HAW HAMBURG

MASTER THESIS
Nadia Hintze

Einsatz synthetisch erzeugter Daten zur Verbesserung von Deep Learning Modellen zur Objekterkennung

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Nadia Hintze

Einsatz synthetisch erzeugter Daten zur Verbesserung von Deep Learning Modellen zur Objekterkennung

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Dr. Susanne Draheim

Eingereicht am: 19.08.2024

Nadia Hintze

Thema der Arbeit

Einsatz synthetisch erzeugter Daten zur Verbesserung von Deep Learning Modellen zur Objekterkennung

Stichworte

Objekterkennung, Synthetische Trainingsdaten, YOLOv8, Unity3D Engine

Kurzzusammenfassung

Neuronale Netze zur Objekterkennung sind auf eine ausreichende Anzahl qualitativ hochwertiger Trainingsdaten angewiesen. Solche Trainingsdaten mit Aufnahmen realer Umgebungen zu beschaffen ist jedoch ein langwieriger und aufwändiger Prozess. Die gebrauchten Trainingsdaten können alternativ in einer virtuellen 3D-Umgebung synthetisch generiert werden. Diese Arbeit beantwortet die Frage, inwiefern solche Daten genaue und robuste Modelle erzeugen können. Dabei setzt sie sich mit den Einflüssen verschiedener Aspekte der Umgebungsgestaltung, wie z.B. der Orientierung an der Anwendungsdomäne oder einer besonders hohen Varianz, auseinander. Zu diesem Zweck wird eine Anwendung implementiert, die synthetische Datensätze mit unterschiedlich gestalteten Umgebungen generiert. Der Vergleich von Modellen, die mit diesen Datensätzen trainiert wurden, zeigt, dass synthetische Daten als eigenständige Datensätze gute Trainingsergebnisse liefern können. Der gemeinsame Einsatz von synthetischen und real aufgenommenen Daten kann darüber hinaus einen Zusatz an Genauigkeit und Robustheit gegenüber ausschließlich real angefertigten Daten bewirken.

Nadia Hintze

Title of Thesis

Use of synthetically generated data to improve deep learning models for object recognition

Keywords

Object Recognition, Synthetic Data, YOLOv8, Unity3D Engine

Abstract

Neural networks for object recognition are dependent on a sufficient amount of high-quality training data. However, obtaining such training data with images of real environments is a lengthy and time-consuming process. Alternatively, the required training data can be generated synthetically in a virtual 3D environment. This thesis answers the question to what extent such data can generate precise and robust models. It deals with the influences of different aspects of the environment design, such as the orientation to the application domain or a particularly high variance. For this purpose, an application is implemented that generates synthetic data sets with differently designed environments. The comparison of models trained with these data sets shows that synthetic data can provide good training results as independent data sets. The joint use of synthetic and real recorded data can also result in additional accuracy and robustness compared to exclusively real data.

Inhaltsverzeichnis

1 Einleitung

Deep Learning Modelle sind in der Lage, komplexe Aufgaben anhand von Beispielen zu lösen. Um gute Ergebnisse zu erzielen, sind sie darauf angewiesen, über ausreichend viele, qualitativ hochwertige und variierende Trainingsdaten zu verfügen. Wenn dies nicht der Fall ist, drohen Ungenauigkeit und eine unzureichende Generalisierbarkeit der trainierten Funktion [31]. Um beispielsweise Objekte auf RGB-Bildern zu erkennen, ist es für ein optimales Trainings-Ergebnis nicht unüblich, dass sich der Umfang des benötigten Datensatzes im fünf-stelligen Bereich bewegt [5, 7, 31]. Die Beschaffung eben jener Daten gestaltet sich jedoch in der Regel aufwändig.

Trainingsdaten zur Erkennung von Objekten zu erstellen lässt sich in zwei Aufgaben unterteilen: Die Aufnahme von Kamerabildern und die Anfertigung der dazugehörigen Labels. Beide Aufgaben bringen ihre eigenen Herausforderungen mit sich.
Für die Labels müssen die Ground Truth Informationen über die gesuchten Objekte auf jedem Kamerabild ermittelt werden. Wenn beispielsweise die Objekt-Pose erkannt werden soll, muss diese Pose auf jedem Bild möglichst exakt bestimmt werden. Gleichzeitig sollten aber keine Hilfsmittel auf den Kamerabildern zu sehen sein, die nicht zu der Umgebung gehören, in denen das fertig trainierte Modell angewendet wird, da sich das trainierte Modell ansonsten primär an der Ansicht der Hilfsmittel orientieren könnte. Das erfordert den Einsatz eines Systems, das diesen Anforderungen gerecht wird [13]. Auch im Fall einer angestrebten Erkennung von 2D-Bounding-Boxen von Objekten auf Kamerabildern, müssen diese erst einmal auf den Trainingsbildern bestimmt werden, was entweder ein gesondertes System oder die manuelle Definition der Bounding Boxen auf jedem einzelnen Bild erfordert.
Obwohl Hintze [13] ein System entwickelte und nutzte, das die Erzeugung von Labels komplett automatisiert, blieb die Aufnahme der Bilder aufwändig. Die Variation der Umgebung vor der Kamera, z.B. hinsichtlich der Lichtverhältnisse oder der Anordnung der sichtbaren Objekte, musste für jedes Bild händisch umgesetzt werden. So entstand in dem Projekt in ca. 30 Minuten ein Datensatz von 53 Bildern. In diesem Fall waren für die

Umgebungsgestaltung nur die Beleuchtung in Form künstlicher Innenraum Beleuchtung und Objekte, die für einen Menschen leicht beweglich sind, relevant [13]. Jedoch gibt es auch Anwendungsfälle, bei denen die Veränderung der Umgebung deutlich umfangreicher ist, wie z.B. bei der visuellen Analyse im Straßenverkehr. Hier sind Fahrzeuge, die Beleuchtung des Straßenverkehrs und die Infrastruktur der Umgebung die zu variierenden Elemente [29]. Auch Fehlerfälle, wie z.B. ein verwackeltes Kamerabild, ungünstige Verdeckungen der Objekte und Rauschen im Kamerabild müssen explizit im Datensatz vertreten sein [5]. Selbst, wenn die Variationen gründlich im Voraus geplant werden, ist die Aufnahme zehntausender Bilder ein umfangreiches Unterfangen. Zudem kann die Umgebung, die fotografiert wird, aus anderen Gründen limitiert sein. Beispielsweise, weil man Schwierigkeiten hat, sich in der Umgebung aufzuhalten, in der später das Modell angewandt wird, weil besonderes Expertenwissen benötigt wird, um die Umgebung zu gestalten oder, weil es physikalische Limitationen der Varianz in der dargestellten Umgebung gibt. So ist es z.B. schwierig, auf Fotos reale Objekte gezielt zu fotografieren, während sie sich in der Luft befinden. Die Beschaffung real aufgenommener Trainingsdaten ist aus diesen Gründen in der Regel qualitativ und quantitativ eingeschränkt.

Um diese Einschränkungen auszugleichen, existieren Strategien, Trainingsdaten auf einem synthetischen Weg zu generieren. Diese Generierung kann auf real angefertigten Bildern basieren, indem z.B. generative Netze dahingehend trainiert werden, ähnliche Bilder zu produzieren [36]. Ein weiteres Beispiel ist die Data Augmentation, die die Kamerabilder mithilfe von optischen Filtern, wie z.B. modifizierten Kontrastwerten oder Verschwommenheit, verändert [23]. Es gibt allerdings auch die Option, Daten rein synthetisch, also ohne die Basis real aufgenommener Trainingsdaten, aufzubauen. Auf diese Option konzentriert sich diese vorliegende Arbeit. Dafür wird in einer virtuellen, dreidimensionalen Szene, z.B. über eine Game Engine, eine Umgebung aufgebaut, die unter anderem ein 3D-Modell des gesuchten Objekts beinhaltet. Anschließend werden Schnappschüsse davon angefertigt und mit automatisch ermittelbaren Informationen gelabelt. Diese Informationen können z.B. aus Posen, 2D Bounding Boxen, Tiefeninformationen oder Segmentierungen der Kamerabilder bestehen [30][7]. Dieses Vorgehen verspricht den Aufwand der Gestaltung der fotografierten Umgebung und der Bestimmung der Ground Truth zu verringern, während die erzeugte Varianz steigt [7][2][29].

Die Verwendung dieser rein synthetisch konstruierten Daten birgt jedoch das Problem des *Reality Gaps*. Tremblay u. a. [31] beschreiben diesen als Differenz zwischen real aufgenommenen und synthetischen Bilddaten. Der Reality Gap kann dafür verantwortlich sein, dass die reale Umgebung, in der das fertig trainierte Modell angewandt werden

soll, so wenig von den Trainingsdaten repräsentiert wird, dass es dort nicht funktionieren kann. Mittlerweile existieren Arbeiten, deren Trainingsergebnisse rein synthetischer Daten mit denen real aufgenommener Bilddaten mithalten können. Dazu gehören die Arbeiten von Damian u. a. [7] und die von Tremblay u. a. [31]. Beide haben den Reality Gap überbrückt, indem eine möglichst realistische Umgebung virtuell nachgebaut wurde. Diese virtuelle Szene aufzubauen bedeutet jedoch auch einen relativ großen Aufwand, insbesondere dann, wenn großer Wert auf visuellen Realismus gelegt wird. Daher ergibt sich die Frage, welche Eigenschaften synthetische Daten mitbringen müssen, um gute Ergebnisse zu erzielen.

Die vorliegende Arbeit verfolgt das Ziel, zu ergründen, wie synthetisch generierte Bilddaten verwendet werden können, um genaue und robuste Modelle zur Objekterkennung zu erstellen. Das Thema wird in vier Forschungsfragen unterteilt, die wie folgt lauten:

1. Lässt sich das Trainingsergebnis realer Daten durch den zusätzlichen Einsatz synthetischer Daten verbessern?
2. Reichen synthetische Daten für ein gutes Trainingsergebnis aus?
3. Wie stark hängen der visuelle Realismus und die Varianz der Trainingsdaten mit dem Trainingsergebnis zusammen?
4. Wie viel Aufwand ist bei der Gestaltung der synthetischen Daten notwendig, um das Trainingsergebnis gegenüber der Verwendung realer Daten zu verbessern?

