ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften, Informatik und Psychologie
Institut für Datenbanken und Informationssysteme

Objekterkennung mit ARKit und TensorFlow: Anwendungsfälle, Konzepte, Evaluierung

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Kristijan Biro
kristijan.biro@uni-ulm.de

Gutachter:
Prof. Dr. Manfred Reichert

Betreuer:
Klaus Kammerer

2019

Fassung 17. Dezember 2019

Satz: PDF-LaTeX 2$_\varepsilon$

Kurzfassung

In den Bereichen Computer Vision und Neuroinformatik gab es in den letzten Jahren große Fortschritte, die neuartige Augmented Reality Anwendungen ermöglichen. So können beispielsweise im industriellen Umfeld Smartphones dazu verwendet werden, Maschinenbauteile über eine Kamera zu erkennen, zu klassifizieren und weitere Informationen über diese bereitzustellen. Hierfür existieren verschiedene Objekterkennungs-Frameworks, deren industrielle Verwendbarkeit jedoch nicht hinreichend evaluiert wurde.

In dieser Abschlussarbeit werden die Frameworks TensorFlow Lite und ARKit näher betrachtet. Ziel ist es, die Erkennungsrate und -geschwindigkeit der beiden Frameworks mit Hilfe eines Experiments zu evaluieren. In dem Experiment werden Maschinenteile einer pharmazeutischen Verpackungsmaschine als Subjekte verwendet und deren Erkennungsleistung auf Basis verschiedener Umgebungsfaktoren, wie Blickwinkel und Entfernung evaluiert.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die mich während der Verfassung dieser Ausarbeitung unterstützt haben.

Ein großer Dank geht an Klaus Kammerer, der diese Ausarbeitung betreut hat. Für seine Unterstützung und Geduld möchte ich mich hiermit herzlich bedanken.

Ebenso möchte ich mich beim Herrn Prof. Dr. Manfred Reichert für die Begutachtung meiner Bachelorarbeit bedanken.

Für die Bereitstellung der Maschinenteile möchte ich mich hiermit auch bei der Firma Uhlmann bedanken.

Zuletzt möchte ich mich bei meiner Familie bedanken, vor allem bei meinen Eltern Antal und Ivana die mich während dem Studium unterstützt haben, meinen Großeltern Balaž und Katarina ohne die mein Studium in Deutschland nicht möglich wäre. Ebenso möchte ich mich bei meiner Großmutter Tereza bedanken, die mich immer mit guten Ratschlägen beraten hat.

Inhaltsverzeichnis

1

Einleitung

Menschen sind in der Lage Objekte, die sich in ihrer Umgebung befindet, leicht zu erkennen und zu identifizieren. Moderne Algorithmen und Technologien ermöglichen es auch Computern Objekte in einem Videobild zu erkennen und zu identifizieren. Fortschritte in Computer Vision und Neuroinformatik haben dazu beigetragen, die Erkennungsleistung stetig zu verbessern, sodass sich das Potential dieser Technologie vor allem für industrielle Anwendungen erschließt. Beispielsweise können Maschinen über eine Kameraeingabe Produkte eines Produktionsprozesses kontrollieren. Hierfür existieren bereits Software-Frameworks, die eine Objekterkennung durchführen können. Jedoch besteht die Frage ob sich diese für einen zuverlässigen und effizienten Einsatz in der Industrie eignen. Im Rahmen dieser Arbeit werden zwei Frameworks für die Objekterkennung im Rahmen eines Experiments untersucht.

1.1 Problemstellung

In dem Kontext dieser Arbeit werden Maschinenbauteile einer Maschine für Medikamentenverpackungen betrachtet. Diese werden im Rahmen einer Wartung abgebaut oder ausgetauscht. Um ein Maschinenteil bei einem Tausch in kurzer Zeit identifizieren zu können, bietet sich eine Objekterkennung an. In der pharmazeutischen Industrie gibt es sehr strenge Richtlinien, die es nicht ermöglichen die Maschinenteile beispielsweise mit Stanzmarken oder Aufklebern zu markieren, da solche Methoden eine effektive sterile Reinigung der Maschinenteile verhindern würden. Mit Hilfe eines Objekterkennungs-Frameworks könnte eine Software entwickelt werden, die eine Handhabung der Bauteile erleichtern würde. Diese Software könnte zum Beispiel die Verwaltung der Bauteile

unterstützen, indem sie diese identifiziert und weitere Informationen bereitstellen. Somit hätte die Benutzer dieser Software in kürzester Zeit alle Informationen über das Maschinenteil, eine manuelle Suche in gedruckten Dokumentationen würde entfallen. Es existieren viele Frameworks zur Objekterkennung, doch es ist fraglich ob sich alle für industrielle Anwendungen, wie die oben skizzierte, eignen.

1.2 Zielsetzung

Im Rahmen dieser Arbeit wird ein Experiment definiert, durchgeführt und ausgewertet, das die beiden Frameworks TensorFlow Lite und ARKit bezüglich ihrer Leistung zur Objekterkennung evaluiert. Die Frameworks werden in einer experimenteller Umgebung untersucht und evaluiert. Nach der Ausführung und Evaluierung des Experiments wird ein Vergleich zwischen den beiden Frameworks durchführt. Ziel des Experimentes ist es, Metriken zu erheben, welche die Erkennungsleistung der beiden Frameworks quantitativ darstellen können.

1.3 Struktur der Arbeit

Kapitel 2 führt Grundlagen für die Themen Computer Vision, Machine Learning, Mixed Reality sowie die betrachteten Frameworks ein. Kapitel 3 betrachtet einen Anwendungsfall für Serviceprozesse von cyber-physischen Produktionssystemen. Kapitel 4 beschreibt den Aufbau der entwickelten Anwendungen für das Experiment. Kapitel 5 führt den Experimentalaufbau und dessen Durchführung ein. Anschließend werden verwandte Arbeiten in Kapitel 6 diskutiert. Kapitel 7 fasst diese Abschlussarbeit zusammen.

2

Grundlagen

In diesem Kapitel werden Grundlagen zu Computer Vision, Machine Learning Algorithmen sowie Mixed Reality Technologien erläutert.

2.1 Computer Vision

Computer Vision befasst sich mit dem Problem, die Umgebung anhand von Bildern in dem Kontext von Rechnern zu verstehen [1]. Menschen können mit Leichtigkeit ihr visuelle Umgebung beschreiben, in der sie sich befinden. Sie haben ein Verständnis für die Umgebung und können die Objekte anhand ihrer Eigenschaften beschreiben. Wissenschaftler haben sich mit dem Thema befasst und über die Jahre Methoden, beispielsweise Algorithmen zur Objekterkennung, Gesichtserkennung und Rekonstruktion von 3D Umgebungen entwickelt.

Dennoch ist das Forschungsfeld Computer Vision ein anspruchsvolles Forschungsgebiet, das sich mit komplexen Problemen befasst. Szeliski begründet dies damit, dass Computer Vision ein inverses Problem ist: es wird nach Informationen gesucht in Daten die nicht genug Informationen enthalten, um eine Problemstellung zu lösen [2]. Deswegen werden physikalische und statistische Modelle verwendet, um bei potenziellen Ergebnissen Unterschiede verstehen zu können. Modelle, die in Computer Vision verwendet werden, kommen aus den Gebieten der Physik und der Computergrafik. In diesem Gebieten wurden Modelle entwickelt, die beschreiben wie sich Objekte bewegen und animieren lassen, wie sich Licht auf verschiedenen Oberflächen reflektiert und wie Projektionen auf eine Oberfläche dargestellt werden können.

Computer Vision ein praktisches Gebiet, das unter Anderem in der Industrie [3], Medizin [4] und Wissenschaft [5] Verwendung findet. Beispiele hierfür sind die optische Zeichenerkennung (engl.: optical character recognition) [6], Gesichtserkennung [7] und Videoüberwachung [8]. Im Folgenden werden die Grundkonzepte von Computer Vision beschrieben und erläutert.

2.1.1 Bildverarbeitung

Ein Bild kann als eine zweidimensionale Funktion $f(x, y)$ definiert werden, wobei x und y räumliche (Ebenen-)Koordinaten sind. Die Amplitude von f eines beliebigen Paares von Koordinaten (x, y) werden als Intensität oder Grauwert des Bildes an diesem Punkt bezeichnet. Wenn (x, y) und die Amplitudenwerte von f endliche, diskrete Größen sind, wird dies ein *digitales Bild* genannt [9].

Die häufigsten Methoden bei der Bearbeitung von digitalen Bildern sind Weißabgleich, Rauschreduktion, Anpassung der Bildschärfe und Bildrotation [2]. Die einfachste Art ein Bild zu bearbeiten ist die *Punktoperation* [2]. Hierbei handelt es sich um eine Vorgehensweise, bei der jeder Ausgabepixelwert von dem entsprechenden Eingabepixelwert abhängig ist. Diese Methode wird meistens bei der Anpassung von Helligkeit und Kontrast im Bild verwendet. Als konkretes Beispiel kann ein Farbbild betrachtetet werden, in dem die Helligkeit erhöht werden soll. Farbbilder haben drei Kanäle auf den nun ein konstanter Wert aufaddiert werden soll. Nach der Addition hat sich die Intensität der jeweiligen Pixels erhöht und damit auch die Helligkeit.

Eine weitere Methode, die bei der Verarbeitung von Bildern verwendet wird, ist die *Nachbarschaftsoperation*. Hierbei wird im Gegensatz zur Punktoperation die Nachbarschaftsumgebung eines Pixels betrachtet, um den Wert des Ausgabepixels zu berechnen. Unter der Nachbarschaftsumgebung werden die Pixels in der Umgebung des betrachteten Pixels definiert. Nachbarschaftsoperationen werden für die Anpassung der Bildschärfe, Kantenhervorhebung und Rauschreduktion verwendet. Die meist verwendete Nachbarschaftsoperation ist die *lineare Filterung* [10]. Bei der linearen Filterung wird der Ausgabepixelwert als eine Summe der gewichteten Eingabepixelwerten berechnet.

Dies wird mit der Formel

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

berechnet, wobei $h(k,l)$ die Filtermaske ist. Die Einträge von $h(k,l)$ werden auch Filterkoefziente genannt. Weiter kann die Formel auch in

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l)$$

umgeschrieben werden, wobei das Vorzeichen von f umgekehrt wurde. Dies wird in der Mathematik *Faltung* (engl.: *convolution*) genannt und wird mit der Notation

$$g = f * g$$

bezeichnet, während h als Gewichtsfunktion bezeichnet wird. Faltungen haben die Eigenschaft, dass sie kommutativ und assoziativ sind. Ferner ist die Fouriertransformation zweier gefalteter Bilder gleich dem Produkt der einzelnen Fouriertransformationen der Bilder. Diese Eigenschaften erleichtert die Berechnungen, die meist bei einer Bildverarbeitung stattfinden. In Abbildung 2.1 wird eine Faltungsoperation veranschaulicht: das linke Bild wird mit einem Filter in der Mitte gefaltet; es entsteht das Bild auf der rechten Seite. Die blauen Pixel bezeichnen die Nachbarschaft des Eingabepixels, aus dem dann der in der Abbildung entstehende Ausgabepixel grün gefärbt ist.

In diesem Abschnitt wurden einige Methoden für die Bildverarbeitung betrachtet und diskutiert. Nachfolgend werden die bearbeiteten Bilder analysiert, um Informationen aus dem Bild zu erhalten. Im nächsten Abschnitt werden ein paar dafür geeignete Methoden vorgestellt.

2.1.2 Feature-Erkennung

Als *Feature* im Kontext von Computer Vision werden Merkmale in einem Bild bezeichnet, die das Bild prägen. So können zum Beispiel Features in einem Bild Kanten, verschiedene geometrische Formen und Flächen sein. Da Features vom Anwendungsfall abhängig

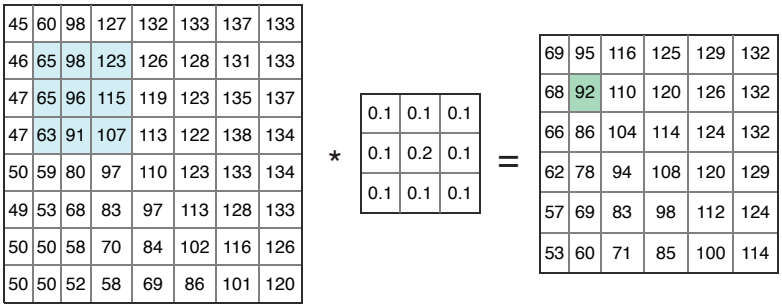

Abbildung 2.1: Beispiel einer Faltung angewandt auf einem Bild [2]

sind, wird bei der praktischen Anwendung festgestellt, welche Features in einem bestimmten Anwendungsfall betrachtet werden sollen. So können Features die nicht von Interesse sind ausgefiltert werden. Features können in Klassen aufgeteilt werden [10]. Eine Klasse beschreibt dabei spezifische Bereiche im Bild, wie zum Beispiel Berggipfel, Bauwerke und andere Formen. Diese lokalen Features werden auch *interest points* genannt und werden von der Nachbarschaftsumgebung des Features beschrieben. Eine weitere wichtige Klasse sind die *erkannten Kanten* im Bild. Kanten können anhand der Orientierung und lokaler Darstellung erkannt werden und beschreiben oft die Grenzen eines Objektes. Im Folgenden werden die einzelnen Phasen der Feature-Erkennung beschrieben.

Punktfeatures werden verwendet, um eine Menge von entsprechenden Features in verschiedenen Bildern zu finden. Die Herangehensweise, um Punktfeatures zu finden, kann in zwei Arten aufgeteilt werden [2]. Die erste Herangehensweise ist es Features in einem Bild zu finden, die leicht verfolgbar sind, wobei für die lokalen Features eine bekannte Suchmethode verwendet wird. Die zweite Herangehensweise ist es unabhängig in allen betrachteten Bildern die Features zu finden und dann gegen die lokale Darstellung zu vergleichen. Die erste Herangehensweise wird meistens bei der Analyse von Videoeingaben verwendet, während letztere sich eher eignet, wenn bei der Aufnahme der Bilder kontinuierliche Bewegungen und Szenenänderungen stattfinden.

Der Prozess der Feature-Erkennung bei Punktfeatures kann in vier Phasen aufgeteilt werden: *Feature-Erkennung*, *Feature-Beschreibung*, *Feature-Übereinstimmung* und *Feature-*

Verfolgung [2]. Abbildung 2.2 stellt die einzelnen Schritte dar. Die erste Phase besteht aus der Feature-Erkennung. Es wird nach Bereichen im Bild gesucht, die mit anderen Bildern übereinstimmen könnten. In der Feature-Beschreibungsphase werden die erkannten Features in ein Format umgewandelt, das sich leicht mit dem anderen Formaten vergleichen lässt. In Abbildung 2.2 wird die Orientierung und Länge der Gradienten verwendet, um die einzelnen Pixelwerte zu beschreiben. Der Gradient ist die mehrdimensionale Ableitung in einem Punkt P und der Begriff wird genauer im nächsten Abschnitt definiert. Bei der Feature-Übereinstummungsphase werden die Features im vorhandenen Bild nach einer Übereinstimmung gesucht. In Abbildung 2.2 bezeichnen die Farben der Zahlen die Genauigkeit der Übereinstimmung. Zuletzt wird bei der Feature-Verfolgung die Nachbarschaft betrachtet, in der sich das Feature befindet, um bei neuen *Frames* die Features zu bestimmen. Unter *Frames* versteht man eine Bildeinheit in einer Bildsequenz von mehreren Bildern.

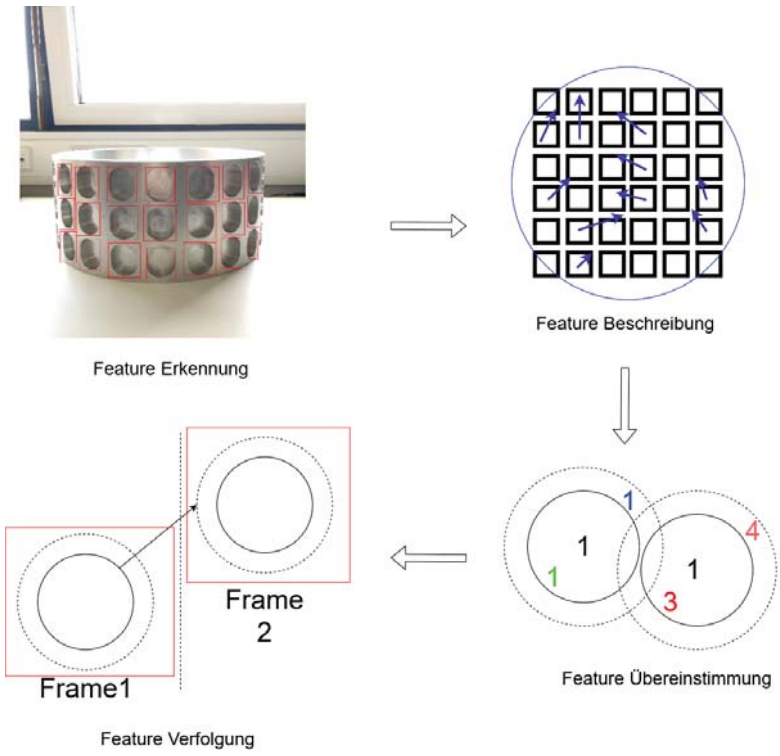

Abbildung 2.2: Punktfeatures: Prozess der Feature-Erkennung [2]

Während die Punktfeatures gut für die Erkennung von Features in zweidimeensionalen Räumen sind, eignet sich die Kantenerkennung für die Erkennung von Features in 3D besser. Das liegt daran, dass Objekte, die sich im 3D Raum befinden, von ihren Konturen geprägt sind und diese dem Objekt die eigentlichen Features definieren. Kanten werden in Bilder als Grenzen zwischen zwei verschiedenen Kontrasten, Intensitäten und Texturen gesehen. Eine Kante wird als ein Bereich der rapiden Variation der Intensität definiert [2]. Ein Bild wird als ein *Höhenfeld* betrachtet, das Flächen enthält, wobei an Stellen mit großer Steigung oder im Bereich von dicht gestellten Höhenlinien sich in meisten Fällen die Kanten befinden. Ein *Höhenfeld* ist eine dreidimensionale Fläche auf der Werte einer zweidimensionaler Funktion abgebildet werden. In der Mathematik können Flächen die eine Richtung und Steigung haben, durch den Gradienten definiert werden [10].

$$J(x) = \nabla I(x) = (\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y})(x)$$

Der lokale Gradientvektor J zeigt in die Richtung des steilsten Anstiegs in der *Intensitätsfunktion*. Die *Intensitätsfunktion* bildet die Koordinaten (x, y) eines Pixels auf den Wert der Intensität ab. Die Größe des Vektors beschreibt die Steigung oder die Varianzstärke, während die Ausrichtung des Vektors in eine Richtung senkrecht zur lokalen Kontur zeigt. Ein Problem bei der Berechnung der Ableitung eines Bildes besteht darin, dass bei der Berechnung hohe Frequenzen betont werden und damit bestehendes Rauschen verstärkt wird. Deshalb wird vor der Berechnung des Gradienten zuerst das Bild mit einem Filter optimiert.

2.1.3 Objekterkennung

Nachdem im letzten Abschnitt die Schritte für die Bildverarbeitung und Feature-Erkennung beschreiben wurden, können diese Schritte verwendet werden, um eine *Objekterkennung* durchzuführen. Diese gilt immer noch als eines der schwierigsten Probleme im Bereich der Computer Vision [2]. Die Welt besteht aus trivialen, aber auch komplexen Objekten, die verschiedene Formen besitzen. Da der Komplexitätsgrad der Objekte unterschiedlich ist, ist es sehr schwer Bilder mit Bildern aus einer Datenbank zu vergleichen und Ähnlichkeiten zwischen den Bildern zu berechnen. Die Erkennungsmethoden

können sich deswegen von einander ziemlich unterscheiden. Im Fokus steht welche Features erkannt werden sollen. Wenn bekannt ist, welche Features erkannt werden sollen, kann eine Bildanalyse durchgeführt werden und Features, die im Bild erkannt wurden, können mit bekannten Features verglichen werden.

Die generelle Objekterkennung kann in zwei Kategorien aufgeteilt werden, nämlich die *Klassenerkennung* und die *Instanzerkennung* [2]. Bei der Klassenerkennung steht die Erkennung des Objektes und dessen Zuweisung zu einer Klasse im Fokus. So kann zum Beispiel ein Motorrad oder Auto im Bild zur Klasse “Fahrzeug” eingeordnet werden, da beide einige Features miteinander teilen. In der Instanzerkennung ist das Ziel ein bekanntes Objekt zu erkennen, das aus einer anderen Umgebung oder Perspektive betrachtet wird, dessen Merkmale und Form aber bereits bekannt sind. Im Weiteren wird die Instanzerkennung näher diskutiert.

Um eine oder mehrere Instanzen eines bekannten Objektes zu erkennen, muss das Erkennungssystem zuerst die Menge an Features des Objektes finden und diese in einer indexierten Struktur speichern, beispielsweise als *Suchbaum*. Ein *Suchbaum* ist eine Datenstruktur, bei der die Menge der zu durchsuchenden Elemente in einer Baumstruktur dargestellt ist. Beim Erkennungsprozess werden dann die Features aus dem neuen Bild extrahiert und gegen die gespeicherten Daten verglichen. Wenn eine bestimmte Anzahl an Übereinstimmungen gefunden wurde, muss diese auf potenzielle Fehler überprüft werden. Wegen der hohen Fehlerquote kann beispielsweise die sogenannte *Hough-Transformation* zur Erkennung der Übereinstimmung der geometrischen Ähnlichkeiten verwendet werden, die eine Methode zur Kantenerkennung darstellt [11]. Dabei wird die affine Transformation verwendet, um Daten aus der Datenbank mit den Daten zu vergleichen, die aus einem Bild extrahiert wurden. Eine affine Transformation ist eine Abbildung zwischen zwei affinen Räumen. Die von Lowe et al. verwendeten Features haben Informationen über die Position, Skalierung und Orientierung, was die Nutzung einer vierdimensionalen Ähnlichkeitstransformation ermöglicht. Jedes Feature beschreibt dabei die Position des Mittelpunktes, Skalierung und Orientierung des Objektes. Diese werden dann mit einem anderen Bild verglichen. Dabei werden die Umgebungen der Features analysiert um die Spitzenwerte in einer affinen Übereinstimmung auswählen zu können [11].

2.1.4 Harris Ecken- und Kantenerkennung

In der realen, dreidimensionalen Umgebung existieren zu viele diverse Objekte, als dass anhand des Kontextes des Bildes eine Erkennung der Objekte im Bild möglich wäre. Harris schlägt deswegen ein Computer Vision System vor, das Bewegungen einer Bildsequenz der Kamera analysiert [12]. Um ein explizites Tracking der Features im Bild zu erreichen, müssen die Features im Bild diskret sein und nicht aus einem Kontinuum wie Textur oder Kantenpixel folgen. Die Repräsentation der Kanten als eine Menge von Fragmenten einer geraden Linie als die diskrete Repräsentation der Features hat den Nachteil, dass in den Bildsequenzen die kurvigen Linien und Texturen in jedem Frame unterschiedlich fragmentiert werden und somit nicht verfolgbar sind.

Eine Lösung hierfür stellt die gleichzeitige Erkennung von Kanten und Ecken in einem Frame dar [12]. Die Verbindungen bestehen dann aus Kanten, die sich auf den Ecken begegnen. Um die Verbindungen von Kanten und Ecken zu erkennen, wird die Moravec-Eckenerkennung durchgeführt [13]. Bei der Moravec-Eckenerkennung wird ein lokaler Bereich im Bild betrachtet. Dabei werden die durchschnittlichen Änderungen in der Intensität des Bildes festgestellt, die aus der Bewegung des lokalen Bereichs in die verschiedene Richtungen folgt. Es werden drei Fälle betrachtet: bei lokalen Bereichen mit konstanter Intensität werden alle Verschiebungen des lokalen Bereiches kleine Änderungen der Intensität liefern, da der betrachtete lokale Bereich zur einer Einheit gehört. Wenn der lokale Bereich im Bild eine Kante berührt, dann wird die Bewegung entlang der Kante eine kleine Änderung der Intensität bewirken, aber eine Bewegung die senkrecht zur Kante ist, wird eine große Änderung der Intensität bewirken. Im dritten Fall ist das Eck des lokalen Bereiches eine Ecke oder ein isolierter Punkt; dabei werden alle Bewegungen eine große Änderung der Intensität bewirken. Eine Ecke kann somit erkannt werden, wenn eine minimale Änderung der Bewegung eine große Änderung der Intensität verursacht.

