten Bild verglichen wird, werden feine Details herausstechen. Zwischen zwei stark geglätteten Bildern werden hingegen grobe Strukturen herausstechen. Durch diese Berechnungen können mithilfe von unterschiedlich geglätteten und skalierten Bildern die gesuchten Merkmalspunkte in Bildern gefunden werden [18].

Ein Problem bei der Erkennung dieser Punkte ist das Erkennen einer perfekt geraden Kante [18]. Wenn die Kante nämlich perfekt gerade ist und die komplette Länge über durchgängig gleich aussieht, werden alle Punkte der Kante identisch erkannt. Dies bedeutet, dass identifiziert werden kann, dass die Kante existiert, nicht jedoch wo entlang der Kante sich der Punkt befindet. Um dieses Problem zu lösen, werden im SIFT Algorithmus im dritten Schritt die Maxima an den Kanten unterdrückt, wodurch letztendlich nur noch die wirklichen Merkmalspunkte bestehen bleiben.

Wenn die Merkmalspunkte gefunden wurden, werden die Punkte um den Merkmalspunkt betrachtet und basierend darauf, wie diese aussehen, die Deskriptoren berechnet [18]. Diese Deskriptoren sind invariant in Hinsicht auf Verschiebung oder Translation des Merkmals. Ein Objekt wird mit diesem Merkmal auch an einem anderen Ort erkannt. Selbst bei Drehung oder Rotation eines Merkmals, wenn z. B. die Kamera auf dem Kopf steht, wird das Objekt mit dem Merkmal erkannt, ebenso bei Skalierung des Merkmals, also wenn es sich weiter entfernt von der Kamera befindet (natürlich nur bis zu einem gewissen Maße). Auch sind die Deskriptoren bis zu einem gewissen Grad unverändert in Bezug auf Belichtungsveränderungen, verschiedene Betrachtungswinkel und unterschiedliche Kameras.

Um nun also ein Objekt in einem anderen Bild zu finden, werden die Deskriptorwerte in beiden Bildern verglichen und die Deskriptoren mit den geringsten Unterschieden als Übereinstimmungen gesehen [18]. Dies kann viele korrekte Ergebnisse bringen, kann aber auch zu falschen Übereinstimmungen führen, wenn zwei Punkte sehr ähnlich, nicht jedoch gleich sind, beispielsweise bei sich wiederholenden Strukturen. Dieses Problem zu lösen ist aber nicht mehr Teil des SIFT-Algorithmus, sondern wird von anderen Algorithmen betrachtet [18] (wie z. B. dem Random-Sample-Consensus-Algorithmus (RANSAC)). Der SIFT-Algorithmus und vergleichbare Algorithmen bieten also die Möglichkeit, Übereinstimmungen von Merkmalen in verschiedenen Bildern zu finden und dadurch zu erkennen, ob ein Objekt in einem Bild vorhanden ist oder nicht.

2.3.2 Neuronale Netze zur Objekterkennung

Andere Ansätze für Objekterkennung sind solche, die auf Machine Learning (ML) im Allgemeinen oder Deep Learning im Speziellen setzen. Diese haben in jüngeren Jahren immer mehr Zulauf erhalten, da sie eine höhere Genauigkeit bei der Erkennung von Objekten aufweisen als z. B. SIFT oder SURF [6].

ML bezeichnet das Lernen von Regeln oder Repräsentationen anhand von Beispielen vorhandener Daten [19]. Für ML werden also Eingabedaten, beispielhafte Ausgaben sowie ein Ansatz zur Messung, wie akkurat der ML-Algorithmus arbeitet, benötigt.

Deep Learning ist ein Unterfeld des Machine Learnings. Für DL werden sogenannte Deep Neural Networks (DNN) verwendet, die aus vielen geschichteten Abstraktionsebenen bestehen. Die Anzahl dieser Ebenen wird als Tiefe des Netzwerks bezeichnet, wodurch sich auch der Name DNN herleitet. Neuronale Netze sind Funktionsapproximatoren, die zur Lösung einer gegebenen Aufgabenstellung das Extrahieren hierarchischer Merkmalsrepräsentationen aus Daten erlernen können. Das heißt das, was zuvor durch Algorithmen wie SIFT übernommen wurde, können die neuronalen Netze automatisiert durchführen.

Für das Training eines Netzwerkes wird ein Datenbestand benötigt, der zum Lernen verwendet werden kann. So muss, um ein DNN zur Objekterkennung zu erhalten, dieses zunächst mit Informationen darüber versorgt werden, wie das zu erkennende Objekt in verschiedenen Kontexten aussieht. Hierfür werden bereits vorhandene Bilder des Objektes verwendet. Aus diesen Daten werden Merkmale automatisch extrahiert und sind entsprechend nicht angewiesen auf Expertenwissen. Dies ist der größte Unterschied zwischen der Erkennung durch ein neuronales Netz und den in 2.3.1 beschriebenen hartkodierten Algorithmen zur Merkmalserkennung.

Es gibt aber auch verschiedene Möglichkeiten von Data Augmentation, also der Verbesserung der Daten mithilfe von bestimmten Techniken [20]. Selbst wenn heutzutage der Zugriff auf viele Bilddaten gegeben ist, ist die Anzahl doch auch noch immer begrenzt. Neuronale Netzwerke werden allgemein gesagt aber präziser, je mehr unterschiedliche Daten für das Training zur Verfügung stehen. Ein Problem ist jedoch, dass mehr Daten zu bekommen immer auch mit Kosten zusammenhängt. Diese können Zeit, monetäre Mittel oder Rechenleistung sein. Daher werden stattdessen die bereits vorhandenen Daten mithilfe von Data Augmentation vermehrt [20].

Hierbei ist es wichtig, dass nur Data-Augmentation-Techniken verwendet werden, die im Kontext des zu erkennenden Objektes Sinn ergeben. Wenn beispielsweise ein Hund erkannt werden soll, so können die Bilder des Hundes auch gespiegelt werden. Ein gespiegeltes Bild eines Hundes ist immer noch ein Bild eines Hundes. Auch gibt es Möglichkeiten den Kontrast eines Bildes zu erhöhen oder zu verringern oder das Bild zu rotieren. Eine weitere Option der Data Augmentation ist das Generieren von Daten durch das Platzieren des Objektes auf Hintergrundbildern. Es gibt verschiedene Bildbibliotheken wie ImageNet [13], die eine Vielzahl von kategorisierten Bildern anbieten, um Hintergründe für ein Objekt zu erstellen.

Ein zu beachtender Punkt ist, dass ein Objekt auch nach der Augmentation weiterhin als das gleiche Objekt zu erkennen sein muss. Solange dies gegeben ist, sind all diese Techniken möglich und verbessern die Genauigkeit des Erkennens durch das Netzwerk [20].

Insbesondere für die Klassifikation von Bildern hat sich in den letzten Jahren der DL-Ansatz als genauer herausgestellt als die klassischen Verfahren. Der erste Durchbruch wurde mit

dem AlexNet [6] aus 2012 erzielt. Dieses war das erste CNN, das in der ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) eine Top-5-Fehlerrate von 15,4 % erreichte und damit mehr als 10 % geringer als der nächstbeste Eintrag[3]. Seitdem wurden neue und bessere Netzwerkarchitekturen erstellt, die noch geringere Fehlerraten erzielen. Ein Beispiel ist das Inception-V3-Netz [21], das auf demselben Datensatz eine Top-5-Fehlerrate von 5,6 % erreicht.

CNNs sind inspiriert von den Ergebnissen von Hubel und Wiesel von 1962 [7] und daher insbesondere für Bilderkennung relevant. Hubel und Wiesel haben herausgefunden, dass verschiedene Neuronen im Gehirn nur bei Gegenwart von Kanten mit bestimmter Orientierung reagieren. Manche reagieren nur, wenn vertikale Kanten sichtbar sind, andere wiederum nur bei horizontalen Kanten oder Kanten, die zu einem gewissen Grad gedreht wurden. Diese Idee von spezialisierten Komponenten in einem Netzwerk ist die Grundlage für CNNs. Aufgrund dieser Aktualität und dem Potential dieser Technologie wird in dieser Arbeit für die Klassifikation von Objekten ein ML-Ansatz verwendet.

Der generelle Aufbau eines CNNs lässt sich durch eine Reihe von Convolutional-, Pooling- und Fully-Connected-Ebenen, Layer genannt, beschreiben. Der Convolutional Layer, das dem Netzwerk auch seinen Namen verleiht, ist immer die erste Ebene in einem CNN. Die Eingabe in den Layer (z. B. eine Pixelmatrix) wird mit einem Filter (auch Neuron) belegt [19]. Die Größe des Filters ist das Receptive Field. Hat man beispielsweise eine 32 x 32 x 3 RGB-Pixelmatrix so könnte das Receptive Field 5 x 5 x 3 groß sein. Das heißt, der Filter beleuchtet einen entsprechend großen Teil der Matrix. Wichtig ist hierbei, dass die Tiefe des Filters mit der der Eingabe übereinstimmt. Der Filter ist ein Array von Zahlen, die als Gewichte oder Parameter bezeichnet werden. Die Gewichte des Filters werden dann mit den Werten der Pixelmatrix multipliziert. Im Beispiel erhält man hierdurch entsprechend 75 Produkte. Diese werden summiert zu einem einzelnen Wert. Dieser Prozess wird wiederholt für jede mögliche Position des Filters auf der Pixelmatrix, sodass am Ende ein 28 x 28 x 1 Array für alle eindeutigen Positionen entsteht. Dieses Array wird als Activation Map oder Feature Map bezeichnet.

Der Pooling Layer fasst Informationen der Activation Map zusammen. So gibt es z. B. das Max Pooling, welches aus einem 2 x 2 Bereich der Map nur den höchsten Wert behält. Die anderen Werte werden verworfen. Dies führt zu geringerem Speicherbedarf, da nur 25 % der Daten behalten werden, was tiefere Netzwerke erlaubt. Auch wird Overfitting, also das übermäßige Anpassen des Modells an die Daten, verhindert. Das Netz bleibt daher allgemeiner.

Zum Ende des Netzes gibt es den Fully Connected Layer. Dieser nimmt die Ergebnisse des vorangegangenen Layers entgegen und gibt einen Vektor mit der Größe der möglichen Klassen wieder. Die Werte des Vektors repräsentieren die Wahrscheinlichkeit dafür, dass das

[3] Ergebnisse der ILSVRC 2012 wurden veröffentlich unter: http://image-net.org/challenges/LSVRC/2012/results.html

Bild eine Klasse repräsentiert. Dies funktioniert, indem der Layer die Activation Maps des vorherigen Layers anschaut und bestimmt, welche Merkmale in diesen am meisten mit einer bestimmten Klasse übereinstimmen.

2.3.3 Random Sample Consensus (RANSAC)

Der RANSAC-Algorithmus wurde 1981 von Fischler und Bolles [22] beschrieben. Im Kontext des RANSAC-Algorithmus werden die Punkte eines betrachteten Datensatzes in zwei Kategorien aufgeteilt. Zum einen gibt es die Daten, die durch ein Modell mit bestimmten Parametern beschrieben werden können und zum anderen Ausreißer, die nicht durch dieses Modell beschrieben werden. Ausreißer werden allgemein als Outlier bezeichnet, die abgedeckten Daten als Inlier. Das Ziel des RANSAC ist nun, genau ein solches Modell zu finden, dass möglichst viele Inlier beschrieben werden.

Der Algorithmus besteht aus drei Schritten, die so lange wiederholt werden, bis mit hoher Sicherheit das bestmögliche Modell gefunden wurde [18]. Der erste Schritt ist die zufällige Wahl einer Teilmenge von Punkten aus der gesamten Menge vorhandener Datenpunkte. Die Anzahl der zu wählenden Datenpunkte ist abhängig von dem Anwendungsfall, den man betrachtet. Im zweiten Schritt werden die Parameter des Modells berechnet, das alle Punkte dieser Teilmenge beschreibt. Der letzte Schritt ist die Bewertung dieses Modells, indem betrachtet wird, wie viele Inlier durch dieses Modell innerhalb eines zuvor festgelegten Grenzwertes abgebildet werden.

Möchte man z. B. eine Gerade durch eine Menge von Punkten in einem zweidimensionalen Koordinatensystem ziehen, welche die Datenmenge möglichst genau beschreibt, kann der RANSAC-Algorithmus verwendet werden [18]. Dies könnte beispielsweise bei einer Versuchsauswertung verwendet werden.

Um eine Gerade zu beschreiben, werden zwei Punkte, der Stützpunkt und der Richtungspunkt, benötigt, um alle Parameter dieser Gerade bestimmen zu können. Entsprechend würde man hier für den RANSAC immer zwei zufällige Punkte innerhalb des Datensatzes auswählen, um durch diese eine Gerade zu ziehen. Als Nächstes werden alle Punkte gezählt, die innerhalb eines bestimmten Grenzwertes um die Gerade liegen. Dies wird so lange wiederholt, bis die Gerade mit denjenigen Parametern gefunden wird, die die höchstmögliche Wertung hat. Alle Punkte, die dann von diesem Modell beschrieben werden, sind die Inlier. Alle anderen sind Outlier und können entsprechend für weitere Berechnungen ignoriert werden.

Der RANSAC-Algorithmus kann auch sinnvoll in Ergänzung zum SIFT-Algorithmus verwendet werden [18]. Der SIFT-Algorithmus findet lediglich übereinstimmende lokale Merkmalspunkte und kann entsprechend zwei in der Realität verschiedene Punkte einander zuordnen. Dies kann dazu führen, dass man ein Ergebnis erhält, das viele korrekt, aber auch

viele falsch zugeordnete Übereinstimmungen beinhaltet. Um diese falschen Übereinstimmungen von den korrekten zu trennen und anschließend mit den korrekten weiterzuarbeiten, wird RANSAC verwendet.

Angenommen, zwei Bilder sollen so ausgerichtet werden, dass sie sich überlagern, dann können aus den beiden Bildern die Merkmalspunkte und Übereinstimmungen mithilfe von SIFT extrahiert werden [18]. Anschließend wird RANSAC verwendet und möglichst wenig dieser Übereinstimmungen werden ausgewählt, um aus diesen eine Transformation, also Rotation und Verschiebung, zu berechnen, sodass die gewählten Übereinstimmungen übereinanderliegen. Anschließend werden die restlichen Übereinstimmungen in beiden Bildern verglichen, um zu überprüfen, wie viele von diesen ebenfalls durch die Transformation korrekt übereinanderliegen. Wieder wird dies so lange wiederholt, bis das bestmögliche Ergebnis erzielt wurde, also möglichst viele Übereinstimmungen als Inlier modelliert wurden.

Die Anzahl an ausgewählten Punkten ist nicht zufällig, sondern immer das Minimum an benötigten Punkten, um ein Modell für das zu untersuchende Problem zu erstellen [18]. Im Fall der Geraden sind dies zwei, da es nur zwei Geradenparameter gibt. Dies muss entsprechend je nach angewendetem Modell angepasst werden, sodass alle Modellparameter beschrieben werden können. Dadurch kann dieser Algorithmus auch z. B. auf Tiefendaten angewandt werden, um so die genaue Orientierung eines Modells innerhalb einer Punktwolke zu bestimmen.

2.3.4 Iterative Closest Point (ICP)

Speziell für die Objekterkennung in 3D-Daten kann mit Punktwolken gearbeitet werden. Nicht nur der RANSAC-Algorithmus kann auf diese angewendet werden. Es gibt auch andere Ansätze, um mit Punktwolken zu arbeiten, wie z. B. den Iterative-Closest-Point-Algorithmus (ICP). Dieser wurde 1991 von Y. Chen und G. Medioni, [23] sowie 1992 von Paul J. Besl und Neil D. McKay beschrieben. [24]

Der ICP-Algorithmus nimmt als Eingabe eine Zielwolke, die als Referenz dient und nicht verändert wird, sowie eine Modellwolke, die durch Rotation und Translation an die Zielwolke angepasst wird [25].

Im Allgemeinen besteht der Algorithmus aus vier Schritten. Zuerst wird jedem Punkt der Modellwolke der nächste Punkt in Bezug auf Distanz in der Zielwolke zugeordnet. Als Nächstes wird für alle Punkte durch Rotation und Translation eine Transformationsmatrix berechnet, welche die Punkte am besten aneinander angleicht. Anschließend wird die Modellwolke mit dieser Transformationsmatrix angepasst, sodass eine neue Orientierung dieser Wolke entsteht. Zuletzt wird erneut über alle Punkte iteriert und die vorherigen Schritte werden wiederholt. Dies wird so lange durchgeführt, bis eines der Abschlusskriterien

erreicht wurde. Entweder wurde ein zuvor festgelegtes Fehlermaß, etwa die Distanz zwischen den Punkten der beiden Wolken, unterschritten oder die Änderungen durch die Transformationsmatrix sind geringer als ein vorher gesetzter Grenzwert [25].

2.4 Evaluierung vorhandener Soft- und Hardwaretechnologien

Im Folgenden wird auf bereits im Markt verfügbare Technologien für AR eingegangen.

2.4.1 Softwaretechnologien

Für das Erstellen von AR-Anwendungen werden mittlerweile immer mehr Software Development Kits (SDKs) zur Verfügung gestellt. So haben sowohl Apple mit ARKit[4] als auch Google mit ARCore[5] jeweils ein eigenes herausgebracht, aber ebenso Produkte von unabhängigen Entwicklern wie Vuforia[6] oder Open Source Lösungen werden angeboten.

Die meisten dieser Lösungen bieten Software-Stacks zum Tracking durch Visual Inertial Odometry (VIO) an. VIO bedeutet, dass die Software die Position des Nutzers im Raum in Echtzeit verfolgt [26]. Dies erfolgt durch eine Kombination von Daten von Bewegungs- oder Inertialsensoren (zusammen als Inertial Measurement Unit (IMU) bezeichnet) mit optischem Tracking durch die Kamera eines Objektes.

Apples ARKit beispielsweise verwendet VIO-Tracking zur Positionsbestimmung. Diese Position wird zwischen der Aktualisierung der angezeigten Bilder auf dem Display kontinuierlich neu berechnet. Im ARKit werden diese Berechnungen zweimal parallel zueinander durchgeführt und diejenige, die genauer ist, wird verwendet und an das ARKit-SDK weitergegeben. Der große Vorteil von VIO-Systemen ist, dass sie sowohl die Daten der IMU als auch der Kamera benutzen, da diese beiden Systeme die Schwächen des jeweils anderen ausgleichen. Sie basieren auf zwei komplett unterschiedlichen Messsystemen. Dadurch teilen sie sich keinerlei Abhängigkeiten, wodurch, sofern die Werte von einem System nicht genau genug für korrektes Tracking sind, das andere verwendet werden kann. Durch die Verwendung des Kalman-Filters wird immer das beste Ergebnis dieser Messungen zur Positionsbestimmung benutzt [26].