Als Grundlage der Untersuchung wird eine Anwendung in dieser Arbeit entwickelt, die die Generierung von synthetischen Datensätzen unterstützt. Dabei werden unterschiedliche Strategien zur Gestaltung der virtuellen Umgebung umgesetzt. Das neuronale Netz YOLOv8 wird mit allen generierten Datensätzen trainiert und die resultierenden Modelle mit denselben Testdaten auf Genauigkeit und Robustheit geprüft. Gemeinsam mit der Betrachtung verwandter Arbeiten zu dem Thema bietet das die Möglichkeit, die Forschungsfragen zu beantworten.

Aufbau der Arbeit

Die Betrachtung der verwandten Arbeiten findet als erstes in Kap. 2 statt. Darin werden anhand von Beispielen aus der Literatur Methoden zur Gestaltung synthetischer Bilddaten beschrieben und danach kategorisiert, wie stark visueller Realismus in den Datensätzen eine Rolle spielt. Zudem werden die Erkenntnisse der Autoren hinsichtlich der Forschungsfragen vorgestellt. Diese Übersicht dient in Kapitel 3 als Grundlage dafür, eigene Strategien zur Datengestaltung mit unterschiedlich realistischen und variablen Aspekten zu definieren, nach denen in dieser Arbeit eigene Datensätze entwickelt und getestet werden sollen. Auch die Rahmenbedingungen der geplanten Experimente, wie das zu trainierende neuronale Netz, die eingesetzte Software und die Anwendungsdomäne, werden hier ausgewählt. In Kapitel 4 werden die praktische Umsetzung der geplanten Anwendung und die Generierung der synthetischen Datensätze dokumentiert. Mit jedem gewonnenen Datensätzen wird in Kapitel 5 ein YOLOv8-Modell trainiert. Die Modelle werden anschließend mithilfe in realen Umgebungen aufgenommenen Testdatensätzen evaluiert. Zum Abschluss werden in Kapitel 6 die wichtigsten Erkenntnisse dieser Arbeit zusammengefasst und genutzt, um die zuvor gestellten Forschungsfragen zu beantworten. Außerdem wird ein Ausblick auf mögliche Erweiterungen der Experimente gegeben.

2 Methoden zur Gewinnung von Trainingsdaten

In dieser Arbeit wird ergründet, welche Einflüsse der Reality Gap synthetischer Trainingsdaten auf die Qualität von Trainingsergebnissen hat. Hierfür werden in diesem Kapitel bislang erfolgte Arbeiten, die sich mit diesem Thema auseinandergesetzt haben, vorgestellt. Alle Arbeiten haben dabei das Ziel verfolgt, Datensätze synthetisch zu konstruieren, deren Trainingsergebnisse mit denen real angefertigter Trainingsdaten mithalten können. Um den Einfluss der visuellen Realitätsnähe genauer einordnen zu können, werden nun im ersten Schritt diese Arbeiten mit besonderem Augenmerk auf der Gestaltung ihrer virtuellen Umgebungen kategorisiert. Im zweiten Schritt werden die Erkenntnisse der Autoren in Bezug auf die Forschungsfragen vorgestellt, sodass darauf aufbauend in Kap. 3 die in dieser Arbeit erfolgenden Experimente entworfen werden können.

2.1 Kategorien der Umgebungsgestaltung

Die hier betrachteten Strategien zur Gestaltung synthetischer Daten werden in zwei Kategorien eingeteilt: Die Domain Randomization und die Structured Domain Randomization. Die Grenzen zwischen den hier vorgestellten Kategorien sind fließend und die Begrifflichkeiten unterscheiden sich wegen ihrer fehlenden einheitlichen Definition von Autor zu Autor. Zum Zweck der Übersichtlichkeit und Vergleichbarkeit wird hier anhand bestimmter Eigenschaften, die in den folgenden Kapiteln jeweils festgelegt werden, daher eine eigene Gruppierung vorgenommen.

2.1.1 Domain Randomization

Die erste Strategie befreit sich vollständig von dem Anspruch auf visuellen Realismus. Hier wird Wert darauf gelegt, dass die Umgebung möglichst einfach und schnell prozedu-

ral erstellt werden kann, ohne bei der Gestaltung Expertise in dieser Domäne vorauszusetzen. Um das zu gewährleisten, kommen in erster Linie, bis auf die 3D-Modelle der im Voraus bekannten Zielobjekte, rein zufällig gewählte 3D-Objekte und Texturen zum Einsatz. Auch die Anordnung aller 3D-Objekte erfolgt größtenteils zufällig. Auf diese Weise ist die Variation der entstehenden Daten besonders groß, wovon man sich eine daraus resultierende Robustheit des Trainingsergebnisses gegenüber unbekannten Umgebungen erhofft. Inwiefern dieser Zusammenhang in den hier vorgestellten Arbeiten nachgewiesen wurde, wird in Kap. 2.2 erläutert.

Jonathan Tremblay hat sich in mehreren Arbeiten mit der Verwendung von Domain Randomized Trainingsdaten auseinandergesetzt [31, 30, 29]. In der ersten Arbeit haben Tremblay u. a. [31] einen Datensatz erstellt, der die Posen von 21 im voraus bekannten Haushaltsgegenständen mit sechs Freiheitsgraden auf einzelnen RGB-Bildern erkennen soll. Sowohl die 3D-Modelle der Zielobjekte als auch die RGB-Bilder für die Inferenz und die Evaluation des trainierten Modells wurden dem YCB-Datensatz von Calli u. a. [3] entnommen.
Bei der Gestaltung der virtuellen Umgebung gehen Tremblay u. a. [31] wie folgt vor: Zunächst wird eine zufällige Menge virtueller Zielobjekte ausgewählt, die in zufälligen Posen in der Umgebung vor der Kamera verteilt werden. Anschließend wird auf dieselbe Weise mit einer zufälligen Anzahl geometrisch primitiver Objekte verfahren, die ebenfalls zufällig skaliert werden. Diese Objekte bestehen unter anderem aus Quadern, Sphären, Zylindern oder Kegeln. Sie werden auch *Distraktoren* genannt, da sie bewusst so positioniert werden, dass sie Zielobjekte verdecken können und visuell von ihnen ablenken, um vergleichbare reale Umstände zu simulieren. Alle Distraktoren bekommen eine Textur, die entweder einfarbig oder gestreift gestaltet wird. Den Hintergrund bildet eine senkrechte Ebene, die mit einem realen Foto, einer gestreiften oder einer einfarbigen Textur versehen wird. Abschließend erfolgt die Beleuchtung der Umgebung aus einer zufälligen Richtung und der Blickwinkel der Kamera wird ebenfalls zufällig ausgewählt. All diese veränderlichen Aspekte schaffen eine umfangreiche Variabilität der generierten Daten.

Tremblay u. a. [29] haben in der zweiten Arbeit angestrebt, 2D-Bounding-Boxen von 36 bekannten Fahrzeugen im KITTI-Datensatz[11], der Kamerabilder aus dem Straßenverkehr enthält, zu erkennen. Der Ablauf der Umgebungsgestaltung ähnelt dem des anderen Projekts. Ein Unterschied liegt in der Platzierung der gesuchten Fahrzeuge. Sie werden auf einer 2D-Ebene positioniert, die einen realen Boden simuliert. In dieser Hinsicht sind die Resultate verglichen mit der ersten Arbeit von Tremblay u. a. [31] optisch realistischer. Die Distraktoren werden von den Autoren auch hier zufällig in der Luft platziert,

Abbildung 2.1: Auszug aus den von Tremblay u. a. [31] generierten Trainingsdaten
Quelle: Tremblay u. a. [31]

um Verdeckungen der gesuchten Fahrzeuge möglichst vielfältig zu gestalten. Ein weiterer Unterschied ist, dass als Distraktoren auch visuell komplexere Objekte, wie z.B. Palmen, gewählt wurden. Auch auf diese Art und Weise unterscheiden sich die Verdeckungen stärker untereinander. Für eine zusätzliche Variation haben die Autoren außerdem optische Filter für eine Data Augmentation benutzt. Zufallsbasiert kamen 90° Drehungen des Bildes, Spiegelungen, Zuschneidungen, Gaußsches Rauschen sowie Veränderungen von Helligkeit und Kontrast zum Einsatz.

Abbildung 2.2: Auszug aus den von Tremblay u. a. [29] generierten Trainingsdaten
Quelle: Tremblay u. a. [29]

Ein weiteres Beispiel für einen Datensatz, der in der Erstellung stark der Methode von Tremblay u. a. [29] gleicht, ist die Arbeit von Damian u. a. [7]. Hier wurden dieselben Faktoren zufallsbasiert verändert. Zielobjekte sind hier jedoch per Photogrammetrie virtuell erfasste Tonnen.

Die Methodik von Cunha u. a. [5] hebt sich von den anderen hier vorgestellten Arbeiten dadurch ab, dass Bild-Segmente realer Kamerabilder in ihre anderweitig synthetischen Trainingsbilder eingearbeitet werden. Das erreichen die Autoren, indem sie zuerst eine Aufnahme des realen Zielobjektes anfertigen, die Pose dieses Objekts relativ zur Kamera bestimmen, das Bild semantisch segmentieren, das Zielobjekte ausschneiden und abschließend den realen Hintergrund gegen einen synthetisch in der Unity3D Engine ge-

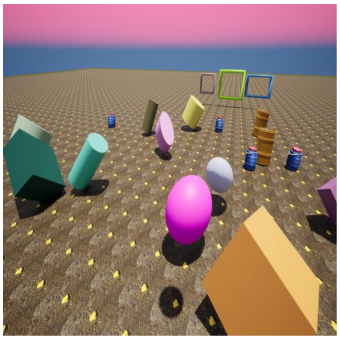

Abbildung 2.3: Beispiel aus den von Damian u. a. [7] generierten Trainingsdaten
Quelle: Damian u. a. [7]

nerierten Hintergrund austauschen. Die virtuelle Umgebung wird dabei entsprechend der Pose des realen Objekts ausgerichtet, sodass es optisch so wirkt, als würde das reale Objekt auf dem virtuellen Boden stehen. Indem ein digitaler Zwilling des realen Objekts in der realen Pose vor der virtuellen Kamera platziert wird, können Informationen über das reale Objekt in der virtuellen Umgebung geschätzt werden. So kann beispielsweise die Maske zur semantischen Segmentierung in der virtuellen Szene automatisch kalkuliert werden. Außerdem können Tiefeninformationen gemessen werden, ohne auf reale Sensorik angewiesen zu sein, falls das verwendete neuronale Netz diese Informationen erfordert. Auf den Trainingsbildern wird in dieser Arbeit ein texturloses, 3D-gedrucktes Spielzeugauto gesucht. Der Aufbau der synthetischen Umgebung gleicht dem der zweiten Arbeit von Tremblay u. a. [29]. Für zufällige Texturen der Distraktoren, des Bodens und des Hintergrunds hat sich diese Arbeit jedoch an dem VOC-Datensatz[8] bedient.