Bei der Moravec-Eckenerkennung treten verschiedene Probleme auf. Die Ergebnisse sind anisotropisch, da nur die diskrete Menge von Bewegungen auf jeweils 45 Grad betrachtet werden. Dies kann vermieden werden, indem alle Bewegungen in Bezug auf den Ursprung betrachtet werden. Das Ergebnis kann zudem Rausch enthalten, da

der lokale Bereich binär und rechteckig ist. Die Nutzung eines kreisförmigen lokalen Bereiches, wie zum Beispiel ein Gauß-Filter, minimieren diesen Effekt [12].

2.1.5 Visuelle Odometrie

Visuelle Odometrie [14] ist ein Methode, die mit Hilfe einer Videoeingabe die Bewegung der Kamera abschätzt. Aus der Videoeingabe werden zwischen zwei nachfolgenden Bildsequenzen die Features verknüpft und auf *Bildtrajektorien* abgebildet. Eine *Bildtrajektorie* bezeichnet den Pfad, der bei der Verfolgung eines Features entsteht. Anhand der Kamerabewegung und der Feature-Verfolgung werden dann Pfade erstellt, welche die Bewegung repräsentieren. Dies erstellt eine sogenannte *visuelle Odometrie*, die Bewegungsdaten anhand von visuellen Eingaben berechnet.

Nach Nister [14] werden in jedem Bild der Videoeingabe Harris-Ecken erkannt. Das Bild wird mit 8 Bits per Pixel dargestellt. Danach wird die Stärke s der Kantenerkennung berechnet. Nachdem die Ecken erkannt wurden, wird die *Non-maximum suppression* Methode verwendet, um Featurepunkte zu definieren. In Abbildung 2.3 wird die Non-maximum suppression Funktion dargestellt. Non-maximum suppression geht den Pfad einer Kante entlang und filtert nur die maximalen Werte des Pixels der am Pfad liegt, während die Nachbarpixel auf null gesetzt werden. Ein Featurepunkt auf einem Pixel wird dann angenommen, wenn die Stärke s größer ist als alle anderen Pixel in eine $5x5$ Umgebung um den Pixel.

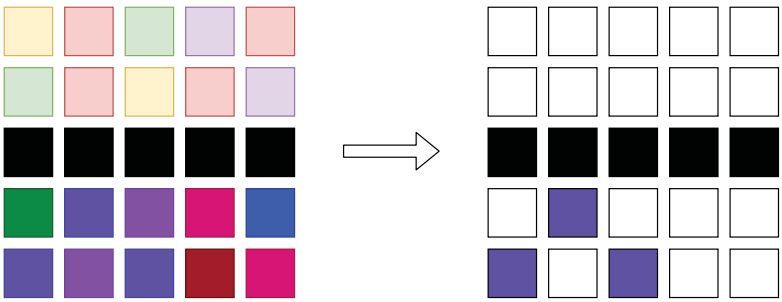

Abbildung 2.3: Non-maximum suppression

Nachdem die Features extrahiert wurden, wird im nächsten Frame nach Übereinstimmung mit den Features gesucht. Features werden in allen Frames erkannt, es werden

nur Features in den jeweiligen Frames miteinander verknüpft. Eine normalisierte Korrelation wird dann über einen $11x11$ Fenster genutzt, um potenzielle Treffer auszuwerten. Die Entscheidung, welche der Treffer akzeptiert werden, wird mit der *gegenseitige Konsistenzprüfung* durchgeführt. Jedes Feature hat mit Features aus anderen Frames eine normalisierte Korrelation. Es wird nun in einem anderen Frame nach Featuren mit der höchsten normalisierten Korrelation gesucht. Wenn dieser gefunden wird, wurde ein Treffer gefunden. Nur diejenigen Paare mit einem Treffer werden als ein valider Treffer akzeptiert. Aus dem Treffern wird dann die Umgebung verknüpft und Pfade werden zwischen dem Frames erstellt.

2.2 Machine Learning

In diesem Kapitel werden grundlegenden Konzepte von Machine Learning diskutiert [15] [16].

2.2.1 Grundlagen

Machine Learning gehört zum wissenschaftlichen Gebiet der Künstlichen Intelligenz und befasst sich mit Algorithmen und statistischen Modellen, die von Computersystemen verwendet werden, um eine Problemstellung zu lösen ohne explizite Angabe der Anweisungen wie das Problem gelöst werden kann. Murphy definiert Machine Learning als eine Menge von Methoden, die automatisch in einer Datenmenge nach bekannten Mustern suchen und dann anhand erkannter Muster die Struktur der zukünftigen Daten abschätzen können oder in der Lage sind Entscheidungen unter Unbewusstheit zu treffen [15].

Murphy teilt Machine Learning in zwei Typen auf: beim *überwachten Lernen* ist das Ziel eine Zuordnung der Eingaben x zu den Ausgaben y zu finden, wobei eine Menge von gekennzeichneten Ein- und Ausgabepaaren $D = \{(x_i, y_i)\}_{i=1}^{N}$ gegeben ist. Hierbei ist D die Trainingsmenge und N ist die Anzahl an Trainingsbeispielen. Jede Trainingseingabe x_i ist meistens ein D-dimensionaler Vektor, deren Elemente *Features*, *Attribute*

oder *Kovariablen* heißen. In der Regel kann x_i auch ein komplexeres Objekt sein. Die Ausgabevariable kann verschiedene Formen annehmen, aber in der Regel ist y_i eine nominelle Variable aus einer endlichen Menge $y_i \in \{1, .., C\}$, oder eine reelle Zahl. Wenn y_i ein nomineller Wert ist, wird dieses Problem als Klassifikation oder *Mustererkennung* bezeichnet, wenn y_i ein reeller Wert ist wird das Problem *Regressionsanalyse* genannt.

Bei der Klassifikation ist das Ziel, eine Zuordnung der Eingaben x zu den Ausgaben y zu erstellen, dabei ist $y \in \{1, ..., C\}$, wobei C die Anzahl der Klassen bezeichnet. Wenn $C = 2$, dann heißt diese Klassifikation die *Binäre Klassifikation*, da zwei Werte angenommen werden können, während eine mehrklassige Klassifikation $C > 2$ voraussetzt. Mit der Klassifikation kann beispielsweise der Wert einer Funktion abgeschätzt werden. Beispiel: gegeben sei eine unbekannte Funktion $y = f(x)$ und das Ziel des Lernens ist es, anhand der gekennzeichneten Trainingsmenge die Funktion abschätzen zu können und Schätzungen des Ergebnisses zu berechnen. So kann das Ergebnis für neue Eingaben vorhergesehen werden. Ein Anwendungsbeispiel für die Klassifikation ist die Bildklassifikation, in der die Eingabe ein Bild ist und die Ausgabe dann eine der bestehenden Klassen ist. Die Regressionsanalyse besteht aus den gleichen Konzepten, die von der Klassifikation verwendet werden. Der einzige Unterschied ist, dass bei der Regressionsanalyse die Ausgabevariable ein stetiger Wert ist, während bei der Klassifikation die Variable diskret ist. Ein Anwendungsbeispiel ist die Vorhersage des Aktienmarktes für einen bestimmten Zeitraum [15].

Der zweite Typ von Machine Learning heißt *unüberwachtes Lernen*, bei dem nur die Eingabedaten $D = \{x_i\}_{i=1}^{N}$ gegeben sind. Ziel ist es in diesem Eingabedaten Muster zu finden. Bei unüberwachtem Lernen wird die Ähnlichkeit der einzelnen Daten aus der Datenmenge analysiert und daraus werden Daten die sehr Ähnlichkeit zu einander sind in einem entsprechenden Cluster zugeordnet.

2.2.2 Deep Learning

Deep Learning ist ein Teilgebiet von Machine Learning und versucht komplexe Eingabedaten durch mehrere Schichten zu vereinfachen und aus den vereinfachten Daten wiederum Schlussfolgerungen zu ziehen. Ein Beispiel für ein Deep Learning Modell ist

das mehrlagige Perzeptron (siehe Abbildung 2.4). Ein mehrlagiges Perzeptron ist eine mathematische Funktion, die eine Eingabemenge auf einen Ausgabewert abbildet. Die Hauptfunktion eines mehrlagigen Perzeptrons bildet sich aus der Verknüpfung mehrerer einfacher mathematischer Funktionen. Jede Anwendung der mathematischen Funktion erstellt eine neue Repräsentation der Eingabedaten.

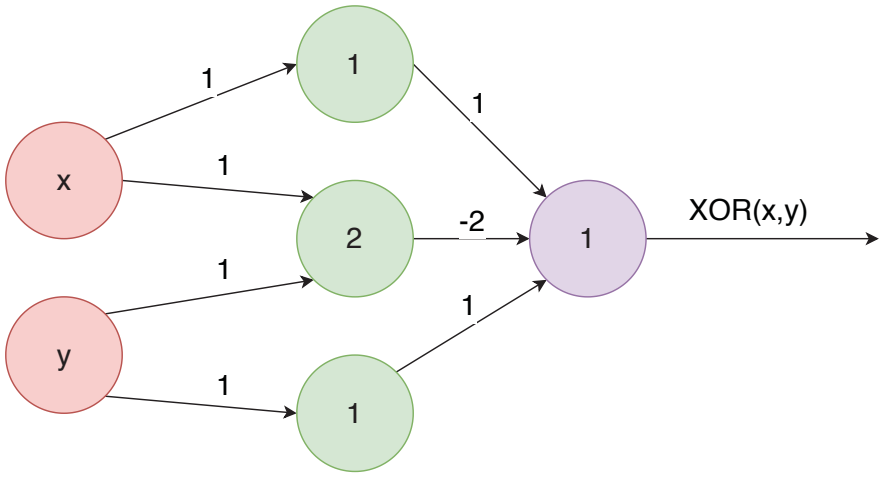

Abbildung 2.4: Mehrlagiges Perzeptron

Die Eingabedaten können dabei beliebig komplex werden (siehe Abbildung 2.5). Als Beispiel kann ein Bild betrachtet werden, das als Eingabe benutzt wird. Das Bild wird als Pixelmenge betrachtet. Aus dieser Menge ein Objekt im Bild zu erkennen ist eine sehr komplexe Aufgabe. Deswegen werden bei Deep Learning mehrere Schichten verwendet, um die Datenrepräsentation zu vereinfachen [16]. So wird die Eingabe in der sichtbaren Schicht definiert und repräsentiert die tatsächlichen Daten, in diesem Beispiel sind das die Pixel des Bildes. Danach besteht das Netzwerk aus einer Folge von versteckten Schichten. Die versteckten Schichten können in diesem Fall bestimmte Features extrahieren, die im Bild vorkommen. Diese Schicht heißt *versteckt*, weil die in dieser Schicht vorkommenden Daten nicht von der Eingabe stammen, sondern der Algorithmus feststellen muss welche Konzepte hilfreich sind, um die Beziehung zwischen dem betrachteten Daten zu analysieren. Bei den gegebenen Pixeln kann die erste Schicht die Kanten im Bild erkennen, sodass sie die Helligkeit der Nachbarpixel vergleicht. Weiter kann anhand der Repräsentation aus der ersten Schicht die zweite

Schicht nach Ecken und Konturen suchen, da diese aus den Kanten bestimmt werden können. Danach kann die dritte Schicht ganze Objekte erkennen, in dem die Menge der Konturen und Ecken die in der zweiten Schicht erkannt wurden, ein spezifisches Objekt beschreiben. Zuletzt kann die Beschreibung für dann das erkannte Objekt ausgegeben werden [17].

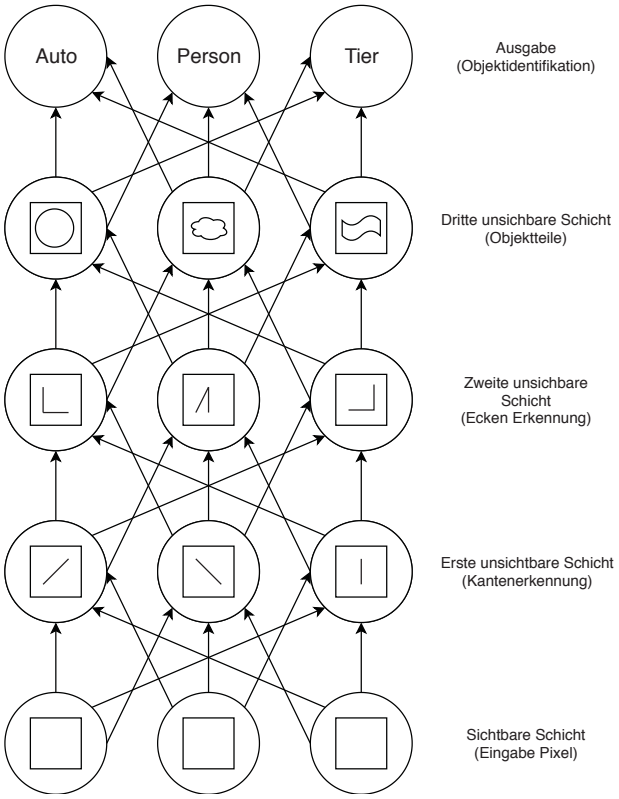

Abbildung 2.5: Neuronales Netz [17]

2.2.3 Convolutional Networks

Convolutional Networks sind eine spezielle Form von neuronalen Netzen, die geeignet sind Daten zu bearbeiten, die in einer Gitterform verteilt sind. Ein Beispiel für Daten die aus einer Gitterform zusammengefasst sind, sind digitale Bilder die als eine Matrix dargestellt werden können. Convolutional Networks eignen sich gut für die Verarbeitung von Bilddaten, die als 2D-Gitter betrachtet werden können. Bei Convolutional Networks

wird statt einer Matrizenmultiplikation eine mathematische Faltung verwendet (siehe Kapitel 2.1.1).

Convolutional Networks implementieren drei Konzepte, die ein Machine Learning Systeme verbessern [16]: *spärliche Eingaben*, *Parameterteilung* und *Äquivarianz-Repräsentation*. Gewöhnliche Schichten der neuronalen Netze verwenden Matritzenmultiplikation mit einer Matrize von Parametern mit einem separaten Parameter, der die Interaktion zwischen der Eingabeeinheit und jeder Ausgabeeinheit beschreibt. Das bedeutet, dass jede Ausgabeeinheit mit jeder Eingabeeinheit interagiert. Convolutional Networks verwenden im Gegensatz zu gewöhnlichen neuronalen Netzen spärliche Eingaben. Um dies zu erreichen, wird der *Filterkern* verkleinert, sodass dieser kleiner als die Eingabe des Convolutional Networks ist. Ein *Filterkern* ist eine quadratische Matrize, die in der Faltungsoperation als Filtermaske verwendet wird. Beispielsweise kann bei der Bildverarbeitung ein großes Bild vorliegen, für die Featureerkennung jedoch werden nur separate kleine Bereiche im Bild betrachtet, da die zu erkennenden Objekte nur aus einer kleinen Menge an Pixel bestehen. Wenn zum Beispiel die Kanten in einem Bild zu erkennen sind, könnte ein Filterkern verwendet werden, der viel kleiner ist, als das gesamte Bild, da die Kanten aus einer kleinen Menge an Pixel bestehen als das gesamte Bild. Das bedeutet, dass weniger Parameter gespeichert werden müssen und das Modell insgesamt weniger Speicher benötigt. Ebenso benötigt die Berechnung der Ausgabe eine kleinere Anzahl an Operationen. Diese Verbesserungen der Effizienz sind groß, da die Algorithmen in der Praxis häufig die Laufzeit von $O(n \times m)$, wobei n die Eingaben und m die Ausgaben sind.

Bei der Parameter-Teilung wird derselbe Parameter für mehr als eine Funktion in einem Modell verwendet. In gewöhnlichen neuronalen Netzen wird jedes Element der Gewichtmatrize genau einmal – bei der Berechnung der Ausgabe eine Schicht – verwendet. Es wird mit einem Element der Eingabe multipliziert und danach nicht mehr verwendet. In einem Convolutional Network wird der Filterkern auf jede Position der Eingabe angewandt. Die Parameter-Teilung, die von der Faltungsoperation verwendet wird bedeutet, dass anstatt bei dem Lernprozess für jede Position ein separate Menge an Parameter auf jede Position zum lernen benutzt wird, der Lernprozess nur auf einer Menge ausgeübt wird.

Bei der mathematischen Faltung führt diese Art von Parameter-Teilung dazu, dass die Schicht die Eigenschaft der Äquivarianz in Bezug zur Translation annimmt. Eine Funktion ist äquivariant, wenn auf die Eingabe der Funktion eine Änderung vorgenommen wurde, die Ausgabe sich auch verändert. Besonders ist eine Funktion $f(x)$ äquivariant zur eine Funktion $g(x)$, wenn $f(g(x)) = g(f(x))$. Nun im Falle der Faltung, wenn eine Funktion g die Funktion ist, die Eingaben verschiebt, dann ist die Faltungsfunktion äquivariant zu der Funktion g. Zum Beispiel kann eine Funktion I betrachtet werden, die den Helligkeitswert an der Koordinaten eines Pixels im Bild berechnet. Sei g eine Funktion die eine Bildfunktion auf eine andere Bildfunktion abbildet, sodass $I' = g(I)$ die Bildfunktion ist mit $I'(x, y) = I(x - 1, y)$. Diese Funktion verschiebt jeden Pixel von I um einen Pixel nach rechts. Wenn diese Transformation zuerst auf I angewandt wird, und danach die Faltungsoperation auf I durchgeführt wird, so ist das Ergebnis gleich, wie wenn die Faltungsoperation auf I' zuerst angewandt wird, und dann die Transformation g auf die Ausgabe. Dies ist nützlich wenn eine Funktion bekannt ist die Features in einer bestimmten Umgebung eines Pixels bewertet.

2.3 Mixed Reality

In diesem Kapitel werden Anwendungen und Technologien im Bereich Mixed Reality eingeführt und diskutiert.

2.3.1 Mixed Reality

Mixed Reality kann als Oberbegriff für die verschiedenen Typen von virtueller oder erweiterter Realitäten gesehen werden (siehe Abbildung 2.6) [18]. Es existieren verschiedene Arten von Realitäten, die computergestützt generiert werden. So hat zum Beispiel die *virtuelle Realität (VR)* eine Umgebung die komplett künstlich erschaffen wurde, während *Augmented Reality (AR)* versucht die reale und die virtuelle Welt zusammen zu bringen und die virtuellen Objekte in der echten Welt darzustellen. Die Virtuelle Realität ist dabei eine elektronische Simulation einer realen Umgebung, die über eine Kopfbrille und

kabelgebundene Kleidung erlebt werden kann, so dass der Nutzer in realistischen dreidimensionalen Situationen interagieren kann [19]. Im folgenden Abschnitt wird Augmented Reality weiter betrachtet.

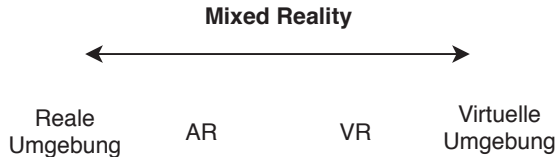

Abbildung 2.6: Mixed Reality [18]

2.3.2 Augmented Reality

Augmented Reality ist eine Variation der virtuellen Realität. Während ein Benutzer in einer virtuellen Realität komplett abgekapselt von der Außenwelt ist, nimmt er bei Augmented Reality seine reale Umgebung wahr [20]. Idealerweise werden virtuelle Objekte in der realen Umgebung platziert und es besteht das Gefühl, dass die virtuellen und realen Objekte in der Umgebung koexistieren. Ferner wird Augmented Reality, als ein System definiert, das folgende Charakteristiken besitzt: Kombination von realen und virtuellen Komponenten, Echtzeit-Verwendbarkeit und Bewusstsein des AR-Systems über die drei räumlichen Dimensionen. Im Weiteren werden die Eigenschaften von Augmented Reality diskutiert [20].

Um Augmented Reality Applikationen zu entwickeln, werden verschiedene Displaytechnologien, Bewegungstracker, Grafikcomputer und Software benötigt. Augmented Reality Displays zeigen dem Benutzer eine Mischung aus der realen und virtuellen Umgebung. Die reale Umgebung wird meistens durch Kameras aufgenommen, aus der dann die reale und virtuelle Umgebung erstellt und auf den Displays angezeigt wird. Van Krevelen unterscheidet zwei Darstellungsmethoden, nämlich die Nutzung einer Kamera als die einzige Eingabequelle der realen Umgebung, deren Videostream auf ein undurchsichtiges Display abgebildet wird und zweitens, die Nutzung von optischen Displays die transparent sind, auf die dann die virtuelle Komponente projiziert wird [21]. Die Nutzung der Kameraeingabe, um die reale Umgebung auf einem Display darzustellen, hat den

Vorteil, dass die Kosten geringer sind als bei transparenten Displays. Weiter können die realen Objekte in der Videoeingabe manipuliert werden. So können unerwünschte Markierungen entfernt werden, die oft für die Platzierung von virtuellen Objekten verwendet werden. Verzögerungen bei der Wahrnehmung der virtuellen und realen Umgebung können behoben werden und so dem Benutzer ein besseres Erlebnis bieten. Dennoch bestehen auch Nachteile bei der Nutzung der Kameraeingabe. Beispielsweise ist die Auflösung im Vergleich zu transparenten Displays viel geringer. Weiter hängt die Größe des Sichtfeldes von der Kamera ab. Eine weiter Möglichkeit für die Darstellung ist es *transparente Displays* zu verwenden. Diese stellen die reale Umgebung unverändert dar. Weiter ist diese Displaytechnlogie im Gegensatz zur Kameraeingabe viel sicherer zu verwenden, da der Benutzer immer die reale Umgebung unverändert sieht. Trotzdem werden Kameras benötigt, um die reale Umgebung zu analysieren um entsprechend die virtuellen Objekte in der gemeinsamen Sicht darzustellen. Da das Display transparent ist, erschwert es die realistische Darstellung der virtuellen Objekte, gemeinsam mit dem realen Objekten, da wegen der Transparenz die Helligkeit und der Kontrast auf dem Display reduziert wird. So werden die virtuellen Objekte in einer realen Umgebung, die sehr hell ist, schwer dargestellt. Ein weiteres Problem ist das Sichtfeld, da der Benutzer die gesamte reale Umgebung sieht, werden die virtuellen Objekte an den Kanten des Displays abgeschnitten. Die Nutzung der jeweiligen Displaytechnoligen hängen sehr vom Anwendungsfall ab, so eignet sich die Videoeingabe zum Beispiel mehr für Mobilgeräte wie Handys, während sich die transparenten Displays eher für die Nutzung bei Augmented Reality Brillen eignen.

Um die virtuellen Objekte auf den Display anzuzeigen, muss das AR-System sich seiner Umgebung bewusst sein. Im Idealfall ist das AR-System in der Lage, bewusst die reale Umgebung wahrzunehmen und die Bewegungen des Benutzers zu verfolgen [21]. Die Position sollte in Variablen gespeichert werden, die zu der realen Umgebung identisch sind, während auch der sogenannte Roll-Nick-Gier-Winkel des Benutzers betrachtet wird und dementsprechend in Variablen gespeichert wird. Die in diesen Variablen gespeicherten Informationen werden von der Sensorik des AR-Systems erhoben. Aus den Sensordaten werden dann Modelle der Umgebung erzeugt, sodass das System die Position des Benutzers und sein Sichtfeld kennt und in dem Umgebungsmodell die

virtuellen Objekte platzieren kann. Es werden üblicherweise 3D Modelle der geometrischen Umgebung erstellt. Um diese zu erstellen können verschiedene *manuelle* und *automatische* Methoden verwendet werden. Bei der manuellen Erzeugen der Umgebung wird das 3D-Modell strikt nach einer bekannten Umgebung, wie zum Beispiel einem bestimmten Zimmer, entworfen und in die Software eingebettet. Dieser Ansatz ist leider sehr eingeschränkt, da das AR-System nur in zuvor definierten Umgebungen funktioniert. Bei der automatischen Modellerzeugung der Umgebung wird anhand von Kameradaten und anderen Sensordaten ein Umgebungsmodell erzeugt. Somit wird der Freiheitsgrad erhöht und die Umgebung, in der sich der Benutzer befindet, kann beliebig verändert werden. Eine automatische Erzeugung des Modells der Umgebung wird von Patel demonstriert, wobei für die Erstellung des Modells ein Entfernungsmesser verwendet wurde, um Notizen in der Umgebung zu platzieren [22].