Ein weiterer Bestandteil des ARKits, der auch von vielen anderen SDKs unterstützt wird, ist die Plane-Detection oder Oberflächenerkennung. Hierdurch werden Flächen in der Welt

[4] Informationen zu ARKit zu finden unter: https://developer.apple.com/documentation/arkit
[5] Informationen zu ARCore zu finden unter: https://developers.google.com/ar/develop/
[6] Informationen zu Vuforia zu finden unter: https://developer.vuforia.com

erkannt, um Objekte korrekt in dieser zu registrieren. Dies verhindert, dass beispielsweise ein AR-Element über dem Boden schwebt, anstatt auf demselben zu stehen.

Googles ARCore ist sehr ähnlich zu ARKit mit einigen Vor- und Nachteilen. ARCore basiert auf Google Tango, das eine Referenzarchitektur und Software-Stack für Smartphones ist. 2017 wurde Tango zugunsten ARCores eingestellt. Der Software-Stack für VIO und Tiefenwahrnehmung sowie Oberflächenerkennung wurde weitestgehend von Tango übernommen [27].

Letztendlich lässt sich also sagen, dass beide Technologien mit Sicht auf UX nicht wirklich zu unterscheiden sind. ARKit hat aufgrund der hauseigenen Smartphones einen Vorteil hinsichtlich der Hardware-Kalibrierung und Software-Integration. Dadurch kann ein zuverlässigeres Tracking garantiert werden, während ARCore genauer in den Bereichen Spatial Mapping und Wiederherstellung verlorenen Trackings ist [27]. Die Funktionen, die beide SDKs zusätzlich anbieten, sind ebenfalls ähnlich, wodurch es sich nicht sagen lässt, welches SDK generell besser ist. ARKit bietet Unterstützung für alle moderneren Apple Smartphones ab dem iPhone 6s[7], während ARCore nur eine Auswahl von Android-Geräten unterstützt (zum Zeitpunkt der Recherche waren es 33 unterschiedliche Modelle[8]).

Vuforia ist ein weiteres kommerzielles SDK, das hardwareunabhängig eingesetzt werden kann und hierdurch für viele verschiedene Smartphones und AR-Headsets eine Alternative bietet[9]. Auch Vuforia setzt auf VIO-Tracking, ist aber aufgrund der Hardwareunabhängigkeit nicht so sehr auf spezielle Smartphones optimiert wie beispielsweise das ARKit [26]. Ein Herausstellungsmerkmal von Vuforia ist, dass es von Haus aus Objekterkennung unterstützt. Dies funktioniert aber nur für kleine Objekte und für das Scannen der Objekte können nur eine geringe Anzahl Smartphones benutzt werden[10].

Die Open Source SDKs verwenden lediglich optisches Tracking durch Anwendung von Computer-Vision-Algorithmen (CV), um die Position des Nutzers festzustellen. Das macht sie ebenfalls hardwareunabhängig, jedoch ist auch hier das Tracking nicht so qualitativ hochwertig wie bei spezialisierten SDKs [26].

---

[7] Informationen zu den unterstützten Geräten zu finden unter (Stand 12. Mai 2018): https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/DeviceCompatibilityMatrix/DeviceCompatibilityMatrix.html

[8] Informationen zu den unterstützten Geräten zu finden unter (Stand 12. Mai 2018): https://developers.google.com/ar/discover/supported-devices

[9] Informationen zu den unterstützten Geräten zu finden unter (Stand 12. Mai 2018): https://library.vuforia.com/articles/Solution/Vuforia-Supported-Versions

[10] Informationen zu Vuforias Objekterkennung: https://library.vuforia.com/articles/Training/Object-Recognition

Das Mixed Reality Toolkit[11] (MRTK) von Microsoft ist genau genommen kein SDK, sondern eine Zusammenstellung von Skripten, welche die Entwicklung von Anwendungen für die HoloLens vereinfachen sollen. Dies beinhaltet einfache Wege zum Behandeln von verschiedenen Eingabeformen wie Gesten- oder Spracheingaben, aber auch Sound-Ausgabe, Spatial Mapping oder UX. Es bietet einen guten Einstieg in die Entwicklung von AR-Anwendungen für Microsofts Produkte im Bereich Mixed Reality.

2.4.2 Hardwaretechnologien

Die Hardware, die für AR-Anwendungen benutzt werden kann, hängt sehr vom Anwendungsfall ab, der abgedeckt werden soll. Handheld-Geräte wie Smartphones oder Tablets können z. B. verwendet werden, wenn die Hände nicht unbedingt für die Aufgabe gebraucht werden, die von der AR-Anwendung vereinfacht werden soll. Ein AR-Headset mit See-Through-Display hingegen ist geeigneter, wenn während der Verwendung der Anwendung beide Hände frei bleiben sollen, was im Kontext dieser Arbeit auch der Fall ist.

Daher wird sich in diesem Teilabschnitt auf die verfügbaren AR-Headsets beschränkt. Viele der Headsets, die als AR-Headsets bezeichnet werden, sind oft Datenbrillen, die Informationen an statischen Positionen im Blickfeld des Benutzers darstellen. Zwar werden sie dadurch auch AR-Headsets genannt, sie sind jedoch für den Anwendungsfall dieser Arbeit nicht geeignet.

Dadurch gibt es nur wenige Headsets, die tatsächlich alle Funktionalitäten erfüllen, die für eine echte AR-Anwendung benötigt werden.

Zum einen gibt es mit diesen Kriterien die Microsoft HoloLens, die vom Unternehmen im Januar 2015 angekündigt wurde und seit Oktober 2016 in Deutschland erhältlich ist. Die Magic Leap One[12] soll einen ähnlichen Funktionsumfang wie die HoloLens haben, ist jedoch zum Zeitpunkt dieser Recherche nicht verfügbar. Aufgrund dieser Nachforschung wird die Anwendung dieser Arbeit auf Basis der Microsoft HoloLens entwickelt.

2.5 Betrachtetes Szenario

Basierend auf dieser Analyse wird im Folgenden ein Szenario ausgearbeitet, das im dritten Kapitel praktisch bearbeitet wird. Ein Prüfling soll in einer kontrollierten Umgebung erkannt werden. Dieser ist im Vorfeld bekannt. Entsprechend ist es möglich, ein neuronales Netz

[11] Das Mixed Reality Toolkit ist verfügbar unter: https://github.com/Microsoft/MixedRealityToolkit-Unity/

[12] Die Magic Leap One ist zu finden unter: https://www.magicleap.com/

dahingehend zu trainieren, nur dieses bestimmte Objekt zu erkennen. Um möglichst realitätsnah zu arbeiten, soll ein Modell eines Motors verwendet werden.

Die Erkennung des Objektes soll mithilfe der HoloLens geschehen. Es ist also nötig sicherzustellen, dass die Daten dieser weiterverarbeitet werden können. Das bedeutet, dass diese entweder direkt auf dem Gerät verarbeitet werden oder ein Kommunikationsweg zwischen der HoloLens und einem externen Rechner evaluiert werden muss. Die Erkennung des Prüflings soll initial einmalig geschehen, sodass ein digitaler Klon des Prüflings an dessen Stelle gesetzt werden kann. Dies hat den Vorteil, dass der Prüfling nicht durchgängig erkannt und die Rotation bestimmt werden muss.

Im Rahmen der Arbeit werden verschiedene Annahmen gemacht, um das zu lösende Szenario zu vereinfachen. Alle Vereinfachungen, die getroffen werden, sind realitätsnah und können daher in dieser Arbeit angewendet werden. So kann erwartet werden, dass sich das zu untersuchende Objekt zum Zeitpunkt der Erkennung mittig im Sichtfeld befindet, da es in diesem Moment im Fokus des Nutzers ist. Auch kann angenommen werden, dass der Prüfling nicht von anderen Objekten verdeckt ist und dies daher nicht im Training eines Modells beachtet werden muss. Eine weitere Annahme ist, dass der Prüfling immer auf dem Boden steht, sodass das Erkennen seiner Unterseite nicht benötigt wird.

Eine Begrenzung durch die Technologie der aktuellen Version der HoloLens ist, dass sich das Objekt entweder zwischen 80 cm und 4 m vor dem Benutzer, für den fernen oder zwischen 0 und 80 cm für den nahen Tiefensensor befinden muss, sofern die Informationen der Tiefenkamera der HoloLens verwendet werden sollen.

3 Praktische Durchführung

Dieses Kapitel beschreibt die praktischen Umsetzungen, die notwendig sind, um das erstellte Szenario zu bearbeiten. Zunächst werden die verwendeten Technologien genauer beschrieben. Anschließend wird die Softwarestruktur einer ganzheitlichen Anwendung für das Szenario erläutert. Hierbei wird darauf geachtet, dass die Anwendung in Komponenten aufgeteilt ist, sodass Teilprobleme weitestgehend unabhängig voneinander bearbeitet werden können. Dies vereinfacht auch das Testen der einzelnen Komponenten sowie potenzielle Aktualisierungen.

Passend zu dem betrachteten Szenario wird ein Objekt als Prüfling gewählt und prototypisch gedruckt. Aufgrund der Erkenntnisse aus 2.3.2 soll das Erkennen des Objektes anhand eines CNNs geschehen. Die anschließende Erkennung der Orientierung wird mithilfe von 3D-Modellen, Tiefeninformationen von der HoloLens und den RANSAC- und ICP-Algorithmen durchgeführt.

Für das Training des CNNs auf die Erkennung der Ansichten wird ein Datensatz an Bildern des zu erkennenden Objektes benötigt. Die RANSAC- und ICP-Algorithmen brauchen ein Modell des Objektes sowie Tiefeninformationen des betrachteten Objektes im Raum. Der genaue Vorgang und die Umsetzung beider Verfahren sowie die Erstellung eines nötigen Datensatzes werden im Folgenden in eigenen Kapiteln beschrieben.

3.1 Beschreibung verwendeter Hard- und Softwaretechnologien

3.1.1 Microsoft HoloLens

Alle Angaben zur Technologie der HoloLens beziehen sich auf die hier in der Arbeit verwendete aktuelle Version der HoloLens. Es ist durchaus denkbar, dass zukünftige Versionen anders aufgebaut sind oder andere Technologien verwenden, wodurch aktuelle Probleme möglicherweise behoben werden.

Die HoloLens ist ein AR-Headset, entwickelt und vertrieben von Microsoft [28]. Sie ist komplett kabellos einsetzbar, da sie einen eingebauten Computer besitzt, auf dem ein für die HoloLens spezialisiertes Windows namens Windows Holographic als Betriebssystem läuft.

Die HoloLens betreibt von sich aus VIO-Tracking mithilfe der verbauten Kameras und Sensoren. Neben normalen Farbkameras sind auch zwei Tiefenkameras für die Erkennung von nahen und fernen Tiefendaten vorhanden, sowie Infrarotkameras zum Erkennen von Reflektivität. Die HoloLens berechnet auf Basis der Daten, die durch diese Kameras erstellt werden, selbstständig kontinuierlich ein Modell der Umgebung, das über die Weboberfläche heruntergeladen werden kann [29]. Die HoloLens besitzt eingebaute Lautsprecher und ein Mikrofon für Sprachaus- und eingabe, die für die Benutzerinteraktion verwendet werden können.

Neben einer CPU und GPU, wie sie in alltäglichen Computern zur Berechnung und Darstellung von Aufgaben verbaut sind, hat die HoloLens zusätzliche Application-Specific Integrated Circuits (ASIC), die von Microsoft Holographic Processing Units (HPU) genannt werden und durch das Übernehmen von spezialisierten Prozessen die CPU und GPU entlasten [28].

Um auf bestimmte Funktionen der HoloLens von einem externen Rechner zuzugreifen, läuft auf der HoloLens permanent ein Web Server, der das Windows Device Portal zur Verfügung stellt [29]. Auf dieses kann dann, entweder durch eine USB-Verbindung zwischen einem PC und der HoloLens oder durch einen Internetzugriff auf die IP-Adresse der HoloLens, zugegriffen werden. Dies ermöglicht das Fernsteuern von bestimmten Funktionen der HoloLens, wie das Installieren von Applikationen, das Aktivieren des Research Modes oder das Herunterladen von Umgebungsmodellen.

Im Windows 10 April 2018 Update wurde der Research Mode für die HoloLens eingeführt [30]. Die größte Ergänzung durch das Update ist die Möglichkeit, direkt auf die Sensordaten der HoloLens zuzugreifen. Diese Sensordaten kommen von vier Kameras, die für das Tracking der Umgebung verwendet werden, sowie zwei Tiefenkameras. Eine der Tiefenkameras wird benutzt für das Tracking von nahen Tiefendaten, wie beispielsweise Hände zur Gestensteuerung. Die andere ist für ferne Daten zuständig, die für Spatial Mapping verwendet werden. Des Weiteren sind zwei Infrarot-Reflektivitätskameras, die ebenfalls von der HoloLens zur Berechnung von Tiefeninformationen genutzt werden, vorhanden.

Seit Mai 2018 gibt es eine grundlegende Dokumentation und Code-Beispiele zur Verwendung der Sensordaten. Entsprechend wird in dieser Arbeit die Verwendung dieser zur Bewältigung des beschriebenen Problems evaluiert [30].

3.1.2 TensorFlow

TensorFlow [31] ist eine Open Source Bibliothek, die für numerische Berechnungen verwendet wird und dabei hohe Leistungen erreicht. Dies ist möglich, da TensorFlow Berechnungen in Datenflussgraphen repräsentiert, die effizient auf spezielle Hardware zugeschnitten werden können. Beispiele für solche Hardware sind GPUs oder die speziell von Google für ML entwickelten Tensor Processing Units (TPUs) [32]. Sie hat eine gute

Unterstützung für viele Abstraktionslayer für neuronale Netze und bietet viele Optimierungsfunktionen. Außerdem bietet TensorFlow ein großes bestehendes Ökosystem mit verschiedenen Werkzeugen von Erstellung bis Auswertung von NNs. Deshalb wird TensorFlow im Rahmen dieser Arbeit zur Implementierung und Anwendung für die Klassifikation von Objekten eingesetzt.

3.1.3 Unity 3D Engine

Unity ist eine plattformübergreifende 3D-Entwicklungsumgebung zur Entwicklung von 2D- und 3D-Anwendungen sowie Videospielen. Sie unterstützt zum Zeitpunkt der Erstellung dieser Arbeit 27 Plattformen, zu welchen auch die gängigen AR-Plattformen wie Microsoft HoloLens, ARCore, ARKit und Vuforia zählen[13].

In dieser Arbeit wird Unity zur Entwicklung von Anwendungen für die HoloLens verwendet. Für grundlegende Funktionen wie die Spatial Perception oder Eingabemethoden wie Gesten- oder Spracheingaben wird das MRTK benutzt. Für spezielle Funktionen müssen sogenannte Capabilities[14] aktiviert werden. So wird für den Zugriff auf das Internet die Capability `internet (client/server)` gebraucht. Für den Zugriff auf die Kamera der HoloLens wird die `webcam`-Capability benötigt und um auf die Scans der Umgebung zuzugreifen die `spatial perception`-Capability.

Die genannten Capabilities sind alle öffentlich und können über die Einstellungen innerhalb eines Unity-Projektes aktiviert oder deaktiviert werden. Es gibt auch nicht öffentliche Capabilities, sogenannte `restricted capabilities`[15]. Diese müssen durch Eintragen von Zeilen in eine Projektkonfigurationsdatei aktiviert werden.

3.2 Softwarestruktur

Die HoloLens ist zwar ein Rechner, jedoch ist die Version des Windows 10 Betriebssystems, das auf dieser läuft, begrenzt. Beispielsweise kann nicht auf alle Daten im Dateisystem zugegriffen werden und nicht jedes Programm, das mit einem normalen Windows 10 kompatibel ist, kann frei installiert werden. So ist man begrenzt auf die Anwendungen, die im Windows Store verfügbar sind, sowie diejenigen, die man selbst für die HoloLens entwickelt.

---

[13] Informationen zu Unity unter: https://unity3d.com/unity Stand: 1 November 2018.

[14] Informationen zu Capabilities: https://docs.microsoft.com/en-us/windows/mixed-reality/unity-development-overview

[15] Informationen zu restricted capabilities unter: https://docs.microsoft.com/de-de/windows/uwp/packaging/app-capability-declarations

Daher wird in den folgenden Kapiteln evaluiert, welche Programme und Funktionen bereits erhältlich sind und wie diese genutzt werden können, sowie welche zusätzlich erstellt werden müssen. Zunächst jedoch wird die Gesamtstruktur der zu erstellenden Software erläutert. Die Software ist unterteilt in verschiedene Komponenten, die miteinander interagieren.