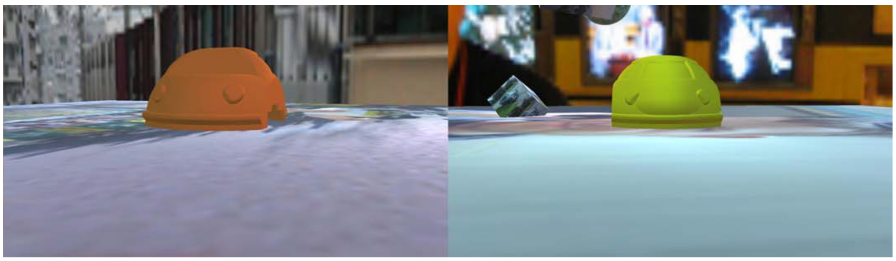

Abbildung 2.4: Auszug aus den von Cunha u. a. [5] generierten Trainingsdaten
Quelle: Cunha u. a. [5]

Die synthetischen Daten von Borkman u. a. [1] fallen im Vergleich zu den anderen Datensätzen dadurch auf, dass die Hintergründe hier nicht durch einfache, zweidimensionale Texturen dargestellt werden, sondern mehrere dreidimensionale Objekte zu einem Hin-

tergrund zusammengefügt werden. Dafür werden so viele geometrisch primitive Objekte zufällig in einem begrenzten Raum platziert, bis es aus Sicht der Kamera keine Lücken mehr gibt und diese Objekte eine Wand bilden. Gesucht werden in diesem Datensatz Produkte aus einem Lebensmittel-Supermarkt. Als Texturen der Hintergrund-Objekte und der Distraktoren haben die Autoren zufällige Bilder aus dem Datensatz von Klasson u. a. [17], der ebenfalls Darstellungen von verpackten und unverpackten Lebensmitteln enthält, verwendet. Durch diese Strategie werden Kamerabilder generiert, die einem Anwendungsfall mit besonders starker visueller Unordnung nahekommen, bei der einzelne Supermarkt-Produkte unter vielen erkannt werden müssen. Ähnlich wie in den anderen zuvor vorgestellten Arbeiten wurden hier neben den Texturen auch die Farbgebungen und Skalierungen der einzelnen Distraktoren und der Hintergrund-Objekte sowie die Platzierung von ihnen und den Zielobjekten zufallsbasiert gestaltet. Auch die Beleuchtung wurde immer wieder verändert. Ähnlich wie in der Arbeit von Tremblay u. a. [29] wird auch hier mit unterschiedlichen Kameraeinstellungen, wie z.B. hinsichtlich des Kontrasts und der Sättigung, experimentiert.

Abbildung 2.5: Beispiel aus den von Borkman u. a. [1] generierten Trainingsdaten
Quelle: Borkman u. a. [1]

2.1.2 Structured Domain Randomization

Im Gegensatz zur Domain Randomization ist die Structured Domain Randomization an der Anwendungsdomäne orientiert. Prakash u. a. [21] definieren diese Methode durch die Aufrechterhaltung des Kontextes dieser Domäne bei der Umgebungsgestaltung trotz des Einsatzes zufällig gestalteter Elemente. Hier wird also eine gewisse Expertise über die Anwendungsdomäne vorausgesetzt, jedoch kann der letztendliche, genaue Einsatzort außer Acht gelassen werden.

In ihrer Arbeit generieren Prakash u. a. [21] beispielsweise Bilder, die den Straßenverkehr widerspiegeln sollen. Dafür haben sie zufällige Kurven im virtuellen 3D-Raum festgelegt, an denen nebeneinander in einer zufälligen Reihenfolge Straßen, Baum-Reihen, Häuser-Blöcke, Bürgersteige oder Parkzonen verlaufen. Zusätzlich werden Fahrzeuge, Menschen oder andere Objekte darauf jeweils zufällig verteilt und der Lichteinfall einer zufälligen Tageszeit simuliert. So entstehen Szenen, die zwar vom Zufall abhängen und eine gewisse Variation beinhalten, die jedoch auch immer als realistischer Straßenverkehr zu identifizieren sind.

Abbildung 2.6: Auszug aus den von Prakash u. a. [21] generierten Trainingsdaten
Quelle: Prakash u. a. [21]

Tobin u. a. [28] generieren hingegen Trainingsdaten für die Steuerung eines Roboterarms, der zum Greifen realer Bauklötze diese zuvor auf einem Bild lokalisieren muss. Die reale Umgebung besteht aus einem Tisch, auf dem die Bauklötze liegen, einem Stuhl, der dahinter steht und dem Boden. Um die anvisierte, reale Umgebung virtuell nachzustellen, haben die Autoren zunächst den Tisch durch einen Quader abstrahiert, den Stuhl virtuell aus mehreren 3D-Elementen zusammengesetzt, für den Boden eine waagerechte und für den weiteren Hintergrund eine senkrechte Ebene platziert. Variiert haben sie nun die Texturen und Farbgebungen aller virtuellen Elemente sowie die Anzahl und die Positionen der virtuellen Bauklötze. Ebenso wurden der Blickwinkel der Kamera und die Beleuchtung zufällig gestaltet, ergänzt durch die zufallsbasierte Simulation von unterschiedlich starkem Bildrauschen.

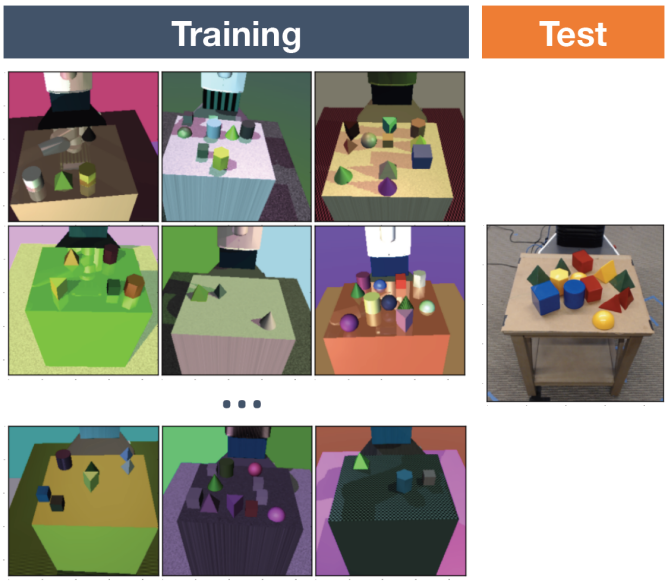

Abbildung 2.7: Auszug aus den generierten Trainingsdaten im Vergleich zu realen Testdaten
Quelle: Tobin u. a. [28]

Tobin u. a. [28] haben ihre Methode als Domain Randomization betitelt und werden auch in mehreren anderen Arbeiten als erstes Beispiel für einen rein synthetischen Domain Randomised Datensatz, der in einer virtuellen 3D-Umgebungen generiert wurde, zitiert [7, 31, 21]. Die vorliegende Arbeit ordnet die Methode von Tobin u. a. [28] im Vergleich zu den anderen Arbeiten eher als Structured Domain Randomization ein, da der Aufbau der virtuellen Umgebung stark an der anvisierten Domäne orientiert ist. Jedoch ist sie verglichen mit den restlichen, hier als SDR-Datensätze eingestuften Arbeiten weniger realistisch, da der Tisch und der Stuhl abstrahiert und auch die Texturen willkürlich gewählt wurden.

Neben dem Domain Randomised Datensatz aus Kap. 2.1.1 haben Tremblay u. a. [31] auch einen Datensatz konstruiert, der visuell an der Anwendungsdomäne orientiert wurde. Ziel des Datensatzes ist es, Haushaltsgegenstände auf Kamerabildern zu erkennen, die in drei unterschiedlichen Umgebungen aufgenommen wurden. Diese Umgebungen bestehen aus einem Wohnzimmer, einem Waldabschnitt und einem Raum in einem Tempel. Dafür haben die Autoren zunächst die drei Umgebungen aufgebaut, indem die meisten erforderlichen Objekte, inklusive einer realistischen Beleuchtung, statisch dort platziert wurden. Innerhalb dieser Umgebungen wurden anschließend ebenfalls statisch jeweils fünf Bereiche festgelegt, die für die anzufertigenden Kamerabilder im Vordergrund ste-

hen sollen. Diese Bereiche wurden so ausgewählt, dass sich die Beleuchtungsverhältnisse und Hintergründe jeweils zwischen den Bereichen unterscheiden. Für die Platzierung der Zielobjekte und der Distraktoren haben die Entwickler simuliert, dass diese Objekte physikalisch korrekt in die Szene fallen und realistisch miteinander kollidieren. Die Ausgangsposition, von der aus sie fallen gelassen wurden, befand sich innerhalb einer Box, die in dem zuvor festgelegten Bereich definiert wurde. Die genaue Ausgangspose in der Box wurde zufällig bestimmt, um eine Variation zwischen den Daten zu schaffen. Während die Objekte zur Laufzeit fielen, änderte die virtuelle Kamera konstant ihre Position und Ausrichtung und machte laufend Aufnahmen der Szene, während die Informationen für die Labels festgehalten wurden. Der auf diese Weise gewonnene Datensatz wurde separat unter dem Namen *Fallen Things („FAT“)* veröffentlicht und zusätzlich in einer eigenen Arbeit vorgestellt [30].

Abbildung 2.8: Auszug aus dem FAT-Datensatz, Quelle: Tremblay u. a. [31]

Auch Damian u. a. [7] haben neben Domain Randomised Daten noch einen Datensatz generiert, der visuell die Anwendungsdomäne widerspiegelt. Als Umgebungen haben sie einen Wüsten- und einen Waldabschnitt gewählt. Eine Diversifizierung der Daten erzielen sie durch die zufällige Änderung der Position und der Anzahl der Zielobjekte, ihrer Texturen, des Kamera-Blickwinkels und der zufälligen Wahl der Tageszeit, die auf den Bildern simuliert wird. Im Gegensatz zu Tremblay u. a. [29] ließen die Entwickler die Zielobjekte nicht in die Szene fallen, sondern platzierten diese gezielt.