Ein weiterer wichtiger Aspekt in Augmented Reality ist die Verfolgung der Benutzerbewegung. Während sich der Benutzer in der realen Umgebung bewegt, soll diese Bewegung dementsprechend in dem Umgebungsmodell abgebildet werden. Nach Kervelen [21] kann die Verfolgung des Benutzers *mechanisch* und *magnetisch*, *inertial* und *optisch* erfolgen. Bei der mechanischen Verfolgung des Benutzers wird der Benutzer mit einem mechanischen Gerät verbunden, das sich bewegt wenn der Benutzer seine Position im Raum verändert. Seine Bewegungen werden dann anhand der Differenz der initialen Position des Geräts berechnet. Die erste mechanische Bewegungsverfolgung funktionierte, indem an einer Raumdecke ein Gerät befestigt war und der Benutzer mit diesem Gerät mechanisch verbunden war [23]. Die magnetische Bewegungsverfolgung berechnete die Entfernung zwischen zwei elektromagnetischen Feldern. Somit konnte der Benutzer anhand der Stärke des elektromagnetischen Feldes in der Umgebung positioniert werden [24]. Die inertielle Bewegungsverfolgung verwendet Beschleunigungssensoren und ein Gyroskop, um die Bewegung des Benutzers zu verfolgen. Diese Instrumente befinden sich auch in modernen Smartphones und werden zum Beispiel bei der Navigation benutzt, um die Richtung des Smartphones zu bestimmen. Anhand der konstanten Bewegungen des Gerätes kann berechnet werden in welche Richtung sich das Gerät bewegt. Bei der optischen Verfolgung werden die Konzepte aus dem Kapitel 2.1 benutzt, um den Benutzer zu verfolgen. Anhand der Kameraeingabe werden

Features detektiert und eine Bewegung in den aufeinander folgenden Frames festgestellt. Naimark [25] verwendet Marker, um die Bewegung durch die Umgebung festzustellen, während Chia [26] auf Marker verzichtet und die Feature die aus dem nachfolgenden Bildern der Kameraeingabe betrachtet. Das Verzichten von Markern hat den Vorteil, dass kein Aufwand betrieben werden muss, um die Marker zu erstellen und in der Umgebung zu positionieren. Genauso ist die Abhängigkeit von Markern nicht mehr vorhanden, da die Bewegungsverfolgung auch in Räumen stattfinden kann, die keine Marker enthalten. In modernen Augmented Reality Systemen werden mehrere dieser Methoden kombiniert. So wird zum Beispiel in dem Framework ARKit die inertielle und optische Bewegungsverfolgung verwendet, um die Bewegungen des Benutzers zu verfolgen.

Um ein Nutzen von den virtuellen Objekten in der realen Umgebung zu gewinnen, wird eine Benutzerschnittstelle vorausgesetzt. Während bei klassischen Computersystemen die typischen Eingabegeräte Maus und Tastatur sind, werden bei Augmented Reality andere Methoden vorausgesetzt. So werden nach Ishii und Ullmer [27] greifbare Benutzerschnittstellen vorgestellt. Es werden physikalische Objekte verwendet, die mit einem virtuellen Objekt überlagert werden, sodass die Schnittstelle ein Gefühl von Haptik vermittelt. Dieses überlagerte physikalische Objekt kann dann als Schnittstelle zur Eingabe für das Augmented Reality System verwendet werden. Eine weitere Schnittstelle wird von Buchmann [28] vorgeschlagen, indem elektronische Handschuhe verwendet werden, die bei der Berührung eines virtuellen Objektes an den Fingerspitzen der Handschuhe eine kleine Vibration auslösen und somit haptisches Feedback erzeugen. Eine alternative zu Handschuhen ist nach Mayol [29] die Hände des Benutzers anhand von der Kameraeingabe zu beobachten und Gestenerkennung zu betreiben. Eine wichtige Schnittstelle in jedem Computer System ist die Schnittstelle für die Texteingabe. Um eine Texteingabe zu ermöglichen bieten sich verschiedene Methoden an die vom Gerätetyp abhängig sind. So können bei mobilen Geräten wie zum Beispiel dem Handy, die vom Betriebssystem vorhandene Tastatur verwendet werden, während sich bei Augumented Reality Brillen Spracherkennung besser eignet.

2.3.3 Frameworks

In diesem Abschnitt werden zwei Frameworks vorgestellt, mit denen Objekte in einem Augumented Reality Videodatenstrom erkannt und deren Bewegung verfolgt werden kann.

Apple ARKit

Auf der Worldwide Developers Conference 2017, einer von Apple organisierten Entwicklerkonferenz, wurde zum ersten Mal das Framework ARKit vorgestellt. ARKit ist ein Augumented Reality Framework, das für mobile Geräte entwickelt wurde. Genauer wurde das Framework von Apple, für das mobile Betriebssystem iOS entwickelt. Das Framework bietet Entwicklern eine Programmierschnittstelle, das den Entwicklern die Entwicklung von Augumented Reality Anwendungen erleichtert. ARKit ist *closed source* Software, Entwickler haben folglich keinen genauen Einblick über die Funktionsweise des Frameworks. Im weiteren Text werden die Merkmale des Frameworks diskutiert.

Das grundlegende Merkmal von ARKit ist die Möglichkeit die Sensordaten des Gerätes in Echtzeit zu ermitteln. Mit den erfassten Daten können dann die relative Position des Gerätes in der realen Umgebung bestimmt werden. ARKit verwendet visuelle Odometrie [14], sodass mit Kamerabildern und Bewegungsdaten des Beschleunigungssensor festgestellt werden kann, wo sich das Gerät in der realen Welt befindet. Weiter kann ARKit an bestimmten Eigenschaften die Umgebung interpretieren, in der sich das Gerät befindet. So können Oberflächen in der realen Umgebung erkannt werden. Ein weiteres Merkmal von ARKit ist die Möglichkeit einen *Hit Test* durchzuführen. Dies bedeutet, dass ein virtuelles Objekt mit der erkannten Oberfläche anhand von ihren berechneten Positionen überprüfen kann, ob sie sich gegenseitig berühren. Um die virtuellen Objekte in der realen Umgebung realistisch darzustellen, schätzt ARKit die Lichtintensität der Umgebung. So werden virtuelle Objekte in die reale Umgebung mit derselben Lichtintensität dargestellt. Und zuletzt hat ARKit einen *Renderer*, der die virtuellen Objekte in der realen Umgebung darstellt.

ARKit ist eine sitzungsorientierte API. Zuerst wird eine ARSession erstellt, ein Objekt in dem alle Berechnungen stattfinden, die für eine Augmented Reality Umgebung benötigt werden. Um genauer zu definieren was in der ARSession betrachtet wird, kann man dem Objekt ARSession ein ARSessionConfiguration-Objekt übergeben. Ein ARSessionConfiguration-Objekt enthält die Konfiguration, die in der Anwendung verwendet werden soll. So kann z.B. konfiguriert werden, dass in der ARSession ein Gesicht erkannt werden soll. Um die Sitzung zu starten beinhaltet jedes ARSession Objekt eine $run()$ Methode. Mit der Ausführung der Methode beginnt auch die Bearbeitung der Kamera- und Sensordaten. Nachdem die Bearbeitung abgeschlossen ist, werden ARFrames ausgegeben. In einem ARFrame-Objekt befindet sich der Zustand der Sitzung und alle benötigten Daten für das Rendering der virtuellen Objekte. Mit den Informationen, die man von den ARFrames bekommt kann die Augumented Reality Szene manipuliert werden. Ein Beispiel dafür sind ARAnchros. Dies sind Positionen in der virtuellen Welt, die durch ARKit relativ zu den realen Positionen platziert werden. So kann man mit Hilfe von ARAnchors virtuelle Objekte in der realen Welt positionieren. Dies sind die vier wichtigsten Klassen, die bei der Entwicklung von ARKit Anwendungen benötigt werden. Da in dieser Ausarbeitung die Objekterkennung im Fokus steht, wird nachfolgend diese Funktion von ARKit diskutiert.

Bevor ein Objekt mit dem Framework ARKit erkannt werden kann, benötigt man einen Scan des Objektes. Dieser kann mit Hilfe einer Anwendung durchgeführt werden, die auf den Entwicklerseiten von ARKit verfügbar ist. Die Anwendung erstellt eine Punktwolke, die Merkmale des Objektes markieren. Anhand dieses Scans kann nun eine Objekterkennung durchführt werden. Um eine Objekterkennung durchzuführen muss die Datei, die beim Scan erstellt wurde, in das Projekt eingebunden und in den Ordner ARResourceGroup gespeichert werden. Danach kann für das ARSession-Objekt, ein ARSessionConfiguation-Objekt erstellt werden, indem das gescannte Objekt als ARReferenceObjects angegeben ist. Nachdem kann die ARSession beginnen. Gesucht wird nach einem ARObjectAnchor, der erstellt wird wenn das Objekt erkannt wurde. Ein ARObjectAnchor ist ähnlich zu einem ARAnchor, der die Position des erkannten Objektes in der virtuellen Umgebung liefert.

TensorFlow

TensorFlow [30] ist ein Framework, das sich für den Entwurf von Machine Learning Algorithmen eignet. Gleichzeitig ist TensorFlow auch ein Framework für die Ausführung dieser Algorithmen. Eine Berechnung, die mit TensorFlow definiert wurde, kann auf einer Vielzahl von Systemen ausgeführt werden, wobei für die verschiedenen Systeme keine oder nur kleine Änderungen durchgeführt werden müssen. Das Framework ist flexibel konfigurierbar und kann für eine Vielzahl von Algorithmen verwendet werden, dazu gehören auch Algorithmen für das Training von neuronalen Netzen und die Algorithmen für die Auswertung von neuen Daten anhand der trainierten Daten.

TensorFlow entstand aus dem Google Brain Projekt, das im Jahr 2011 gestartet wurde. Ziel des Projektes war es die Nutzung großer, skalierbarer neuronaler Netze auf Google Produkte zu untersuchen. In den frühen Phasen des Projektes entstand das erste verteilte, skalierbare System für das Training und die *Inferenz* mit dem Namen DistBelief [31]. Der Begriff *Inferenz* bezieht sich auf den Prozess der Ausführung eines Modells auf einem Gerät, um Vorhersagen basierend auf Eingangsdaten zu treffen [30]. Mit DistBelief wurden Forschungen im Bereich von unüberwachtem Lernen, Bildklassifikation, Objekterkennung und vielen mehr betrieben. Aus diesen Forschungen haben sich praktische Anwendungen für die Google Produkte erwiesen. So wurden mit Hilfe von DistBelief neuronale Netze in den beliebtesten Google Produkten eingebettet.

Mit der Erfahrung die im DistBelief Projekt die Entwickler gesammelt haben, wurde der Nachfolger mit dem Namen TensorFlow entwickelt [30]. In TensorFlow werden unter Verwendung von Datenfluss-orientierten Modellen Berechnungen beschrieben, die dann auf verschiedene Hardware abgebildet werden. So können Training und Inferenz auf mobilen, aber auch auf großen, verteilten Clustersystem durchgeführt werden.

Im Folgenden werden das Programmiermodell und die Grundkonzepte von TensorFlow näher erläutert. Eine Berechnung wird in TensorFlow mit einem gerichteten Graphen beschrieben, der aus einer Menge von Knoten und Kanten besteht. Der Graph stellt eine Abfolge von Berechnung dar, mit der Knoten ihren Zustand aktualisieren können. In einem TensorFlow Graphen hat jeder Knoten keinen oder mehrere Eingaben und Ausgaben und stellt eine Instanz einer Operation dar. Werte, die entlang der üblichen

Kanten im Graph fließen, heißen *Tensors*. Tensors sind beliebig große Arrays in dem der grundlegende Datentyp bei der Erstellung des Graphen definiert ist. Eine *Operation* besitzt einen Namen und stellt eine abstrakte Berechnung dar. Eine solche Berechnung kann zum Beispiel die Matrizenmultiplikation sein. Eine Operation kann Attribute beinhalten, diese müssen bei der Erstellung des Graphen definiert werden. Ein *Kernel* ist eine bestimmte Implementierung eine Operation die auf einem bestimmten Gerät ausgeführt werden kann. Die kompilierte TensorFlow-Binärdatei definiert eine Menge von Operationen und Kernels, die über einen Registrierungs-Mechanismus verfügbar sind. Diese Menge kann mit zusätzlichen Operationen und Kernels erweitert werden. Client-Programme interagieren mit TensorFlow. Die grundlegende Funktion der Sitzung ist Run, das eine Menge von Ausgabenamen nimmt, die berechnet werden müssen und eine optionale Menge an Tensors die als Eingabe des Graphen auf bestimmten Ausgabeknoten platziert werden. Mit diesen vorhandenen Parametern kann nun die TensorFlow-Implementierung die transitive Hülle aller Knoten berechnen, die ausgeführt werden müssen, um die angeforderten Ausgaben zu erhalten. Nachdem die transitive Hülle berechnet wurde, kann eine Reihenfolge festgelegt werden, in der die Knoten fehlerfrei nach ihren Abhängigkeiten ausgeführt werden.

Die Hauptkomponente in einem TensorFlow-System ist ein *Client*, der über die Sitzungschnittstelle mit ein *Master* kommuniziert [30]. Eine weitere wichtige Komponente sind die *Arbeitsprozesse*, bei welchen jeder arbeitende Prozess dafür verantwortlich ist, den Zugang zu einem Gerät für die Berechnung zu stellen. Außerdem ist der Arbeitsprozess für die Ausführung der Knoten des Graphen in der richtigen Reihenfolge zuständig. Die Reihenfolge wird vom Master bestimmt. Die Geräte betreiben alle Berechnungen und jeder Arbeitsprozess ist für ein oder mehrere Geräte verantwortlich. Als Gerät werden CPUs oder GPUs verwendet. Für Tensor-Arrays werden verschiedene Datentypen unterstützt, wie z. B. IEEE float und double Typen [30]. Die voraussichtlichen Größen hängen von dem Gerät ab auf dem der Tensor vorliegt. Deswegen wird die passende Größe mit Hilfe von einer Speicherallokation durchgeführt, die für jedes Gerät individuell implementiert werden muss.

3

Anwendungsfall: Serviceprozesse von cyber-physischen Produktionssystemen

Ein *cyber-physisches System* ist ein System von zusammenarbeitenden Recheneinheiten, die stark mit der physikalischen Umgebung interagieren. Cyber-physische Systeme nehmen die laufenden Prozesse in der physikalischen Umgebung wahr und stellen Datenzugriffs- und Datenverarbeitungsdienste über das Internet bereit [32]. In dieser Ausarbeitung wird der Fokus auf cyber-physische Systeme gesetzt, die in der Industrieproduktion eingesetzt werden. Diese werden auch *cyber-physische Produktionssysteme* genannt. Ein *cyber-physisches Produktionssystem* ist eine Klasse von Systemen, die eine Interaktion zwischen der Cyber- und Physischenkomponente ermöglichen und in der Industrieproduktion eingesetzt werden [33]. Die Cyberkomponente besteht meistens aus einer Software die das bestehende System verwaltet, während die physische Komponente aus gewöhnlichen Maschinen aus der Industrie besteht. Ziel cyber-physischer Produktionssysteme ist es bestehende Prozesse in der Produktion zu automatisieren und einen Grad an autonomen Eingreifen des Systems bei Abweichungen von den Prozessschritten zu erreichen [34]. Im Folgenden werden cyber-physische Produktionssysteme eines Pharmaverpackungsmaschinenherstellers betrachtet.

Im Rahmen eines studentischen Projekts wurde ein Assistent für die Wartung von Pharmaverpackungsmaschinen entwickelt. Der Assistent bestand aus eine Anwendung die in einer Augmented Reality Brille ausgeführt wurde. In dem Assistent wurden die jeweiligen Arbeitsschritte angezeigt, die bei einer Wartung durchgeführt werden müssen. Eine Interaktion zwischen den Assistenten und der Maschine existierte nicht. Um das zu ändern, muss der Assistent Informationen über die Maschine, bzw. Maschinenteile, verfügen. Mit

einer Objekterkennung wäre es möglich, eine Interaktion zwischen dem Assistenten und der Maschine zu erreichen. Somit hätte auch der Assistent die Möglichkeit dem Benutzer genauer zu Beschreiben was er in einem Prozessschritt zu erledigen hat [35, 36].

In der Pharmaindustrie werden sehr hohe Anforderungen vorausgesetzt. So werden von dem Bundesministerium der Justiz und für Verbraucherschutz bestimmte Richtlinien für die Verpackung von Arzneimittel gegeben. Ein Beispiel dafür ist das Gesetz über den Verkehr mit Arzneimitteln. Auch Hersteller der Verpackungsmaschinen sind an diese Richtlinien gebunden. Aus diesen Richtlinien folgt auch, dass die Maschinen nicht mit Aufklebern markiert werden können, sondern alle Markierungen in den einzelnen Maschinenteilen dort eingeprägt sind, wo keine mikrobiologische Reinigung erforderlich ist. Dieses Problem wird in folgendem Anwendungsfall näher betrachtet.

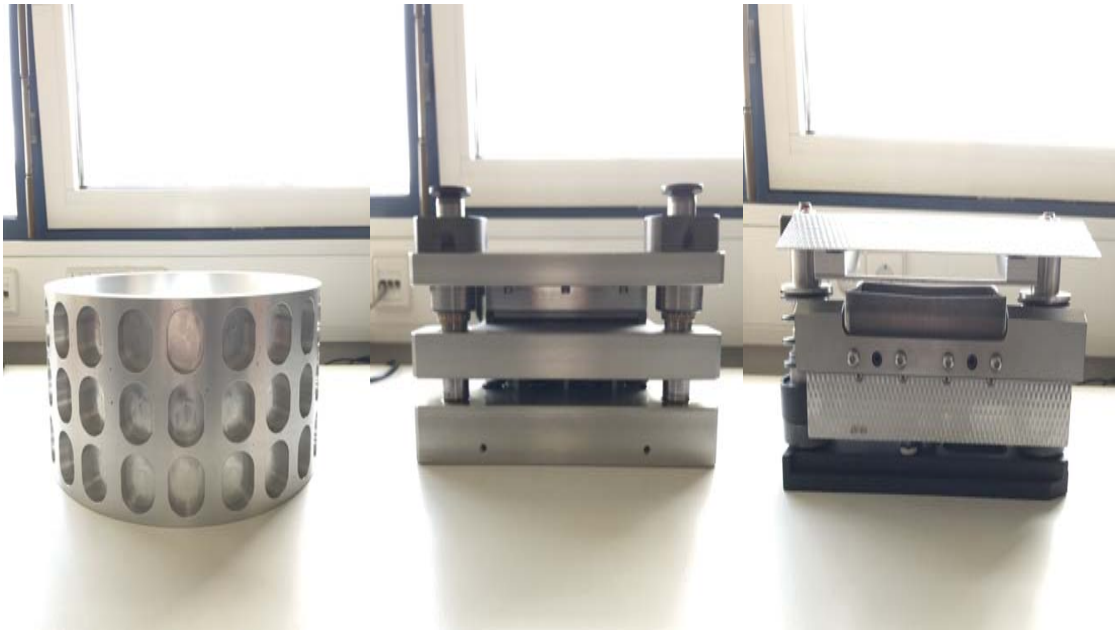

Abbildung 3.1: Formatteile

Für dieser Ausarbeitung wurden drei *Formatteile* einer Verpackungsmaschine zu Verfügung gestellt, die in Abbildung 3.1 zu sehen sind. *Formatteile* sind auswechselbare Maschinenteile einer Verpackungsmaschine, die für das Verpacken eines bestimmten Produktes entworfen wurden. Formatteile bieten eine hohe Flexibilität im Verpackungsprozess, da mit dem Tausch eines Formatteiles ein anderes Produkt mit derselben

Verpackungsmaschine verpackt werden kann. Dies impliziert, dass die Formatteile im Verpackungsprozess oft ausgetauscht werden. Formatteile, die in dieser Ausarbeitung näher betrachtet werden, unterscheiden sich in ihrer geometrischen Form und sind von einander in einem geringen Grad nur auf der Oberfläche ähnlich.

Ziel der Objekterkennung ist es dem Nutzer bei einer Wartung der Maschine zusätzliche Informationen über die Maschinenteile zu geben und bei einem fest definiertem Prozess den Nutzer zu warnen, wenn er vom Prozessablauf abweicht und zum Beispiel Maschinenteile verwendet, welche für diesen Prozessschritt nicht erlaubt sind. Die Objekterkennung kann folglich einem Prozessmanagementsystem eine Rückmeldung liefern, sowie dem Benutzer direkt Informationen über die Maschinenteile übermitteln (vgl. Abbildung 3.2).

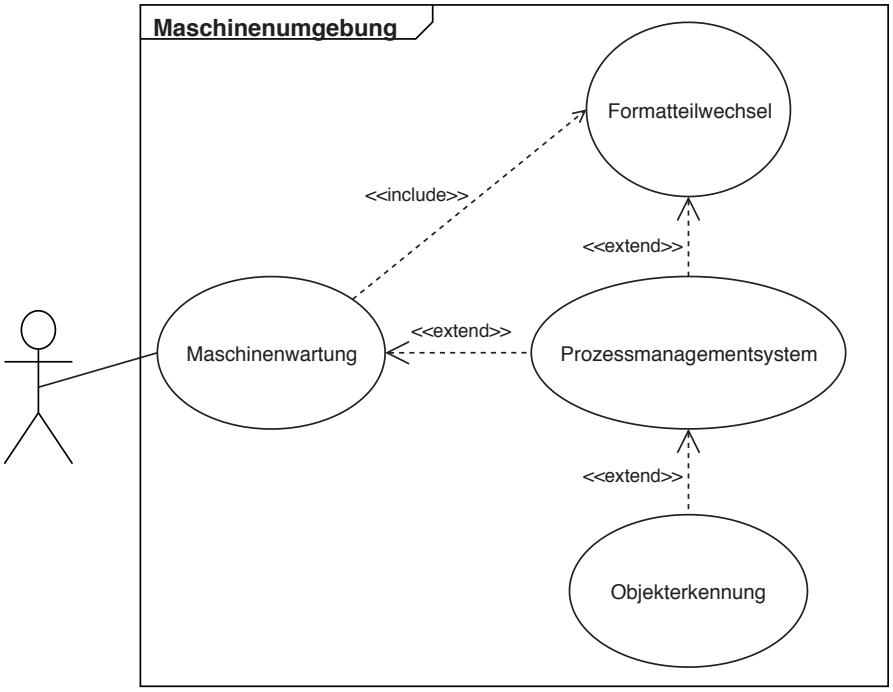

Abbildung 3.2: UML Anwendungsfalldiagramm einer Objekterkennung

Um festzustellen, ob sich eine Objekterkennung der Formatteile eignet, wird in Kapitel 5 ein Experiment beschrieben, das die Erkennung von 3D-Objekten evaluiert. Die betrachteten 3D-Objekte sind die Formatteile der Verpackungsmaschine. Alle Formatteile werden in gleichen Bedingungen in diesem Experiment getestet und für die Objekterken-

nung werden die Frameworks ARKit und TensorFlow Lite verwendet. Die Ergebnisse des Experimentes sollen dabei eine feste Aussage liefern, ob die Frameworks für eine Objekterkennung brauchbar sind und welches der beiden Frameworks sich für diesen spezifischen Anwendungsfall besser eignet.