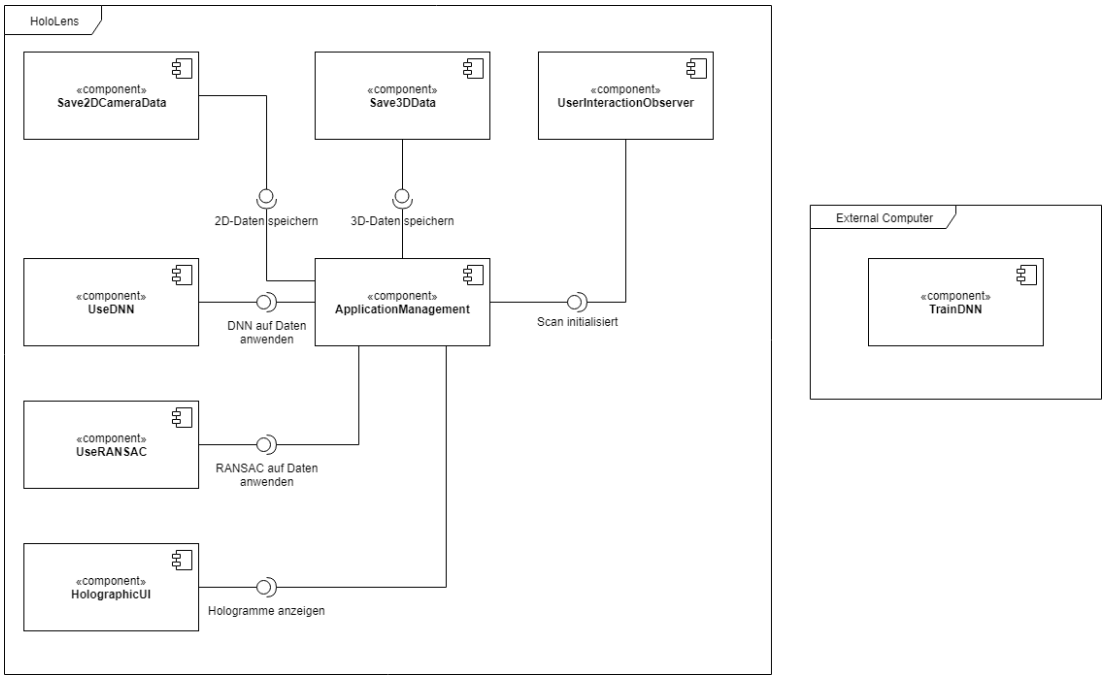

Abbildung 3.1: Komponentendiagramm

Der Großteil der Anwendung soll direkt auf der HoloLens laufen. Lediglich die Komponente zum Trainieren des NN ist ausgelagert auf einen externen Rechner, da hierfür viel Rechenleistung gebraucht wird, welche die HoloLens nicht aufbringen kann. Für alle HoloLens-Anwendungen wird das HoloToolkit benutzt. Dieses abstrahiert verschiedene grundlegende Funktionen der HoloLens und bildet somit eine API, um diese zu verwenden.

Die Kernkomponente der Anwendung ist die `ApplicationManagement`-Komponente. Von dieser werden die anderen Komponenten angesprochen. Sie ist es auch, die durch die `UserInteractionObserver`-Komponente benachrichtigt wird, wenn der Benutzer eine Interaktion wie z. B. eine Geste oder einen Sprachbefehl durchführt. Weitere Komponenten sind die `Save2DCameraData`– und `Save3DData`-Komponenten, welche die Speicherung der Daten der Kamera und der Tiefenkamera übernehmen. Außerdem existieren noch die `UseDNN`- und `UseRANSAC`-Komponenten für das Anwenden des NN auf die Daten der 2D-Kamera und des RANSAC und ICP auf die 3D-Daten der Tiefenkamera. Als letzte Komponente innerhalb der HoloLens gibt es weiterhin die `HolographicUI`-Komponente zur Darstellung der Hologramme für den Benutzer.

3.3 Erstellen des Datensatzes

Dieses Kapitel beschreibt die Erstellung der für die Experimente benötigten Daten. Hierfür wird zunächst die Auswahl und Erstellung des verwendeten Prüflings erklärt. Außerdem wird beschrieben, wie die Bilder zum Training des CNNs aufgenommen und verbessert wurden. Anschließend werden die benötigten Schritte zur Generierung von Tiefendaten erläutert.

Vor dem Ansatz der Verwendung der rohen Sensordaten der HoloLens wurde ein anderer verfolgt. Die HoloLens scannt kontinuierlich die Umgebung, um daraus ein 3D-Modell zu generieren. Dieses Modell lässt sich dann über die Weboberfläche herunterladen. Die Qualität des Scans ist jedoch nicht über diese einstellbar, kann aber verändert werden, wenn eine Anwendung auf der HoloLens dies einstellt. Hierfür muss in dieser zunächst in Unity die `SpatialPerception` Capability aktiviert werden. Anschließend kann durch die Anwendung auf die Funktionen der HoloLens zugegriffen werden. Um nun die Generierung der Raummodelle zu beeinflussen, muss anschließend zu der `SpatialMappingObserver.cs` aus dem HoloToolkit eine Zeile hinzugefügt werden, welche die Variable `TrianglesPerCubicMeter` setzt. Der Wert kann auf bis zu 2000 hochgesetzt werden.

Die Qualität der Modelle lässt jedoch selbst bei den höchsten Einstellungen zu wünschen übrig. So kann dasselbe Objekt aus derselben oder zumindest sehr ähnlichen Richtung unterschiedlich aussehen und kleine Gegenstände können oft nicht als solche erkannt werden. Ein weiteres Problem, das während der Evaluierung der Modelle aufgefallen ist, ist das Automatisieren der Erstellung dieser. Für diesen Teil des Device Portals ist keine API spezifiziert worden[16]. Dies bedeutet, dass die Modelle nur durch das Simulieren eines Webbrowsers automatisiert heruntergeladen werden können. Aus diesen Gründen wurde dieser Ansatz nicht tiefgehender verfolgt.

3.3.1 Erstellen der Bilddaten

Zur Erstellung der 2D-Bilddaten können sogenannte `MediaStreams` verwendet werden. Diese werden im Detail im Teilabschnitt 3.3.2 behandelt. Hierzu muss lediglich der entsprechende Stream der Kamera gewechselt werden. Für diese Bilddaten ist außerdem keine Konvertierung notwendig.

Der grundlegende Versuchsaufbau zur Erstellung der Bilder ist zunächst unabhängig vom tatsächlich verwendeten Prüfling. Letztendlich soll das Objekt auf einer flachen Oberfläche, wie einem Boden oder einer Tischplatte, platziert werden. Nun müssen aus den zuvor festgelegten Ansichten Aufnahmen vom Objekt gemacht werden. Hierbei ist darauf zu achten, dass die Entfernung zum Objekt, nicht aber der Winkel pro Ansicht, variiert werden sollte.

[16] GitHub-Issue zum Thema: https://github.com/MicrosoftDocs/mixed-reality/issues/500

Bei dem verwendeten Prüfling handelt es sich um das Modell eines Ford-Motorblocks, das online unter der Creative Commons – Attribution – Share Alike Lizenz veröffentlicht wurde[17]. Dieses wurde mit einem Ultimaker 3 Extended gedruckt. Als Material wurde ein Polyactid (PLA) verwendet, sowie ein Polyvinylalkohol (PVA) für wasserlösliche Stützstrukturen. Das gedruckte Objekt ist ungefähr 16,8 cm x 13,8 cm x 10 cm groß.

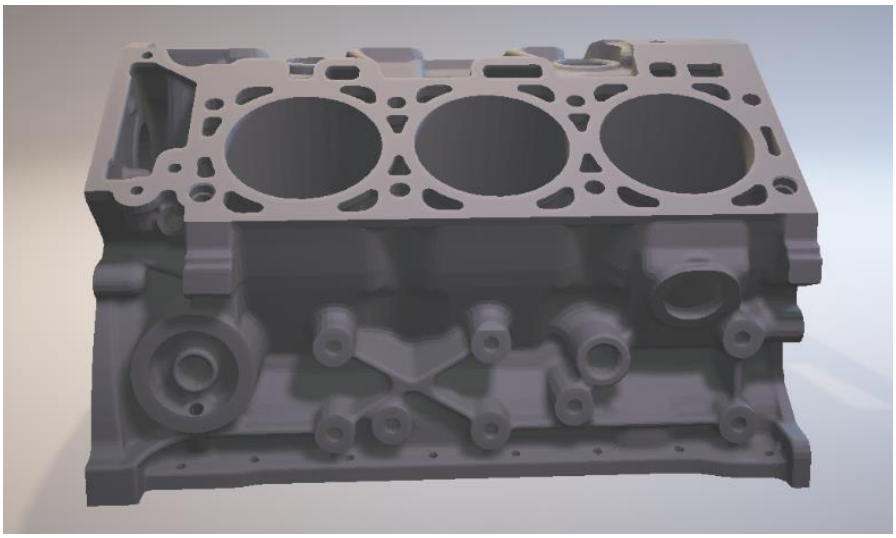

Abbildung 3.2: Modell eines Ford Motorblocks

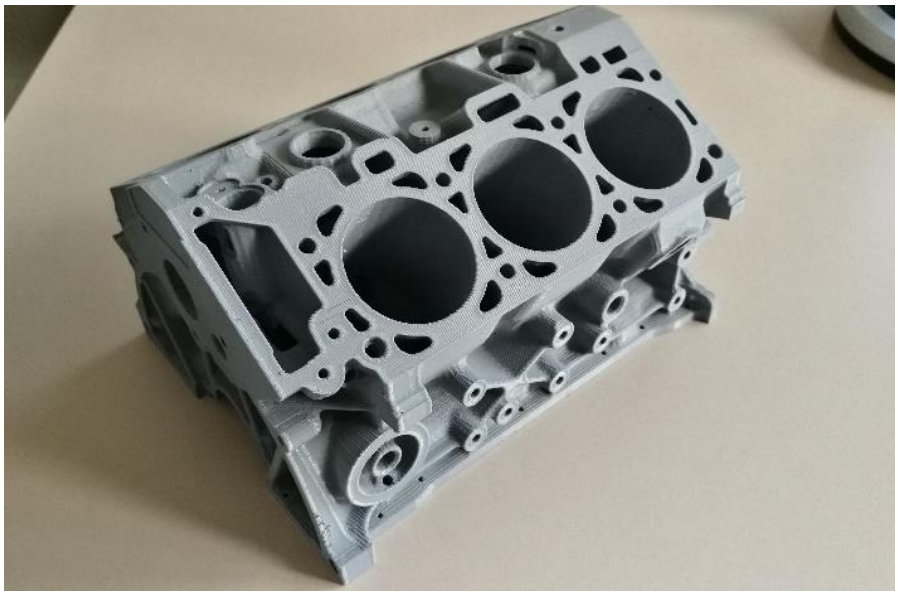

Abbildung 3.3: Gedrucktes Modell

[17] Das verwendete Modell ist zu finden unter: https://www.thingiverse.com/thing:40257

Zur Erstellung der charakteristischen Ansichten des Motors wurde dieser mehrfach aus acht verschiedenen Perspektiven unter unterschiedlichen Lichtbedingungen fotografiert. Hierbei wurde darauf geachtet, dass die Entfernung zwischen der HoloLens und dem Modell gleich bleibt. Auf diese Weise wurden pro Seite ungefähr 150 Bilder erstellt.

Auf Basis dieser Aufnahmen wurden anschließend mithilfe der Open Source Computer Vision Bibliothek OpenCV [33] weitere Bilder generiert. Hierfür wurden die aufgenommenen Fotos verändert und die veränderten Bilder gespeichert.

Zum einen wurde mit Skalierung, Verschiebung und Weichzeichnern gearbeitet, um so verschiedene Entfernungen zum Objekt zu simulieren. Zum anderen wurde zufälliges Rauschen über die Bilder gelegt, um dadurch das Trainieren basierend auf wiederholende Hintergrundmerkmale zu erschweren. Auch wurde die Helligkeit verändert, um noch weitere zusätzliche Lichtbedingungen zu simulieren. Aus diesen verschiedenen Techniken wurden zufällig einige ausgewählt und auf die aufgenommenen Fotos angewendet. Diese Data-Augmentation-Techniken wurden alle unter Beachtung der Annahmen aus 2.5 gewählt. So werden für das erstmalige Training des Netzes 10.000 Bilder pro Ansicht generiert. Der Datensatz für das Training besteht also aus 80.000 Bildern.

3.3.2 Extrahieren der Sensordaten

Es gibt die Möglichkeit, direkt die Sensordaten der HoloLens zu verwenden. Die Dokumentation des Codes wurde von Microsoft in einem Repository zur Verfügung gestellt. Diese ist jedoch nicht vollständig nachzuvollziehen, weshalb viele der benötigten Schritte mithilfe von Artikeln oder Blog-Einträgen Dritter[18] [19] besser verständlich sind.

Für den Zugriff auf die Daten muss mit Windows `MediaStreams` gearbeitet werden. Die HoloLens verfügt über viele unterschiedliche `MediaStreams`. So gibt es Streams für alle eingebauten Kameras, die z. T. verschiedene Auflösungen und Bildwiederholungsraten unterstützen.

Damit die Initialisierung der `MediaStreams` funktioniert, muss zunächst eine weitere Capability zur Anwendung hinzugefügt werden. Diese war aber zum Zeitpunkt des Bearbeitens dieser Arbeit nicht in der Dokumentation von Microsoft aufgeführt. Da jedoch alle Capabilities auch als Registry Keys auf einem Windows System vorliegen, kann innerhalb

[18] M. Taulty: “Experimenting with Research Mode and Sensor Streams on HoloLens Redstone 4 Preview” zu finden unter: https://mtaulty.com/2018/04/06/experimenting-with-research-mode-and-sensor-streams-on-hololens-redstone-4-preview/

[19] M. Taulty: “Sketchy Experiments with HoloLens, Facial Tracking, Research Mode, RGB Streams, Depth Streams.,” zu finden unter: https://mtaulty.com/2018/06/19/sketchy-experiments-with-hololens-facial-tracking-research-mode-rgb-streams-depth-streams/

dieser nachgeschaut werden, welche Capabilities vorhanden sind. So kann unter den Restricted Capabilities `perceptionSensorsExperimental` gefunden werden. Nachdem diese zur Anwendung hinzugefügt wurde, kann also die Initialisierung der `MediaStreams` beendet werden.

Nach Auswahl des korrekten Streams für die Tiefenkamera können die einzelnen Bilder, die von der Kamera aufgenommen werden, gespeichert werden. Diese Aufnahmen sind jedoch in ihrer grundlegenden Form von wenig Nutzen, da sie nahezu komplett Schwarz sind. Um dieses Bild nun in eine Form zu konvertieren, mit der einfacher zu arbeiten ist, hat Microsoft in ihrem Repository ein Beispiel. Dieses befindet sich jedoch in einer reinen C++-Anwendung, die direkt auf der HoloLens läuft. Da aber in dieser Arbeit mit Unity und damit mit einer C#-Anwendung gearbeitet wird, sind einige Anpassungen notwendig, damit der Code von Microsoft funktioniert.

Zunächst wurde versucht, den C++-Code soweit wie möglich zu übernehmen. Dies stellte sich jedoch als Problem heraus, da bei einigen der Anweisungen, die direkt auf Pointern im Speicher der HoloLens arbeiten, Fehler geworfen wurden. Dies war auf fehlende Berechtigungen für den Zugriff auf den Speicher zurückzuführen. Entsprechend musste der Code von Microsoft so umgeschrieben werden, dass keine Pointerarithmetik mehr verwendet wurde. Nach der erfolgreichen Durchführung sind die erhaltenen Bilder in einer auswertbaren Falschfarbendarstellung vorhanden.

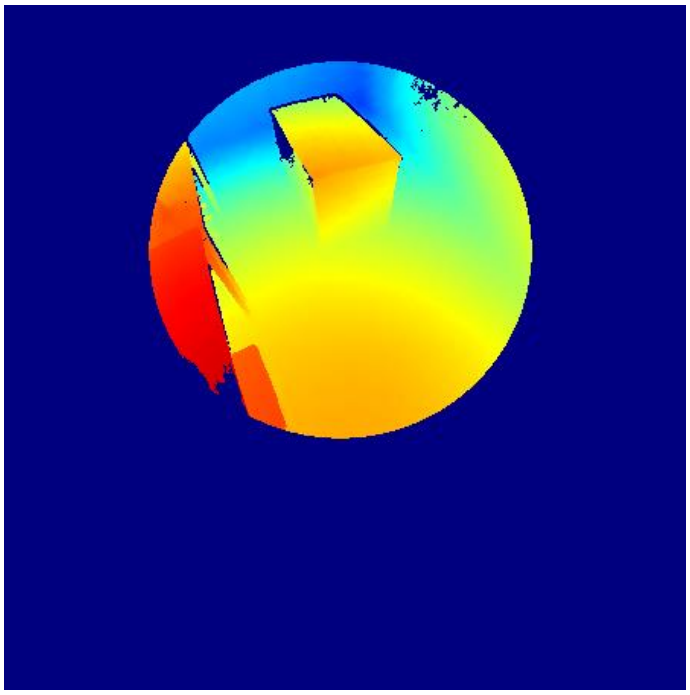

Abbildung 3.4: Konvertiertes Tiefenbild

Diese Bilder müssen noch gespeichert werden. Zur Speicherung der Daten auf der HoloLens müssen je nach Speicherort verschiedene Capabilities aktiviert werden, damit die Anwendung berechtigt ist, auf diese Orte zuzugreifen. In dieser Arbeit wurden die Bilder in der Bilder-Bibliothek auf der HoloLens gespeichert. Auf diese kann einfach von einem externen Rechner zugegriffen werden und so können die Daten beispielsweise auch in Batches übertragen werden.

Nachdem alle diese notwendigen Schritte vollzogen wurden, können nun also alle Daten direkt von der HoloLens erzeugt werden. Der nächste Schritt ist damit das Erstellen von Massendaten, um mit diesen alle benötigten Modelle zu trainieren.

3.3.3 Konvertierung der 3D-Daten

Um den ICP-Algorithmus auf die Daten anzuwenden, werden ein Datensatz sowie ein vorhandenes Modell benötigt. Daher werden die von der HoloLens erstellten Tiefenbilder verwendet, um einen Datensatz zu erhalten.

Die HoloLens hat zwei verschiedene Tiefensensoren, einen für nahe Tiefendaten bis zu 0,8 m und einen für weiter entfernte zwischen 0,8 und 4 m. Die Auflösung des fernen Tiefensensors hat sich im Praxistest als zu gering herausgestellt, um sie für die Erkennung des verwendeten Modells zu benutzen. Daher werden für die weiteren Experimente die Daten des nahen Sensors verwendet.