Abbildung 2.9: Auszug aus den generierten Daten, Quelle: Damian u. a. [7]

2.2 Trainingsresultate

In jeder vorgestellten Arbeit wurden neuronale Netze mit den jeweils erstellten Datensätzen trainiert und die Datensätze anhand der Ergebnisse evaluiert. Alle dafür notwendigen Metriken, die die Leistung eines Modells beurteilen können, wie z.B. der F1-Score und die mittlere Präzision, werden in Kap. 5.2 erläutert. An dieser Stelle werden die Schlussfolgerungen der verschiedenen Arbeiten einander gegenübergestellt. Die Erkenntnisse aus dieser gemeinsamen Betrachtung sind die Grundlage für den Entwurf der eigenen Experimente in den folgenden Kapiteln. Die Einschätzungen der verschiedenen Arbeiten sind nur begrenzt miteinander vergleichbar, da sich beispielsweise bei einigen die Architekturen der trainierten Netze unterscheiden und auch die Testdaten nur zum Teil dem Leser bekannt sind. Daher ist es schwierig zu beurteilen, wodurch die Ergebnisse in den einzelnen Fällen genau zustande kommen. Jedoch liefern sie eine erste Basis, um zu entscheiden, welche Herangehensweisen in dieser Arbeit weiter verfolgt und ausgewertet werden sollen.

Einfluss der Realitätsnähe

Insgesamt wurden die Datensätze in vielerlei Hinsicht evaluiert. Ein essenzieller Aspekt in fast allen betrachteten Arbeiten ist der Vergleich der resultierenden Genauigkeit und Robustheit des trainierten Modells bei Gebrauch von realen angefertigten Kamerabildern, SDR- und DR-Daten[1]. Die Abbildung 2.10 bietet eine Übersicht aller synthetischen Da-

[1] „Domain Randomized“ wird im Folgenden durch „DR“ und „Structured Domain Randomized“ durch „SDR“ abgekürzt.

tensätze aus den referenzierten Arbeiten. Darin wurden die Datensätze danach sortiert, wie realistisch die generierte Umgebung jeweils gestaltet wurde.

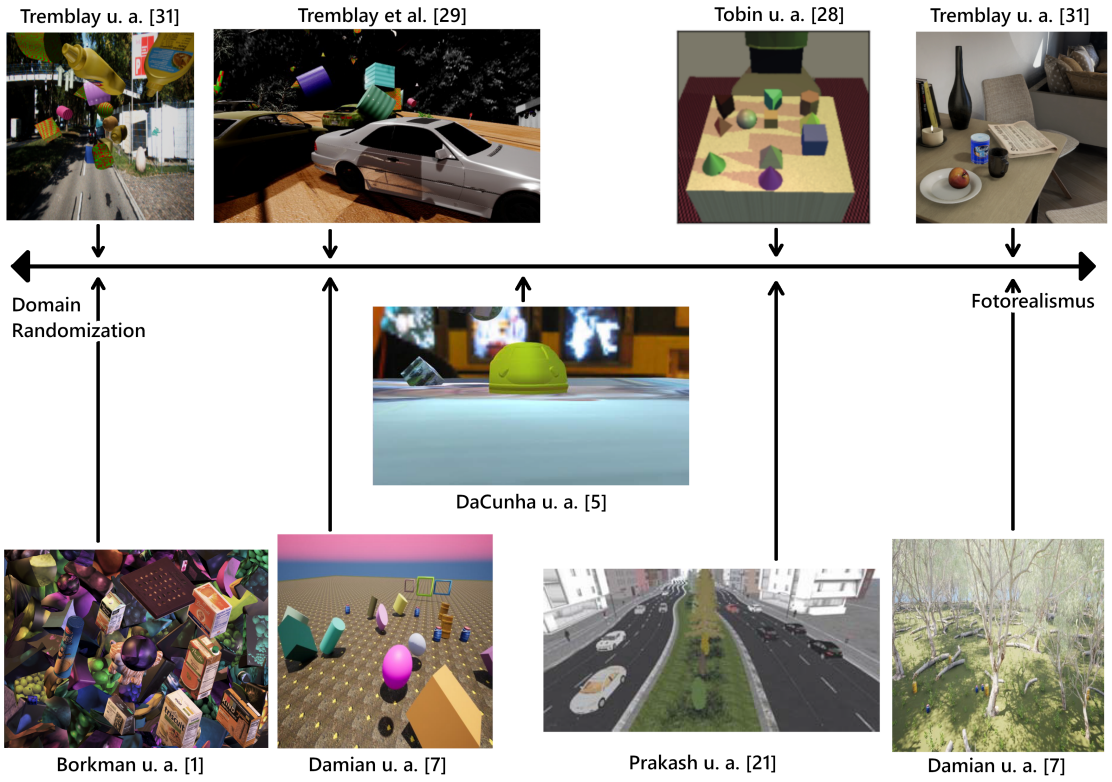

Abbildung 2.10: Übersicht aller vorgestellten Arbeiten, Quelle: Eigene Abbildung mit Auszügen aller in dem Kapitel 2.1 vorgestellten Arbeiten

Tremblay u. a. [31] haben als ersten Versuch einen Datensatz erstellt, der zu gleichen Teilen fotorealistische SDR-Daten und DR-Daten enthält. Damit haben sie anschließend ihre eigene Netzwerk-Architektur trainiert und das Modell mit dem YCB-Datensatz[3] getestet. Das Ergebnis stellten sie der Arbeit von Xiang u. a. [37] gegenüber. Darin wurde das Netz *PoseCNN* entwickelt und mit ausschließlich real aufgenommenen Bildern trainiert, um so ebenfalls Haushaltsgegenstände aus dem YCB-Datensatz zu erkennen. Tremblay u. a. [31] kamen zu genaueren und gegenüber extremen Beleuchtungen auch robusteren Ergebnissen. Allerdings haben beide Arbeiten unterschiedliche Netz-Architekturen verwendet. Als Tremblay u. a. [31] PoseCNN mit ihren synthetischen Daten trainierten, wurde die Erkennung wesentlich unzuverlässiger. Daher ist unklar, in welchem Umfang ihre sehr guten Ergebnisse dem alleinigen Einfluss des Datensatzes zugeschrieben werden können. Die Autoren haben weitere, synthetische Datensätze ausprobiert, deren Anteile

an SDR- und DR-Daten variiert haben. Dabei bekamen sie die besten Ergebnisse, wenn mindestens 40% SDR- und DR-Daten Teil des Datensatzes waren und die beiden Anteile somit relativ ausgewogen gewählt wurden. Durchläufe, in denen ausschließlich eine der beiden Kategorien verwendet wurde, schnitten durchgehend deutlich schlechter ab als die gemischten Datensätze.
In der zweiten, rein auf Domain Randomization konzentrierten Arbeit von Tremblay u. a. [29] wurde betrachtet, wie Trainings des Netzes *Faster R-CNN*[22] mit ihrem DR-Datensatz und mit dem Datensatz *Virtual KITTI*[10] verlaufen. Virtual KITTI besteht aus Bildern, in denen virtuell die Bilder des Datensatzes *KITTI*[11] möglichst exakt nachgebaut wurden. Tremblay u. a. [29] testeten die beiden entstandenen Modelle mit den Daten aus KITTI. Virtual KITTI schnitt zunächst mit einer mittleren Präzision von 79% um 1.6% besser ab als die DR-Variante. Mit einer anschließenden Feinjustierung mit realen Daten kam die DR-Variante jedoch auf 98.5%, das Training mit Virtual KITTI dagegen auf 1.5% weniger. Ein Training mit ausschließlich realen Daten, die ebenfalls aus KITTI entnommen wurden, kam bloß auf 96.4%. Hier kam dem Modell Tremblay u. a. [29] zufolge die große Variation im DR-Datensatz zugute.
Prakash u. a. [21] haben denselben Versuchsaufbau erweitert. Hinzu kamen Trainings mit ihren SDR-Daten und einem Datensatz, der Aufnahmen aus dem fotorealistisch simulierten Straßenverkehr des Computerspiels *GTA V* enthält[16]. In ihren Experimenten übertraf SDR die Resultate aller anderen synthetischen Datensätze. Allerdings wurde ein Modell, das mit 6.000 Bildern des KITTI-Datensatzes trainiert wurde, noch deutlich präziser. Prakash u. a. [21] kamen allerdings bei einem Vergleich mit einem weiteren realen Datensatz aus dem Straßenverkehr, der unabhängig von KITTI entstand, zu der Erkenntnis, dass hiermit nicht nur schlechtere Ergebnisse geliefert wurden als mit dem KITTI Datensatz, sondern auch schlechtere als mit dem SDR-Datensatz. Die somit bessere Generalisierungsfähigkeit der SDR-Daten lässt sich durch die größere Varianz erklären. Bei dem Versuch, nach dem Training mit SDR-Daten eine Feinjustierung mit realen KITTI-Daten durchzuführen, stellte sich dieser Ansatz als die präziseste Objekterkennung unter allen evaluierten Varianten heraus.
Damian u. a. [7] haben drei Versionen ihres Datensatzes beim Training des Netzwerks *YOLOv8*[24] beobachtet. Die erste Version bestand aus 1.000 fotorealistischen Daten, die sich als nicht gut für die realen Daten generalisierbar erwiesen haben. Die Hinzunahme von ca. 6.000 DR-Daten ohne Distraktoren verbesserte das Ergebnis, jedoch erzielte es weiterhin nur einen F1-Score von 0,43. Erst bei der Hinzunahme von ca. 10.000 weiteren DR-Daten inklusive variierender Distraktoren stieg der F1-Score auf 0,8 an. Die Autoren führen dieses Ergebnis auf die starke Varianz zurück. Jedoch haben sie keine