4

Implementierung

In diesem Kapitel werden die Implementierung der Anwendungen ARKit und TensorFlow Lite erläutert.

4.1 TensorFlow Lite Anwendung

In diesem Abschnitt wird die TensorFlow Lite Anwendung zur Objekterkennung vorgestellt. Anhand eines Bildes oder einer Kameraeingabe kann ein Objekterkennungsmodell identifizieren, welche Objekte in dem betrachteten Bild sind, und kann Informationen über ihre Positionen innerhalb des Bildes liefern. Das Ergebnis besteht aus einer bekannten Menge von Objekten, die das Objekterkennungsmodell erkennen kann. Ein Objekterkennungsmodell wird trainiert, um das Vorhandensein und die Position mehrerer Objekte in einer Kameraeingabe zu erkennen. Für jedes erfasstes Objekt gibt das Modell ein Array von vier Zahlen zurück. Diese vier Zahlen sind Koordinaten eines Rechtecks, welches das erkannte Objekte umrahmt. Ein vom Objekterkennungsmodell erkanntes Objekt ist beispielsweise in Abbildung 4.1 zu sehen.

4.1.1 Vorbereitung des Modells

In dieser Ausarbeitung wurde die Python API von TensorFlow verwendet. Zuerst wird der Quellcode von TensorFlow und TensorFlow Modelle heruntergeladen. TensorFlow wird danach aus dem Quellcode kompiliert. Für die Verwendung der TensorFlow API werden zusätzlich die Python-Bibliotheken *matplotlib*, *pillow*, *lxml* und *jupyter* benötigt.

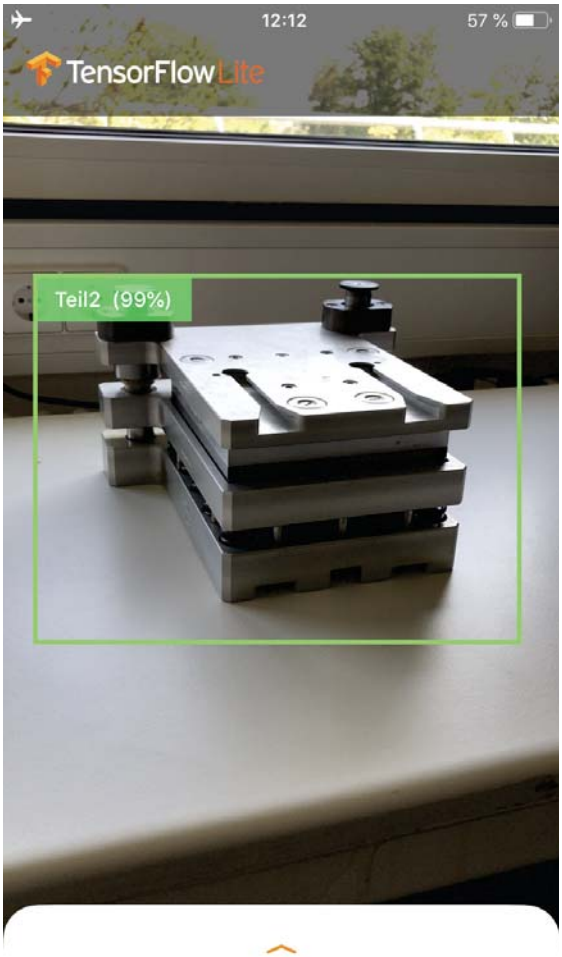

Abbildung 4.1: TensorFlow: Erkanntes Objekt mit Annotation

Die Tensorflow Object Detection API verwendet Protobufs, um Modell- und Trainingsparameter zu konfigurieren. Bevor das Framework genutzt werden kann, müssen die Protobuf-Bibliotheken kompiliert werden. Protobuf ist ein Methode, um Daten strukturiert zu deserialisieren und zu serialisieren. Protobuf stellt fest wie die zu serialisierenden Daten strukturiert werden sollen, indem Sie Protobuf-Nachrichtentypen in .proto-Dateien definieren. Jede Protobuf-Nachricht ist ein kleiner logischer Datensatz von Informationen, der eine Reihe von Name-Wert-Paaren enthält. Nachdem alle benötigten Komponenten installiert wurden, müssen nun die Trainingsdaten vorbereitet werden. Der vollständigen Traningsprozess wird in Abbildung 4.2 dargestellt. Für die Erstellung der Bilder, die für

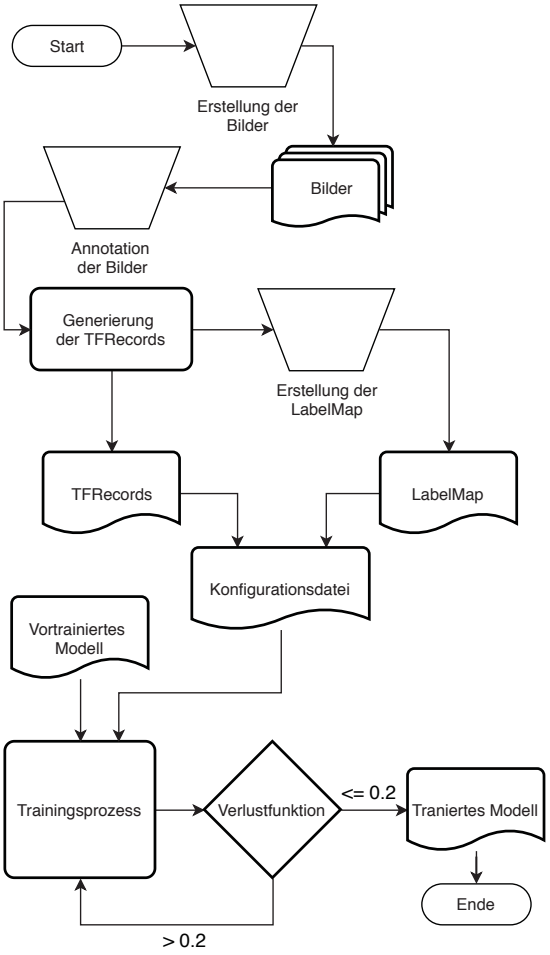

Abbildung 4.2: TensorFlow: Flussdiagramm des Traningsprozesses

das Training verwendet werden, wurde ein iPhone 7 verwendet. Für jedes Objekt, dass erkannt werden soll, wurden 120 Bilder verwendet, da diese Anzahl an Bildern für das Training des Modells empfohlen wird [37]. Diese Bilder werden dann einzeln bearbeitet, sodass alle Bilder die Größe von 300×300 Pixel haben, da die Eingabe des Modells die Größe 300 hat.

Danach müssen Objekte markiert werden, die in dem Bild vorkommen. Dies wird für alle Bilder durchgeführt und in diesem Fall wurde die Anwendung *LabelImg* verwendet. *LabelImg* erstellt für jedes Bild anhand der Markierung eine *XML* Datei die Daten über die Markierung besitzt. Aus den *XML* Dateien werden dann *TFRecord*s für jede *XML* Datei erstellt. Das *TFRecord* Format ist die Eingabe für den TensorFlow Trainer. Nachdem die

*TFRecord*s erstellt wurden muss eine *Label Map* Datei erstellen. Eine *Label Map* Datei sagt dem Trainer was jedes Objekt bezeichnet und definiert eine Abbildung zwischen den *IDs* und den Namen des Erkannten Objekts. Nun sind die Trainingsdaten vorbereitet und das Training kann gestartet werden [37]. Es wird zudem ein vortrainiertes Modell benötigt, auf dem ein neuer Trainingsprozess mit den generierten Daten gestartet wird, sodass das Modell nach dem Training die Objekte erkennen kann. In dieser Ausarbeitung wird das Modell SSD [38] verwendet, dass auf dem COCO-Datensatz [39] trainiert wurde. Nachdem das vortrainierte Modell heruntergeladen wurde, kann der Trainingsprozess konfiguriert werden. In der Konfigurationsdatei muss die Anzahl an Klassen angepasst werden, die erkannt werden sollen. In diesem Fall sind es 3 Klassen, die erkannt werden sollen. Weiter muss das *TFRecord* und die *Label Map* referenziert werden. Nun kann der Trainingsprozess mithilfe von der TensorFlow *train.py*-Skripte gestartet werden. Die Dauer des Trainingsprozesses hängt dabei von der Leistung der Hardware ab, auf der der Trainingsprozesses läuft. In diesem Beispiel wurde das Training auf einer Grafikkarte betrieben, die $6GB$ VRAM besitzt. Das Abbruchkriterium für das Training ist erreicht, wenn die Verlustfunktion im Trainingsprozess einen Wert von ungefähr 0.2 erreicht [37]. Dieser Wert wird periodisch vom Skript *train.py* ausgegeben. In diesem Fall hat das Training 3 Stunden gedauert, bis der Wert der Verlustfunktion angefangen hat zu stagnieren. Nachdem das Training beendet wurden ist, muss aus den Trainingsdaten nun ein *frozen inference Graph* erstellt werden der dann den Klassifier für die Objekterkennung der bereits trainierten Objekte besitzt.

4.1.2 Implementierung

Die implementierte Anwendung baut auf der bestehende TensorFlow Lite Beispielanwendung für Objekterkennung auf, die für die Plattform iOS entwickelt wurde. Um das im letzten Abschnitt beschriebene Modell in dieser Beispielanwendung zu verwendet, muss der *frozen inference Graph* in das *.tflite* Format umgewandelt werden. TensorFlow stellt ein Skript zu Verfügung, das diese Umwandlung durchführt. Nachdem das Modell in das *.tflite* Format umgewandelt wurde kann das Modell aus der Beispielanwendung mit den

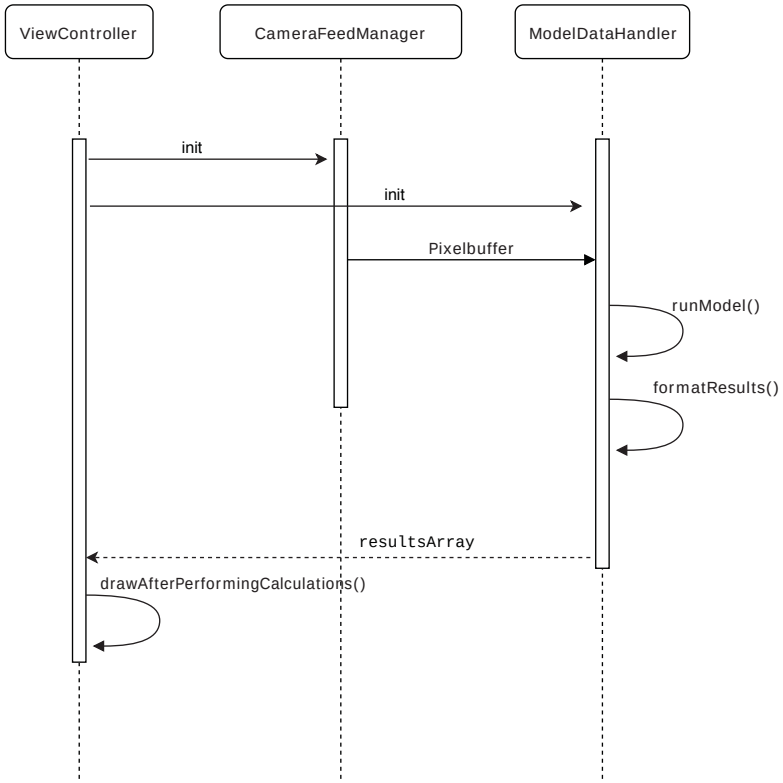

Abbildung 4.3: TensorFlow: Sequenzdiagramm der Objekterkennung in der Anwendung

eigenen Modell ersetzt werden. Abbildung 4.3 zeigt den Objekterkennungsablauf der Anwendung mit Hilfe eines UML-Sequenzdiagramms.

In der Datei *ModelDataHandler.swift* wird der Enum *MobileNetSSD* auf den Namen des Modells angepasst und die *Label Map* wird mit der neuen ersetzt. Dies Schritte genügen, um die Objekterkennung mit dem eigenen Modell zu starten. Der genaue Ablauf stellt sich dabei wie folgt dar: in der Datei *CameraFeedManager.swift* wird die Kameraeingabe verwaltet. Da das Modell nur Bilder der Größe 300×300 als Eingabe annimmt müssen die Bilder auf die Dimension 300×300 Pixel verkleinert werden, bevor sie durch das Modell bewertet werden. Die Funktion *runModel()* in der Datei *ModelDataHandler.swift* übernimmt den Pixelpuffer des *CameraFeedManagers* und verkleinert die Größe des Bildes auf 300×300 Pixel, da die Eingabe des Modells die Größe 300 hat . Weiter wird in der gleich Funktion die Inferenz auf das Modell durchgeführt und die Ergebnisse werden in einer *Result*-Struktur gespeichert. In dieser Struktur befindet sich ein Array

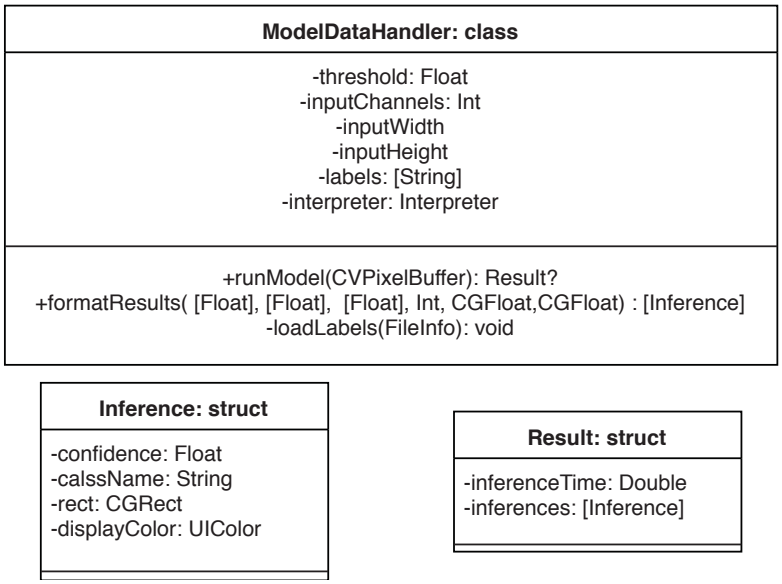

Abbildung 4.4: TensorFlow: UML-Klassendiagramm für ModelDataHandler.swift

von Inferenzen, die aus dem Bild folgen. Die Inferenz wurde als eine Datenstruktur definiert, die eine formatierte Ausgabe des Modells darstellt. So ist das Feld *rect* in dieser Struktur beinhaltet und repräsentiert die Box, die das erkannte Objekt umrahmt. Nachdem das Bild vom Modell bewertet wird, gibt das Modell diese Struktur zurück, in der die Informationen der Objekterkennung vorliegen. Bevor die Struktur *Results* zurückgegeben wird, wird noch die Funktion *formatResults()* aufgerufen, die dazu dient Ergebnisse zu filtern, in denen die *confidence* unter einem Schwellwert liegt. In dieser Implementierung liegt der Schwellwert bei 0.98, sodass das Ergebnis nur akzeptiert wird wenn das Objekt mit einer Wahrscheinlichkeit von 98% erkannt wurde. Dieser Wert wurde eigenständig ausgewählt um eine Fehlertoleranz zu erlauben. Nachdem von dem Modell die Daten geliefert wurden, kann die Funktion *drawAfterPerformingCalculations()* aufgerufen werden und die Box aus dem Feld *rect* der Struktur *Inference* auf der Kameraeingabe zeichnen. Dies sieht dann wie in der Abbildung 4.1 aus. Abbildung 4.4 zeigt das UML-Klassendiagram des *ModelDataHandler*.

4.2 ARKit Anwendung

In diesem Abschnitt wird die ARKit Anwendung zur Objekterkennung vorgestellt. ARKit kombiniert Bewegungssensoren, Kameraaufnahmen und Szenenverarbeitung, um die Erstellung einer AR-Anwendung zu vereinfachen. ARKit kann genutzt werden, um verschiedene Arten von AR-Erlebnissen zu erzeugen. Durch die Erfassung der räumlichen Features eines Objektes kann ein Modell erstellt werden, das zur Objekterkennung verwendet werden kann. Dieses erstellte Modell wird in einem Objekterkennungsprozess als Referenz verwendet, um das genaue Objekt in der realen Umgebung des Benutzers zu erkennen. Abbildung 4.5 zeigt das Ergebnis eines Objekterkennungsprozesses.

Abbildung 4.5: ARKit: Erkanntes Objekt mit Annotation

4.2.1 Vorbereitung des Modells

Im ARKit Framework wird das Modell für die Objekterkennung in einem **.arobject** Format gespeichert. Das Format speichert eine Punktwolke, die aus dem Features eines gescannten Objektes stammen. Das Objekt erhält man, indem man mit der Beispielsanwendung für *Scanning and Detecting 3D Objects* aus der Apple Dokumentation ein Objekt scannt und danach exportiert. Der ganze Prozess wird in folgenden Schritten beschrieben.

In Abbildung 4.6 werden die einzelnen Schritte zur Erstellung eines Referenzobjektes dargestellt. Nachdem die Anwendung gestartet wurde, beginnt die Analyse der Umgebung (1). Falls ein Objekt in der Mitte der Kamerasicht erkannt wurde, wird eine Box um das Objekt gezeichnet (2). Diese Box definiert den Rahmen, in dem der Scanvorgang stattfindet. Diese Box kann dann von einem Benutzer beliebig vergrößert und bewegt werden, sodass das Objekt in den Rahmen der Box passt und zentriert ist. Danach kann durch Druck auf einen Button der Scanvorgang begonnen werden. Das verwendete Gerät, in diesem Fall ein iPhone, muss bewegt werden, sodass alle Seiten des Objektes gescannt werden. Am Rahmen der Box werden hervorgehobene Seiten angezeigt, wenn dieser Bereich erfolgreich gescannt wurde (3). Es müssen alle Seiten der Box hervorgehoben werden, damit der Scan erfolgreich abgeschlossen werden kann. Nachdem der Scan abgeschlossen ist, muss der Ursprung des Objektes angepasst werden (4). Danach kann der Scan gespeichert werden. Wenn der Scanvorgang abgeschlossen ist, kann der Scan in einer Anwendung verwendet werden.

4.2.2 Implementierung

Für die Implementierung der Anwendung wurde die Vorlage *Augmented Reality App* aus Xcode verwendet. Nachdem das Projekt erstellt wurde, müssen die Scans der Objekte, welche erkannt werden sollen, in den Ordner *Objects.arresourcegroup* platziert werden. In diesem Ordner wird bei dem Erkennunsprozess nach Übereinstimmungen gesucht. Von der Vorlage werden die wichtigsten Methoden bereitgestellt. Es wird ein *ARWorldTrackingConfiguration*-Objekt erstellt das dazu dient, die gescannten Objekte

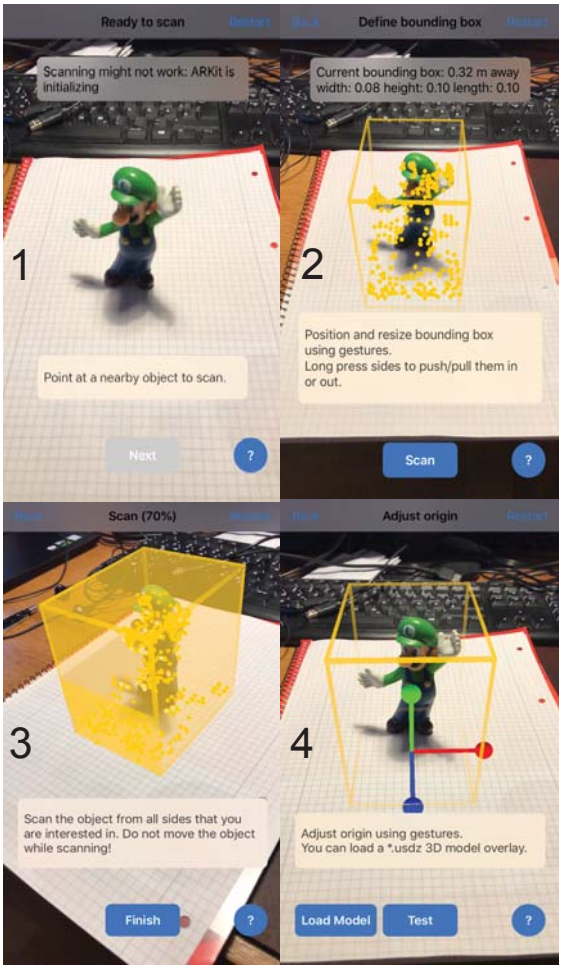

Abbildung 4.6: ARKit: Modellerstellungsprozess

die sich im *Objects.arresourcegroup* befinden, in die Konfiguration der *ARSession* einzubinden. Ein ARSession-Objekt koordiniert die wichtigsten Prozesse, die ARKit durchführt um ein Augmented-Reality-Erlebnis zu schaffen. Während der *ARSession* wird dann die Funktion *renderer()* periodisch aufgerufen und es wird überprüft, ob *ARObjectAnchor* erstellt wurde. Wenn ein *ARObjectAnchor* erstellt wurde bedeutet das, dass der Scan eine Übereinstimmung mit dem betrachteten Objekt ist und das Framework das Objekt erkannt hat (vgl. Abbildung 4.5). Einen besseren Überblick über die Anwendung gibt das UML-Klassendiagramm in Abbildung 4.7.

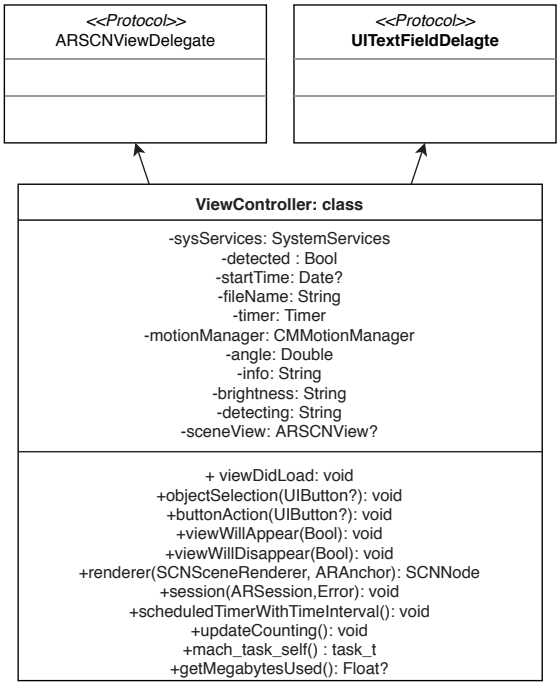

Abbildung 4.7: ARKit: UML-Klassendiagramm für ViewController.swift

4.3 Gemeinsame Komponenten

Da in dieser Arbeit der Fokus auf der Messung verschiedener Parameter liegt, wird der Quellcode für die Messung der zwei Anwendungen gleich implementiert. Dabei ist der Quellcode für die Messungen, als auch die grafische Benutzerschnittstelle in beiden Anwendungen, identisch. Weiter verwenden beide Anwendungen die Bibliothek **iOS-System-Services**, welche für die Messung der Prozessorauslastung zuständig ist. Diese Bibliothek wird mit einem Podfile-Eintrag in das Projekt eingebettet. Die Poddatei ist eine Spezifikation, die die Abhängigkeiten eines oder mehrerer Xcode-Projekte beschreibt.

Beim Start beider Anwendungen wird die Initialisierung von verschiedenen, für die Messung benötigten Variablen, durchgeführt. Danach wird die grafische Benutzerschnittstelle angezeigt (vgl. Abbildung 4.8). In der Benutzerschnittstelle werden Daten eingegeben, die nicht automatisiert erfasst werden können. So wird die Richtung, Heiligkeit und welches Maschinenteil betrachtet wird manuell eingegeben. Nach Auswahl der Richtung

wird anschließend eine Schedulerfunktion *scheduledTimerWithTimeInterval()* gestartet, welche wiederum die Funktion *updateCounting()* in einem Intervall von 0.1 Sekunden startet. In der Funktion *updateCounting()* werden abschließend die erfassten Daten im *CSV*-Format in eine Textdatei geschrieben.