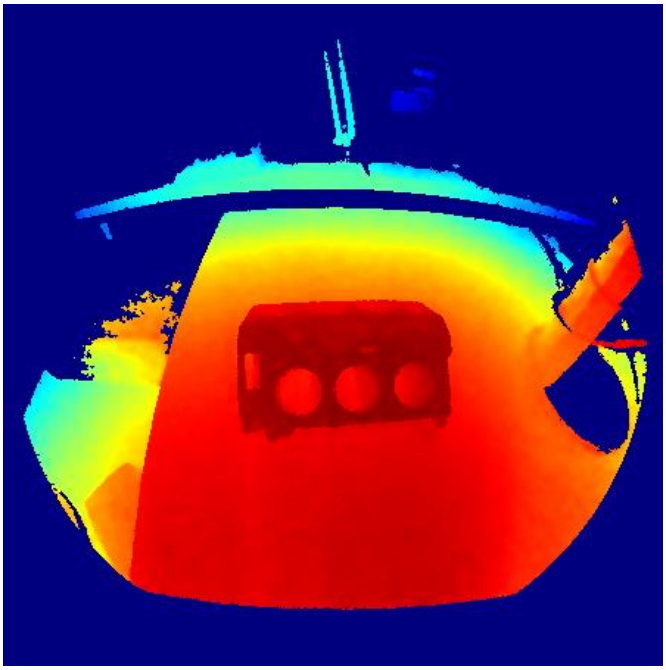

Abbildung 3.5: Aufnahme der Tiefendaten des nahen Sensors

Sowohl die Tiefendaten als auch das Motormodell müssen in das gleiche Format gebracht werden. Hierfür wird das Point Cloud Data (PCD) Format verwendet, das 3D-Daten als Punktwolken repräsentiert. Das PCD-Format ist Teil der Point Cloud Library (PCL) [34], die für die Anwendung der in dieser Arbeit benutzten Algorithmen verwendet wird. Für die Konvertierung der STL-Datei des Motormodells in das PCD-Format hat die PCL ein eigenes Programm. Die aufgenommenen Tiefendaten der HoloLens müssen jedoch auf eine andere Weise konvertiert werden. Dies geschieht durch Verwendung von Trigonometrie [35].

Die Daten, die von den Sensoren der HoloLens zurückkommen, liegen in Form von Ganzzahlen zwischen 0 und 4095 vor. Diese repräsentieren die Entfernung zur HoloLens, letztendlich in der Punktwolke also die Z-Koordinate oder Tiefe. Die X- und Y-Koordinaten müssen berechnet werden. Die Berechnungen werden durch die folgenden zwei Abbildungen besser veranschaulicht.

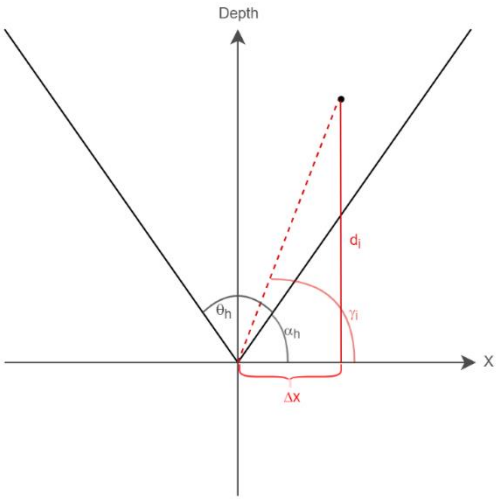

Abbildung 3.6: Berechnung der X-Koordinate

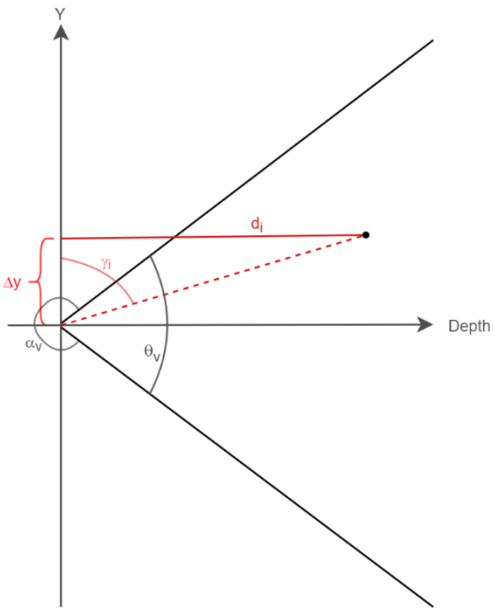

Abbildung 3.7: Berechnung der Y-Koordinate

Die X-Koordinate lässt sich also wie folgt berechnen:

$$\tan \gamma_i = \frac{d_i}{\Delta x} \rightarrow \Delta x = \frac{d_i}{\tan \gamma_i}.$$

d_i entspricht dem Wert, der von dem Tiefensensor für den betrachteten Punkt i zurückgegeben wurde [35]. Für die Berechnung des Winkels γ_i wird angenommen, dass jede Spalte der Pixelmatrix einen bestimmten Winkel innerhalb des horizontalen Blickfelds des Sensors repräsentiert. Der Winkel, der das gesamte horizontale Blickfeld repräsentiert, wird

hier als θ_h bezeichnet und beträgt für den Tiefensensor der HoloLens 120°. Für Berechnungen muss dieser Wert in Radianten umgerechnet werden, was für diesen θ_h ungefähr 2,0944 rad entspricht. [35] Die Spalte des betrachteten Pixels i wird als c_i bezeichnet, die Anzahl aller Spalten als n_c und der erste Winkel als α_h. Somit erhalten wir:

$$\gamma_i = \alpha_h + \frac{c_i \theta_h}{n_c}.$$

α_h ist eine Konstante, die sich wie folgt berechnen lässt:

$$\alpha_h = \frac{\pi - \theta_h}{2}.$$

Die Berechnung der Y-Koordinate wird ähnlich zur X-Koordinate durchgeführt:

$$\tan \gamma_i = \frac{d_i}{\Delta \mathrm{y}} \rightarrow \Delta y = \frac{d_i}{\tan \gamma_i},$$

wobei für die Berechnung von γ_i in diesem Fall die Reihe des Pixels i betrachtet wird, die mit r_i bezeichnet wird. Auch wird dementsprechend die Anzahl aller Reihen n_r statt der Spalten benötigt. Ebenso wird für die Y-Koordinate das vertikale Blickfeld θ_v der Sensoren verwendet. Der Wert für das vertikale und horizontale Blickfeld des Tiefensensors ist derselbe mit 120° [35]. Dementsprechend ergeben sich folgende Gleichungen für γ_i und α_v:

$$\gamma_i = \alpha_v + \frac{r_i \theta_v}{n_r},$$

$$\alpha_v = 2\pi - \frac{\theta_v}{2}.$$

Nach Implementierung dieser Gleichungen lassen sich die von der HoloLens erhaltenen Tiefendaten in Punktwolken umwandeln, sodass mit diesen weitergehend gearbeitet werden kann [35]. Da die Daten des nahen Sensors verwendet werden, können außerdem alle Werte um 4095 gefiltert werden, da diese entweder zu weit entfernt oder zu nah am Sensor sind. Hierdurch wird die Verarbeitungszeit der Tiefendaten reduziert. Abbildung 3.8 und 3.9 zeigen beispielhaft die Konvertierung in eine Punktwolke. Für die Anwendung der RANSAC- und ICP-Algorithmen werden Tiefendaten mit demselben Versuchsaufbau wie in 3.3.1 erstellt. Da beide Ansätze mit MediaStreams arbeiten, kann einfach zwischen der Aufnahme von Bilddaten und Tiefendaten gewechselt werden.

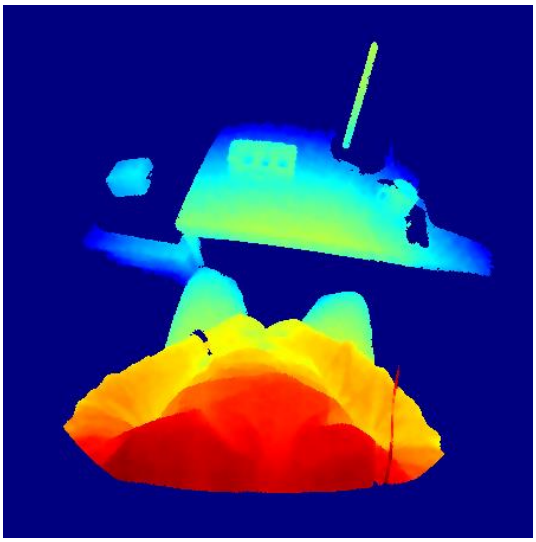

Abbildung 3.8: Tiefendaten des HoloLens-Sensors in Falschfarbendarstellung

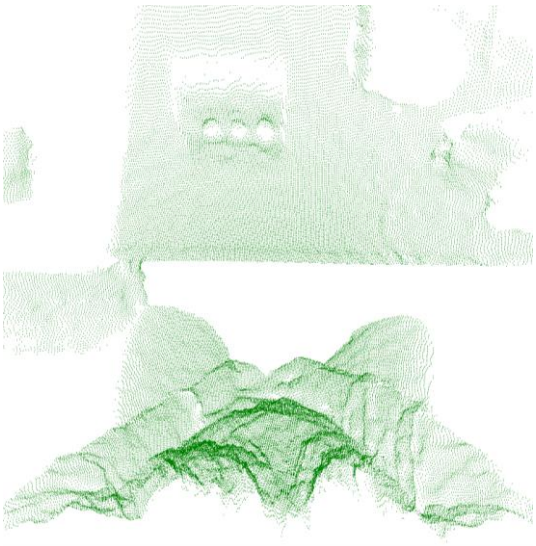

Abbildung 3.9: Konvertierte Punktwolke

3.4 Klassifikation des Objektes

Im Folgenden werden die benötigten Schritte zur Klassifikation des gewählten Modells in der Praxis erläutert.

3.4.1 Evaluierung von TensorFlow auf der HoloLens

Für die Klassifikation der charakteristischen Ansichten muss ein neuronales Netz trainiert werden. Hierfür wird TensorFlow als Software-Bibliothek verwendet. TensorFlow unterstützt nativ jedoch nur 64-Bit Systeme. Dies bedeutet, dass verschiedene Änderungen an Einstellungen vorgenommen werden müssen, damit das Kompilieren problemlos funktioniert und entsprechend TensorFlow auf der HoloLens ausgeführt werden kann[20].

Ein aufgetretenes Problem nach dem fehlerlosen Kompilieren und Erstellen der DLL-Datei für die 32-Bit TensorFlow Version ist, dass diese nicht erfolgreich auf die HoloLens gebracht werden konnte. Im Rahmen dieser Arbeit konnte kein Weg gefunden werden, die zur Anwendung von TensorFlow notwendige 32-Bit DLL auf die HoloLens zu kopieren. Daher wird nun im Folgenden die Option untersucht, die benötigten Daten von der HoloLens auf einen externen Rechner zu übertragen. Auf diesem können dann die TensorFlow-Berechnungen durchgeführt werden und. Die Ergebnisse werden anschließend wieder zurück an die

[20] Anleitung zum Kompilieren von Tensorflow in einer 32-bit Version: https://anyline.com/news/consume-tensorflow-net/

HoloLens gesendet. Aus diesem Grund sind folgende Änderungen am Komponentendiagramm aus Abbildung 3.1 notwendig.

Das Anwenden des DNN soll nun ebenfalls auf einen externen Rechner ausgelagert werden, was bedeutet, dass auch eine Form der Kommunikation zwischen diesem und der HoloLens erstellt werden muss. Daher wird auf der HoloLens eine `CommunicationClient`-Komponente hinzugefügt, während der externe Rechner die entsprechende Serverkomponente erhält.

Die HoloLens kann wie ein normaler PC auf das Internet zugreifen. Entsprechend kann sie auch beispielsweise REST-Schnittstellen aufrufen. Daher ist es möglich, auf einem externen Rechner einen Webserver laufen lassen und so die Daten, die auf diesem benötigt werden, zu übertragen. Zur Verwendung von REST-Schnittstellen gibt es auch bereits implementierte Lösungen für die HoloLens[21], weshalb dieser Teil nicht weitergehend vertieft wird.

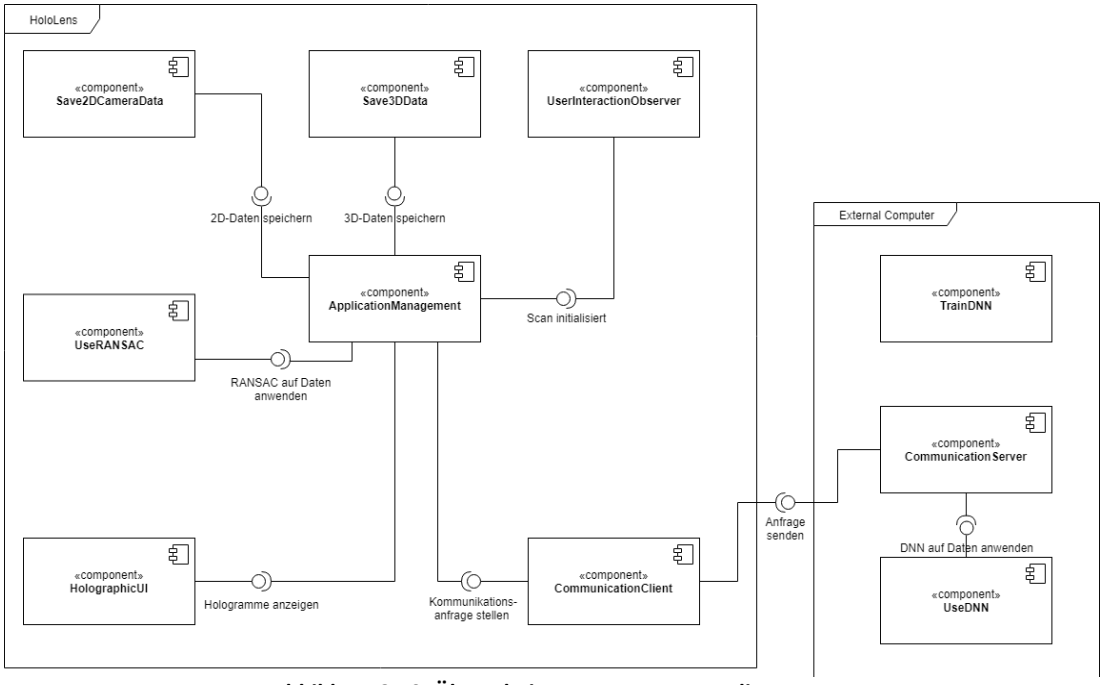

Abbildung 3.10: Überarbeitetes Komponentendiagramm

[21] Beispiel für eine vorhandenen REST-Client:
https://assetstore.unity.com/packages/tools/network/rest-client-for-unity-102501

3.4.2 Charakteristische Ansichten mit TensorFlow

Charakteristische Ansichten bezeichnen in diesem Kontext Bilder des Prüflings aus verschiedenen Perspektiven. Dies kann man sich so vorstellen, als wäre eine vielflächige geometrische Figur um den Prüfling gelegt, beispielsweise ein Ikosaeder. Eine charakteristische Ansicht des Prüflings ist dann der Blick auf das Objekt von einer der Seiten.

Es gibt aber auch Einschränkungen zu den Ansichten, die betrachtet werden müssen. So ist es z. B. nicht nötig, für die Unterseite des Modells eine charakteristische Ansicht zu erstellen, da im betrachteten Szenario das Erkennen dieser nicht notwendig ist.

Die Erstellung der Ansichten erfolgt in dieser Arbeit mithilfe einer der Frontkameras der HoloLens. Der Prüfling wird aus verschiedenen Perspektiven und aus unterschiedlichen Entfernungen aufgenommen. Die so erstellten Ansichten können dann entsprechend je nach Perspektive mit eigenen Namen oder anderen Arten der Identifikation, wie z. B. IDs, versehen werden. Anschließend wird ein NN dahingehend trainiert, dass diese Ansichten erkannt werden, sodass eine grobe Ersteinschätzung der Rotation des Objektes in Relation zur Kamera gemacht werden kann.

3.4.3 Training des Neuronalen Netzes

Nachdem die Daten erzeugt wurden, muss ein neuronales Netz trainiert werden, damit dieses erkennen kann, welche Ansicht des Objektes zu sehen ist. Jede der Ansichten erhält eine eindeutige ID, in diesem Fall eine Bezeichnung der jeweiligen Perspektive, aus der das Objekt betrachtet wurde. Da zunächst mit acht Ansichten gearbeitet wird, werden sie der Verständlichkeit wegen nach den Himmelsrichtungen benannt, z. B. `north` oder `south_east`. Die Bilder werden entsprechend zu den Ansichten zugeordnet. Anschließend wird ein Inception-V3 [21] vortrainiertes Netz auf die acht Ansichten trainiert.

Das Inception-V3-Netz hat 42 Schichten und weniger als 25 Millionen Parameter. Vortrainiert wurde das Netz auf ungefähr 1,2 Millionen Trainingsdaten, verteilt über 1000 Kategorien. Das Modell besteht aus einer Kombination von Convolution, MaxPool, AvgPool, Concat, Dropout, Fully Connected und Softmax Layern. Als Input nimmt es Bilder mit den Dimensionen 299 x 299 x 3. Die Bilder der HoloLens werden daher zunächst auf diese Größe skaliert. Der genaue Aufbau des Inception-V3 wird in Abbildung 3.11 dargestellt.

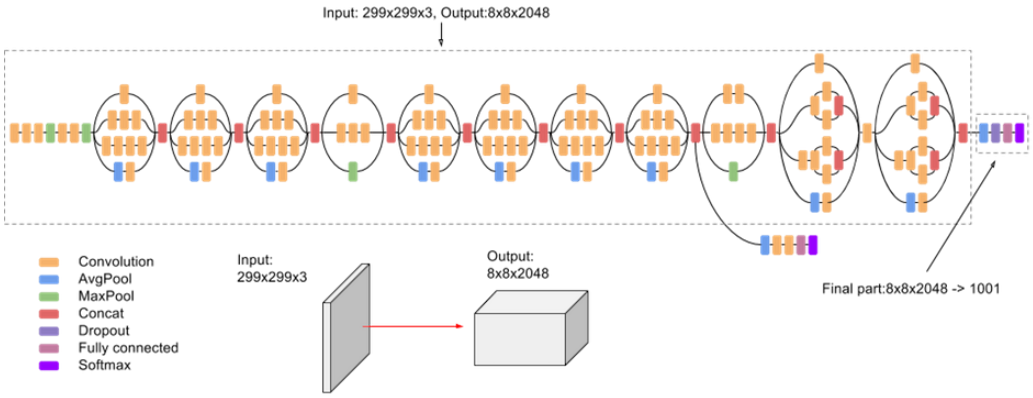

Abbildung 3.11: High-Level-Diagramm[22] des Inception-V3-Modells [21]

Es wurde ein vortrainiertes Netz gewählt, da für das Training von Grund auf wesentlich mehr Daten benötigt werden und der Rechenaufwand um ein Vielfaches erhöht wird. Beim Verwenden des vortrainierten Netzes wird der Teil der Merkmalsextraktion, der auf ImageNet-Daten [13] trainiert wurde, verwendet. Lediglich die finale Schicht für die Klassifikation der neuen Klassen wird trainiert. Diese Technik wird als Transfer Learning bezeichnet [36].