Experimente dazu durchgeführt, ob die Veränderung auch mit dem Umfang des Datensatzes zusammenhängen könnte.
Cunha u. a. [5] haben zehn Trainings des neuronalen Netzes von Tekin u. a. [27] durchgeführt. Ein Training fand mit 7.000 real aufgenommenen Bildern statt, ein weiteres mit 4.000 DR-Daten, vier Trainings wurden mit unterschiedlichen Anteilen real aufgenommener Bilder und DR-Daten umgesetzt und zuletzt vier weitere mit DR-Daten und einer anschließenden Feinjustierung mit real aufgenommenen Bildern. Das Training, das nur DR-Daten verwendete, lieferte eine Erkennung, die die Autoren als nicht zufriedenstellend erachteten und dies mit dem großen Reality Gap erklärten. Mit dem realen Datensatz erreichten Cunha u. a. [5] eine hohe Präzision, wenn die gezeigte Umgebung derjenigen der Testdaten glich. Bei einer starken Abweichung, z.B. durch Bewegungsunschärfe, extreme Beleuchtung oder die Verwendung einer anderen Kamera sank die Genauigkeit jedoch um 90%. Bei den gemischten Datensätzen stieg die erzielte Genauigkeit, sowohl bei einem ähnlichen Umgebungs-Aufbau als auch hinsichtlich der Robustheit, wobei bei einer Bewegungsunschärfe oder extremer Beleuchtung weiterhin die Erkennung deutlich ungenauer blieb, da beides nicht in den Datensätzen vorhanden war. Die besten Ergebnisse lieferte die Mischung aus 40% realen und 60% synthetischen Daten. Eine weitere Verbesserung brachte das Training mit DR-Daten kombiniert mit einer anschließenden Feinjustierung mit 30% bis 60% realen Daten. Nicht nur bei einer ähnlichen Umgebung, sondern auch bei den Tests hinsichtlich der Robustheit gegenüber unvorhergesehenen Veränderungen war die Präzision nun hoch, obwohl diese Umstände weiterhin nicht im Trainingsdatensatz repräsentiert wurden.
Borkman u. a. [1] haben ebenfalls neben einem real angefertigten Datensatz und einem reinen DR-Datensatz auch eine Feinjustierung der DR-Daten mit realen Daten geprüft. Sie haben dafür 400.000 DR-Daten generiert und 1.267 reale Daten aufgenommen. Genau wie Tremblay u. a. [29] haben sie Faster R-CNN zur Evaluation benutzt. Für die Feinjustierung mit realen Daten wurden unterschiedlich große Datensätze getestet. Durch die Hinzunahme synthetischer Daten wurden alle diese Versuche durchweg gegenüber einer Variante mit ausschließlich realen oder synthetischen Daten verbessert. Ein Optimum wurde bei der Kombination 760 realer mit 400.000 synthetischen Daten gemessen.

Es lassen sich mehrere Schlüsse aus den Arbeiten ziehen. Zunächst weisen alle darauf hin, dass eine Kombination synthetischer und real aufgenommener Daten zu einem genaueren und robusteren Modell führt als die alleinige Verwendung der Kategorien. Die Kombination kann entweder durch eine Mischung der verwendeten Kategorien zu einem

gemeinsamen Datensatz oder durch eine Feinjustierung mit realen Daten im Anschluss an ein Training mit synthetischen Daten stattfinden. Für Cunha u. a. [5] besteht das beste Mischverhältnis zu 40% aus realen und zu 60% aus DR-Daten. Tremblay u. a. [29], Prakash u. a. [21], Borkman u. a. [1] und Cunha u. a. [5] setzten erfolgreich auf eine Feinjustierung mit realen Daten. Cunha u. a. [5] verglichen als einzige die beiden Strategien zur Kombination miteinander und favorisierten die Feinjustierung mit realen Daten. Es gibt auch die Möglichkeit, anstelle real aufgenommener Daten realistische SDR-Daten für diese Strategien zu verwenden. Dies haben Damian u. a. [7] und Tremblay u. a. [31] getan und kamen auch hier zu besseren Ergebnissen als bei der alleinigen Verwendung einzelner Kategorien. Das optimale Mischverhältnis lag für Tremblay u. a. [31] bei mindestens 40% SDR- und 40% DR-Anteilen. Keine der betrachteten Arbeiten hat jedoch ausführlich evaluiert, welche Kategorien in Kombination das beste Ergebnis bringen, da für die Kombinationen maximal die selbst entwickelten Datensätze und die real angefertigten Testdaten eingesetzt wurden.

Falls nur eine synthetische Kategorie alleine verwendet wird, kommen Cunha u. a. [5], Damian u. a. [7] und Tremblay u. a. [31, 29] zu dem Schluss, dass weder fotorealistische noch DR-Daten Ergebnisse liefern, die an die Präzision realer Daten heranreichen. Bei Prakash u. a. [21] erzielt SDR zwar präzisere Resultate als die anderen beiden Varianten, aber auch diese kommen nicht an die Verwendung real aufgenommener Daten heran. Die Ergebnisse lassen sich mit der These begründen, dass in fotorealistischen Daten die Varianz nicht ausreicht, während in DR-Daten der Reality Gap zu groß ist und SDR einem Kompromiss zwischen diesen Eigenschaften am nächsten kommt.

Prakash u. a. [21] machten eine weitere interessante Beobachtung, als sie feststellten, dass die Verwendung realer Daten, die in einer anderen Umgebung, aber in derselben Anwendungs-Domäne aufgenommen wurden, zu ungenaueren Ergebnissen führen können als ein synthetischer SDR-Datensatz. Das kann darauf hinweisen, dass die fehlende Varianz in einem real aufgenommenen Datensatz zu einer unzureichenden Generalisierungsfähigkeit führen kann, selbst wenn der Reality Gap kleiner ist. Auch Cunha u. a. [5] gelangten zu der Erkenntnis, dass selbst ein Training mit realen Daten, die in derselben Umgebung aufgenommen wurden wie die Testdaten, nicht robust ist gegenüber unvorhergesehenen Bedingungen wie z.B. extremen Beleuchtungen, Bewegungsunschärfe oder der Verwendung einer anderen Kamera, die im Datensatz nicht repräsentiert werden. Auch das zeigt, dass die fehlende Varianz eines real aufgenommenen Datensatzes Schwächen mit sich bringt, die eine Kombination aus synthetischen und realen Daten abschwächen kann.

Einfluss der einzelnen Gestaltungs-Aspekte

Ein zweiter Gesichtspunkt, der in einigen Arbeiten untersucht wurde, besteht darin, herauszufinden, inwiefern die einzelnen implementierten Eigenschaften der virtuellen Umgebungen die Trainingsresultate beeinflussen. Zu diesem Zweck wurden von den Autoren jeweils Ablationsstudien durchgeführt. Für diese Studien werden spezielle Datensätze generiert, in denen einzelne Eigenschaften, wie z.B. die Beleuchtung oder Nutzung bestimmter Texturen, bei der Umgebungsgestaltung ausgelassen werden. Durch den Vergleich des damit erzielten Modells mit einem, bei dem die Eigenschaft beibehalten wurde, kann der Einfluss der Eigenschaft ermittelt werden.
Als Tremblay u. a. [29] beispielsweise ihre virtuelle Beleuchtung im DR-Datensatz unverändert ließen, verschlechterte sich die durchschnittliche Präzision um 6.1%. Das war in ihren Versuchen, verglichen mit dem Weglassen zufälliger Texturen und Distraktoren, die Eigenschaft mit den größten Auswirkungen. Prakash u. a. [21] verzeichneten bei ihrem SDR-Datensatz hingegen bloß einen Verlust von 2.1%. Bei ihnen stellte sich heraus, dass die Gestaltung des Umgebungskontextes eine wesentlich größere Rolle spielt. Wenn z.B. die dafür relevanten 3D-Objekte, wie Straßen, Gebäude und Fußwege, durch eine zweidimensionale Textur im Hintergrund ausgetauscht wurden, wie es in der Gestaltung von DR-Daten üblich ist (siehe Kap. 2.1.1), kam ein Verlust von 7.2% zustande. Dies kann man als weiteren Hinweis für den positiven Einfluss der realitätsnahen Kontext-Darstellung in SDR-Umgebungen sehen. In einem weiteren Versuch ließen Prakash u. a. [21] den variierend großen Anteil an Gestaltungselementen, die ansonsten zu einer entweder urbanen, vorstädtischen oder dörflichen Umgebung führen würden, unverändert. Durch die fehlende Varianz wurden 6.5% eingebüßt. Als letzten Aspekt positionierten die Autoren alle Fahrzeuge ausschließlich in einer gemeinsamen Spur, was zu einer um 2.8% geringeren Genauigkeit führte, da z.B. parkende Fahrzeuge schlechter erkannt wurden. Das kann man ebenfalls als Hinweis für die Relevanz der am Kontext orientierten Verteilung von Eigenschaften im Datensatz sehen.
Neben der Beleuchtung testeten Tremblay u. a. [29] die Auswirkung zufällig gewählter Texturen für ihre virtuellen Zielobjekte im DR-Datensatz. Ohne ihre Verwendung erzielte das Modell eine um 4.7% geringere durchschnittliche Präzision und bei der Verwendung von 4.000 statt 8.000 Texturen kam es immer noch zu einem Rückgang von 2.2%. Tobin u. a. [28] haben ebenfalls mit der Verwendung von Texturen in ihrem SDR-Datensatz experimentiert. Zunächst haben sie versucht, die Texturen besonders realistisch auszuwählen. Als sie das resultierende Modell mit Fotos testeten, die in einer dazu passenden Umgebung angefertigt wurden, veränderte sich die Präzision kaum. Die Autoren kamen

daher zu dem Schluss, dass das Modell invariant gegenüber den genauen Texturen ist. Jedoch veränderte die Anzahl und Variation an Texturen das Ergebnis stark. Wenn weniger als 1.000 Texturen verwendet wurden, sank die Genauigkeit stark, während sie bei dem maximal getesteten Umfang von 10.000 Texturen ihr Optimum hatte. Bei der Verwendung von 1.000 Texturen stellten die Autoren zudem fest, dass das Resultat bei der Verwendung von 1.000 oder 10.000 generierten Daten gleich blieb. Das ist ein Hinweis darauf, was für eine große Rolle Varianz spielt.
Zuletzt wurde von Tremblay u. a. [29] das Weglassen der fliegenden Distraktoren erprobt, das sich mit einem Verlust von 1.1% durchschnittlicher Präzision niederschlug. Im Zusammenhang damit ist die zuvor geschilderte Evaluation von Damian u. a. [7] interessant. Hier wurde der ca. 7.000 Einträge umfassende Datensatz um ca. 10.000 weitere, nun erstmals variierende Distraktoren enthaltende DR-Daten erweitert. Dadurch verbesserte sich der resultiere F1 Score von 0,43 auf 0,8. Diese große Veränderung kann jedoch zusätzlich mit der Datensatzgröße zusammenhängen, was Damian u. a. [7] nicht testeten. Als Tobin u. a. [28] in ihren Trainings keine Distraktoren verwendeten, wurden reale Zielobjekte, die verdeckt wurden oder inmitten vieler unbekannter Objekte standen, deutlich schlechter erkannt. Bei Test-Bildern, auf denen die Zielobjekte jedoch unverdeckt und für sich genommen zu sehen sind, machte sich das Weglassen der Distraktoren in den Trainingsdaten kaum bemerkbar.
Diese Beobachtung ist ein gutes Beispiel dafür, wie sehr die Beurteilung der gemessenen Ergebnisse von den Testdaten abhängt. Das Weglassen von variierenden Verdeckungen fällt weniger ins Gewicht, wenn im Testdatensatz keine Verdeckungen auftauchen. Die Vermutung liegt nahe, dass dieser Sachverhalt auch auf die anderen Eigenschaften zutrifft. Da dem Leser der Arbeiten nur begrenzte Informationen dazu zur Verfügung stehen, mit welchen Daten im Detail getestet wurde, lassen sich die hier aufgezeigten Beobachtungen der Autoren nur bedingt verallgemeinern. Jedoch sind die Hinweise, dass die Eigenschaften mitunter große Auswirkungen auf das trainierte Modell haben, Grund genug, dies in den folgenden Kapiteln weiter zu untersuchen.