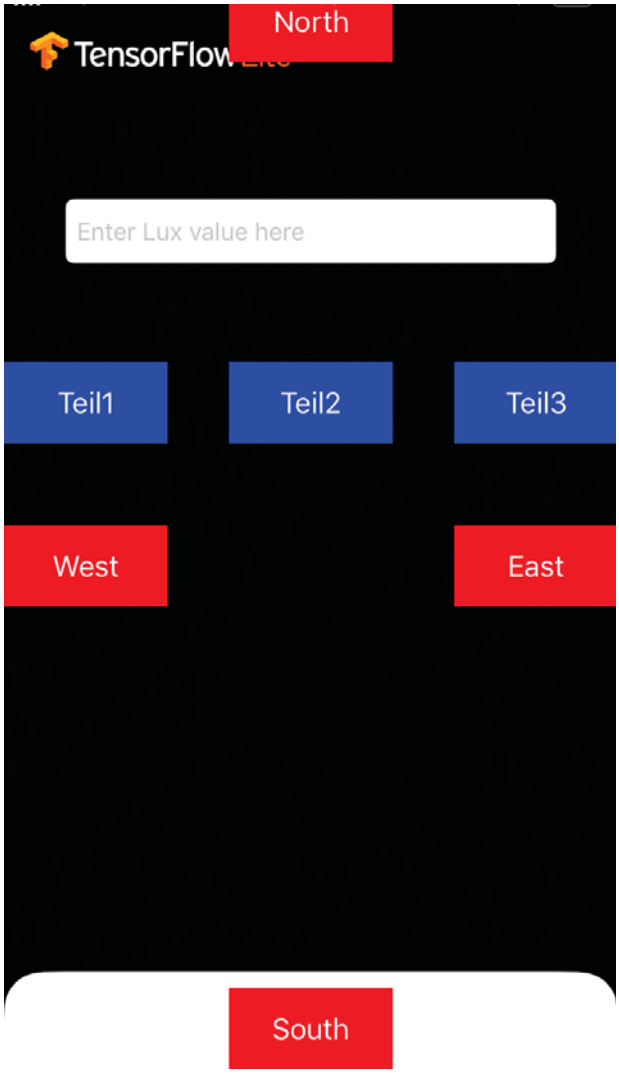

Abbildung 4.8: Grafische Benutzerschnittstelle

5

Evaluation

In diesem Kapitel wird die Evaluation der betrachteten Frameworks mit Hilfe eines Experiments durchgeführt. Das Experiment wird definiert, geplant und ausgeführt. Der gesamte Experimentalprozess orientiert sich dabei an Empfehlungen von Wohlin [40].

5.1 Umfang des Experiments

In diesem Abschnitt wird der Umfang des Experimentes definiert. Im Vordergrund stehen die Aspekte *Untersuchungsobjekt*, *Ziel*, *Perspektive*, *Qualitätsfokus* und *Kontext* des Experimentes.

5.1.1 Zielsetzung

Ziel dieses Experimentes ist es anhand von empirischen Daten die Leistung der Frameworks ARKit und TensorFlow Lite bei Erkennung von 3D-Objekten festzustellen. Dabei ist es wichtig die beiden Frameworks einzeln zu untersuchen, um Schlussfolgerungen über die Leistung der jeweiligen Frameworks ziehen zu können. Die bei diesem Experiment erfassten Ergebnisse sollen Software-Entwicklern die Auswahl des Frameworks erleichtern.

Untersuchungsobjekt. Die Untersuchungsobjekte in diesem Experiment sind die Frameworks ARKit und TensorFlow Lite.

Ziel. Das Ziel dieses Experimentes ist es anhand von empirischen Daten die Untersuchungsobjekte auf ihre Leistung zu evaluieren. Das Experiment gibt ein Überblick was erwartet werden kann; im Sinne der Leistung des einzelnen Frameworks.

Perspektive. Das Experiment wird aus der Perspektive eines Software-Entwicklers durchgeführt. Damit soll die Entscheidung erleichtert werden, welches der beiden Frameworks für die Entwicklung einer Anwendung im industriellem Kontext passend ist.

Qualitätsfokus. Im Mittelpunkt dieser Studie steht die Leistung des jeweiligen Frameworks bei der Durchführung einer 3D-Objekterkennung. Dabei wird die *Prozessorauslastung*, *Speichernutzung*, die *benötigte Zeit* um ein 3D-Objekt zu erkennen sowie die *Erkennungsrate* betrachtet.

Kontext. Der Kontext dieses Experimentes ist die vorliegende Bachelorarbeit, die ihm Rahmen des Studiengangs Software Engineering vom Institut für Datenbanken und Informationssysteme der Universität Ulm angeboten wurde. Das Experiment wird als „Blocked subject-object study“ kategorisiert. Grund hierfür ist, dass zwei Frameworks vorhanden sind, die Objekte des Experiments repräsentieren und drei verschiedene 3D-Objekte die Subjekte des Experiments repräsentieren. Weiter handelt es sich bei diesem Experiment um ein Technologie-orientiertes Experiment, da keine Personen im Experiment involviert sind. Daraus folgt, dass das Experiment als deterministisch eingeordnet werden kann und deswegen als ein *Quasi-Experiment* einzuordnen ist. Ein *Quasi-Experiment* ist eine empirische Untersuchung ähnlich einem Experiment, wenn die Zuordnung von Behandlungen zu den Probanden nicht auf einer Zufallsgenerierung beruhen kann, sondern ergibt sich aus den Eigenschaften der Subjekte oder Objekte selbst [40].

5.2 Planung des Experiments

In diesem Abschnitt werden die benötigten Schritte diskutiert, welche vor dem Experiment definiert werden sollen.

5.2.1 Kontext Auswahl des Experiments

Der Kontext dieses Experimentes ist die vorliegende Bachelorarbeit. Somit handelt es sich um eine off-line Ausführung, da das Experiment außerhalb einer praktischen Umgebung (wie z.B. Unternehmen) durchgeführt wird. Das Experiment wird von dem Bacheloranden in einer definierter Umgebung ausgeführt und das Experiment befasst sich mit einem realen Problem, nämlich die Evaluierung der Leistung der Frameworks ARKit und TensorFlow Lite in einem industriellen Kontext.

5.2.2 Hypothesenformulierung

ARKit und TensorFlow Lite sind die Frameworks die näher betrachtet werden. Die Frameworks werden miteinander verglichen, um nachvollziehen zu können, welches der beiden Frameworks performanter ist. Deswegen muss die Hypothese genau aufgesetzt werden, um nach Evaluierung der aus dem Experiment gesammelten Daten Schlussfolgerungen ziehen zu können. Es ist schwer eine Annahme zu definieren, ob ARKit oder TensorFlow Lite performanter ist, ohne das Experiment auszuführen. Deswegen wird eine Null-Hypothese aufgesetzt, welche dann widerlegt werden soll. Weiter wird dann eine alternative Hypothese (Hypothese 1) definiert die als gültig betrachtet wird, wenn die Null-Hypothese widerlegt wurde.

Hypothese 0. *Beide Frameworks erreichen die gleiche Leistung. Das bedeutet, dass die Prozessorauslastung μ_p, Speichernutzung μ_s, Zeit für die 3D-Objekterkennung μ_t und ob das 3D-Objekt erkannt wurde μ_e für die Frameworks ARKit und Tensorflow Lite alle paarweise die gleichen Werte annehmen. Es gelten folgende Annahmen:*

$$\mu_{p_{tf}} = \mu_{p_{ak}}$$
$$\mu_{s_{tf}} = \mu_{s_{ak}}$$
$$\mu_{t_{tf}} = \mu_{t_{ak}}$$
$$\mu_{e_{tf}} = \mu_{e_{ak}}$$

Hypothese 1. *Beide Frameworks erreichen nicht die gleiche Leistung. Es gelten folgende Annahmen:*

$$\mu_{p_{tf}} <> \mu_{p_{ak}}$$

$$\mu_{s_{tf}} <> \mu_{s_{ak}}$$
$$\mu_{t_{tf}} <> \mu_{t_{ak}}$$
$$\mu_{e_{tf}} <> \mu_{e_{ak}}$$

Aus der Hypothese folgt, dass folgende Daten erfasst werden müssen: Prozessorauslastung μ_p, Speichernutzung μ_s, Zeit für die 3D-Objekterkennung μ_t und ob das 3D-Objekt erkannt wurde μ_e. Dies gilt für das jeweilige Framework. Die Prozessorauslastung μ_p wird in Prozenten gemessen, Speichernutzung μ_s wird in Megabytes gemessen, Zeit für die 3D-Objekterkennung μ_t wird in Sekunden gemessen und ob das 3D-Objekt erkannt wurde μ_e wird mit einem boolschen Wert `Wahr`/`Falsch` gemessen.

5.2.3 Auswahl der Variablen

Aus der Hypothesenformulierung kann gefolgert werden, dass die abhängigen Variablen Prozessorauslastung, Speichernutzung, Zeit die für die 3D-Objekterkennung benötigt wird und ob das 3D-Objekt überhaupt erkannt wurde. Weiter müssen die unabhängigen Variablen definiert werden. Die unabhängigen Variablen sind die Richtung, zu der das Objekt betrachtet wird, der Winkel des Handys zum Objekt und die Entfernung des Fotostativs zum Objekt.

5.2.4 Auswahl der Subjekte

Die Subjekte in diesem Experiment sind die 3D-Objekte, die mit dem jeweiligen Framework erkannt werden sollen. Es handelt sich um drei Maschinenformatteile einer Medikamentenverpackungsmaschine. Eine genauere Beschreibung der Maschinenformatteile findet sich in Kapitel 3.

5.2.5 Experimententwurf

Nachdem in den letzten Abschnitt die Hypothese, abhängige und unabhängige Variablen definiert wurden, ist es nun notwendig, das Experiment so zu gestalten, dass nach der

Durchführung des Experimentes die Null-Hypothese widerlegt werden kann. Um die Null-Hypothese zu widerlegen, wird eine statistische Analyse der gesammelten Daten durchführt. Weiterhin ist nach Wohlin [40] bei dem Experimententwurf darauf zu achten, dass die Aspekte *Zufälligkeit*, *Blockierung* und *Balancierung* betrachtet werden.

Zufälligkeit bei einem Experiment dient dazu, dass unerwünschte Nebeneffekte, die das Ergebnis beeinflussen können, soweit wie möglich zu reduzieren. Ein Beispiel dafür wäre die zufällige Gruppenzuordnung bei einem Experiment, in dem die Subjekte als Personen definiert sind. Da dieses Experiment als ein *Quasi-Experiment* einzuordnen ist und die Vorläufe alle deterministisch sind, kann der Aspekt Zufälligkeit ignoriert werden. Unter Blockierung versteht man die Eliminierung von unerwünschten Faktoren, die das Ergebnis beeinflussen könnten. In diesem Experiment wird die unabhängige Variable Prozessorauslastung definiert. Ein Einfluss auf die Prozessorauslastung könnte der Akkustand sein, da das Handy bei niedrigen Akkustand in einem Energiesparmodus wechseln kann und die Leistung damit beeinflusst werden könnte. Deswegen soll bei der Durchführung des Experimentes das Handy immer mit einem Ladekabel verbunden werden und den Akkustand auf 100% halten. Abschließend wird die Balancierung betrachtet. Balancierung ist eine Gleichstellung der jeweiligen Versuchen in einem Experiment [40]. In diesem Experiment ist die Balancierung mit einem hohen Grad vertreten, da 3 Subjekte vorhanden sind, nämlich die Maschinenteile auf die jeweils ein Versuch identisch durchgeführt wurde.

Nachdem die Aspekte Zufälligkeit, Blockierung und Balancierung auf das Experiment angepasst wurden, wird der Fokus auf den expliziten Entwurf des Experimentes gesetzt. Bei der Durchführung des Experimentes werden folgende unabhängige Variablen betrachtet: Die Richtung, aus der das Objekt betrachtet wird, der Winkel des Handys zum Objekt und die Entfernung des Fotostatives zum Objekt. Als Richtung versteht man die Richtung, in die das Handy im Bezug auf die Markierungen zeigt, die auf dem Tisch angebracht sind. Die Abbildung 5.2 zeigt wie auf dem Tisch die Markierungen gesetzt sind. Genauer wird dies in dem nächsten Unterabschnitt 5.2.6 definiert. Es werden vier Richtungen betrachtet, dass bedeutet das ein Faktor von 4 verschiedenen Messungen vertreten ist, welche alle durchlauft werden. Die Winkel die bei dem Versuchen betrachtet werden, hängen mit der Entfernung des Fotostativs und dem Maschinenteil das betrach-

tet wird. Die Entfernungen aus den die Versuche durchführt werden sind $45cm$, $60cm$ und $80cm$, wobei die Wahl der Winkel von der Entfernung abhängen. Diese Abhängigkeit folgt daraus, dass für die gegebenen Entfernungen die Kamera des Handys, das Maschinenteil komplett abdecken soll, und deswegen für die Entfernung von $45cm$ und $80cm$ einige Winkel bei der Messung ausfallen. Für die Entfernung von $45cm$ wird der Winkel von $40°$ betrachtet. Bei der Entfernung von $60cm$ wird der der Winkel von $45°, 60°$ und $90°$ betrachtet. Zuletzt werden die Messungen aus $80cm$ mit einem Winkel von $60°$ gemessen.

Zusammengefasst müssen bei jedem Maschinenteil alle Kombination der Versuche durchlaufen werden, d.h. insgesamt $4 * 3 * 4 = 48$ Versuche. Da bei den Versuchen einige Winkel nicht betrachtet werden bedeutet, dass pro Maschinenteil für das jeweilige Framework $4 * 1 * 1 + 4 * 1 * 3 + 4 * 1 * 1 = 20$ Messungen durchgeführt werden. Im Folgenden wird ein *fraktionsfaktorieller Entwurf* für das Experiment verfolgt [40]. Ein fraktionsfaktorieller Entwurf eignet sich für dieses Experiment, da die betrachteten Faktoren als faktorielles Muster wachsen. Genauer wird das Entwurfsmuster halber fraktionsfaktorieller Entwurf des 2^k faktoriellen Entwurfs gewählt, da mit der Filterung der bestimmten Winkel nur ungefähr die Hälfte der möglichen Kombinationen betrachtet werden. Zusammengefasst werden die Messungen in einer Tabelle der folgenden Form:

Richtung	Entfernung	Winkel	Prozessorlast	Speichernutzung	Erkennungszeit	Erkannt

5.2.6 Instrumentierung

Für die Durchführung des Experimentes werden Hardware- und Softwareinstrumente benötigt. Hardwareinstrumente sind Instrumente die unabhängig von der Anwendung eingesetzt werden, während Softwareinstrumente Anwendungen bezeichnen, die während der Ausführung einer Anwendung protokollieren, wie diese die Ressourcen des Geräts belasten. Ebenso wird ein Handy benötigt, auf dem die Anwendungen die im Experiment getestet werden, installiert sind. Weiter wird als Hardwareinstrument ein *Luxmeter* benötigt, um die Helligkeit im Raum zu messen. Für die Messungen der Prozessorlast, Speichernutzung, Erkennungszeit und ob das Maschinenteil erkannt

wurde, werden *Softwareinstrumente* benötigt oder die Messungen können direkt in dem Quellcode der Anwendungen implementiert werden.

5.3 Gültigkeitsbewertung

Ein zentrale Frage bei der Durchführung des Experimentes ist die Validierung der Ergebnisse des Experimentes [40]. Es ist wichtig die Gefahren, die das Experiment beeinflussen könnten, schon in der Planungsphase zu identifizieren, und wenn möglich diese Faktoren zu minimieren oder so gar zu eliminieren. Dieser Schritt soll in der Planungsphase durchgeführt werden. Dies hat den Vorteil, dass die Gefahr das Experiment wiederholen zu müssen, vermieden werden kann. Falls Fehler bei der Durchführung auftreten ist die Gültigkeit des ganzen Experimentes infrage gestellt und das Experiment muss bearbeitet, und wieder durchgeführt werden. Es bestehen viele Gefahren welche die Gültigkeit eines Experimentes beeinflussen können, diese können in vier Gruppen aufgeteilt werden [41]: *Gültigkeit des Schlussfolgerung*, *Interne Gültigkeit*, *Konstruktion der Gültigkeit* und *Externe Gültigkeit*.

Gültigkeit der Schlussfolgerung. Diese Gültigkeit beschäftigt sich mit dem Zusammenhang zwischen den Versuchen und dem Ergebnis. Es soll ein signifikanter statistischer Zusammenhang zwischen den Versuchen und dem Ergebnis existieren.

Interne Gültigkeit. Wenn ein Zusammenhang zwischen den Versuchen und dem Ergebnis festgestellt wurde, soll dieser eine Konsequenz der kausalen Beziehung zwischen den Versuch und dem Ergebnis sein. Weiter soll sichergestellt werden, dass dieser Zusammenhang nicht aus einem Faktor folgt, der nicht kontrollierbar ist.

Konstruktion der Gültigkeit. Bei der Konstruktion der Gültigkeit wird der Zusammenhang zwischen der Theorie und den Bemerkungen überprüft, die aus dem Experiment folgen sollen. Wenn die Beziehung zwischen der Ursache und dem Effekt kausal bedingt ist, müssen zwei Aspekte in Acht genommen werden: Zum Ersten dass der Versuch mit der Ursache des Ergebnisses zusammenhängt und zum Zweiten, dass das Ergebnis den Effekt widerspiegelt. **Externe Gültigkeit.** Die externe Gültigkeit beschäftigt sich mit der Generalisierung der Ergebnisse. Wenn eine kausale Beziehung zwischen der

Konstruktion der Gültigkeit und dem Effekt besteht, stellt sich die Frage, ob das Ergebnis des Experimentes auch außerhalb des Kontextes des Experimentes verwendet werden kann.

Nun werden die beschriebenen Gültigkeitsbewertungen an das im letzten Abschnitt definierte Experiment durchgeführt. Cook und Campbell haben eine Liste der Gefahren zusammengestellt, die eine Experimentgültigkeit in Frage stellen [41]. Je nach Experiment kann diese Liste als Orientierung genutzt werden, um die einzelnen Gültigkeitsbewertungen zu hinterfragen. In den nächsten Abschnitten wird diese Liste, als Checkliste durchgegangen, um die Gefahren der Gültigkeit zu identifizieren.

5.3.1 Gültigkeit der Schlussfolgerung

Niedrige statistische Mächtigkeit. Die Mächtigkeit eines statistischen Testes ist die Möglichkeit ein eindeutiges Muster in den analysierten Daten zu finden. Dieser Fall kann in dem vorliegenden Experiment auftreten, da Daten von den abhängigen Variablen Prozessorauslastung, Speichernutzung, Zeit die für die 3D-Objekterkennung benötigt wird und ob das 3D-Objekt erkannt wurde, analysiert werden. Wenn bei einer der abhängigen Variablen kein Muster gefunden wurde, besteht die Gefahr, dass keine Schlussfolgerungen gezogen werden kann.

Zuverlässigkeit der Messungen. Es besteht die Gefahr, dass bei den Versuchsdurchführungen einige Fehler mit den Instrumenten aufgetreten sind. Ein weiteres Problem könnte die Implementierung der Messungen der Software darstellen.

Zufällige Störungen im Umfeld des Experimentes. Objekte die nicht zum Umfeld des Experimentes gehören können einen Einfluss auf das Ergebnis des Experimentes haben. Das Experiment wurde in einem Seminarraum durchgeführt. Wenn die Abbildung 5.2 als Referenz genommen wird, befindet sich in Richtung Süd ein Fenster, in Richtung West befindet sich ein Tisch mit einem Projektionsapparat, während in Richtung Ost eine weiße Wand vorliegt, zuletzt befindet sich in Richtung Nord ein großer Tisch mit Stühlen und eine Wand mit einer Tür und einer Projektionsfläche. Das ganze Umfeld

könnte Einfluss auf das Experiment haben, da im Experiment Frameworks für Computer Vision verwendet werden.

5.3.2 Interne Gültigkeit

Vergangenheit des Ergebnisses. Bei der Wiederholung der Versuche besteht die Gefahr, dass die Ergebnisse anders ausfallen, als bei den vorherigen Versuchen. Dies liegt daran, dass das Umfeld in dem das Experiment durchgeführt wird, sich leicht verändert hat. In diesem Experiment wird die unabhängige Variable Helligkeit definiert. Diese ist trotz im Experiment erfassten absoluten Werten, nicht leicht zu reproduzieren.

Testen. Wenn das Experiment wiederholt wird, besteht die Gefahr, dass bei den Versuchen ein Grad an Vertraulichkeit mit der Ausführung des Experimentes besteht. In diesem Fall muss bei der Durchführung des Experimentes eine leichte Rotationsbewegung betreiben werden. Es besteht also die Gefahr, dass man bei dem wiederholten Versuchen schon anvertraut ist, mit welchem Tempo die Rotationsbewegung durchgeführt werden muss, um das Maschinenteil zu erkennen.

Statistische Regression. Statisch werden bei der Auswertung die Daten der ersten 30 Sekunden betrachtet. Dies könnte einen Einfluss auf die statistische Analyse haben, da vielleicht positive Ergebnisse nach 30 Sekunden vorkommen könnten.

Instrumentation. Bei der internen Gültigkeit wird die Sammlung der Daten betrachtet. In diesem Experiment werden alle gesammelten Daten in einer Textdatei, Zeile für Zeile geschrieben. Es kann also durchaus vorkommen, dass die Datei nicht erstellt wird, oder es zu Ein- und Ausgabefehlern kommen kann, und die erfassten Daten nicht gespeichert werden.

Soziale Gefahren. Da das Experiment ein Quasi-Experiment ist und alle Abläufe deterministisch sind, da keine Personen am Experiment beteiligt sind, bestehen lediglich keine sozialen Gefahren, die das Experiment beeinflussen könnten.

5.3.3 Konstruktion der Gültigkeit

Eingeschränkte Verallgemeinerbarkeit. Das Experiment befasst sich mit der Erkennung von Maschinenteilen, jedoch wird nicht Rücksicht auf den Aufwand gesetzt, der eingesetzt werden muss, um das Maschinenteil zu erkennen. Daraus folgt, dass als Ergebnis des Experimentes, ob das Framework ARKit oder TensorFlow Lite besser Maschinenteile erkennt, eins der Frameworks als besser eingestuft wird, aber anderseits der Aufwand der für die Vorbereitung betreiben wird, für das bessere Framework deutlich höher ist, als das Framework das schlechter die Maschinenteile erkennt.

5.3.4 Externe Gültigkeit

Zusammenhang des Umfelds mit dem Versuch. Das Experiment wird in einem möglichst nah aufgestelltem Laborumfeld durchgeführt. Im Fokus stehen aber Maschinenteile, die in einem Industrie-Umfeld eingesetzt werden. Deswegen ist es möglich, dass die Ergebnisse aus diesem Experiment in einem anderen Umfeld nicht repräsentierend sind.

5.4 Durchführung des Experiments

Nachdem im Letzten Abschnitten das Experiment entworfen und geplant wurde, ist es nun Zeit, das Experiment durchzuführen. Nach [40] kann die Durchführung des Experimentes in drei Phasen aufgeteilt werden. Zuerst werden die Vorbereitungen durchgeführt. Nachdem sicher gestellt ist, dass alle benötigten Vorbereitungen erfolgreich abgeschlossen sind, kann die Durchführung des Experimentes beginnen. Und zuletzt, nachdem das Experiment durchgeführt wurde, muss die Gültigkeit der gesammelten Daten geprüft werden.

5.4.1 Vorbereitung

Die Vorbereitungen für das Experiment sind essenziell für eine erfolgreiche Durchführung des Experimentes. Zuerst muss überprüft werden, ob alle benötigten Materialien

vorhanden sind. Es werden folgende Sachen benötigt: Ein Handy mit dem vorinstallierten Anwendungen, ein Fotostativ mit dem entsprechenden Handy-Adapter, ein Luxmeter, Klebeband und ein Permanent Marker, um den Tisch zu markieren (siehe Abbildung 5.2). Weiter muss der Raum abgedunkelt werden und nur das künstliche Licht als Lichtquelle verwendet werden. Nachdem festgestellt wurde, dass alle Mittel vorhanden sind, muss der Tisch markiert und das Fotostativ aufgebaut werden. Das Handy wird auf das Fotostativ platziert und mit einem Stromkabel verbunden, danach kann das Fotostativ für den spezifischen Versuch angepasst werden.