TensorFlow bietet mit dem TensorFlow Hub[23] eine Bibliothek für Entwickler, die ML-Modelle zur Verfügung stellt. Diese wurde für das Training in dieser Arbeit verwendet. Aus den verfügbaren Modellen[24] zur wurde das Inception-V3 ausgewählt, da es die höchste Genauigkeit in TensorFlows Benchmark[25] erreicht hat.

Der Datensatz der 80.000 erstellten Bilder wurde in 10 % Validierungs-, 10 % Test- und 80 % Trainingsdaten aufgeteilt. Diese Aufteilung wird gemacht, damit bereits während des Trainings gesehen werden kann, ob das Modell ein Overfitting betreibt. Das Training mithilfe des Stochastic Gradient Descent Optimizer (SGD) wurde nach 10.000 Schritten beendet, da es konvergiert hat. Es wurden die Standard-Hyper-Parameter der Implementation verwendet. Ausgenommen hiervon sind die Batch-Size und Lernrate. Diese wurden experimentell ermittelt und liegen für die Batch-Size bei 100 Bildern und für die Lernrate bei 0,01. Mit diesen Einstellungen dauerte das Training ungefähr 23 Minuten auf einem Rechner mit einer Nvidia Geforce GTX 1070 mit 8 GB GDDR5 Speicher, einem 8-Kern Ryzen R7 1700 und 32 GB DDR4 RAM.

---

[22] Abbildung entnommen aus: https://cloud.google.com/tpu/docs/inception-v3-advanced (aufgerufen am 3.5.2019)

[23] Zu finden unter: https://tfhub.dev/

[24] Liste der im TensorFlow Hub verfügbaren Modellen: https://tfhub.dev/s?module-type=image-feature-vector

[25] Benchmark unter: https://github.com/tensorflow/models/tree/master/research/slim

3.5 Objektausrichtung

In diesem Kapitel wird erläutert, wie die Erkennung der Orientierung des gewählten Modells praktisch umgesetzt wurde. Das CNN aus 3.4 erkennt zwar grob die Ausrichtung des Modells, jedoch nicht genau. Daher wird nun mit einem algorithmischen Ansatz versucht diese Orientierung genauer zu bestimmen. Die Ergebnisse der Klassifizierung können hier auch genutzt werden, um das digitale Modell mit einer groben Rotation zu initialisieren.

Für den Vergleich der Punktwolken des Modells und der Tiefendaten werden zwei verschiedene Algorithmen getestet. Sowohl der RANSAC-Algorithmus als auch der ICP-Algorithmus wurden in der Theorie bereits in Kapitel 2.3.3 und 2.3.4 beschrieben, weshalb dieser Abschnitt lediglich den praktischen Einsatz behandelt. Für die Implementierung des RANSAC-Algorithmus, der in dieser Arbeit verwendet wird, müssen die Punktwolken zuerst vorverarbeitet werden. Zunächst werden die Normalvektoren für die Oberflächen beider Punktwolken berechnet, damit anschließend die Merkmalsdeskriptoren für die Punkte berechnet werden können. Für die Erstellung der Normalvektoren wird die `pcl::NormalEstimationOMP`-Klasse[26] verwendet, für die Merkmalsdeskriptoren die `pcl::FPFHEstimationOMP`-Klasse[27].

Der RANSAC-Algorithmus ist als `pcl::SampleConsensusPrerejective`[28] in der PCL implementiert. Dieser hat die Eigenschaft, dass in der Schleife im RANSAC-Algorithmus ein zusätzlicher Schritt eingefügt wird, der Orientierungen, die wahrscheinlich falsch sind, überspringt. Die Anfälligkeit dieser Erkennung kann durch einen Parameter eingestellt werden, sodass beispielsweise eine Punktwolke an eine andere angepasst werden soll, die besonders viel Rauschen enthält. In diesem Fall kann diese Anfälligkeit niedriger gesetzt werden. Getestet wurde der Algorithmus in dieser Arbeit mit verschiedenen Einstellungen für diese Parameter.

Es können aber auch weitere Parameter gesetzt werden: Zum einen die Anzahl der maximalen Iterationen des RANSAC-Algorithmus und die Anzahl der Punkte, die für das Erkennen der Orientierung benötigt werden, was den Standardparametern des RANSAC-Algorithmus entspricht. Außerdem gibt es noch weitere Parameter, die in dieser Implementierung des RANSAC-Algorithmus festgelegt werden können. So kann ein Parameter so gesetzt werden, dass statt dem nächsten passenden Feature zufällig zwischen den besten N passenden Features gewählt werden kann. N entspricht hier dem gewählten

[26] Dokumentation der Klasse verfügbar unter:
http://docs.pointclouds.org/trunk/classpcl_1_1_normal_estimation_o_m_p.html
[27] Dokumentation der Klasse verfügbar unter:
http://docs.pointclouds.org/trunk/classpcl_1_1_f_p_f_h_estimation_o_m_p.html
[28] Dokumentation der Klasse verfügbar unter:
http://docs.pointclouds.org/trunk/classpcl_1_1_sample_consensus_prerejective.html

Parameter. Auch kann ein Parameter gesetzt werden, der einen Grenzwert festlegt, nach welchem ein Punkt der transformierten Punktwolke mit einem Punkt der Zielwolke übereinstimmt. Dieser Grenzwert entspricht der euklidischen Distanz zwischen beiden Punkten. Das heißt, solange die Distanz zwischen beiden Punkten geringer ist als der Grenzwert, werden sie als übereinstimmend angenommen. Der letzte zusätzliche Parameter, der gesetzt werden kann, ist der Anteil an Inliern, der vorhanden sein muss, damit eine Orientierung als korrekt angesehen wird.

Für die Implementierung des ICP-Algorithmus wird die `pcl::IterativeClosestPoint`[29]-Klasse verwendet, welche die beiden Wolken, die aneinander angeglichen werden sollen, als Eingaben nimmt. Außerdem können die Abschlusskriterien gesetzt werden. Diese sind die maximale Anzahl Iterationen, ein Epsilon, ein Maximum für Fehler sowie eine Distanz. Das Epsilon bezeichnet die minimale Transformation, die noch als Änderung gilt. Die Distanz beschreibt den maximalen Abstand, bis zu dem zwei Punkte als übereinstimmend angesehen werden. Diese wurden in der Durchführung mit verschiedenen Werten besetzt, haben aber das Ergebnis nur wenig beeinflusst.

[29] Dokumentation der Klasse zu finden unter:
http://docs.pointclouds.org/trunk/classpcl_1_1_iterative_closest_point.html

4 Auswertung

Im Folgenden wird eine Auswertung der praktischen Ergebnisse durchgeführt. Zunächst wird das trainierte CNN-Modell zur Klassifizierung der charakteristischen Ansichten hinsichtlich ihrer Genauigkeit und Übertragbarkeit auf die Realität ausgewertet. Anschließend werden die RANSAC- und ICP-Algorithmen zur genauen Bestimmung der Objektausrichtung in Hinblick auf ihre Aussagekraft geprüft. Für beide Auswertungen wird auch die Methodik beschrieben, die verwendet wurde, um auf die hier erzielten Ergebnisse zu kommen.

Zuletzt werden noch die Rahmenbedingungen der Durchführung ausgewertet. Es wird betrachtet, welche Probleme mit der verwendeten Soft- und Hardware aufgetreten sind und mögliche Ansätze genannt diese Umstände zu verbessern.

4.1 Klassifikation des Objektes

Das neuronale Netz lässt sich auf verschiedene Aspekte hin auswerten. Zum einen kann erfasst werden, wie präzise es in der Erkennung der Ansichten ist. Hierzu wurden zusätzliche Bilder nach der gleichen Methode, wie in 3.3.1 beschrieben, erstellt. Für jede Klasse sind so 1.000 neue Bilder vorhanden, also 8.000 insgesamt. Diese werden vom trainierten Netz klassifiziert. Anschließend wird überprüft, welche Bilder korrekt und welche inkorrekt klassifiziert wurden. Getestet wird auf einem Rechner mit einer Nvidia GeForce GTX 1070 mit 8 GB GDDR5 Grafikspeicher. Die durchschnittliche Zeit zur Klassifizierung eines Bildes liegt bei ungefähr 0,49 Sekunden.

Zur Überprüfung der Genauigkeit wird eine Confusion Matrix [37] (dt. Wahrheitsmatrix) verwendet. Diese stellt für alle Klassen die tatsächlichen Klassen der Bilder die vom Netz Vorhergesagten entgegen. Je höher die diagonalen Werte sind, desto mehr korrekte Zuordnungen wurden vom Netz getroffen. Korrekte Zuordnungen werden als True Positive bezeichnet. Werte, die in der Spalte der tatsächlichen Klasse liegen, aber nicht als solche klassifiziert wurden, werden False Negative genannt. False Positive sind entsprechend diejenigen, die nicht zu einer Klasse gehören, aber trotzdem dieser zugeordnet wurden. Eine Confusion Matrix kann auch für eine einzelne Klasse erstellt werden. Diese wird dann als 2-mal-2-Matrix dargestellt, wobei die Kategorien schlicht als „Klasse“ und „nicht Klasse“ bezeichnet werden. Hier kann man entsprechend noch die True Negatives sehen, also die Anzahl der Voraussagungen, die korrekt nicht die tatsächliche Klasse vorausgesagt haben.

Die gezeigte Confusion Matrix ist normalisiert. Aus Gründen der Veranschaulichung wurde ein logarithmischer Farbverlauf gewählt, sodass auch die geringen Werte in der Matrix auf einen Blick sichtbar sind. Im Anhang befindet sich zusätzlich Abbildung A.1, welche die Confusion Matrix mit den absoluten Werten darstellt. Die Matrizen sowie die Angaben zur Genauigkeit des Netzes wurden mithilfe der Scikit-learn [38] Software-Bibliothek erstellt.

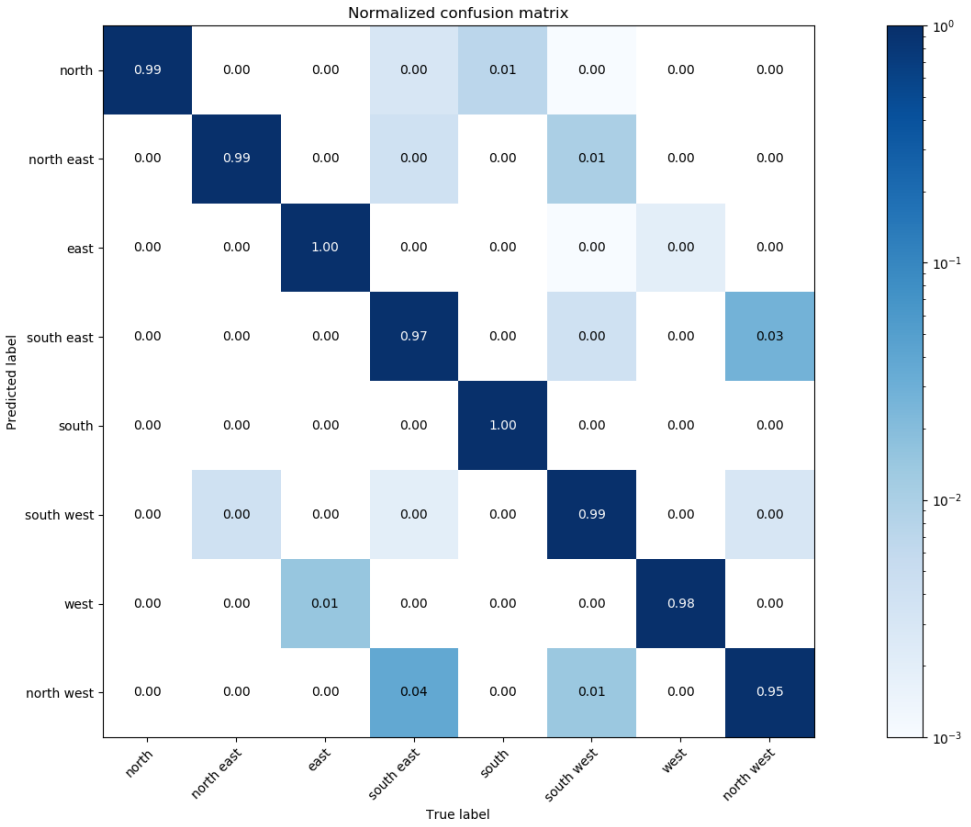

Abbildung 4.1: Confusion Matrix der getesteten Bilder (normalisiert)

In dieser Matrix sieht man, dass die Genauigkeit des Netzes für die Testdaten sehr hoch ist. Die meisten Fehleinschätzungen des Netzes gibt es bei gegenüberliegenden Seiten. Das lässt sich dadurch erklären, dass bei den jeweiligen Seiten die groben Strukturen des Motorblocks übereinstimmen. So unterscheidet sich eine breite Seite des Blocks von der anderen nur in Details, während sich die breiten Seiten von den schmalen schon im Seitenverhältnis voneinander abheben. Eines der Bilder, das z. B. mit `south` klassifiziert wurde, eigentlich aber in die Klasse `north` gehört, war so sehr von der Augmentation in Hinsicht auf Helligkeit und Gamma verändert worden, dass die Details auf der Seite nahezu nicht mehr zu erkennen sind.

Auf Basis dieser Matrix wurde auch die Genauigkeit der korrekten Zuordnungen für alle Klassen durch das Netz berechnet. Dieser Wert beträgt 0,983 oder 98,3 % auf den zusätzlich generierten Testdaten.

Diese Ergebnisse zeigen, dass das Modell gut für die Klassifizierung der charakteristischen Ansichten geeignet ist. Die benötigte Zeit von ungefähr einer halben Sekunde pro Bild ist kurz genug, als dass es im Kontext der Arbeit kein Problem darstellt, da diese Aufgabe nur einmal ausgeführt werden soll. Die Klassifikation kann in der Praxis z. B. parallel zur Konvertierung der Tiefendaten geschehen.

Ein zu beachtender Punkt ist, dass mit viel Data Augmentation gearbeitet wurde. Außerdem wurden nur acht fixe Perspektiven augmentiert. Unter Umständen ist das Modell in der Praxis nicht so genau, da hier andere Bedingungen herrschen. So werden Bilder z. B. aus verschiedenen Perspektiven mit großer Varianz aufgenommen. Dies müsste unter realen Bedingungen getestet werden. Sollte sich dies als Problem herausstellen, ist ein möglicher Lösungsansatz das Training des Modells mit mehr Daten aus diesen Perspektiven.

4.2 Objektausrichtung

Für die Auswertung der RANSAC- und ICP-Algorithmen wurden zunächst Aufnahmen der Tiefenkameras von dem Objekt gemacht. Der Prüfling wurde aus verschiedenen Richtungen und Entfernungen betrachtet. Es wurde darauf geachtet, dass er aber immer zentral im Blickfeld sowie innerhalb der vom Sensor unterstützten Reichweite positioniert wurde und komplett zu sehen war. Für die schriftliche Auswertung wurden drei unterschiedliche Punktwolken ausgewählt, welche die allgemeinen Ergebnisse abdecken. Neben den Abbildungen in diesem Kapitel finden sich noch weitere im Anhang.

Sowohl der RANSAC- als auch der ICP-Algorithmus wurden mit verschiedenen Werten für die in 3.5 genannten Parametern getestet. Ebenso wurde geprüft, wie sich die Ergebnisse ändern, wenn ein Downsampling vor den eigentlichen Berechnungen durchgeführt wird. Downsampling bezeichnet die Reduktion der Datenmenge zur Verbesserung der Bearbeitungsgeschwindigkeit. Hierbei bleibt die generelle Struktur der Wolke erhalten, Details gehen aber ggf. verloren. Die Ergebnisse zeigen, dass in bestimmten Fällen die Berechnung der Orientierung besser funktioniert als in anderen. Eine weitere Beobachtung ist, dass sich die Resultate des RANSAC-Algorithmus nicht als sinnvoll herausgestellt haben. Auch das Downsampling hat für die Qualität der Ergebnisse wenig Änderung gebracht. Die Bearbeitungsgeschwindigkeit hat sich dafür wesentlich (bis zu zehnmal schneller) verringert. Dies ist ein erstes Indiz dafür, dass die Qualität der Daten bereits vor dem Downsampling zu gering ist, um Details zu erkennen.

Der ICP ist durch seine iterative Natur anfällig dafür, auf lokale Minima für die Fehlerberechnung hinzuarbeiten, auch wenn global auf den gesamten Daten eine bessere

Lösung möglich wäre. Dies ist zu erkennen, wenn beispielsweise in der Nähe des anzupassenden Motormodells eine andere Erhöhung zu finden ist, wie in Abbildung 4.2 zu sehen. Der Algorithmus hat ein lokales Minimum an Fehlern erkannt und dahingehend das Modell orientiert. Dieses Problem ist bei allen Aufnahmen aufgetreten, bei denen in der Nähe der Kamera eine solche Erhöhung zu finden war. Im Rahmen der Arbeit konnten weitere Optimierungsversuche nicht mehr durchgeführt werden.