Einfluss der Trainingseinstellungen

Als Drittes haben die Arbeiten ausgewertet, wie einzelne Einstellungen des Trainings die Ergebnisse verändern.
Die ersten beiden Einstellungen betreffen die Anzahl eingesetzter Trainingsdaten und die Verwendung von vortrainierten Gewichten, die mithilfe anderer Datensätze gewonnen

wurden. Tremblay u. a. [31] testeten verschiedene Größen zwischen 2.000 und 1.000.000. Ihr DR-Datensatz erbrachte bei 300.000 Daten das genaueste Ergebnis, während die Vergrößerung des Umfangs bis zu 10.000 Daten den größten Zuwachs an Genauigkeit lieferte. Bei der Verwendung von ausschließlich fotorealistischen Daten wurde das Optimum erst bei 600.000 Daten erreicht, während bei einer Mischung der Kategorien schon 120.000 Daten ausreichten. In ihrer zweiten, rein auf Domain Randomization konzentrierten Arbeit kamen Tremblay u. a. [29] mit 50.000 Daten auf das beste Ergebnis, wenn die Gewichte beim Training zufällig initialisiert wurden. Als vortrainierte Gewichte des COCO-Datensatzes, der unter anderem Bilder von Fahrzeugen enthält, verwendet wurden, reichten schon 10.000 Daten. In ihren Versuchen reichten die Trainingsergebnisse unabhängig von der Datensatzgröße nie an das vortrainierte Äquivalent heran. Im Gegensatz dazu stehen die Beobachtungen von Tobin u. a. [28]. Ihr auf SDR-Daten basierendes Modell verbesserte sich bis zur Benutzung von 50.000 Daten stetig. Sowohl vortrainierte als auch zufällig initialisierte Gewichte kamen ab ca. 8.000 Bildern zu ähnlichen Resultaten. Bei weniger Bildern galt jedoch: Je weniger Daten vorhanden waren, desto schlechter schnitt das nicht vortrainierte Modell ab. Beim SDR-Datensatz von Prakash u. a. [21] war die Leistung ebenfalls mit 10.000 SDR-Daten gesättigt, wobei die Präzision für jede untersuchte Datensatzgröße ungefähr doppelt so hoch war wie beim DR-Datensatz von Tremblay u. a. [29]. Borkman u. a. [1] erreichten ihr bestes Ergebnis bei 400.000 DR-Daten und einer Feinjustierung mit 760 real aufgenommenen Daten. Das war die von ihnen getestete Menge mit dem größten Umfang. Jedoch wurde ab einer Verwendung von 100.000 DR-Daten bei einer gleichbleibenden Menge realer Daten das Ergebnis nicht mehr viel besser. Die Menge an realen Daten hatte hingegen einen deutlich größeren Einfluss.
Um beurteilen zu können, woran die genauen Unterschiede in den benötigten Umfängen der Datensätze liegen, fehlen auch hier die genauen Informationen über die Testdaten und eine stärkere Vergleichbarkeit der Trainings-Einstellungen und Anwendungsdomänen untereinander. Jedoch zeigen die Messungen, dass die Anzahl der benötigten Daten durch mehrere Faktoren beeinflusst wird. Beispielsweise spielt der Einsatz eines Vortrainings eine Rolle. Auch die genaue Gestaltung der Daten scheint das zu tun, wie der Fall von Tremblay u. a. [31] zeigt. Alle Arbeiten brauchten jedoch mindestens 10.000 synthetische Daten für ihre jeweils besten Ergebnisse.

Eine weitere, in einigen Arbeiten getestete Einstellung ist die Verwendung von einer zusätzlichen Data Augmentation. Tremblay u. a. [29] verglichen einen DR-Datensatz, der die virtuell simulierte Beleuchtung beinhaltete, mit einem, bei dem stattdessen Helligkeits-

und Kontrastwerte der generierten Bilder verändert wurden. Sie beobachteten, dass die simulierte Beleuchtung deutlich ausschlaggebender für das Ergebnis ist als die Data Augmentation. Ohne die Beleuchtungs-Veränderung sank die durchschnittliche Präzision um 6.1%, während die Autoren ohne Helligkeits-/Kontrast-Data Augmentation nur 0.1% weniger maßen. Als zusätzlich Zuschneidungen, Spiegelungen und gaußsches Rauschen außer Acht gelassen wurden, kamen Tremblay u. a. [29] auf 1.7% Verlust. Im Gegensatz dazu maßen Prakash u. a. [21] ohne die Veränderung von Kontrastwerten eine um 3.9% geringere durchschnittliche Präzision und ohne die zufällig gestaltete Sättigung sogar 6.5% weniger, als bei dem vollständigen SDR-Datensatz. Hier fielen also beide Faktoren durchaus ins Gewicht. Tobin u. a. [28] untersuchten in ihrer Arbeit nur den Einsatz von zufälligem Rauschen. Bei ihnen hatte es jedoch kaum Auswirkungen auf das Trainingsresultat.

Zuletzt führten Tremblay u. a. [29] Experimente durch, die zeigen sollten, inwiefern sich ihre Beobachtungen bei der Verwendung unterschiedlicher neuronaler Netze veränderten. Zu diesem Zweck trainierten sie neben Faster R-CNN auch die beiden neuronalen Netze R-FCN[6] und SSD[19] mit ihren DR-Daten. Die trainierten Modelle unterschieden sich deutlich voneinander, obwohl dieselben Trainings- und Testdaten verwendet wurden. SSD lieferte mit einer durchschnittlichen Präzision von 46,3% einen sehr viel niedrigeren Wert als R-FCN mit 71,5% und Faster R-CNN mit 78,1%. Auch die Trainings, in denen VKITTI-Bilder als Trainingsdaten verwendet wurden, veränderten sich. Während die erzielte Präzision bei Faster R-CNN etwas höher war, fiel sie bei den anderen beiden Netzen deutlich niedriger aus als bei dem DR-Datensatz. Diese von den jeweiligen Netzwerken abhängigen Unterschiede sind ein weiteres Zeichen dafür, dass die Vergleichbarkeit der Ergebnisse in den einzelnen vorgestellten Arbeiten nur bedingt vorhanden ist. Daher ist es ein Ziel der vorliegenden Arbeit, eine solche Vergleichbarkeit verschiedener Datensätze zu schaffen, indem sie unter ähnlichen Umständen evaluiert werden.

3 Planung der Experimente

Alle betrachteten Arbeiten nutzen den Vergleich von Trainingsergebnissen, um Datensätze zu evaluieren. Jedoch zeigte Kap. 2.2, wie schwierig solche Vergleiche zwischen Arbeiten mit unterschiedlichen Bedingungen sind. Gerade wenn die Bedingungen, wie z.B. der genaue Aufbau der Testdaten, unbekannt sind, erschwert dies den Umstand, der durch die Nutzung von unterschiedlichen neuronalen Netzen oder vortrainierten Gewichten ohnehin schon gegeben ist. Ein Vergleich zwischen den verschiedenen Herangehensweisen wäre jedoch hilfreich um herauszufinden, welche Vorzüge die Methoden jeweils haben. Daher soll in dieser vorliegenden Arbeit eine Umgebung geschaffen werden, in der Datensätze mit unterschiedlichen Eigenschaften unter denselben Bedingungen getestet werden, um Vergleichbarkeit zu schaffen. Für diese Experimente werden die in Abb. 3.1 gezeigten Schritte durchlaufen.

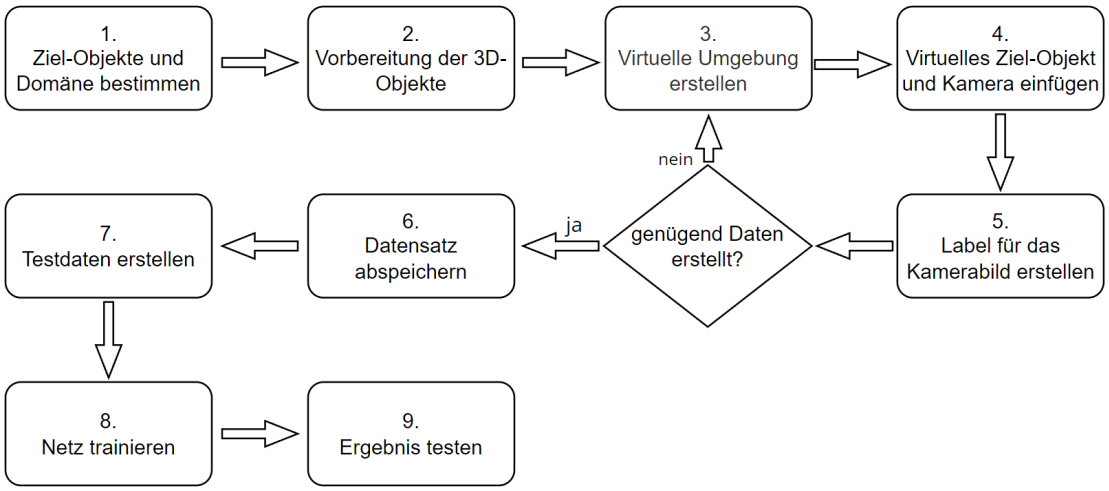

Abbildung 3.1: Ablauf der Experimente, Quelle: Eigene Abbildung

In diesem Kapitel findet die Planung dieser praktischen Schritte statt. Dazu gehört im ersten Schritt die Bestimmung der Anwendungsdomäne und der dazugehörenden Zielobjekte. Auf dieser Grundlage folgt die Skizzierung der Anforderungen an die in Schritt

drei und vier zu gestaltenden Datensätze. Anschließend wird entschieden, welche Technologien sich für die praktische Umsetzung der Experimente eignen. Die letzte Planung in diesem Kapitel betrifft die Entscheidung, welches neuronale Netz in Schritt acht und neun zur Evaluation der Datensätze verwendet wird.