5.4.2 Durchführung des Experimentes

Das Experiment wird in einem Raum durchgeführt, in dem möglichst realitätsnah ein Laborumfeld simuliert werden soll. Ein Laborumfeld ist erwünscht, um das Experiment soweit wie möglich reproduzierbar zu machen. Deswegen wird im Raum nur künstliches Licht verwendet. Das Licht wird durch einen Luxmeter in der Einheit Lux gemessen, um absolute Werte ermitteln zu können. Vor jedem Versuch wird dieser Wert mit dem Lichtsensor TSL2561 gemessen, der mit einem Raspberry Pi 3 Modell B+ verbunden ist. Abbildung 5.1 zeigt den genauen Aufbau des Luxmeters. In dem Raum befindet sich ein Tisch, auf dem die Mitte und die Richtungen markiert sind. Ein besseren Überblick zeigt die Abbildung 5.2. Auf der Mitte des Tisches wird der Lichtsensor bei jeder Messung platziert. Dieselbe Markierung wird auch für die Platzierung der Maschinenteile bei jeweiligen Experimentenversuch verwendet, wobei die Maschinenteile zentral zu der Markierung positioniert sind. Die Frameworks ARKit und TensorFlow Lite werden auf einem iPhone 7 ausgeführt, dass mit einem Fotostativ und einem Handyadapter befestigt ist. In Folge kann die Entfernung, als Entfernung von der Mitte des Tisches bis zum Fotostativ definiert werden, während die Einheit in Zentimeter gemessen wird. Um die Prozessorauslastung in Prozenten zu messen, wurde das Framework iOS-System-Services[1] verwendet. Die Speicher-, Zeit- und Winkelmessung wurden in den jeweiligen Anwendung identisch implementiert und können auf dem Repository näher betrachtet werden. Für die Messung der Speichernutzung wird die Einheit Megabyte verwendet,

[1] iOS-System-Services Stand:3.8.2019

Zeit wird in Sekunden gemessen und der Winkel wird in Grad gemessen. Einen besseren Überblick auf den gesamten Aufbau des Experimentes verdeutlicht die Abbildung 5.3.

Nun können die Schritte des Experimentes definiert werden. Zuerst wird der Luxmeter auf die Mitte des Tisches positioniert und der Wert der Helligkeit wird abgelesen. Danach wird der Luxmeter vom Tisch entfernt und das entsprechende Maschinenteil wird in die Mitte des Tisches positioniert. Bei dem jeweiligen Versuch wird das Fotostativ auf eine vorgegebene Entfernung positioniert und das Handy wird auf einem vorgegebenen Winkel eingestellt. Danach wird in der Anwendung auf der Benutzerschnittstelle in einem Textfeld der Helligkeitswert eingegeben, der davor von dem Luxmeter abgelesen wurde. Im nächsten Schritt wird dann die Richtung zu der das Maschinenteil betrachtet wird in der Benutzerschnittstelle gewählt. Die Richtung orientiert sich nach der Abbildung 5.2. Mit der Wahl der Richtung beginnt die Objekterkennung und Messung der abhängigen Variablen. Sobald die Objekterkennung beginnt, ist es notwendig, eine Rotationsbewegung des Fotostativs zu betreiben. Dies ist notwendig, weil das Framework ARKit eine Initialisierungsphase benötigt in der eine Fläche erkannt werden muss. Um die Bedingungen gleich zu stellen, wird dieselbe Rotationsbewegung auch bei dem Framework TensorFlow Lite durchgeführt. Einen Überblick dieser Rotationsbewegung gibt die Abbildung 5.4. Somit endet auch die Beschreibung der Experimentdurchführung. Diese Schritte werden für alle möglichen Konfigurationen ausgeführt, die im Abschnitt 5.2.5 definiert wurden.

5.4.3 Datenüberprüfung

Nachdem das Experiment komplett durchgeführt wurde, steht die Überprüfung der gesammelten Daten an. Für jede Konfiguration des Experimentes wurde eine Textdatei erstellt, in der die Messungen geschrieben wurden. Der Name der Textdatei besteht aus einem Zeitstempel der besagt, wann der Versuch durchgeführt wurde. Alle Textdateien werden auf dem Handy in den Dokumentenspeicher der Anwendung gespeichert. Die Daten werden dann mit der Anwendung iTunes auf einen PC übertragen. Die Daten werden dann einzeln mit einem Texteditor überprüft.

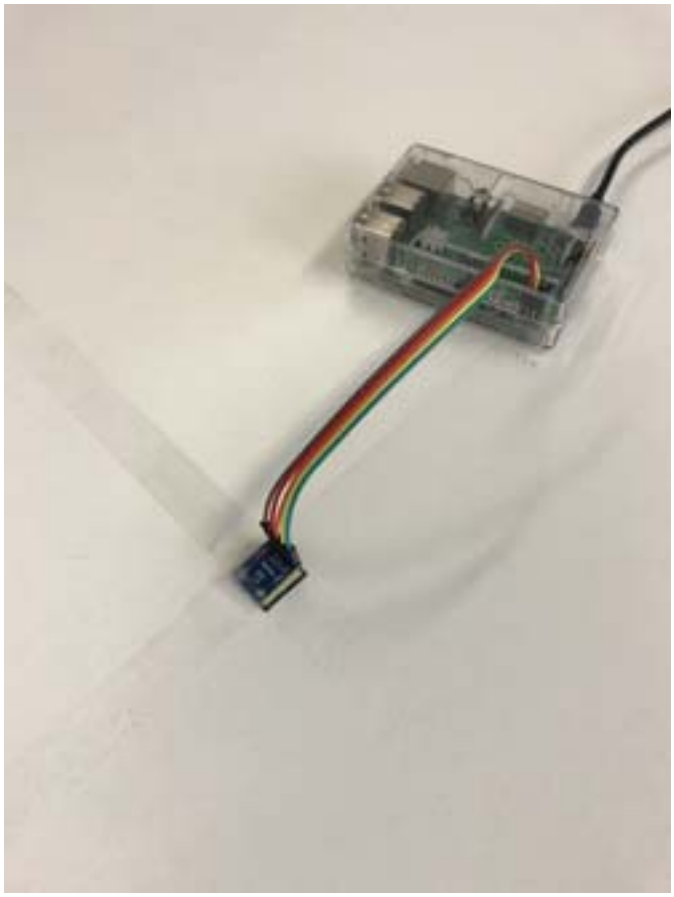

Abbildung 5.1: Raspberry Pi mit einem verbundenen Luxmetersensor

In einer Messungen fehlte der Eintrag für die Helligkeit, d.h. der Knopf zur Auswahl der Richtung wurde in der Benutzerschnittstelle gedrückt, bevor in dem Textfeld die Helligkeit eingetragen wurde. Ein weiterer Fehler war, dass in manchen Messungen die Richtung aus Versehen falsch eingegeben wurde. Diese Fehler können leicht korrigiert werden, da das Experiment deterministisch ist und genaue Ablauf der Schritte bekannt sind. Es werden periodisch in derselben Reihenfolge die Richtungen bei den Messungen durchgegangen, und zwar in folgender Reihenfolge: Süd, West, Nord, Ost. Dank dem Zeitstempel kann anhand der vorherigen oder nachträglichen Messung festgestellt werden, welche Seite bei diesem Versuch wirklich gemessen werden sollte. Genauso kann bei der fehlenden Messung der Helligkeit, anhand von den vorherigen oder nach-

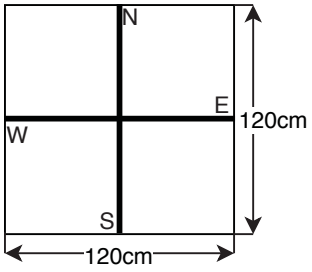

Abbildung 5.2: Tischaufbau aus der Vogelperspektive

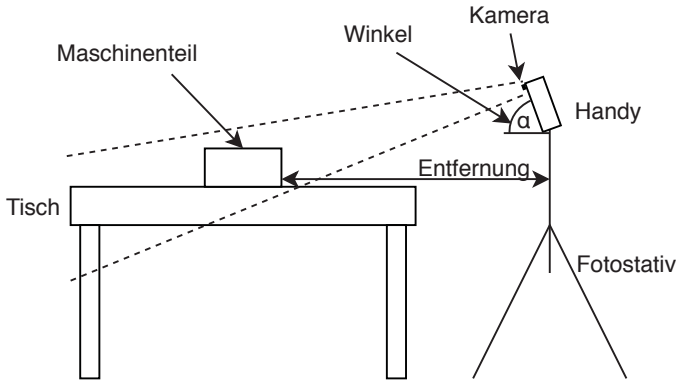

Abbildung 5.3: Versuchsaufbau

träglichen Messung festgestellt werden, wie hoch die ungefähre Helligkeit bei diesem Versuch war.

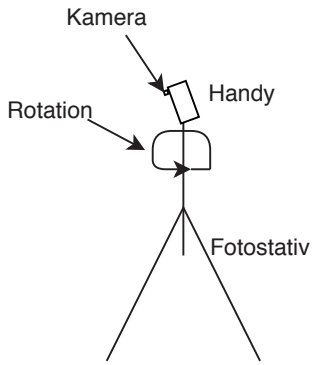

Abbildung 5.4: Rotationsbewegung

5.5 Analyse und Auswertung

Nach der Durchführung des Experimentes wurde eine Datenmenge an Messungen gesammelt, die nun analysiert werden muss. Wie in Wohlin [40] vorgeschlagen wird, kann die Analyse der Daten in drei Schritten ausgeführt werden. Zuerst werden statistische Methoden auf die Daten angewandt. Im zweiten Schritt werden ungültige Ergebnisse aus der Ergebnismenge entfernt. Und zuletzt wird anhand der analysierten Daten die Hypothese getestet.

5.5.1 Statistische Auswertung

Mit den statistischen Methoden werden die gesammelten Daten analysiert, um ein besseres Verständnis der Ergebnismenge zu erhalten. Es wird für jede Konfiguration des Experimentes eine Tabelle erstellt, die in dem Entwurf des Experiments definiert wurde. Für die Berechnung des Mittelwerts der abhängigen Variablen wird die folgende Formel verwendet:

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

Jetzt können für jede Konfiguration des Experimentes die Daten ausgewertet werden.

Maschinenteil 1

Die folgende Tabelle 5.1 zeigt die gemessenen Werte für das Framework ARKit und die Tabelle 5.2 zeigt die gemessenen Werte für das Framework TensorFlow Lite. Für einen bessern Überblick wurden auch Graphen erstellt die visuell die Daten veranschaulichen. So kann die durchschnittliche Prozessorauslastung über den Messungen in der Abbildung 5.5 eingesehen werden, wobei der durchschnittliche Wert für $\overline{\mu_{p_{tf}}} = 97.91\%$ ist, und für $\overline{\mu_{p_{ak}}} = 63.08\%$. Eine Übersicht für die durchschnittliche Speichernutzung ist in der Abbildung 5.6 einsehbar, die absoluten durchschnittlichen Werte sind $\overline{\mu_{s_{tf}}} = 96.62MB$ und $\overline{\mu_{s_{ak}}} = 145.19MB$. Die Erkennungsrate der beiden Frameworks ist in der Abbildung 5.7 sichtbar, wobei die absoluten Werte $\mu_{\hat{e_{tf}}} = 75\%$ und $\mu_{\hat{e_{ak}}} = 5\%$ sind. Und zuletzt bietet die Abbildung 5.8 ein Überblick über die durchschnittliche Erkennungszeit der Frameworks, wobei die durchschnittlichen Werte $\overline{\mu_{t_{tf}}} = 0.636s$ und $\overline{\mu_{t_{ak}}} = 15.9s$ sind.

Tabelle 5.1: Werte für das Framework ARKit und Maschinenteil 1

Richtung	Entfernung	Winkel	$\overline{\mu_{p_{ar}}}$	$\overline{\mu_{s_{ar}}}$	$\mu_{t_{ar}}$	$\mu_{e_{ar}}$	Helligkeit
Süd	45cm	$40°$	71.96	149.07	∞	Nein	970
West	45cm	$40°$	76.67	148.82	15.90	Ja	970
Nord	45cm	$40°$	72.96	147.31	∞	Nein	970
Ost	45cm	$40°$	74.38	149.05	∞	Nein	970
Süd	60cm	$47°$	59.64	145.93	∞	Nein	1073
West	60cm	$47°$	59.32	146.33	∞	Nein	1073
Nord	60cm	$47°$	54.08	141.73	∞	Nein	1073
Ost	60cm	$47°$	53.72	143.57	∞	Nein	1073
Süd	60cm	$88°$	66.12	147.88	∞	Nein	969
West	60cm	$88°$	70.32	148.34	∞	Nein	969
Nord	60cm	$88°$	75.10	148.61	∞	Nein	969
Ost	60cm	$88°$	64.42	147.12	∞	Nein	969
Süd	60cm	$55°$	49.33	138.33	∞	Nein	1013
West	60cm	$55°$	60.83	145.67	∞	Nein	1013
Nord	60cm	$55°$	60.35	144.45	∞	Nein	1013
Ost	60cm	$55°$	53.63	142.44	∞	Nein	1013
Süd	80cm	$59°$	57.50	144.00	∞	Nein	988
West	80cm	$59°$	65.65	143.99	∞	Nein	988
Nord	80cm	$59°$	63.70	143.25	∞	Nein	988
Ost	80cm	$59°$	51.94	137.93	∞	Nein	988

Tabelle 5.2: Werte für das Framework TensorFlow Lite und Maschinenteil 1

Richtung	Entfernung	Winkel	$\overline{\mu_{p_{tf}}}$	$\overline{\mu_{s_{tf}}}$	$\mu_{t_{tf}}$	$\mu_{e_{tf}}$	Helligkeit
Süd	45cm	$40°$	97.53	100.97	0.502	Ja	970
West	45cm	$40°$	98.22	100.33	0.502	Ja	970
Nord	45cm	$40°$	97.30	100.31	0.407	Ja	970
Ost	45cm	$40°$	98.27	100.39	0.402	Ja	970
Süd	60cm	$47°$	97.90	100.28	0.502	Ja	1073
West	60cm	$47°$	98.38	100.45	0.401	Ja	1073
Nord	60cm	$47°$	97.19	95.582	0.502	Ja	1073
Ost	60cm	$47°$	98.47	100.30	0.402	Ja	1073
Süd	60cm	$88°$	97.91	91.532	∞	Nein	969
West	60cm	$88°$	97.77	85.736	∞	Nein	969
Nord	60cm	$88°$	97.03	85.429	∞	Nein	969
Ost	60cm	$88°$	98.27	85.632	∞	Nein	969
Süd	60cm	$55°$	98.08	100.31	0.502	Ja	1013
West	60cm	$55°$	97.85	98.917	0.402	Ja	1013
Nord	60cm	$55°$	97.51	99.728	1.000	Ja	1013
Ost	60cm	$55°$	98.32	100.49	0.402	Ja	1013
Süd	80cm	$59°$	98.30	100.58	0.502	Ja	988
West	80cm	$59°$	97.91	86.471	∞	Nein	988
Nord	80cm	$59°$	97.52	98.588	2.701	Ja	988
Ost	80cm	$59°$	98.38	100.46	0.404	Ja	988

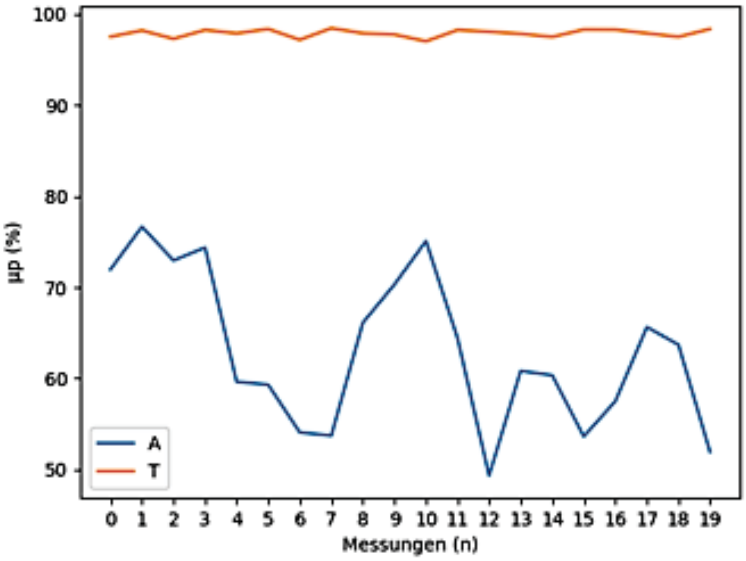

Abbildung 5.5: durchschnittliche Prozessorauslastung in den einzelnen Messungen für Maschinenteil 1

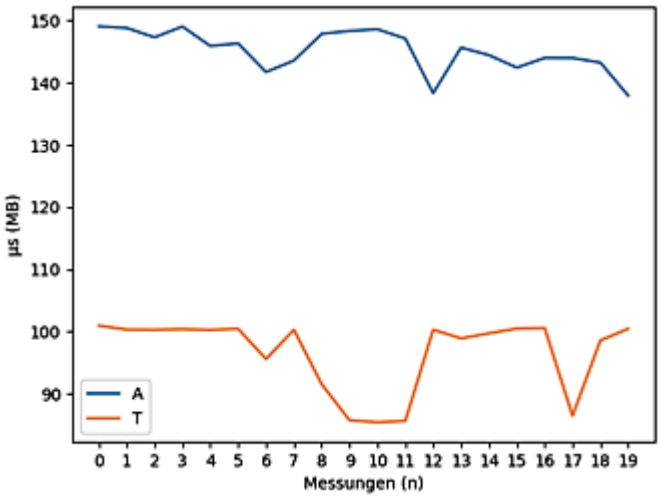

Abbildung 5.6: durchschnittliche Speichernutzung in den einzelnen Messungen für Maschinenteil 1

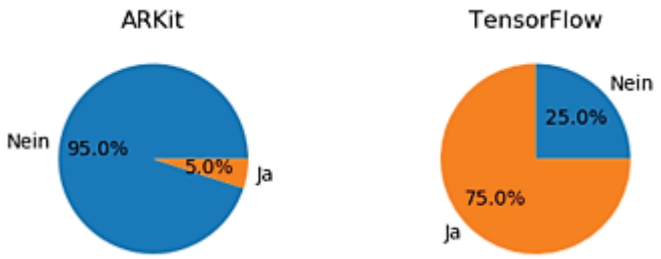

Abbildung 5.7: Verteilung der Erkennungsrate für Maschinenteil 1

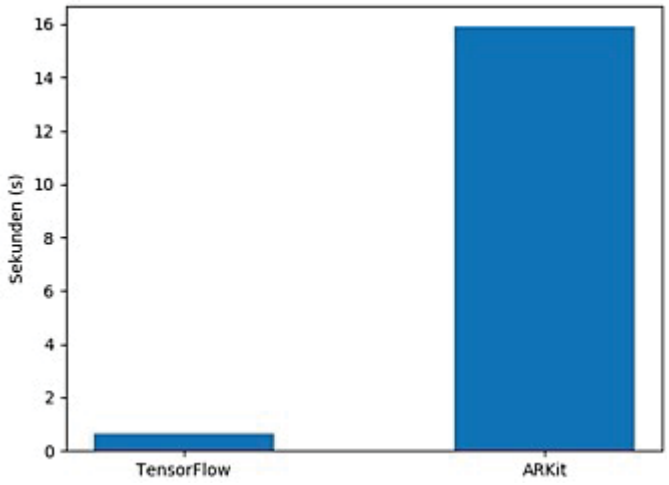

Abbildung 5.8: Durchschnittliche Erkennungszeit für Maschinenteil 1

Maschinenteil 2

Die folgende Tabelle 5.3 zeigt die gemessenen Werte für das Framework ARKit und die Tabelle 5.4 zeigt die gemessenen Werte für das Framework TensorFlow Lite. Für einen bessern Überblick wurden auch Graphen erstellt die visuell die Daten veranschaulichen. So kann die durchschnittliche Prozessorauslastung über den Messungen in der Abbildung 5.9 eingesehen werden, wobei der durchschnittliche Wert für $\overline{\mu_{p_{tf}}} = 98.53\%$ ist, und für $\overline{\mu_{p_{ak}}} = 59.66\%$. Eine Übersicht für die durchschnittliche Speichernutzung ist in der Abbildung 5.10 einsehbar, die absoluten durchschnittlichen Werte sind $\overline{\mu_{s_{tf}}} = 98.87MB$ und $\overline{\mu_{s_{ak}}} = 146.42MB$. Die Erkennungsrate der beiden Frameworks ist in der Abbildung 5.11 sichtbar, wobei die absoluten Werte $\hat{\mu_{e_{tf}}} = 95\%$ und $\hat{\mu_{e_{ak}}} = 40\%$ sind. Und zuletzt bietet die Abbildung 5.12 ein Überblick über die durchschnittliche Erkennungszeit der Frameworks, wobei die durchschnittlichen Werte $\overline{\mu_{t_{tf}}} = 0.81s$ und $\overline{\mu_{t_{ak}}} = 5.49s$ sind.