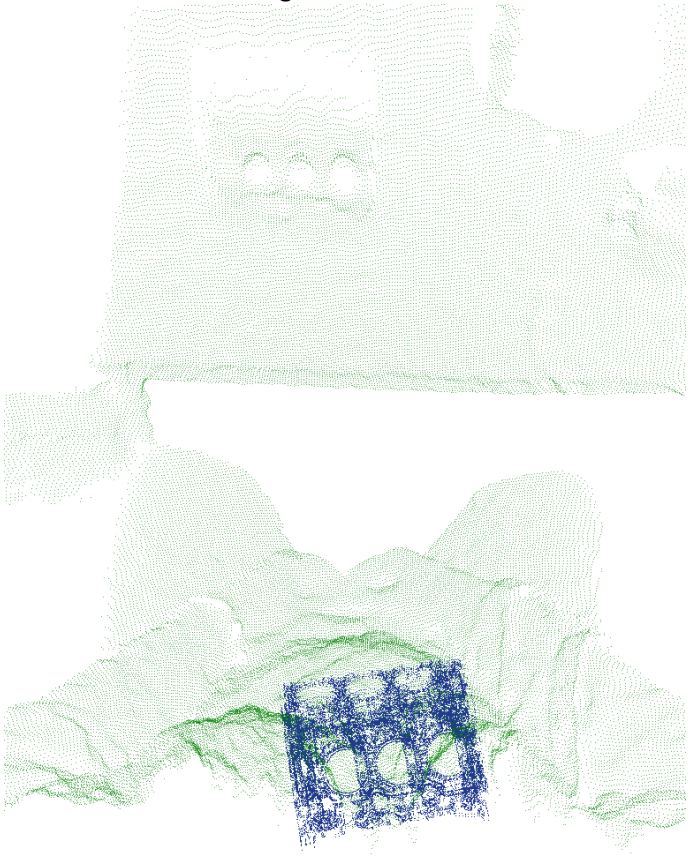

Abbildung 4.2: Punktwolke 1 (grün) und Modellwolke (blau) nach Anpassung von ICP

Aber auch der RANSAC-Algorithmus hat in diesem Fall ein falsches Ergebnis geliefert, wie in Abbildung A.2 im Anhang zu sehen, was sich wahrscheinlich auf die Qualität der Daten zurückführen lässt, insbesondere da der Datensatz, der von der HoloLens zurückgegeben wird, anteilig gesehen einen geringen Anteil Inlier und einen großen Anteil Outlier beinhaltet. Wird beispielsweise die Punktwolke in Abbildung 4.3 betrachtet, so ist zu sehen, dass der Teil der Punktwolke, der das Modell repräsentiert, natürlich nur von einer oder zwei Seiten (vorne und oben) wirklich gesehen werden kann, entsprechend fehlen die Tiefeninformationen der anderen Seiten, welche die Algorithmen verbessern könnten. Auch ist ein Großteil der Punktwolke nicht das eigentliche Objekt. Daher müssen u. U. sehr viele Iterationen durchlaufen werden, bis ein potenziell korrektes Ergebnis erreicht wird. Im

Rahmen dieser Arbeit waren 100.000 die höchste Anzahl Iterationen, die getestet wurden. Diese Durchführung hat bereits zweieinhalb Stunden gedauert und hat das Ergebnis aus Abbildung A.2 geliefert. Eine solche Ausführungsdauer ist aber nicht für Anwendungen im Kontext dieser Arbeit akzeptabel. Daher wurden keine höheren Werte getestet.

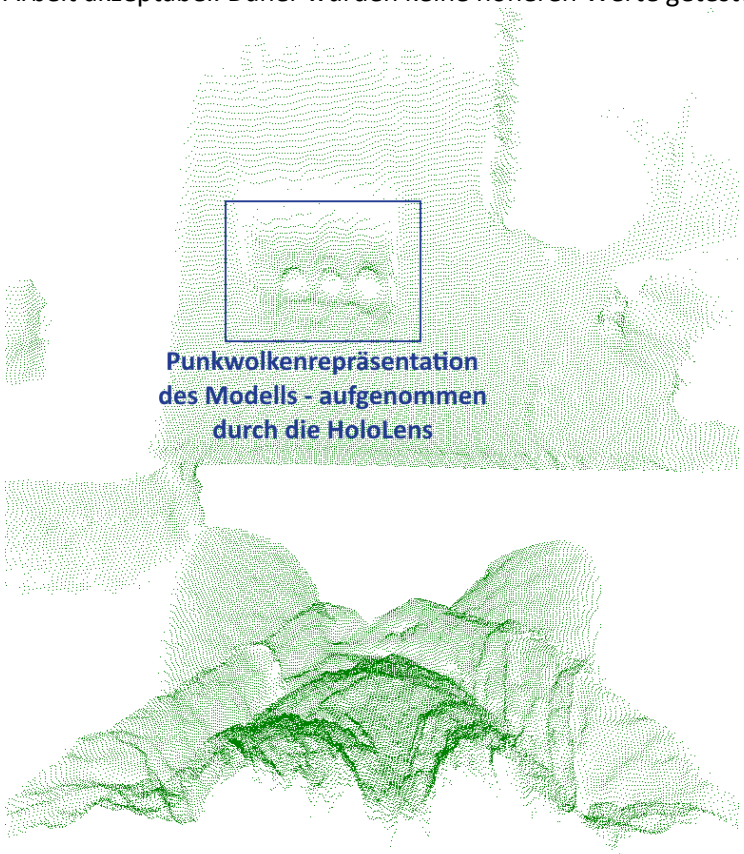

Abbildung 4.3: Veranschaulichung des Motors innerhalb der Punktwolke 1

Bei Verwendung von Tiefendaten, die näher am Objekt erstellt wurden, lässt sich erkennen, dass insbesondere der ICP-Algorithmus brauchbarere Ergebnisse liefert in Bezug auf die Rotation um die Z-Achse. So kann man in Abbildungen 4.4 und A.8 sehen, dass das orientierte Objekt an der korrekten Stelle platziert wurde, jedoch aufgrund der mangelnden Daten für die nicht sichtbaren Seiten des Modells innerhalb der aufgenommenen Daten falsch gekippt ist. Der RANSAC-Algorithmus hingegen ist zwar mit diesen Daten näher am tatsächlichen Objekt, ist aber weiterhin falsch orientiert. Auch wurde festgestellt, dass, wenn beide Punktwolken initial für die Anwendung der Algorithmen geladen werden, das Motormodell mit der Oberseite zur Punktwolke der aufgenommenen Daten steht. Dies hat dafür gesorgt, dass das Modell genau wie in dem in Abbildung 4.4 gezeigten Ergebnis orientiert wurde, jedoch die Unterseite nach oben und die Oberseite nach unten zeigte. Dies war bei allen getesteten Wolken der Fall. Deshalb wurde vor der Anwendung der Algorithmen eine initiale

Rotation um 180° um die Y-Achse hinzugefügt, was zu dem hier gezeigten Ergebnis geführt hat.

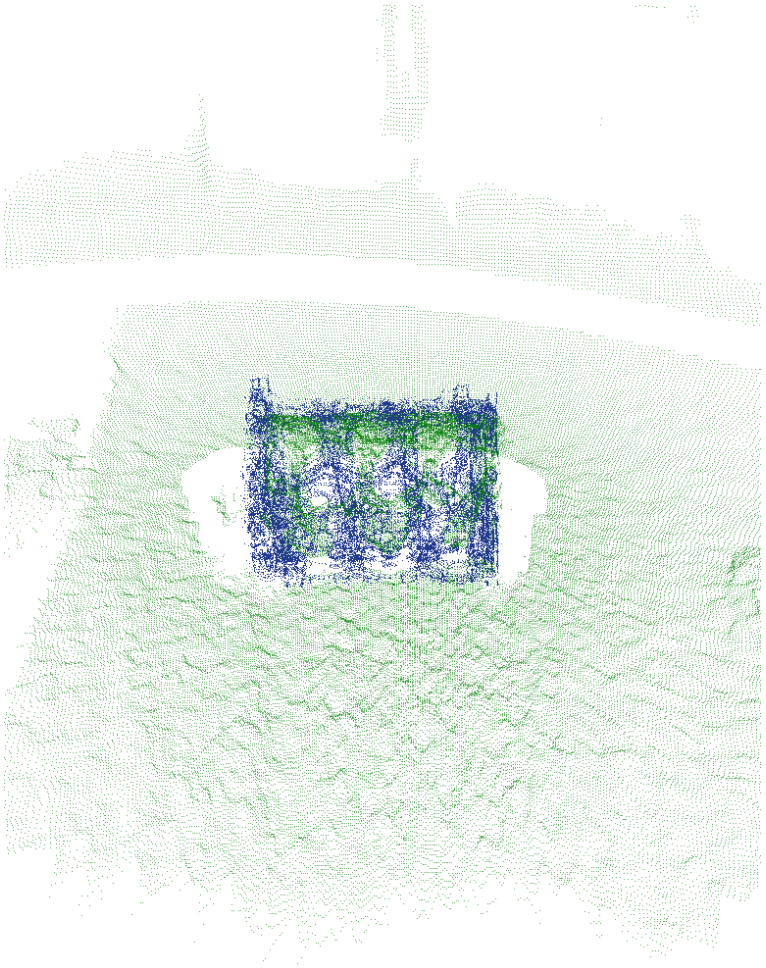

Abbildung 4.4: Punktwolke 2 (grün) und Modellwolke (blau) nach Anwendung von ICP

Ein weiterer aufgefallener Punkt ist, dass der ICP-Algorithmus keine Unterscheidung macht, welche der Seiten nach vorne zeigen. Dies bedeutet, dass das Ergebnis für die Transformationsmatrix, die verwendet wird, um das Modell durch den ICP-Algorithmus zu orientieren, annähernd gleich bleibt. Hierbei ist es egal, ob das Modell nicht initial rotiert ist oder um 180° gedreht, sodass die gegenüberliegende Seite nach vorne zeigt. Dies lässt sich auch wieder auf die niedrige Qualität der aufgenommenen Daten zurückführen. Gleichzeitig kann dadurch aber auch gesagt werden, dass die Vorerkennung der zugewandten Seite durch das neuronale Netz hierdurch noch wichtiger wird, da so die initiale Rotation, die auf das

Modell angewandt werden muss, bestimmt werden kann. Dies bestätigt noch einmal den Einsatz der in dieser Arbeit erarbeiteten Pipeline.

Ein weiterer betrachteter Fall war die Orientierung zu einer der schmalen Seiten des Objektes. Hierbei fällt auf, dass für den RANSAC-Algorithmus genau wie in den anderen Fällen das Ergebnis zu ungenau ist, daher wird auf die Bilder von diesem hier verzichtet. Der ICP-Algorithmus scheint mit den schmalen Seiten des Objektes aber ebenso Probleme zu haben, denn auch mit diesem ist das Ergebnis nicht verwendbar. Hier kann genauso vermutet werden, dass die Qualität der Punktwolken nicht gut genug ist, damit eine korrekte Anpassung vorgenommen werden kann.

Es lässt sich also sagen, dass die Verwendung der Tiefendaten der HoloLens in bestimmten Fällen, wie bei der Betrachtung der breiten Seite, ohne andere störende Objekte im Blickfeld funktioniert. Andere Fälle funktionieren wiederum nicht, wie z. B. die schmale Seite des Motorblocks. Diese Ergebnisse zeigen daher, dass die Erkennung der Orientierung mittels des in dieser Arbeit gewählten Ansatzes grundsätzlich funktioniert. Auch die Verbindung der Technologien zur Klassifikation durch ein NN und der Erkennung der Ausrichtung durch analytische Algorithmen stellte sich als sinnvoll heraus.

Unter Realbedingungen könnten die Ergebnisse sich noch verbessern, da ein realer Motorblock wesentlich größer ist als das hier verwendete Modell. Dadurch würden die Probleme der Auflösung der Sensoren geringer ausfallen. Auch wäre die Verwendung des fernen Tiefensensors nötig, um einen Motor mit realen Ausmaßen aufzunehmen. Dies würde bereits das Inlier-Outlier-Verhältnis verändern, sodass wesentlich mehr Daten des Objektes zur Verfügung stünden.

4.3 Rahmenbedingungen der Durchführung

Während der Durchführung der Experimente gab es verschiedene Rahmenbedingungen, die ebenfalls ausgewertet werden müssen. Allgemein lässt sich sagen, dass die größten Komplikationen beim Erstellen dieser Arbeit dadurch hervorgerufen wurden, dass die HoloLens eine neue Technologie ist, die noch nicht weit verbreitet eingesetzt wird, wodurch es zu bestimmten Problemen keine bekannten Lösungsansätze gibt. Hinzu kommt, dass insbesondere der Research-Mode und der damit einhergehende Zugriff auf die Sensordaten auch erst später hinzugefügt wurde. Dieser wird nicht von bestehenden SDKs oder dem MRTK unterstützt, weshalb hierfür komplett eigens entwickelt werden musste. Zwar gibt es Beispielcode von Microsoft, dieser ist jedoch in C++ geschrieben, was wiederum von Unity nicht unterstützt wird, auch wenn dieses die führende Entwicklungsumgebung für Anwendungen auf der HoloLens ist und sogar von Microsoft selbst in der Entwicklerdokumentation für diese empfohlen wird.

Ebenso wurden einige technische Probleme mit der verwendeten Version der HoloLens deutlich. Die auf Kinect-Technologie basierenden Tiefensensoren und die von diesen erstellten Tiefendaten sind nicht genau genug, um viele Details zu erkennen. Das kann bereits anhand der automatisch angefertigten Raummodelle gesehen werden, die nur Oberflächen gut darstellen, während Einzelheiten nicht erfasst werden können. Dies bedeutet, dass Anwendungen, die insbesondere durch das Beachten von Feinheiten profitieren, nicht genau funktionieren. Das ist bei der Nutzung der RANSAC- und ICP-Algorithmen besonders aufgefallen. Die groben Strukturen können erkannt werden, die Details, die diese Verfahren genauer arbeiten lassen, jedoch nicht.

Eine Möglichkeit, die Algorithmen zu verbessern, wäre eine bessere Datenaufbereitung der 3D-Daten, die von der HoloLens aufgenommen werden. Es kann getestet werden, die Daten genauer auf das aufgenommene Objekt zu beschränken, um mögliche Fehlerminima, die zu falschen Ergebnissen des ICP-Algorithmus führen können, zu eliminieren. So kann z. B. versucht werden, zunächst eine Plane Segmentation (Ebenensegmentierung) auf den Punktwolkendaten [39] durchzuführen, um zumindest die Oberfläche, auf der das Modell steht, aus den Messwerten zu entfernen. Anschließend könnten Werte, die eine bestimmte Distanz vom Zentrum der Aufnahme entfernt sind, gestrichen werden, um noch weiter auf das zu untersuchende Objekt zu fokussieren. Auch könnte versucht werden, statt der direkten Tiefendaten nur mit Merkmalen zu arbeiten, die zuvor durch z. B. SIFT extrahiert wurden.

Zum Zeitpunkt der Erstellung dieser Arbeit hat Microsoft bereits Daten der Tiefensensoren der nächsten Generation der HoloLens veröffentlicht. Diese scheinen qualitativ weit besser zu sein als die der aktuellen Version.

Weitere Aspekte, die das Verwenden der HoloLens in dieser Arbeit erschwert haben, sind UX-Faktoren. So sollte für zukünftige Versionen zum einen das Sichtfeld, in welchem Hologramme angezeigt werden können, vergrößert werden, um eine angenehmere Bedienung zu gewährleisten. Außerdem fühlt sich das Gerät mit 579 g[30] auf Dauer sehr schwer an. Eine weitere Verbesserungsmöglichkeit stellt der Zugriff auf die Sensordaten der HoloLens dar, der verbessert und vereinfacht werden sollte. Gleichermaßen wäre die Unterstützung von gängigen Machine-Learning-Frameworks, oder zumindest ein 64-Bit Betriebssystems zur Vereinfachung der Verwendung dieser, sehr hilfreich. Es gibt neuronale Netze, die darauf ausgelegt sind, dass sie auch auf mobilen Geräten wie z. B. Smartphones verwendbar sind, wie z. B. MobileNetV2 [40]. Diese können daher auch für den in dieser Arbeit beschriebenen Anwendungsfall relevant sein.

[30] Nach Herstellerangaben unter: https://www.microsoft.com/en-us/p/microsoft-hololens-development-edition/8xf18pqz17ts?activetab=pivot:techspecstab

5 Fazit

Das Ziel dieser Arbeit war die Erstellung einer AR-Anwendung zur Erkennung eines Objektes und dessen Orientierung. Dazu wurden verfügbare Hard- und Softwaretechnologien evaluiert. Auf Grundlage dessen wurde zuerst ein Szenario betrachtet, welches das untersuchte Problem abbildet, und ein Konzept entwickelt, das dieses löst. Hierfür musste zunächst ein Objekt gewählt und gedruckt werden, das erkannt werden soll. In dieser Arbeit wurde sich für ein Modell eines Ford-V6 Motorblocks entschieden.

Anschließend wurde ein Datenbestand für das Training eines neuronalen Netzes zur Erkennung der groben Objektausrichtung erstellt. Hierfür wurden zunächst echte Bilder des Objektes aufgenommen, die danach noch durch verschiedene Data-Augmentation-Techniken vervielfacht wurden. Auf Grundlage dieser Daten wurde dann das neuronale Netz trainiert. Es konnte gezeigt werden, dass dieses die Aufgabe der Erkennung der betrachteten Objektseite mit einer hohen Genauigkeit erfüllen kann.

Für die genaue Bestimmung der Orientierung des Objektes wurde mit den Daten der Tiefensensoren der HoloLens gearbeitet. Diese mussten zunächst von dieser extrahiert werden, damit sie anschließend in eine Form gebracht werden konnten, die von den gewählten Algorithmen verarbeitet werden können. Der RANSAC- und ICP-Algorithmus wurden dann auf die aufgenommenen Tiefendaten des Motorblocks angewandt. Die Ergebnisse dieser Ausführungen zeigen, dass in bestimmten Fällen die Algorithmen gute Resultate liefern, während in anderen die niedrige Qualität der Daten zu einem Problem wurde. Besondere Fälle mit geringem Anteil von Daten, die das Modell repräsentieren, zeigen keine guten Ergebnisse.

In dieser Arbeit wurden Lösungen für die Teilprobleme der Objekterkennung und Ausrichtung anhand von Daten der HoloLens entwickelt und evaluiert. Lediglich die Kombination aller Komponenten zu einer kompletten Anwendung wurde im Rahmen dieser Arbeit nicht durchgeführt. Dies sollte aber durch die komponentenorientierte Softwarestruktur unter Beachtung einer strikten Aufgabentrennung kein großes Problem darstellen.

Es ist anzunehmen, dass sich die Erkenntnisse dieser Arbeit auf ähnliche Fälle übertragen lassen. Dies ist insbesondere dadurch gegeben, dass das verwendete zu erkennende Objekt zuerst als digitales Modell vorlag und erst später gedruckt wurde, um so einen realen

Gegenstand zu erhalten. Dies ist nah an der Realität der Industrie, da auch hier zunächst Produkte als digital Modelle vorhanden sind, bevor sie als reale Prototypen erstellt werden.

Die Ergebnisse lassen sich zusätzlich durch leistungsfähigere Technologien und genauere Aufbereitung der Messwerte verbessern. Besonders die Verbesserung der Datenqualität birgt großes Potential für die Anwendung des RANSAC- oder ICP-Algorithmus. Diese zeigen bessere Ergebnisse, wenn die zu vergleichenden Daten auf das eigentliche Objekt beschränkt werden.