3.1 Eingrenzung der Domäne

In Kap. 1 wurde ein vorangegangenes Projekt[13] beschrieben, in dem reale Trainingsdaten für die Posen-Erkennung von Schachfiguren angefertigt wurden. Da in diesem Rahmen drei Schachfiguren mit einem 3D-Drucker gedruckt wurden, liegen realitätsgetreue 3D-Modelle dieser realen Objekte vor.

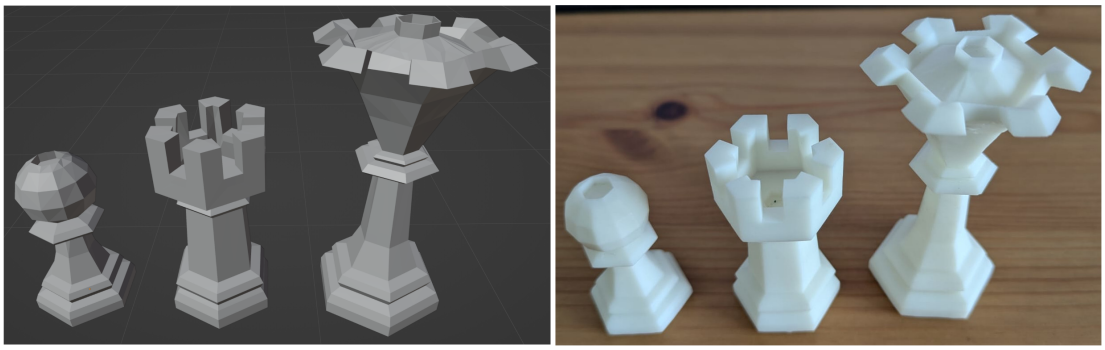

Abbildung 3.2: Die virtuellen 3D-Modelle der Zielobjekte und ihre realen Versionen
Quelle: Eigene Abbildung

Die Existenz von realistischen 3D-Modellen der Zielobjekte ist eine Voraussetzung für die Anfertigung synthetischer Trainingsdaten. In dieser Arbeit wird daher die in dem damaligen Projekt anvisierte Anwendungsdomäne ebenfalls als Grundlage verwendet. Ziel soll hier sein, die zweidimensionale Bounding Box der drei Schachfiguren auf einzelnen RGB-Bildern zu bestimmen. Dies ist vergleichbar mit den in Kap. 2 vorgestellten Arbeiten. Der Anwendungsfall weist hinsichtlich der Zielobjekte insbesondere Ähnlichkeit zu der Arbeit von Cunha u. a. [5] auf, da auch dort ebenfalls ein texturarmes, 3D-gedrucktes Objekt gesucht wurde. Cunha u. a. [5] wiesen darauf hin, was dies besonders herausfordernd und eine erfolgreiche Umsetzung somit besonders aussagekräftig ist.

3.2 Anforderungen an die Datensätze

An dieser Stelle erfolgt die Planung, welche Datensätze generiert und evaluiert werden sollen. Dabei sollen einige in Kap. 2.2 bereits erfolgte Beobachtungen weiter untersucht werden, indem die Evaluations-Bedingungen angeglichen werden. Dazu gehört die Betrachtung, welche synthetischen Daten auch ohne die Verwendung real aufgenommener Daten zu guten Ergebnissen führen und wie gut dabei welche Kategorie im Vergleich abschneidet. Außerdem sollen die entstandenen Kategorien untereinander gemischt werden. Hierdurch kann etwa die Frage beantwortet werden, ob bei Vorliegen realer Daten und relativ leicht zu konstruierenden DR-Daten, sich der Aufwand noch lohnt, SDR-Daten zu erstellen. Auch die Robustheit und Generalisierbarkeit der entstandenen Datensätze soll beurteilt werden. Zuletzt werden, ähnlich wie in den verwandten Arbeiten, Ablationsstudien durchgeführt mit dem Ziel zu ermitteln, welche Einflüsse die einzelnen Gestaltungs-Aspekte der virtuellen Umgebungen auf die Ergebnisse haben. Vor dem Hintergrund dieser Evaluations-Ziele sollen die folgenden Datensätze generiert werden.

Trainingsdatensätze

- **DR1**
  Dies ist die am einfachsten aufgebaute Umgebung. Vergleichbar zu Tremblay u. a. [31] orientieren sich nur die Zielobjekte an der Realität. Als Distraktoren fungieren geometrisch primitive Objekte und der Hintergrund besteht aus einer senkrechten Ebene. Alle Texturen sind einfarbig oder bestehen aus einfachen Mustern. Distraktoren und Zielobjekte werden zufällig im 3D-Raum zwischen Kamera und Hintergrund angeordnet. Die Beleuchtung erfolgt aus einer zufälligen Richtung in einer zufälligen Farbe.

- **DR2**
  Gegenüber DR1 verändert sich hier nur die Wahl der Texturen. Diese werden einem Datensatz zufällig ausgesuchter, möglichst variierender Bilder genommen.

- **DR3**
  Der Aufbau ähnelt dem aus DR2, jedoch werden hier komplexere Distraktoren verwendet, um die Variabilität hinsichtlich der Objektverdeckungen zu erhöhen.

- **DR4**
  Im Gegensatz zu den anderen DR-Szenarios, wird hier etwas mehr Wert auf Realitätsnähe gelegt. Vergleichbar mit Damian u. a. [7] und Tremblay u. a. [29] werden alle Zielobjekte auf einer gemeinsamen, horizontalen Ebene, die einen Boden simuliert, platziert. Zudem orientieren sich die Texturen des Hintergrunds und des Bodens an der Anwendungsdomäne. Dieses Szenario soll die Möglichkeit bieten zu evaluieren, ob in diesem Fall ein Wissen über die Domäne einen Vorteil bieten kann.

- **SDR1**
  Für diesen Datensatz wird ein Wohnzimmer prozedural erstellt. Die genaue Gestaltung, wie z.B. die Größe des Raums sowie die Auswahl und Platzierung der Möbel, verläuft zufallsbasiert. Somit soll eine Umgebung entstehen, die sich an der generellen Anwendungsdomäne orientiert, jedoch kein Wissen über den konkreten Einsatzort der Anwendung voraussetzt.

- **SDR2**
  Die Umgebung aus SDR1 wird um fliegende Distraktoren erweitert, um die Varianz zu erhöhen und die Erkennung verdeckter Zielobjekte zu stabilisieren.

Zusätzlich zu den genannten Datensätzen werden Mischungen der Datensätze mit real aufgenommenen Bilddaten getestet, da in Kap. 2.2 mehrere Arbeiten damit zu besonders guten Ergebnissen kamen.

Testdatensätze

Zur Evaluation dieser Trainingsdaten werden Datensätze gebraucht, die in realen Umgebungen angefertigt werden. Zunächst werden Aufnahmen der realen Zielobjekte aus realen Wohnzimmern benötigt. Zur Beurteilung der Robustheit werden Aufnahmen mit unterschiedlichen Beleuchtungsverhältnissen, mit Verdeckungen der Zielobjekte, aus variierenden Blickwinkeln und in verschiedenen Umgebungen eingesetzt.

3.3 Auswahl der Software und Hardware

Um die Experimente dieser Arbeit durchzuführen, werden einige Technologien gebraucht. Abb. 3.2 zeigt farblich unterlegt die Schritte im Ablauf, die jeweils unterschiedliche Technologien erfordern.

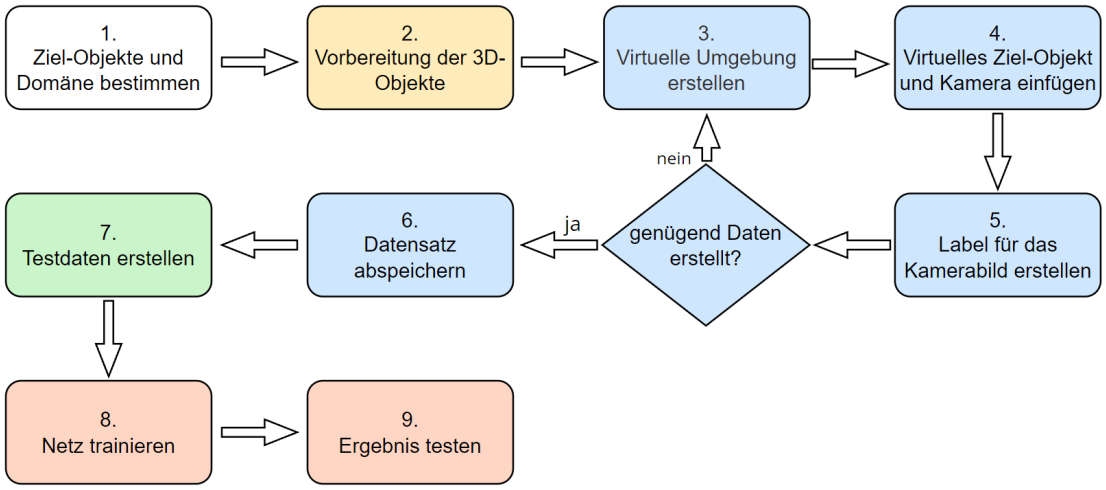

Abbildung 3.3: Ablauf der Experimente, Quelle: Eigene Abbildung

Nachdem nun die Anwendungsdomäne, die Zielobjekte und die groben Anforderungen an die umzusetzenden Datensätze festgelegt wurden, müssen in Schritt zwei 3D-Objekte erstellt werden, mit denen die virtuelle Umgebung in Schritt drei gefüllt werden kann. Da in diesem Projekt bereits 3D-Modelle der Zielobjekte vorliegen, fehlen nur noch entsprechende Modelle für die virtuelle Umgebung. Hier wurden zunächst einige fertige 3D-Modelle von Stühlen, Tischen, Schachbrettern, Fenstern, Decken- und Boden-Lampen **...** sowie fertige Texturen, die jeweils zu der ausgewählten Anwendungsdomäne passen, ausgesucht. Sie wurden den Webseiten Sketchfab[1] und CC0-Textures[2] entnommen. Dabei wurde darauf geachtet, dass die Dateien jeweils unter der Creative Commons Lizenz veröffentlicht wurden. Eine vollständige Liste der verwendeten Dateien ist in dem Projekt-Ordner hinterlegt. In den Fällen, in denen bestehende 3D-Objekte und Texturen bearbeitet oder neu erschaffen werden mussten, wurden die Editoren Blender[3] für 3D-Objekte und Gimp[4] für 2D-Bilder verwendet.