Tabelle 5.3: Werte für das Framework ARKit und Maschinenteil 2

Richtung	Entfernung	Winkel	$\overline{\mu_{p_{ar}}}$	$\overline{\mu_{s_{ar}}}$	$\mu_{t_{ar}}$	$\mu_{e_{ar}}$	Helligkeit
Süd	45cm	$40°$	61.30	150.63	7.500	Ja	970
West	45cm	$40°$	57.40	148.08	4.201	Ja	970
Nord	45cm	$40°$	57.45	147.75	6.101	Ja	970
Ost	45cm	$40°$	59.84	147.31	5.201	Ja	970
Süd	60cm	$47°$	51.71	141.39	∞	Nein	1073
West	60cm	$47°$	64.10	146.06	3.701	Ja	1073
Nord	60cm	$47°$	53.80	145.28	∞	Nein	1073
Ost	60cm	$47°$	50.65	136.81	∞	Nein	1073
Süd	60cm	$88°$	72.48	151.35	∞	Nein	969
West	60cm	$88°$	70.34	153.71	∞	Nein	969
Nord	60cm	$88°$	76.87	151.98	∞	Nein	969
Ost	60cm	$88°$	63.11	154.76	1.901	Ja	969
Süd	60cm	$55°$	60.08	146.12	∞	Nein	1013
West	60cm	$55°$	54.82	140.39	13.40	Ja	1013
Nord	60cm	$55°$	60.48	143.68	∞	Nein	1013
Ost	60cm	$55°$	54.99	147.17	12.90	Ja	1013
Süd	80cm	$59°$	56.10	145.11	∞	Nein	988
West	80cm	$59°$	59.63	142.83	∞	Nein	988
Nord	80cm	$59°$	52.23	140.60	∞	Nein	988
Ost	80cm	$59°$	55.98	147.48	∞	Nein	988

Tabelle 5.4: Werte für das Framework TensorFlow Lite und Maschinenteil 2

Richtung	Entfernung	Winkel	$\overline{\mu_{p_{tf}}}$	$\overline{\mu_{s_{tf}}}$	$\mu_{t_{tf}}$	$\mu_{e_{tf}}$	Helligkeit
Süd	45cm	$40°$	98.36	100.67	0.502	Ja	970
West	45cm	$40°$	98.67	99.885	0.502	Ja	970
Nord	45cm	$40°$	98.41	100.82	0.502	Ja	970
Ost	45cm	$40°$	98.23	100.55	0.503	Ja	970
Süd	60cm	$47°$	98.87	100.74	0.402	Ja	1073
West	60cm	$47°$	98.85	100.96	0.402	Ja	1073
Nord	60cm	$47°$	98.74	100.86	0.407	Ja	1073
Ost	60cm	$47°$	98.87	100.89	0.501	Ja	1073
Süd	60cm	$88°$	98.68	85.864	∞	Nein	969
West	60cm	$88°$	98.63	96.090	0.502	Ja	969
Nord	60cm	$88°$	98.44	90.828	3.201	Ja	969
Ost	60cm	$88°$	98.14	95.716	1.501	Ja	969
Süd	60cm	$55°$	98.61	100.03	2.001	Ja	1013
West	60cm	$55°$	98.71	100.81	0.404	Ja	1013
Nord	60cm	$55°$	97.79	100.52	0.502	Ja	1013
Ost	60cm	$55°$	98.11	100.63	0.502	Ja	1013
Süd	80cm	$59°$	98.45	99.295	1.501	Ja	988
West	80cm	$59°$	98.82	100.87	0.502	Ja	988
Nord	80cm	$59°$	98.74	100.73	0.502	Ja	988
Ost	80cm	$59°$	98.64	100.76	0.502	Ja	988

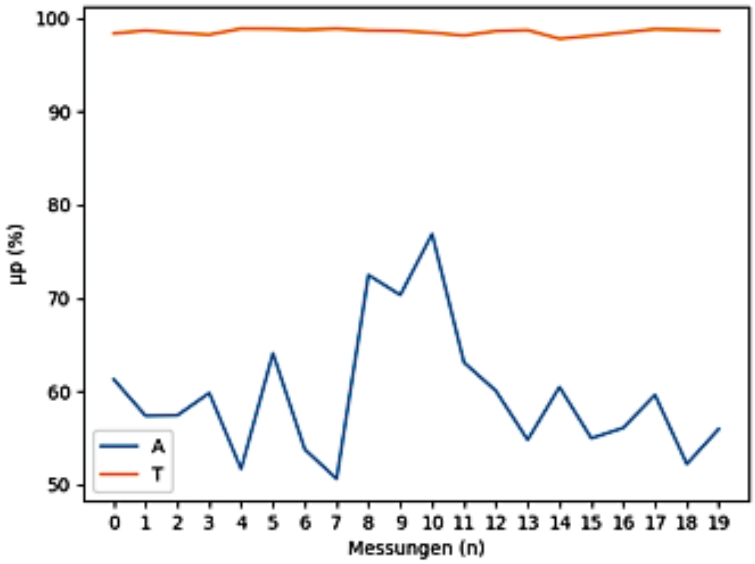

Abbildung 5.9: durchschnittliche Prozessorauslastung in den einzelnen Messungen für Maschinenteil 2

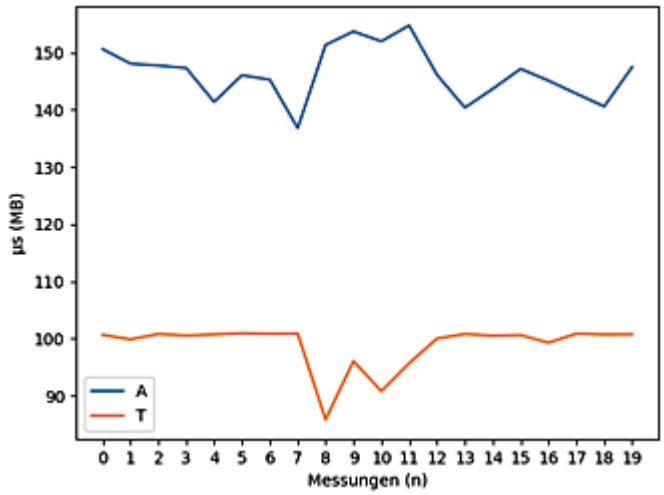

Abbildung 5.10: durchschnittliche Speichernutzung in den einzelnen Messungen für Maschinenteil 2

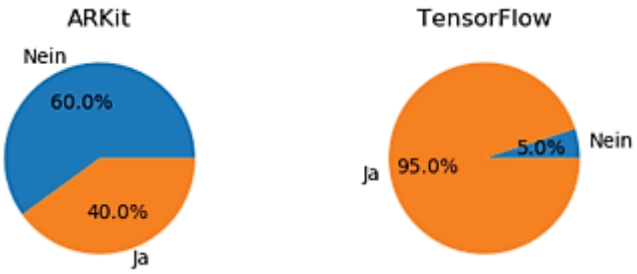

Abbildung 5.11: Verteilung der Erkennungsrate für Maschinenteil 2

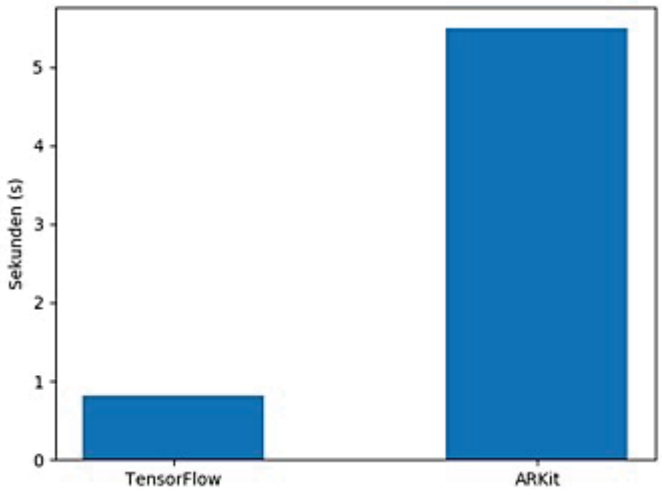

Abbildung 5.12: Durchschnittliche Erkennungszeit für Maschinenteil 2

Maschinenteil 3

Die folgende Tabelle 5.5 zeigt die gemessenen Werte für das Framework ARKit und die Tabelle 5.6 zeigt die gemessenen Werte für das Framework TensorFlow Lite. Für einen bessern Überblick wurden auch Graphen erstellt die visuell die Daten veranschaulichen. So kann die durchschnittliche Prozessorauslastung über den Messungen in der Abbildung 5.13 eingesehen werden, wobei der durchschnittliche Wert für $\overline{\mu_{p_{tf}}} = 98.55\%$ ist, und für $\overline{\mu_{p_{ak}}} = 57.33\%$. Eine Übersicht für die durchschnittliche Speichernutzung ist in der Abbildung 5.14 einsehbar, die absoluten durchschnittlichen Werte sind $\overline{\mu_{s_{tf}}} = 89.58MB$ und $\overline{\mu_{s_{ak}}} = 145.78MB$. Die Erkennungsrate der beiden Frameworks ist in der Abbildung 5.15 sichtbar, wobei die absoluten Werte $\hat{\mu_{e_{tf}}} = 60\%$ und $\hat{\mu_{e_{ak}}} = 35\%$ sind. Und zuletzt bietet die Abbildung 5.16 ein Überblick über die durchschnittliche Erkennungszeit der Frameworks, wobei die durchschnittlichen Werte $\overline{\mu_{t_{tf}}} = 1.71s$ und $\overline{\mu_{t_{ak}}} = 11.08s$ sind.

Tabelle 5.5: Werte für das Framework ARKit und Maschinenteil 3

Richtung	Entfernung	Winkel	$\overline{\mu_{p_{ar}}}$	$\overline{\mu_{s_{ar}}}$	$\mu_{t_{ar}}$	$\mu_{e_{ar}}$	Helligkeit
Süd	45cm	$40°$	68.5	150.41	∞	Nein	970
West	45cm	$40°$	54.80	145.48	4.201	Ja	970
Nord	45cm	$40°$	56.46	143.94	5.501	Ja	970
Ost	45cm	$40°$	72.24	152.21	22.90	Ja	970
Süd	60cm	$47°$	53.38	147.29	∞	Nein	1073
West	60cm	$47°$	62.07	147.07	18.20	Ja	1073
Nord	60cm	$47°$	47.23	135.91	∞	Nein	1073
Ost	60cm	$47°$	51.59	143.25	∞	Nein	1073
Süd	60cm	$88°$	72.46	149.93	∞	Nein	969
West	60cm	$88°$	66.46	150.82	∞	Nein	969
Nord	60cm	$88°$	76.65	150.07	∞	Nein	969
Ost	60cm	$88°$	59.48	155.33	∞	Nein	969
Süd	60cm	$55°$	49.59	143.40	∞	Nein	1013
West	60cm	$55°$	55.36	143.36	5.401	Ja	1013
Nord	60cm	$55°$	54.31	145.79	8.601	Ja	1013
Ost	60cm	$55°$	54.31	144.50	12.80	Ja	1013
Süd	80cm	$59°$	47.93	138.28	∞	Nein	988
West	80cm	$59°$	43.95	141.49	∞	Nein	988
Nord	80cm	$59°$	49.49	140.21	∞	Nein	988
Ost	80cm	$59°$	50.40	146.89	∞	Nein	988

Tabelle 5.6: Werte für das Framework TensorFlow Lite und Maschinenteil 3

Richtung	Entfernung	Winkel	$\overline{\mu_{p_{tf}}}$	$\overline{\mu_{s_{tf}}}$	$\mu_{t_{tf}}$	$\mu_{e_{tf}}$	Helligkeit
Süd	45cm	$40°$	98.91	92.903	0.901	Ja	970
West	45cm	$40°$	98.20	93.324	0.503	Ja	970
Nord	45cm	$40°$	98.16	93.344	0.502	Ja	970
Ost	45cm	$40°$	98.85	92.778	3.101	Ja	970
Süd	60cm	$47°$	98.71	93.489	0.403	Ja	1073
West	60cm	$47°$	98.87	85.931	∞	Nein	1073
Nord	60cm	$47°$	98.62	90.705	4.101	Ja	1073
Ost	60cm	$47°$	98.64	91.632	0.403	Ja	1073
Süd	60cm	$88°$	98.50	85.459	∞	Nein	969
West	60cm	$88°$	98.47	86.056	∞	Nein	969
Nord	60cm	$88°$	98.60	85.907	∞	Nein	969
Ost	60cm	$88°$	98.35	85.493	∞	Nein	969
Süd	60cm	$55°$	98.62	90.805	0.502	Ja	1013
West	60cm	$55°$	98.50	86.102	8.801	Ja	1013
Nord	60cm	$55°$	97.90	93.307	0.502	Ja	1013
Ost	60cm	$55°$	98.94	93.479	0.402	Ja	1013
Süd	80cm	$59°$	98.58	85.935	∞	Nein	988
West	80cm	$59°$	98.73	86.058	∞	Nein	988
Nord	80cm	$59°$	98.39	93.412	0.503	Ja	988
Ost	80cm	$59°$	98.56	85.518	∞	Nein	988

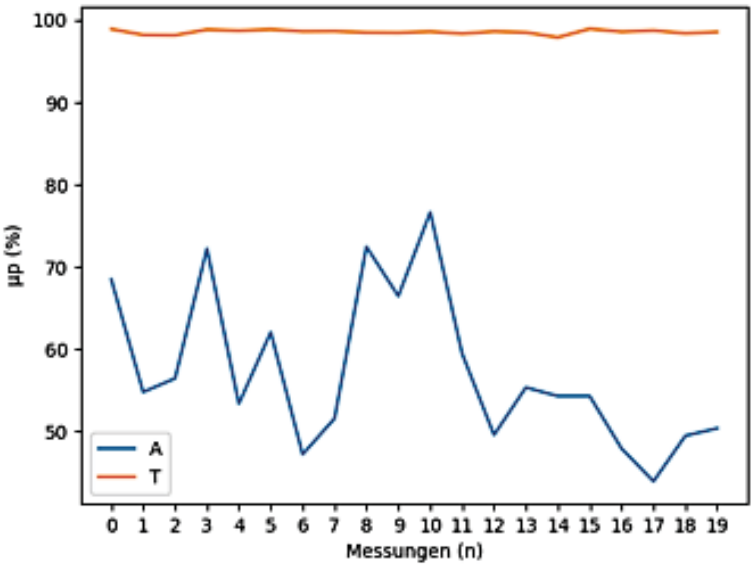

Abbildung 5.13: durchschnittliche Prozessorauslastung in den einzelnen Messungen für Maschinenteil 3

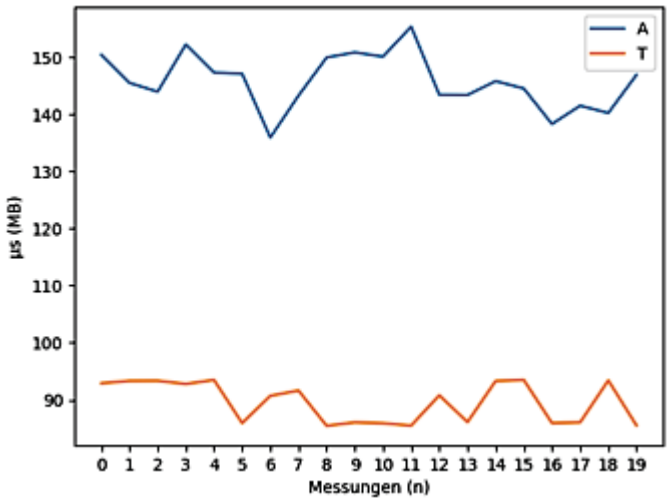

Abbildung 5.14: durchschnittliche Speichernutzung in den einzelnen Messungen für Maschinenteil 3

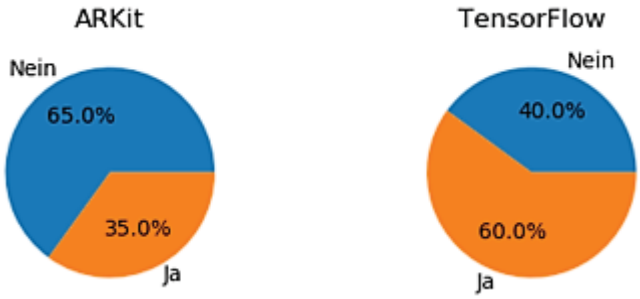

Abbildung 5.15: Verteilung der Erkennungsrate für Maschinenteil 3

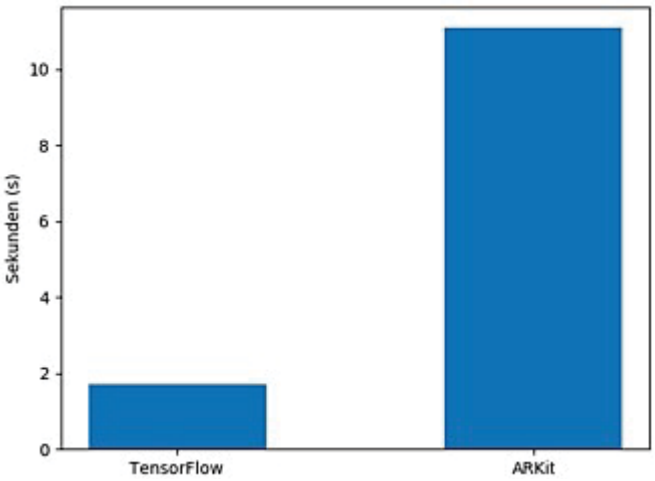

Abbildung 5.16: Durchschnittliche Erkennungszeit für Maschinenteil 3

5.5.2 Reduktion der Ergebnismenge

In dem letzten Abschnitt wurden die Messungen die im Experiment gesammelt wurden, statistisch ausgewertet. Dabei wurden bei der Auswertung einige Messungen ignoriert. Dies war der Fall bei der Auswertung der durchschnittlichen Erkennungszeit $\overline{\mu_t}$. Um einen sinnvollen Wert zu berechnen, wurden Messungen eingeschlossen, bei welchen das Maschinenteil erkannt wurde. Weiter wird in der Auswertung nach Werten gesucht, die aus der Ergebnismenge ausstechen. In der Tabelle 5.1 in der Spalte $\overline{\mu_{ear}}$ befindet sich nur ein Eintrag, der besagt, dass das Maschinenteil erkannt wurde. Das bestätigt auch die Abbildung 5.7 in der deutlich zu sehen ist, dass die Erkennungsrate bei 5% liegt. Nun muss entschieden werden, ob die Messungen und Auswertung von Maschinenteil 1 in die Ergebnismenge einbezogen werden soll. Da drei Maschinenteile in dem Experiment betrachtet werden, besteht die Gefahr, dass bei der Entfernung der Messungen und Auswertungen von Maschinenteil 1 aus der Ergebnismenge, die Varianz des Experiments beeinflusst wird. Weiterhin liegt die Vermutung für die schwache Erkenunngsrate an den Maschinenteil selbst, da wahrscheinlich das Maschinenteil wenige Features besitzt, die dazu führen, dass das Maschinenteil nicht erkannt wird. Dies ist aber kein valider Grund diese Messungen aus der Ergebnismenge zu entfernen.

5.5.3 Testen der Hypothese

In dem Abschnitt 5.2.2 wurde die Hypothese formuliert. Es wurde eine Hypothese 0 definiert und eine alternative Hypothese 1. Nun werden die Auswertungen die im letzten Abschnitt durchgeführt wurden, benutzt, um die Hypothese 0 zu widerlegen. Weiterhin muss bei dem testen der Hypothesen auf den Typ-I-Fehler geachtet werden. In Wohlin [40] wird der Typ-I-Fehler so definiert, dass bei der Durchführung eines statistischen Tests, ein Muster oder Beziehung in den Daten erkannt wurden, obwohl kein Muster oder eine Beziehung existiert. Als Wahrscheinlichkeit ein solchen Fehler zu begehen ist $P(\text{Typ-I-Fehler}) = P(\text{widerlege } H_0 | H_0 \text{ Wahr})$.

Laut Wohlin [40] bietet sich für diesen Experimententwurf der t-Test an. In Wohlin [40] ist der t-Test wie folgt definiert. Als Eingabe werden zwei unabhängige Stichproben

$x_1, x_2, ..., x_n$ und $y_{1,2}, ..., y_m$ definiert. Die Nullhypothese besagt, dass $\mu_x = \mu_y$ gelten muss. Nun ist es notwendig t_0 zu berechnen mit der folgenden Formel:

$$t_0 = \frac{\overline{x} - \overline{y}}{S_p\sqrt{\frac{1}{n} + \frac{1}{m}}}, S_p = \sqrt{\frac{(n-1)S_x^2 + (m-1)S_y^2}{n+m-2}}$$

S_x^2 und S_y^2 sind einzelne Stichprobenvarianzen. Das Kriterium, um die Nullhypothese zu widerlegen ist: Wenn $\mu_x \neq \mu_y$, dann folgt, dass die Nullhypothese widerlegt ist, falls $|t_0| > t_{\alpha/2,n+m-2}$. Der Wert $t_{\alpha,f}$ ist der α Prozentsatz der t Verteilung mit dem Grad f an Freiheit, der dem Wert $n + m - 2$ entspricht. In den folgenden Test der Hypothese wird der Wert $\alpha = 0.05$ vorausgesetzt, was genauer bedeutet, dass wenn einen Wert von $|t_0| > t_{\alpha/2,f}$ gefunden wurde, wurde ein Widerspruch zu H_0 gefunden. Genauer wird die Wahrscheinlichkeit $P(|t_0| \geq |t|) < \alpha$ betrachtet. f_x steht für die Verteilungsfunktion.

Eine Ausnahme beim Testen der Hypothese ist die Erkennungsrate. Da diese Einträge nur mit einem binären Wert bei der Messung eingetragen sind, wird für diese Daten nicht zu dem t-Test zugegriffen. Für diese Form von Daten wird der Exakte Test nach Fisher verwendet [42]. Eine Kontingenztafel der Größe $2x2$ wird für die binären Werte benutzt. Im Test wird die hypergeometrische Verteilung benutzt um die exakte Wahrscheinlichkeit p zu berechnen. Je kleiner der Wert p ist, desto größer ist die Signifikanz unserer Messungen, und somit kann die Hypothese 0 widerlegt werden. Für diesen Test wird auch der Signifikanzwert $\alpha = 0.05$ vorausgesetzt. Wenn also $p < \alpha$ gilt kann die Hypothese 0 widerlegt werden.

Prozessorauslastung

Tabelle 5.7: T-Test Berechnung für die Prozessorauslastung

	$\mu_{p_{ar}}$	$\mu_{p_{tf}}$
Durchschnitt	60.02	98.33
Standardabweichung	8.51739	0.45181
Anzahl der Messungen	60	60

$$S_p = \sqrt{\frac{(60-1)8.51739^2 + (60-1)0.45181^2}{60+60-2}} = 6.03117$$

$$t_0 = \frac{60.02 - 98.33}{6.03117\sqrt{\frac{1}{60}+\frac{1}{60}}} = -34.7913$$

$$\Rightarrow P(|t_0| \geq 34.7913) = \int_{-\infty}^{34.7913} f_X dt \approx 0$$

$$\Rightarrow P(|t_0| \geq 34.7913) < \alpha = 0.05 \Rightarrow \mu_{p_{ar}} \neq \mu_{p_{tf}}$$

$$\Rightarrow H_0 \text{ widerlegt}$$

$$\Rightarrow \text{Es gilt } H_1 : \mu_{p_{ar}} < \mu_{p_{tf}}$$

Speichernutzung

Tabelle 5.8: T-Test Berechnung für die Speichernutzung

	$\mu_{s_{ar}}$	$\mu_{s_{tf}}$
Durchschnitt	145.798	95.027
Standardabweichung	4.29245	6.04753
Anzahl der Messungen	60	60

$$S_p = \sqrt{\frac{(60-1)4.29245^2 + (60-1)6.04753^2}{60+60-2}} = 5.24394$$

$$t_0 = \frac{145.798 - 95.027}{5.24394\sqrt{\frac{1}{60}+\frac{1}{60}}} = 53.0296$$

$$\Rightarrow P(|t_0| \geq 53.0296) = \int_{-\infty}^{53.0296} f_X dt \approx 0$$

$$\Rightarrow P(|t_0| \geq 53.0296) < \alpha = 0.05 \Rightarrow \mu_{s_{ar}} \neq \mu_{s_{tf}}$$

$$\Rightarrow H_0 \text{ widerlegt}$$

$$\Rightarrow \text{Es gilt } H_1 : \mu_{s_{ar}} > \mu_{s_{tf}}$$

Erkennungszeit

Tabelle 5.9: T-Test Berechnung für die Erkennungszeit

	$\mu_{t_{ar}}$	$\mu_{t_{tf}}$
Durchschnitt	8.58806	0.989
Standardabweichung	6.228603	1.4497
Anzahl der Messungen	16	46

$$S_p = \sqrt{\frac{(16-1)6.228603^2 + (46-1)1.4497^2}{16+46-2}} = 3.35784$$

$$t_0 = \frac{8.58806 - 1.4497}{3.35784\sqrt{\frac{1}{16}+\frac{1}{46}}} = 7.32456$$

$$\Rightarrow P(|t_0| \geq 7.32456) = \int_{-\infty}^{7.32456} f_X dt \approx 0$$

$$\Rightarrow P(|t_0| \geq 7.32456) < \alpha = 0.05 \Rightarrow \mu_{t_{ar}} \neq \mu_{t_{tf}}$$

$$\Rightarrow H_0 \text{ widerlegt}$$

$$\Rightarrow \text{Es gilt } H_1 : \mu_{t_{ar}} > \mu_{t_{tf}}$$

Erkennungsrate

Tabelle 5.10: Exakter Test nach Fisher - Berechnung für die Erkennungsrate

	$\mu_{e_{ar}}$	$\mu_{e_{tf}}$	Reihe insgesamt
Erkannt	16	46	62
Nicht erkannt	44	14	58
Spalte insgesamt	60	60	n=120

$$p = \frac{\binom{16+46}{16}\binom{44+14}{44}}{\binom{120}{16+44}} \approx 2.8696 * 10^{-8}$$

$$\Rightarrow p < \alpha = 0.05 \Rightarrow \mu_{e_{ar}} \neq \mu_{e_{tf}}$$

$$\Rightarrow H_0 \text{ widerlegt}$$

$$\Rightarrow \text{Es gilt } H_1 : \mu_{e_{ar}} < \mu_{e_{tf}}$$

5.6 Ergebnisse

Bis zu diesem Zeitpunkt wurde der Entwurf des Experimentes, die Gültigkeitsbewertung, die Ausführung des Experimentes, die statistische Auswertung der Daten und das Testen der Hypothese betrachtet. Nun ist es Zeit, eine Schlussfolgerung über den ganzen Experimentprozess zu ziehen. Bei der Auswertung der Daten wurden drei Maschinenteile betrachtet. Die Ergebnisse für die abhängigen Variablen Prozessorauslastung und Speichernutzung waren bei den Messungen über die drei Maschinenteile ähnlich verteilt.