5.1 Ausblick

Analysen zeigen, dass AR sich immer mehr im Markt behauptet. Insbesondere die Verkaufszahlen von AR-Headsets werden als stark steigend prognostiziert, vor allem in vielen Bereichen der Industrie und Medizin. So wird die jährliche Wachstumsrate des Umsatzes z. T. auf 70 % oder höher bis 2020 geschätzt [2]. Auch werden neue AR-Headsets wie die Magic Leap verkauft. Microsoft hat ebenfalls eine aktualisierte Version der HoloLens für 2019 angekündigt, die einen eigenen dedizierten Chip für Machine-Learning-Aufgaben besitzen soll. Das wäre für den in dieser Arbeit betrachteten Anwendungsfall sehr hilfreich, da zum einen die Klassifikation des Objektes direkt auf der HoloLens durchgeführt werden kann. Dies führt zu weniger Kommunikation mit anderen Systemen, was potenzielle Fehlerquellen eliminiert. Zum anderen können neuronale Netze gut auf spezialisierte Hardware angepasst werden und so die Performance verbessert werden.

Auch soll die kommende Version der HoloLens wesentlich verbesserte Tiefensensoren verbaut haben. Dies würde zu einer Verbesserung der RANSAC- und ICP-Algorithmen führen, da die mangelnde Qualität der Daten als eines der größten Probleme erkannt wurde.

Die Rahmenbedingungen für weiterführende Experimente werden also immer weiter verbessert. Außerdem sind in der Industrie die für die Verwendung der vorgestellten Lösung benötigten Daten schon in Form von CAD-Dateien vorhanden. Daher sollte auch weiterhin in dieser Richtung geforscht und entwickelt werden.

Die Experimente dieser Arbeit konnten zeigen, dass die Erkennung eines Objektes und dessen Orientierung relativ zum Anwender innerhalb einer AR-Anwendung mit aktuellen Technologien möglich ist. Diese Erkenntnisse dienen als Grundlage für weiterführende Schritte.

Eine offensichtliche erste Erweiterung der Ergebnisse dieser Arbeit ist das Zusammensetzen der einzelnen erforschten Komponenten zu einer Gesamtanwendung und die Erforschung der damit einhergehenden Probleme. Auch die hierfür benötigten Benutzerinteraktionen können weitergehend untersucht werden. Darüber hinaus ist von Interesse, inwiefern die

Verwendung einer solchen Applikation die tatsächliche Produktivität in der Realität positiv beeinflusst.

Eine weitere mögliche Verbesserung liegt in der Generierung der Daten für das Training des neuronalen Netzes. So könnte eine Pipeline entwickelt werden, um direkt aus den vorhandenen 3D-Daten der zu erkennenden Objekte die charakteristischen Ansichten zu generieren. Dies würde sowohl den manuellen Aufwand für die Erstellung eliminieren als auch die Möglichkeit bieten, sehr effizient eine weit größere Anzahl verschiedener Ansichten zu erstellen. Dies würde die Robustheit des Modells verbessern, was dazu führen würde, dass die Klassifikation der initialen Orientierung verbessert werden könnte.

Ebenso kann das neuronale Netz selbst präziser gemacht werden. So kann getestet werden, ob andere Architekturen oder ein von Grund trainiertes Netz bessere Ergebnisse liefern. Zusätzlich wurden in dieser Arbeit einige Annahmen getroffen, welche u. U. in der Praxis so nicht haltbar sind. Daher kann es sein, dass die generierten Trainingsdaten nicht die Realität abbilden.

Auch die Verbesserung der Erkennung der Objektausrichtung ist eine mögliche Ergänzung dieser Arbeit. Hierzu gehört zum einen die Vorverarbeitung der Tiefendaten, die ungewollte Punkte aus den Messwerten entfernen kann, um dadurch nur auf das eigentlich zu findende Objekt zu fokussieren. Zum anderen können aber auch die Algorithmen an sich präzisiert werden z. B. durch Verwendung anderer Parameter. Genauso denkbar ist es, dass weitere Algorithmen oder Variationen der verwendeten Algorithmen bessere Ergebnisse liefern und somit ebenfalls erforscht werden können.

Nicht zuletzt können die Ergebnisse dieser Arbeit auch im Kundenumfeld der HyperTest-Plattform angewendet werden. Mit diesen könnte man sich dann ein besseres Bild machen, wie die realen Bedingungen aussehen. So können die bisherigen Prozesse evaluiert und mit den erarbeiteten Ergebnissen verbessert werden.

Literaturverzeichnis

[1] P. Milgram, H. Takemura, A. Utsumi und F. Kishino, „Augmented reality: A class of displays on the reality-virtuality continuum," *Telemanipulator and Telepresence Technologies,* Nr. 2351, Januar 1994.

[2] M. E. Porter und J. E. Heppelmann, „Eine Brücke zwischen physischer und digitaler Welt," *Harvard Business Manager,* Nr. 2, pp. 19-39, Februar 2018.

[3] R. Dörner, W. Broll, P. Grimm und B. Jung, Virtual and Augmented Reality (VR / AR), Berlin, Heidelberg: Springer Vieweg, 2013.

[4] H. Durrant-Whyte und T. Bailey, „Simultaneous localization and mapping: part I," *IEEE Robotics Automation Magazine,* Bd. 13, Nr. 2, pp. 99-110, 2006.

[5] T. Bailey und H. Durrant-Whyte, „Simultaneous localization and mapping (SLAM): part II," *IEEE Robotics Automation Magazine,* Bd. 13, Nr. 3, pp. 108-117, 2006.

[6] A. Krizhevsky, I. Sutskever und G. E. Hinton, „ImageNet Classification with Deep Convolutional Neural Networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, Lake Tahoe, Nevada, USA, Curran Associates Inc., 2012, pp. 1097-1105.

[7] D. H. Hubel und T. N. Wiesel, „Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology,* pp. 574-591, 1959.

[8] L. G. Roberts, *Machine perception of three-dimensional solids,* Cambridge, Massachusetts, USA: Massachusetts Institute of Technology, Dept. of Electrical Engineering, 1963.

[9] D. Marr, Vision: A Computational Investigation into the Human Representation and Processing of Visual Information, Cambridge, Massachusetts, USA: The MIT Press, 1981.

[10] K. Fukushima, „Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics,* Bd. 36, Nr. 4, pp. 193-202, 1980.

[11] D. G. Lowe, „Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, Greece, Greece, IEEE, 1999, pp. 1150-1157.

[12] P. Viola und M. Jones, „Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, IEEE, 2001, pp. I-I.

[13] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li und L. Fei-Fei, „ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, IEEE, 2009, pp. 248-255.

[14] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár und C. L. Zitnick, „Microsoft COCO: Common Objects in Context," in *Lecture Notes in Computer Science*, Springer International Publishing, 2014, pp. 740-755.

[15] D. Lowe, „Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision,* Nr. 60, pp. 91-110, 01 November 2004.

[16] H. Bay, T. Tuytelaars und L. Van Gool, „SURF: Speeded Up Robust Features," in *Computer Vision -- ECCV 2006*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2006, pp. 404-417.

[17] N. Dalal und B. Triggs, „Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, IEEE, 2005, pp. 886-893 vol. 1.

[18] C. Stachniss, „Photogrammetry II - 10 - SIFT Features and RANSAC (2015/16)," Aufzeichnung einer Bachelorvorlesung an der Universität von Bonn im Wintersemester 2015/16, 12 Januar 2016. [Online]. Available: https://www.youtube.com/watch?v=oT9c_LlFBqs. [Zugriff am 8 11 2018].

[19] F. Chollet, Deep Learning with Python, Greenwich: Manning Publications Co., 2017.

[20] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan und Q. V. Le, *AutoAugment: Learning Augmentation Policies from Data,* https://arxiv.org/pdf/1805.09501.pdf, 2019.

[21] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens und Z. Wojna, „Rethinking the Inception Architecture for Computer Vision," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* pp. 2818-2826, 2016.

[22] M. A. Fischler und R. C. Bolles, „Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography," *Commun. ACM,* Nr. 24, pp. 381-395, Juni 1981.

[23] Y. Chen und G. Medioni, „Object modeling by registration of multiple range images," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, Sacramento, CA, USA,, IEEE, 1991, pp. 2724-2729 vol.3.

[24] P. J. Besl und N. D. McKay, „A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* pp. 239-256, Februar 1992.

[25] S. Rusinkiewicz und M. Levoy, „Efficient variants of the ICP algorithm," in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, Quebec City, Quebec, Canada,, IEEE, 2001, pp. 145-152.

[26] M. Miesnieks, „Why is ARKit better than the alternatives?," 31 Juli 2017. [Online]. Available: https://medium.com/6d-ai/why-is-arkit-better-than-the-alternatives-af8871889d6a. [Zugriff am 7 11 2018].

[27] M. Miesnieks, „How is ARCore better than ARKit?," 1 September 2017. [Online]. Available: https://medium.com/6d-ai/how-is-arcore-better-than-arkit-5223e6b3e79d. [Zugriff am 7 11 2018].

[28] M. Zeller und B. Bray, „HoloLens hardware details," Microsoft, 3 März 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/hololens-hardware-details. [Zugriff am 1 November 2018].

[29] P. Farley und M. Satran, „Device Portal for HoloLens," Microsoft, 26 September 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/uwp/debug-test-perf/device-portal-hololens. [Zugriff am 11 November 2018].

[30] D. Gedye, N. Schonning und M. Zeller, „HoloLens Research mode," Microdsoft, 3 Mai 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/research-mode. [Zugriff am 5 November 2018].

[31] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, J. Yangqing, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu und X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems,* Software available from tensorflow.org., 2015.

[32] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox und D. H. Yoon, „In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, New York, NY, USA, ACM, 2017, pp. 1-12.

[33] K. Pulli, A. Baksheev, K. Kornyakov und V. Eruhimov, „Realtime Computer Vision with OpenCV," *Queue,* Bd. 10, Nr. 4, pp. 40-56, 2012.

[34] R. B. Rusu und S. Cousins, „3D is here: Point Cloud Library (PCL)," in *2011 IEEE International Conference on Robotics and Automation*, Shanghai, China, IEEE, 2011, pp. 1-4.

[35] A. Rodriguez, „Transforming a depth map into a 3D point cloud," 6 September 2017. [Online]. Available: https://elcharolin.wordpress.com/2017/09/06/transforming-a-depth-map-into-a-3d-point-cloud/. [Zugriff am 9 Januar 2019].

[36] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng und T. Darrell, *DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition,* 2013.

[37] K. M. Ting, „Confusion Matrix," in *Encyclopedia of Machine Learning and Data Mining*, Boston, MA, Springer US, 2017, pp. 260-260.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot und É. Duchesnay, „Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.,* pp. 2825-2830, 11 2011.

[39] B. Oehler, J. Stueckler, J. Welle, D. Schulz und S. Behnke, „Efficient Multi-resolution Plane Segmentation of 3D Point Clouds," in *Intelligent Robotics and Applications*, Berlin, Heidelberg, Springer Berlin Heidelberg, 2011, pp. 145-156.

[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov und L.-C. Chen, „MobileNetV2: Inverted Residuals and Linear Bottlenecks," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Salt Lake City, UT, USA, IEEE, 2018, pp. 4510-4520.

Abbildungsverzeichnis

A. Anhang

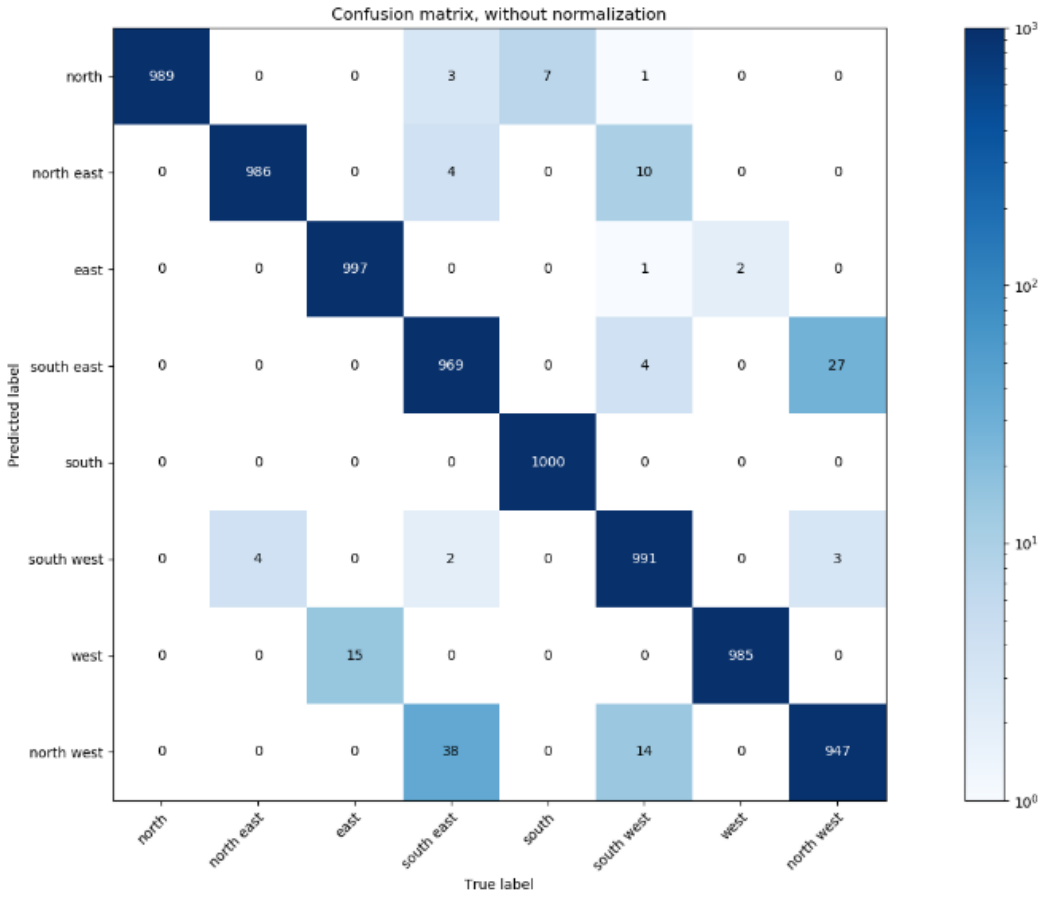

Abbildung A.1: Confusion Matrix der getesteten Bilder (absolut)

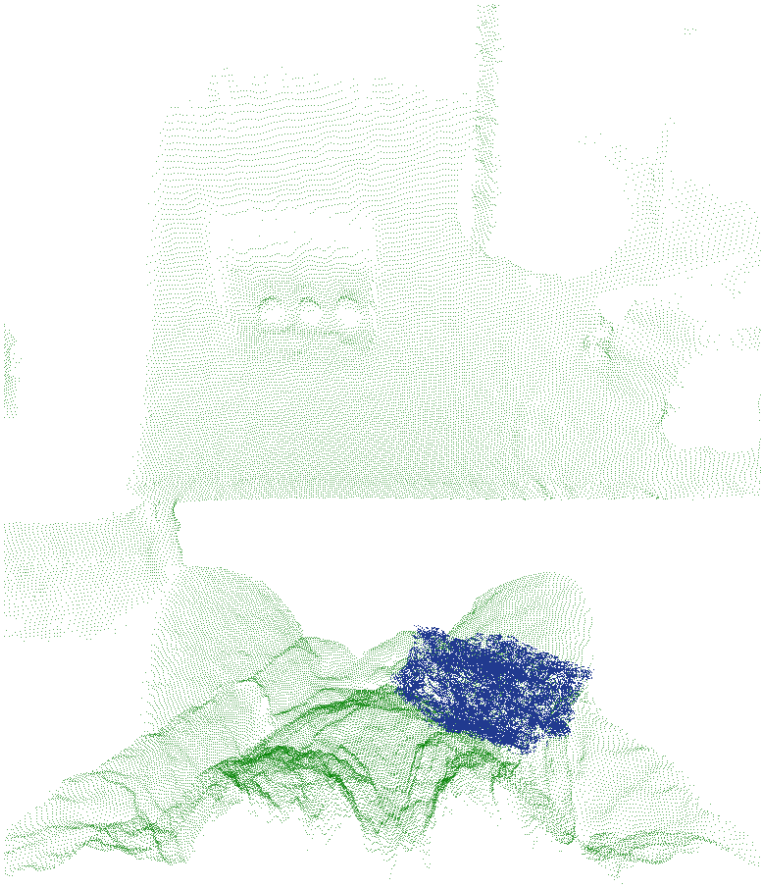

Abbildung A.2: Punktwolke 1 (grün) und Modellwolke (blau) nach Anpassung von RANSAC

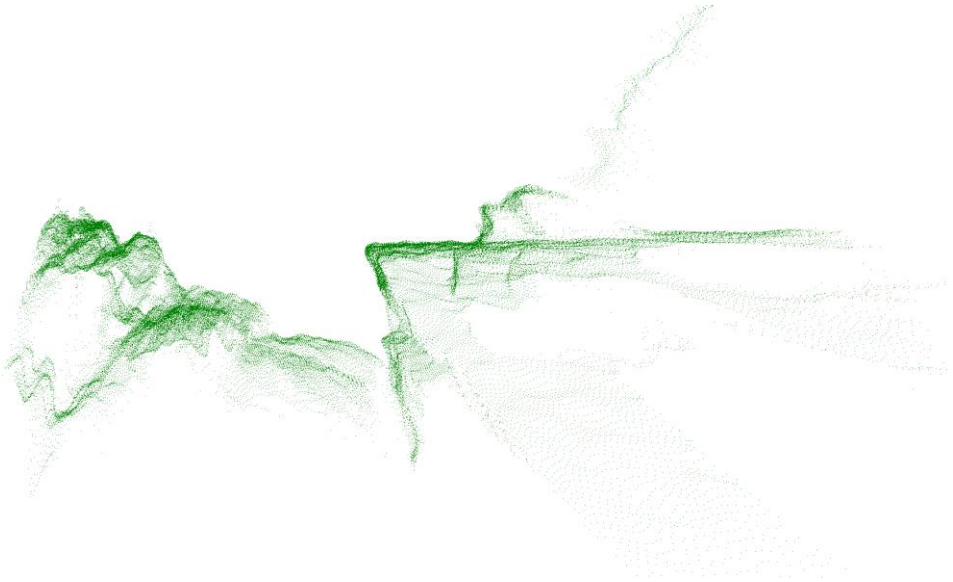

Abbildung A.3: Punkwolke 1, Seitenansicht

Abbildung A.4: Punktwolke 1 (grün) und Modellwolke (blau) nach Anpassung von ICP, Seitenansicht