Nach dem die Nullhypothese für die Prozessorauslastung 5.7 und die Speichernutzung 5.8 widerlegt wurde, können nun Schlussfolgerungen über die beiden abhängigen Variablen ziehen. So wurde eine größere Prozessorauslastung für das Framework TensorFlow Lite im Vergleich zum Framework ARKit festgestellt, während bei der Speichernutzung das Framework ARKit einen größeren Wert an Speicher benötigt, als das Framework TensorFlow Lite. Signifikante Unterschiede zwischen den Messungen gab es bei den abhängigen Variablen Erkennungszeit und Erkennugnsrate. Bei der Betrachtung von Maschinenteil 1, kann aus der Abbildung 5.7 erkannt werden, dass bei Maschinenteil 1 eine Erkennungsrate bei dem Framework ARKit von nur 5% gemessen wurde. Bei den Maschinenteilen 2 und 3 war die Erkennungsrate für das Framework ARKit bei 40% und 35%, was deutlich höhere Werte sind. Im Gegensatz zum Framework ARKit, hat das Framework TensorFlow Lite für die Maschinenteile 1, 2 und 3 bei der Erkennungsrate die Werte von jeweils 75%, 95%, 60% erzielten. Daraus kann gefolgert werden, dass die Erkennung eines Maschinenteils durchaus von seinem Aufbau und Merkmalen abhängt.

Weiterhin kann, dank der Widerlegung der Nullhypothese 5.10, die Werte der jeweiligen Frameworks mit einander vergleicht werden und daraus folgt, dass das Framework TensorFlow Lite, für Maschinenteile die in diesem Experiment verwendet wurden, sich besser für eine Objekterkennung eignet. Die durchschnittliche Erkennungszeit unterscheidet sich auch enorm zwischen den zwei Frameworks. So wurde bei den Maschinenteil 1 eine durchschnittliche Erkennungszeit von $15.9s$ für das Framework ARKit und $0.636s$ für

das Framework TensorFlow Lite festgestellt. Bei dem Maschinenteil 2 hat das Framework ARKit einen durchschnittlichen Wert von $5.49s$ und TensorFlow Lite $0.81s$. Maschinenteil 3 hat einen durchschnittlichen Wert von $11.08s$ für das Framework ARKit und $1.71s$ für TensorFlow Lite. Da die Nullhypothese 5.9 widerlegt wurde, kann gefolgert werden, dass über die ganzen Messungen das Framework TensorFlow Lite eine schnellere Erkennungszeit, als das Framework ARKit hat.

Zusammengefasst kann anhand der jeweiligen Schlussfolgerungen festgestellt werden, dass das Framework ARKit nur beim Merkmal Prozessorauslastung besser abgeschnitten hat als das Framework TensorFlow Lite. Bei allen anderen gemessenen Merkmalen hat das TensorFlow Lite Framework besser abgeschnitten. So kann man schlussfolgern, dass sich für die in diesem Experiment verwendeten Maschinenteile, der verwendeten Umgebung sowie den verwendeten Instrumenten das FrameWork TensorFlow Lite besser für eine allgemeine Objekterkennung der Maschinenteile eignet.

6

Verwandte Arbeiten

Im folgenden Kapitel wird ein Überblick über wissenschaftliche Arbeiten gegeben, die sich mit dem Thema Computer Vision und Augmented Reality mit Schwerpunkt Objekterkennung befassen.

6.1 SSD: Single Shot MultiBox Detector

Liu et al. stellen eine Methode zur Objekterkennung in Bildern vor, die ein einziges neuronales Netz verwendet [38]. Dabei wird der Ausgaberaum der Boxen, die das erkannte Objekt umrunden, diskret dargestellt, nämlich in einer Menge von gewöhnlichen Boxen über verschiedene Seitenverhältnisse und Skalierungen anstelle der Feature im Bild. Bei der Erkennung generiert das neuronale Netz ein Resultat, das die Genauigkeit des erkannten Objektes repräsentiert. Die Box wird dann anhand der erkannten Kontur des Objektes angepasst. Das neuronale Netz kombiniert die Voraussage über mehrere Featurebereiche im Bild, die unterschiedliche Auflösung haben, um in der Lage zu sein Objekte unterschiedlicher Größe zu erkennen.

In der Trainingsphase benötigt das **SSD** Framework ein Eingabebild und eine Box, die das Objekt umrahmt (siehe Abbildung 6.1*a)*. Es werden anhand einer Faltung die Menge der Boxen unterschiedlicher Seitenverhältnisse auf jeder Position der vorhandenen Featuremenge in unterschiedlichen Skalierungen, hier $8x8$ Pixel (Abbildung 6.1*b)* und $4x4$ Pixel (Abbildung 6.1*c)* evaluiert. In jeder Box werden die Abweichungen zur Form des Objektes und die Erkennungsgenauigkeit für alle Objektkategorien $(c_1, c_2, ..., c_p)$ berechnet.

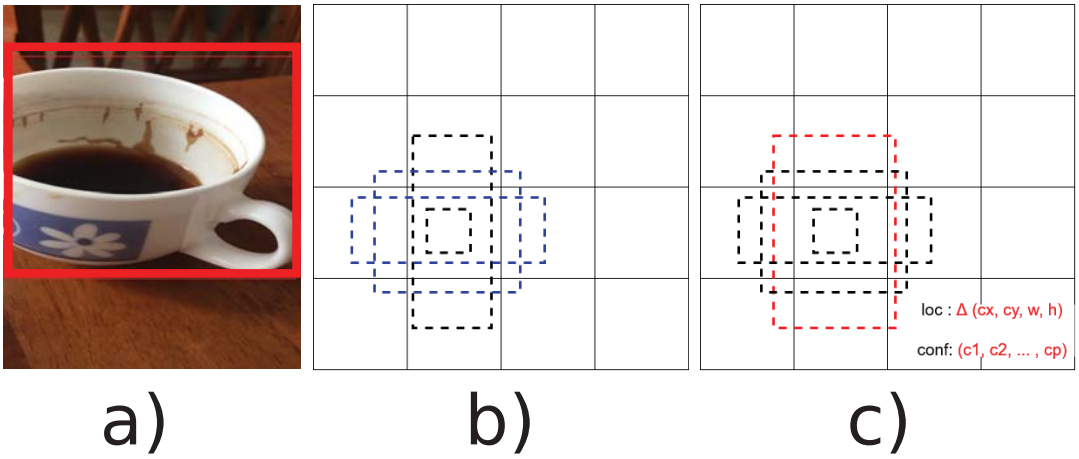

Abbildung 6.1: SSD Framework [38]

6.2 You Only Look Once: Unified, Real-Time Object Detection

Redmon et al. verfolgen einen ähnlichen Ansatz wie das vorgestellte SSD Framework [43]. Ein neuronales Netz berechnet Boxen, die das erkannte Objekt umrahmen, und zum Objekt zugehörige Klassen aus einem Bild in einer Evaluation. Die Funktionsweise besteht aus folgenden Schritten: Ein Convolutional Neural Network sagt gleichzeitig mehrere Boxen und Wahrscheinlichtkeitsklassifizierungen für diese Boxen voraus. **YOLO** (kurz von *You Only Look Once*) verwendet in der Trainingsphase ganze Bilder. Dabei wird mit einer Regressionsanalyse an das Erkennungsproblem herangegangen. **YOLO** ist sehr performant, da es im Gegensatz zu Ansätzen mit rekurrenten neuronalen Netzen keine komplexen Bearbeitungsschritte gibt. In rekurrenten neuronalen Netzen wird zuerst ein Bereich mit potenziellen Boxen generiert, welche das Objekt umrahmen. Danach werden diese Boxen klassifiziert. Nach der Klassifikation folgt die Nachbearbeitung der Boxen und Eliminierung der Duplikate. Für eine Regressionsanalyse werden diese komplexe Schritte nicht benötigt, da ein einziges Convolutional Neural Network mehrere Boxen gleichzeitig vorhersagt.

6.3 Multiple 3D Object Tracking for Augmented Reality

Eine Herangehensweise ohne neuronale Netze wird von Park et al. vorgestellt [44]. Der Schwerpunkt der Arbeit liegt in der Erkennung mehrerer Objekte in einem Bilddatenstrom. Dabei werden Features Frame für Frame erkannt. Für jedes Objekt, das erkannt werden soll, steht ein CAD 3D Modell und eine Menge an Referenzbildern der Feature des Objektes zur Verfügung. Jeder Eingabeframe wird dann mit einer Teilmenge der Features verglichen. Es werden Features verfolgt, die auf dem Objekt erkannt wurden; diese werden dann von Frame zu Frame weiter verfolgt. Die Menge an Treffern wird dann genutzt, um die Position des 3D Objektes in dem nächsten Frame zu schätzen, sodass das Objekt weiter verfolgt werden kann.

Wenn ein neues Objekt in der Kameraeingabe erscheint, initialisiert das System eine Verfolgung des Objektes von eine Bild zur dem nächsten Bild. Um die Verfolgung durchzuführen, werden die Featurepunkte auf der Oberfläche des Objektes erkannt, die dann in den folgenden Bildern verfolgt werden. Von diesem Punkt kann die Schätzung der Position des Objektes stattfinden, während die Objekterkennung keinen Faktor spielt, da das Objekt nur verfolgt wird. Die Verfolgung des Objektes ist im Gegensatz zur Objekterkennung performanter, da bei der Verfolgung eines Objektes, das aktuelle Bild mit dem vorherigen Bild vergleicht werden kann und somit leicht feststellen kann, wo die Pixelbereiche im aktuellen Bild sich im Gegensatz zum vorherigen Bild befinden. Aus der Menge der gefundenen Features werden dann in einem Hintergrundprozess Treffer der Features des betrachteten Objektes und dem Features des CAD 3D Modells und der Refferenzbilder berechnet.

6.4 Augmented Reality Frameworks

Außer dem Framework ARKit existieren noch weitere Frameworks für Augmented Reality, doch nicht alle Frameworks haben eine Unterstützung für Objekterkennung. **ARCore** erfüllt die Grundfunktionalitäten eines Frameworks für Augmented Reality, unterstützt jedoch keine Objekterkennung. Einen ähnlichen Ansatz wie das in Abschnitt 6.3 vorge-

stellte Framework von Park verwendet das Framework **VisionLib** [45], welches CAD Modelldaten analysiert und mit Objekten in der realen Umgebung vergleicht und nach Übereinstimmungen sucht. **Vuforia** [46] ist ebenso ein Framework das Objekterkennung unterstützt. Es erkennt anhand eines 3D Scans ein reales Objekt. Die Funktionsweise ist dabei sehr ähnlich zu der von ARKit.

Gleicherweise gilt auch, dass neben TensorFlow andere Frameworks existieren, die für Machine Learning verwendet werden können. **Core ML** ist ein von Apple entwickeltes Framework. Das TensorFlow Team hat Konverter veröffentlicht, die es ermöglichen TensorFlow Modelle zu konvertieren und in dem Core ML Framework zu verwenden. Eine Open-Source Alternative zu CoreML ist das Framework **PyTorch** [47], das für die Entwicklung von Deep Neuronal Networks geeignet ist.

6.5 Alternative Objekterkennungstechnologien

Eine Objekterkennung kann nicht ausschließlich mit einer Kameraeingabe durchgeführt werden. Es existieren verschiedene Techniken, mit welchen eine Objekterkennung durchgeführt werden kann. Eine Möglichkeit ist es die zu identifizierenden Objekte mit RFID Chips [48] auszustatten. Vogt betrachtet diesen Anwendungsfall, bei dem auch mehrere Objekte gleichzeitig erkannt werden können, wobei die Anzahl der RFID-Tags nicht begrenzt ist [49].

Eine weitere Möglichkeit ist es, Markierungen an die Objekte zu platzieren, die erkannt werden sollen. Garrido-Jurado stellt ein Markierungssystem vor, das aus Referenzmarker besteht, die eine quadratische Form haben. Es wird die Generierung der Markierungen betrachtet und wie genau diese erkannt werden können [50].

7

Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Anwendungsfall betrachtet, in dem Maschinenteile einer Pharmaverpackungsmaschine durch Objekterkennung erkannt werden sollen. Eine zuverlässige Objekterkennung könnte zum Beispiel die Wartung einer Maschine erleichtern, in dem Informationen über die erkannten Teile geliefert werden können. Prozess-orientierte Informationssysteme, die mit einer Objekterkennung überprüfen können, ob zum Beispiel die richtigen Bauteile im entsprechenden Prozessschritt verwendet werden, könnten Service-Abläufe unterstützen und beispielsweise fehlerhafte Prozessausführungen erkennen und vermeiden. Zur Objekterkennung existieren viele Frameworks, von denen im Rahmen dieser Arbeit die beiden Frameworks ARKit und TensorFlow Lite betrachtet wurden. Dabei wurde eine Evaluation der beiden Frameworks durchgeführt, um festzustellen welches der beiden Frameworks sich für die Erkennung von Maschinenteilen besser eignet.

Für die Evaluation der Frameworks wurden zwei Anwendungen entwickelt, welche die beiden Frameworks implementieren. Die Implementierung der Messungen, sowie die grafische Benutzerschnittstelle, wurde in den beiden Anwendungen identisch implementiert. Beide Anwendungen wurden für die iOS Plattform entwickelt, da ARKit nur für die iOS Plattform zur Verfügung steht.

Das Experiment wurde nach bewährten Methoden sorgfältig geplant und beschrieben. Es wurde eine Null-Hypothese aufgesetzt, welche dann im Rahmen des Experiments widerlegt werden sollte. Während des Experiments wurden Sensordaten für Prozessorauslastung, Speichernutzung, Zeit für die 3D-Objekterkennung und ob das 3D-Objekt erkannt wurde, für die Frameworks ARKit und Tensorflow Lite paarweise erfasst. Bei der Durchführung des Experimentes wurden folgende unabhängige Variablen betrachtet:

die Richtung, aus der das Objekt beobachtet wird, der Winkel des Handys zum Objekt und die Entfernung zum betrachteten Objekt. Jedes Maschinenteil wurde aus den Entfernungen von $45cm$, $60cm$ und $80cm$ beobachtet, wobei der Winkel abhängig von der jeweiligen Entfernung war. Weiter wurde jedes Maschinenteil aus jeder Himmelsrichtung beobachtet, um alle Features des einzelnen Maschinenteils bei der Objekterkennung zu betrachten. Der Ablauf eines Versuches bestand aus der Auswahl einer Konfiguration der unabhängigen Variablen, der Auswahl des Maschinenteils und der Auswahl der Anwendung des zu testenden Frameworks.

Nach statistischer Auswertung der erfassten Daten zeigte sich, dass die gesammelten Daten signifikant sind und aus den Ergebnissen des Experiments belastbare Schlussfolgerungen für das betrachtete Experimentalsetup gezogen werden dürfen. Aus den Ergebnissen folgt, dass das ARKit Framework nur beim gemessenen Merkmal Prozessorauslastung besser abgeschnitten hat als das TensorFlow Lite Framework. Bei allen anderen gemessenen Merkmalen hat das TensorFlow Lite Framework besser abgeschnitten. Daraus folgt, dass sich für die in diesem Experiment verwendeten Maschinenteile, der verwendeten Umgebung sowie den verwendeten Instrumenten das TensorFlow Lite Framework besser für eine Objekterkennung dieser Maschinenteile eignet.

Basierend auf diesen Ergebnissen bieten sich weitere Experimente an, die neben veränderten Licht- und Raumparametern auch andere Geräteklassen untersuchen, um weiterreichende Aussagen über die Verwendbarkeit der Objekterkennungsframeworks für industrielle Umgebungen treffen zu können.

Literaturverzeichnis

[1] Ballard, D.H., Brown, C.M.: Computer Vision. 1st edn. Prentice Hall Professional Technical Reference (1982)

[2] Szeliski, R.: Computer Vision: Algorithms and Applications. 1st edn. Springer-Verlag, Berlin, Heidelberg (2010)

[3] Rossol, L. In: Computer Vision in Industry. Springer Berlin Heidelberg, Berlin, Heidelberg (1983) 11–18

[4] Pun, T., Gerig, G., Ratib, O.: Image analysis and computer vision in medicine. Computerized Medical Imaging and Graphics **18** (1994) 85 – 96 Multimedia Techniques in the Medical Environment.

[5] Capitán-Vallvey, L.F., López-Ruiz, N., Martínez-Olmos, A., Erenas, M.M., Palma, A.J.: Recent developments in computer vision-based analytical chemistry: A tutorial review. Analytica Chimica Acta **899** (2015) 23 – 56

[6] Mori, S., Nishida, H., Yamada, H.: Optical Character Recognition. 1st edn. John Wiley & Sons, Inc., New York, NY, USA (1999)

[7] Rowley, H.A., Baluja, S., Kanade, T.: Rotation invariant neural network-based face detection. In: Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231). (1998) 38–44

[8] Collins, R.T., Lipton, A.J., Kanade, T.: Introduction to the special section on video surveillance. IEEE Transactions on pattern analysis and machine intelligence **22** (2000) 745–746

[9] Gonzales, R.C., Woods, R.E.: Digital image processing. Volume 2. Prentice hall New Jersey (2002)

[10] Forsyth, D.A., Ponce, J.: Computer Vision: A Modern Approach. Prentice Hall Professional Technical Reference (2002)

[11] Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision **60** (2004) 91–110

[12] Harris, C., Stephens, M.: A combined corner and edge detector. In: In Proc. of Fourth Alvey Vision Conference. (1988) 147–151

[13] Moravec, H.: Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical Report CMU-RI-TR-80-03, Carnegie Mellon University, Pittsburgh, PA (1980)

[14] Nister, D., Naroditsky, O., Bergen, J.: Visual odometry. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004. Volume 1. (2004) I–I

[15] Murphy, K.P.: Machine Learning: A Probabilistic Perspective. The MIT Press (2012)

[16] Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) `http://www.deeplearningbook.org`.

[17] Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. CoRR **abs/1311.2901** (2013)

[18] Milgram, P., Kishino, F.: A taxonomy of mixed reality visual displays. IEICE TRANSACTIONS on Information and Systems **77** (1994) 1321–1329

[19] Steuer, J.: Defining virtual reality: Dimensions determining telepresence. Journal of communication **42** (1992) 73–93

[20] Azuma, R.T.: A survey of augmented reality. Presence: Teleoperators & Virtual Environments **6** (1997) 355–385

[21] Van Krevelen, D., Poelman, R.: Augmented reality: Technologies, applications, and limitations. Vrije Univ. Amsterdam, Dep. Comput. Sci (2007)

[22] Patel, S.N., Rekimoto, J., Abowd, G.D.: icam: Precise at-a-distance interaction in the physical environment. In Fishkin, K.P., Schiele, B., Nixon, P., Quigley, A., eds.: Pervasive Computing, Berlin, Heidelberg, Springer Berlin Heidelberg (2006) 272–287

[23] Sutherland, I.E.: A head-mounted three dimensional display. In: Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I. AFIPS '68 (Fall, part I), New York, NY, USA, ACM (1968) 757–764

[24] Raab, F.H., Blood, E.B., Steiner, T.O., Jones, H.R.: Magnetic position and orientation tracking system. IEEE Transactions on Aerospace and Electronic Systems **AES-15** (1979) 709–718

[25] Naimark, L., Foxlin, E.: Circular data matrix fiducial system and robust image processing for a wearable vision-inertial self-tracker. In: Proceedings. International Symposium on Mixed and Augmented Reality. (2002) 27–36

[26] Kar Wee Chia, Cheok, A.D., Prince, S.J.D.: Online 6 dof augmented reality registration from natural features. In: Proceedings. International Symposium on Mixed and Augmented Reality. (2002) 305–313

[27] Ishii, H., Ullmer, B.: Tangible bits: Towards seamless interfaces between people, bits and atoms. In: Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems. CHI '97, New York, NY, USA, ACM (1997) 234–241

[28] Buchmann, V., Violich, S., Billinghurst, M., Cockburn, A.: Fingartips: Gesture based direct manipulation in augmented reality. In: Proceedings of the 2Nd International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia. GRAPHITE '04, New York, NY, USA, ACM (2004) 212–221

[29] Mayol, W.W., Murray, D.W.: Wearable hand activity recognition for event summarization. In: Ninth IEEE International Symposium on Wearable Computers (ISWC'05). (2005) 122–129

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

[31] Dean, J., Corrado, G.S., Monga, R., Chen, K., Devin, M., Le, Q.V., Mao, M.Z., Ranzato, M., Senior, A., Tucker, P., Yang, K., Ng, A.Y.: Large scale distributed deep networks. In: NIPS. (2012)

[32] Monostori, L.: Cyber-physical production systems: Roots, expectations and r&d challenges. Procedia Cirp **17** (2014) 9–13

[33] Khaitan, S.K., McCalley, J.D.: Design techniques and applications of cyberphysical systems: A survey. IEEE Systems Journal **9** (2015) 350–365

[34] Hoppenstedt, B., Pryss, R., Stelzer, B., Meyer-Brötz, F., Kammerer, K., Treß, A., Reichert, M.: Techniques and emerging trends for state of the art equipment maintenance systems - a bibliometric analysis. Applied Sciences **8** (2018) 1–29

[35] Kammerer, K., Pryss, R., Sommer, K., Reichert, M.: Towards context-aware process guidance in cyber-physical systems with augmented reality. In: 4th International Workshop on Requirements Engineering for Self-Adaptive, Collaborative, and Cyber Physical Systems (RESACS'18), IEEE Computer Society Press (2018)

[36] Hoppenstedt, B., Kammerer, K., Reichert, M., Spiliopoulou, M., Pryss, R.: Convolutional neural networks for image recognition in mixed reality using voice command labeling. In: 6th International Conference on Augmented Reality, Virtual Reality and Computer Graphics (SALENTO AVR 2019). Lecture Notes in Computer Science, Springer (2019)

[37] Tang, J.: Intelligent Mobile Projects with TensorFlow: Build 10+ Artificial Intelligence Apps Using TensorFlow Mobile and Lite for iOS, Android, and Raspberry Pi. Packt Publishing (2018)

[38] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: SSD: single shot multibox detector. CoRR **abs/1512.02325** (2015)

[39] Lin, T., Maire, M., Belongie, S.J., Bourdev, L.D., Girshick, R.B., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: common objects in context. CoRR **abs/1405.0312** (2014)

[40] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering: An Introduction. Kluwer Academic Publishers, Norwell, MA, USA (2000)

[41] Cook, T., Campbell, D.: Quasi-Experimentation: Design and Analysis Issues for Field Settings. Houghton Mifflin (1979)

[42] Agresti, A.: A survey of exact inference for contingency tables (1992)

[43] Redmon, J., Divvala, S.K., Girshick, R.B., Farhadi, A.: You only look once: Unified, real-time object detection. CoRR **abs/1506.02640** (2015)

[44] Park, Y., Lepetit, V., Woontack Woo: Multiple 3d object tracking for augmented reality. In: 2008 7th IEEE/ACM International Symposium on Mixed and Augmented Reality. (2008) 117–120

[45] visionLib: visionLib – Augmented Reality Tracking Library for industries. `https://visionlib.com/` (2019) [Online; Zugriff 24-Novembar-2019].

[46] Vuforia: Vuforia Engine. `https://developer.vuforia.com/` (2019) [Online; Zugriff 24-Novembar-2019].

[47] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NeurIPS Autodiff Workshop. (2017)

[48] Roh, S., Choi, H.R.: 3-d tag-based rfid system for recognition of object. IEEE Transactions on Automation Science and Engineering **6** (2009) 55–65

[49] Vogt, H.: Multiple object identification with passive rfid tags. In: IEEE International Conference on Systems, Man and Cybernetics. Volume 3. (2002) 6 pp. vol.3–

[50] Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., Marín-Jiménez, M.: Automatic generation and detection of highly reliable fiducial markers under occlusion. Pattern Recognition **47** (2014) 2280–2292

Abbildungsverzeichnis

Tabellenverzeichnis

Name: Kristijan Biro Matrikelnummer: 898418

Erklärung

Ich erkläre, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Ulm, den .....

Kristijan Biro