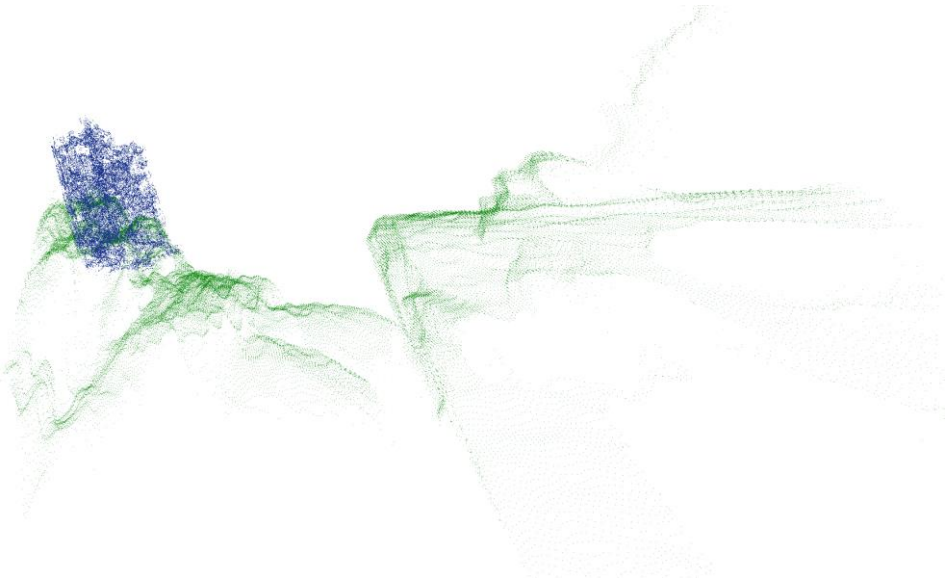

Abbildung A.5: Punktwolke 1 (grün) und Modellwolke (blau) nach Anpassung von RANSAC, Seitenansicht

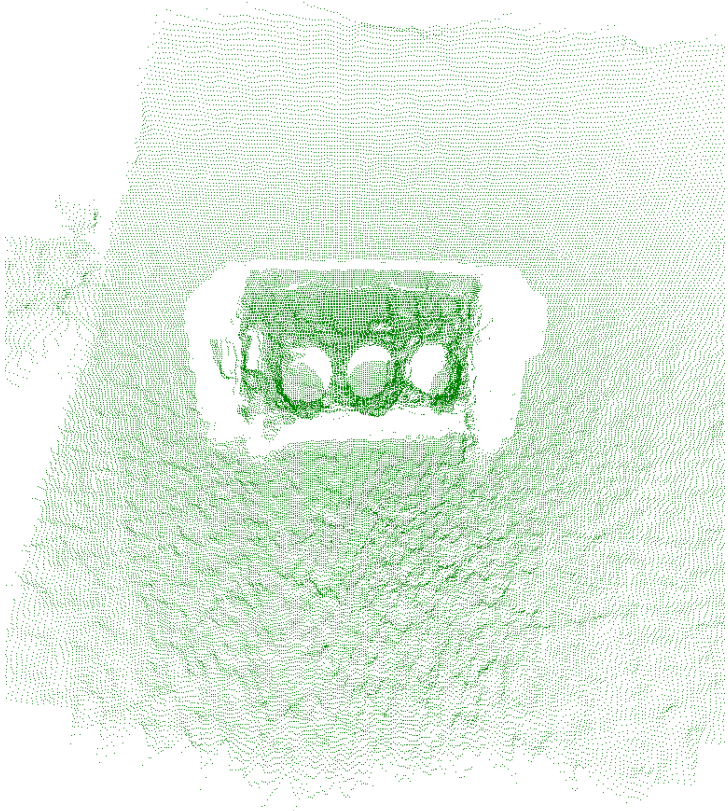

Abbildung A.6: Punktwolke 2

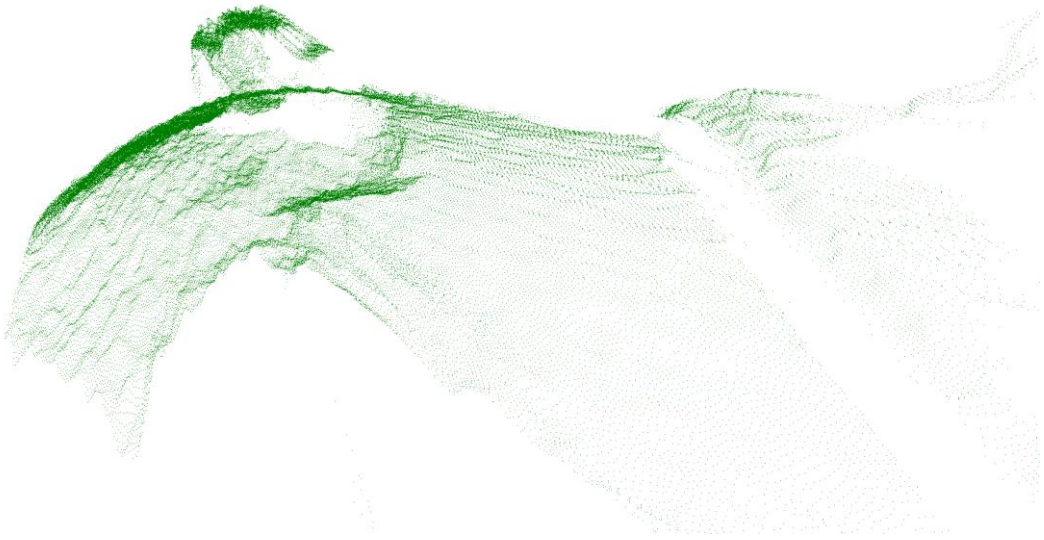

Abbildung A.7: Punktwolke 2, Seitenansicht

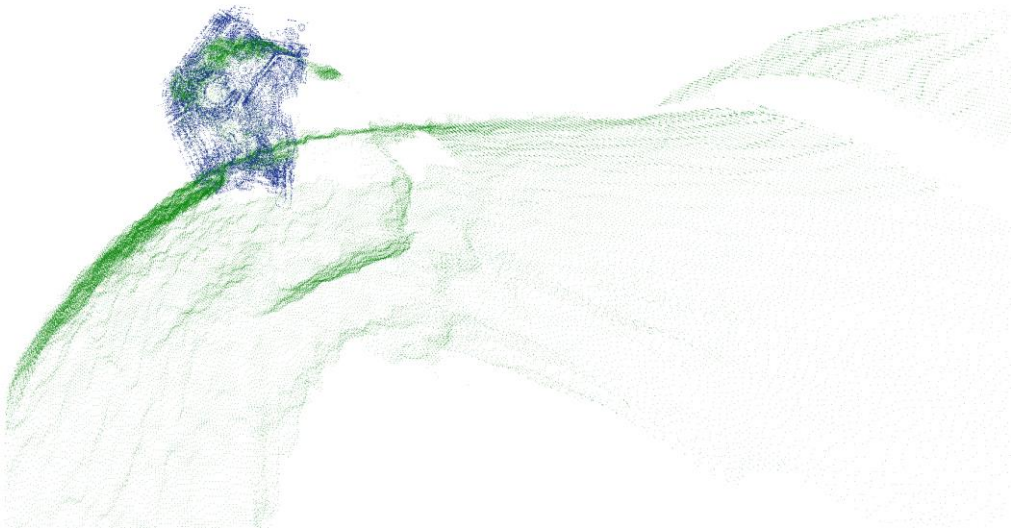

Abbildung A.8: Punktwolke 2 (grün) und Modellwolke (blau) nach Anpassung von ICP, Seitenansicht

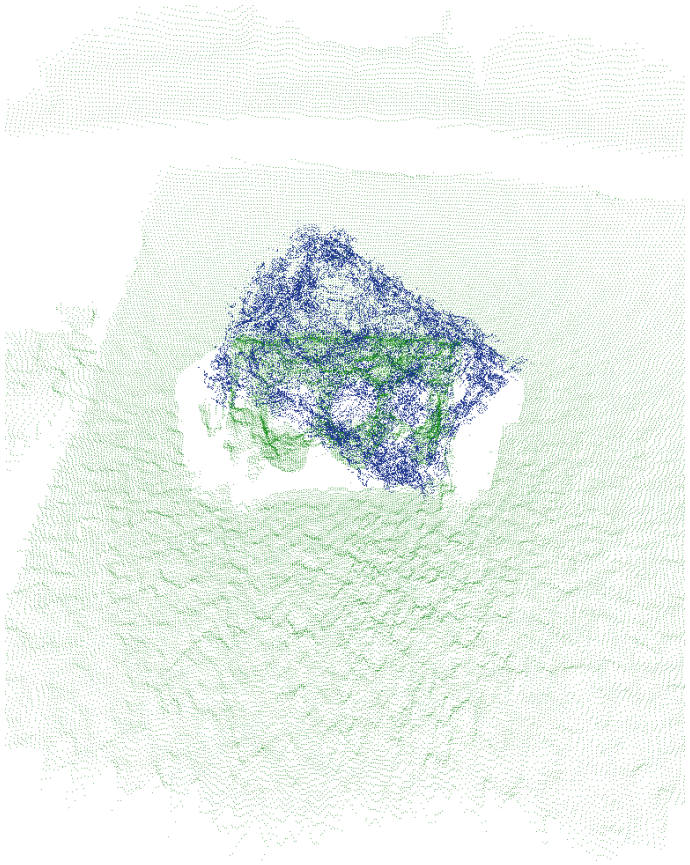

Abbildung A.9: Punktwolke 2 (grün) und Modellwolke (blau) nach Anwendung von RANSAC

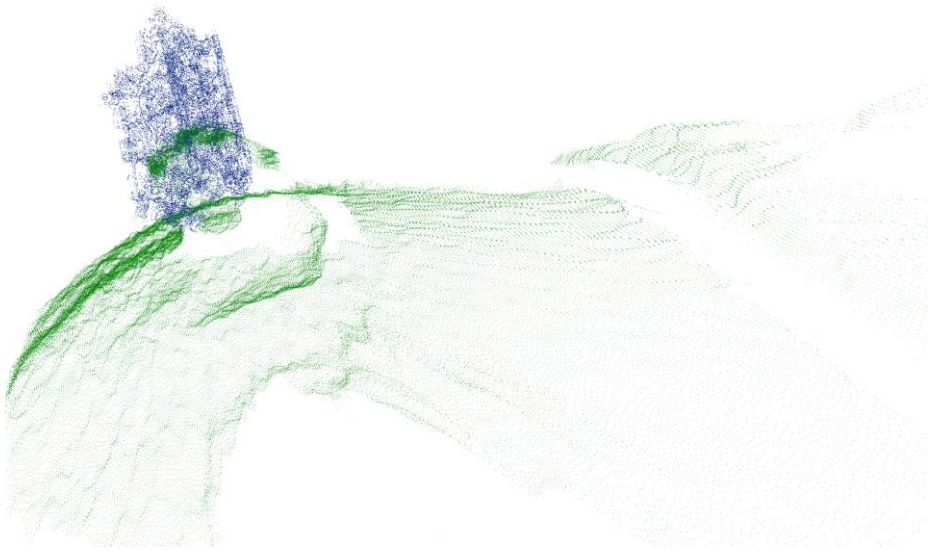

Abbildung A.10: Punktwolke 2 (grün) und Modellwolke (blau) nach Anwendung von RANSAC, Seitenansicht

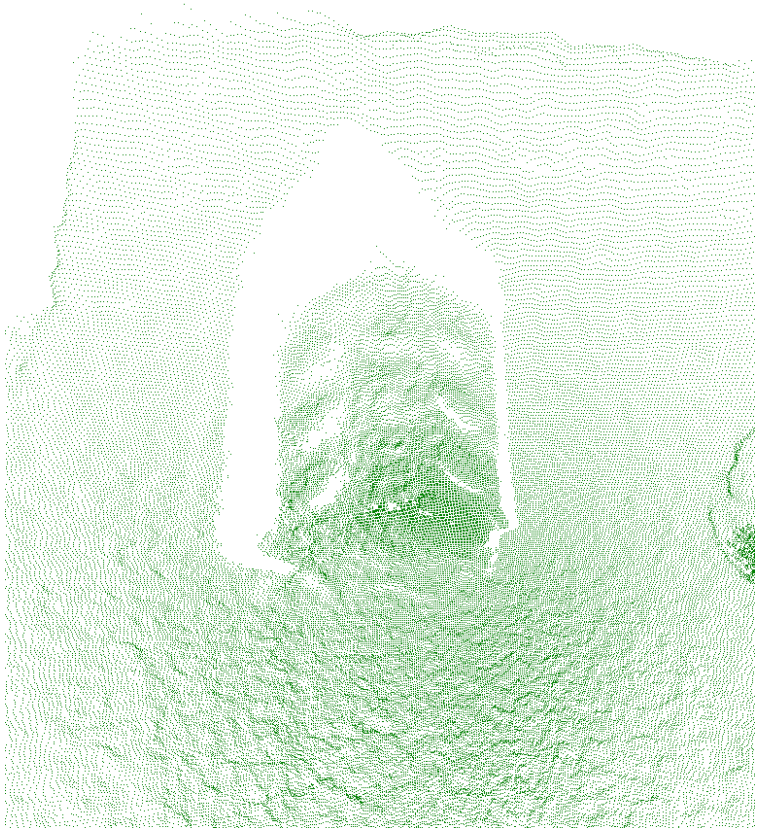

Abbildung A.11 - Punkwolke 3

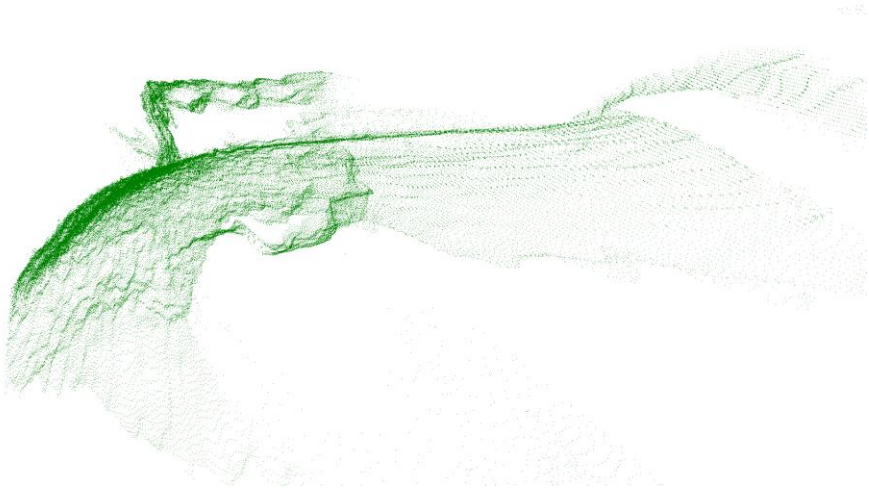

Abbildung A.12 - Punktwolke 3, Seitenansicht

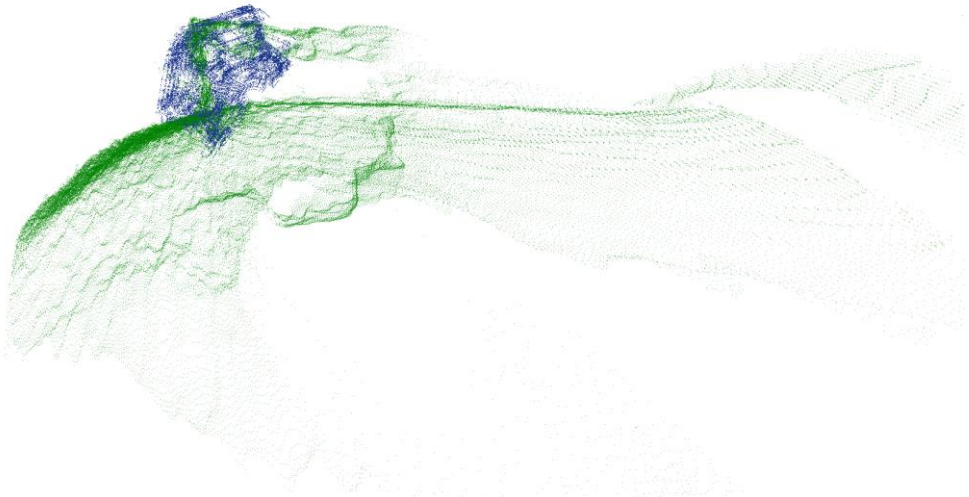

Abbildung A.13 - Punktwolke 3 (grün) und Modellwolke (blau) nach Anwendung von ICP, Seitenansicht

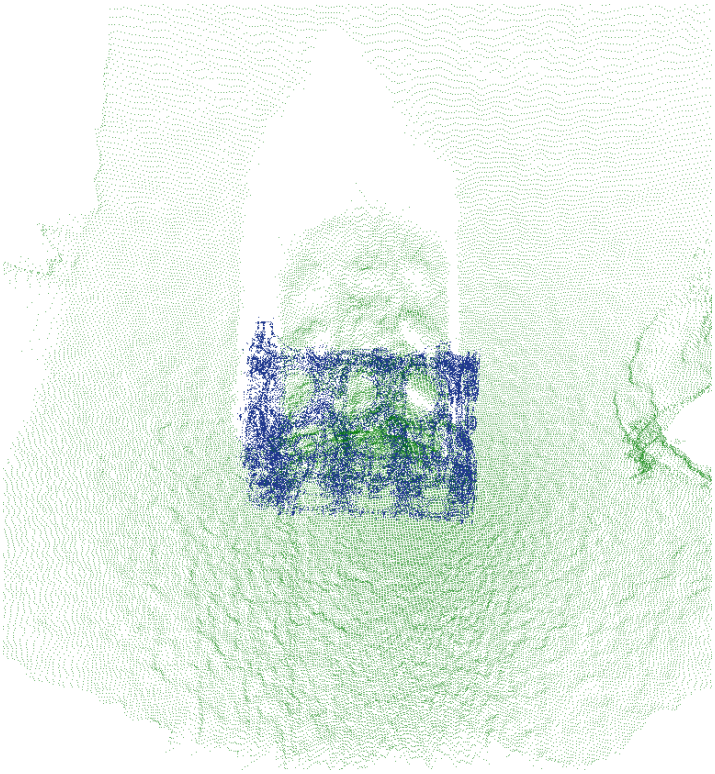

Abbildung A.14 - Punktwolke 3 (grün) und Modellwolke (blau) nach Anwendung von ICP

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 06. Mai 2019 _____

Marcell Klepper