Für die Implementierung der Anwendung, die die Datensätze in den Schritten drei bis sechs generiert, wird eine Entwicklungsumgebung benötigt, in der virtuelle Umgebungen aus 3D-Objekten gestaltet werden können und das Laufzeitverhalten der Umgebung durch Programmierung definierbar ist. Die in Kap. 2 betrachteten Arbeiten haben zu diesem Zweck Game Engines oder Simulations-Umgebungen verwendet. In der vorliegenden

[1] Sketchfab Inc.: „Sketchfab“, https://sketchfab.com/ (Letzter Zugriff 03.07.2024)
[2] „CC0 Textures“, https://cc0-textures.com/ (Letzter Zugriff 03.07.2024)
[3] „Blender 4.1“, https://www.blender.org/ (Letzter Zugriff 03.07.2024)
[4] „Gimp - GNU Image Manipulation Program“, https://www.gimp.org/ (Letzter Zugriff 03.07.2024)

Arbeit wurde sich für die Unity3D Engine[5] entschieden. Unity3D bringt eine Vielzahl von Tools mit sich, die die Entwickler bei der Erstellung einer virtuellen, dreidimensionalen Umgebung unterstützen und ist gleichzeitig flexibel genug konfigurierbar, sodass alle in Kap. 3.2 definierten Anforderungen an die Datensätze damit umgesetzt werden können. In dieser Engine wird mit der Programmiersprache C# gearbeitet, wofür zusätzlich die externe IDE Visual Studio 2022[6] zum Einsatz kommt.
Für die Erstellung der Testdaten in Schritt sieben wird ein Tool benötigt, mit dem reale Aufnahme gelabelt werden können. Dafür wird die Seite Roboflow[7] eingesetzt. Diese beinhaltet neben der Funktion, manuelle Labels anfertigen zu können, auch die Einbindung von Data Augmentation und eine Versionsverwaltung. Daher eignet sich die Seite gut, um reale Datensätze zu erstellen, die den Anforderungen aus Kap. 3.2 gerecht werden können.
Zuletzt wird für das Training in Schritt acht Hardware benötigt, deren Rechenkapazitäten dafür geeignet sind, neuronale Netze zu trainieren, da diese Trainings aufwändig zu berechnen sein können. Zu diesem Zweck werden unterschiedliche virtuelle Maschinen verwendet, die durch die Hochschule für Angewandte Wissenschaften bereitgestellt werden. Um die benötigte Leistung zu erbringen, beinhaltet ihre Cloud-Lösung unter anderem mehrere Nvidia A100 und Quadro P6000 Grafikkarten. Auch die Evaluation der Datensätze in Schritt neun findet auf dieser Hardware statt.

3.4 Auswahl des neuronalen Netzes

In diesem Kapitel wird das neuronale Netz, mit dem die Datensätze evaluiert werden, ausgewählt. Als Ziel des Netzes wurden in dieser Arbeit zwei Alternativen in Betracht gezogen: Die Erkennung der 2D und die der 3D Bounding Boxen der Zielobjekte auf einzelnen RGB-Bildern. Um die Forschungsfragen hinsichtlich der virtuellen Umgebungsgestaltung zu beantworten, eignen sich beide Alternativen gleichermaßen, solange alle hier erarbeiteten Datensätze unter denselben Bedingungen getestet werden. Die in Kap. 2 aufgezeigten verwandten Arbeiten haben alle die Variante mit 2D Bounding Boxen zur Evaluation benutzt. Evaluiert wurde dort unter anderem mit den neuronalen Netzen Faster R-CNN[22] und YOLOv8[24]. Ezzeddini u. a. [9] verglichen für den Anwendungsfall, Fische und Fischer-Ausrüstungen unter Wasser zu erkennen, die Leistung der beiden

[5] Unity Technologies: „Unity3D Engine“, https://unity.com/de (Letzter Zugriff 03.07.2024)
[6] Microsoft: „Visual Studio“, https://visualstudio.microsoft.com/de/ (Letzter Zugriff 03.07.2024)
[7] Sketchfab Inc.: „Roboflow“, https://roboflow.com/ (Letzter Zugriff 03.07.2024)

Netze. In ihren Versuchen lieferte YOLOv8 die präzisere Erkennung. Auch in einem vergleichbaren Artikel[8] von Rustem Glue aus einer anderen Anwendungsdomäne läuft die Erkennung von YOLOv8 am präzisesten ab. Aufgrund dieser guten Leistungen fällt die Entscheidung für die Experimente der vorliegenden Arbeit auf die YOLOv8 Implementierung von Ultralytics[14]. Diese enthält z.B. bereits ein Dockerfile, das eine nützliche Grundlage für das Training des Modells auf der externen Hardware darstellt, sowie mehrere Vorschläge an vortrainierten Backbones, ähnlich zu denen, die in den verwandten Arbeiten zum Einsatz kamen. Somit eignet sich die Implementierung gut, um Experimente durchzuführen, die vergleichbar mit denen der verwandten Arbeiten sind.

[8] Rustem Glue: „YOLOv8, EfficientDet, Faster R-CNN or YOLOv5 for remote sensing“, https://medium.com/@rustemgal/yolov8-efficientdet-faster-r-cnn-or-yolov5-for-remote-sensing-12487c40ef68 (Letzter Zugriff 09.07.2024)

4 Entwicklung der Datensätze

Nachdem die Anforderungen an die Datensätze, die in dieser Arbeit untersucht werden sollen, in Kap. 3.2 definiert wurden, werden diese Datensätze nun erstellt. In Kapitel 4.1 wird zuerst die zur Datengenerierung implementierte Anwendung vorgestellt. Die Anwendung deckt dabei im Ablauf der Experimente die Schritte drei bis sechs ab (siehe Abb. 4.1).

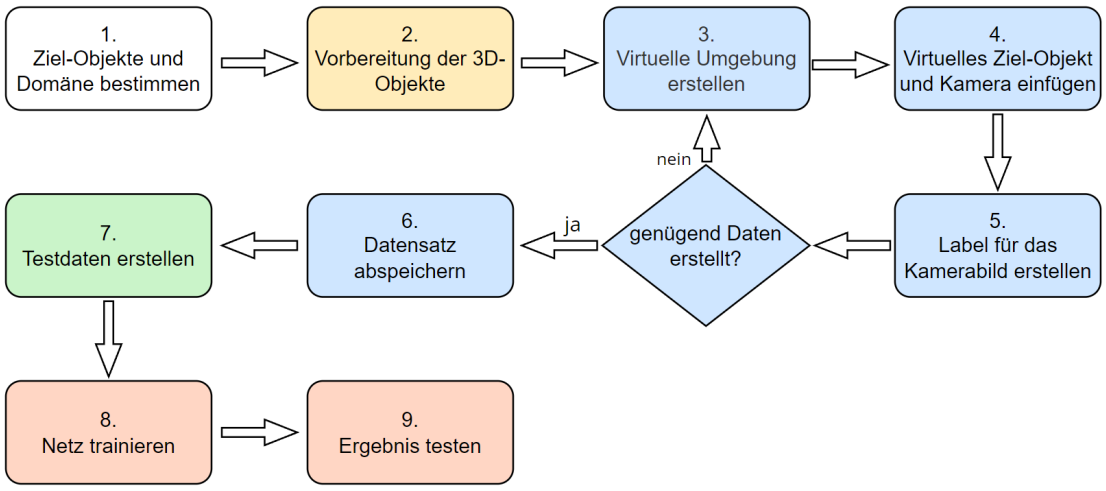

Abbildung 4.1: Ablauf der Experimente, Quelle: Eigene Abbildung

Kapitel 4.2 stellt die generierten Datensätze vor und erläutert im Detail, wie die festgelegten Anforderungen erfüllt wurden. Dabei wird außerdem darauf eingegangen, welche virtuellen 3D-Objekte für den Aufbau der simulierten Umgebung in Schritt zwei der Abb. 4.1) einsatzbereit gemacht wurden. Kapitel 4.3 bietet zuletzt einen Einblick in die Performanz der Anwendung. Zum einen wird ein Eindruck davon vermittelt, wie lang die Generierung der Datensätze gedauert hat, zum anderen werden Herausforderungen aufgezeigt, die während des Prozesses entstanden sind.

4.1 Entwicklung der Unity3D Umgebung

Im Folgenden wird der Aufbau der Anwendung, die zur Datengenerierung in der Unity3D Engine konstruiert wurde, dokumentiert. Zum Einstieg wird erklärt, wie der funktionale Ablauf der Datengenerierung vonstatten geht. Anschließend wird darauf eingegangen, welche Funktionen zur Umgebungsgestaltung implementiert wurden. Daraufhin wird genauer auf den architektonischen Aufbau der Anwendung eingegangen. Eine vollständige Übersicht über die entwickelte Architektur ist in Form eines Klassendiagramms im Anhang zu finden (siehe A.1). Weiterhin wird die Erstellung der Labels thematisiert.

4.1.1 Ablauf der Datengenerierung

Zum Start der Anwendung gibt es eine Initialisierungsphase. In diesem Stadium befinden sich ausschließlich statisch festgelegte Objekte in der virtuellen 3D-Umgebung. Während dieser Phase werden zur Laufzeit einige Variablen zur Verwaltung der Umgebung initialisiert. Anschließend konstruiert die Anwendung nacheinander verschiedene Umgebungen, die für die einzelnen Daten eines Datensatzes fotografiert werden. Für jedes Datum wird die Umgebung neu gestaltet, um eine möglichst hohe Variation innerhalb der Daten zu erzielen. Pro Durchlauf der Anwendung wird genau ein Datensatz mit gleichbleibenden Anforderungen an die Umgebung erstellt. Der allgemeine Ablauf ist in Abb. 4.2 zu sehen.

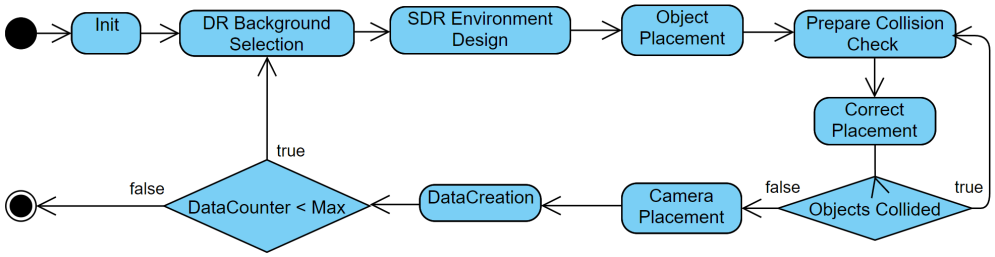

Abbildung 4.2: Zustandsdiagramm der Anwendung, Quelle: Eigene Abbildung

Falls ein DR-Datensatz generiert wird, wird nach der Initialisierungsphase eine Hintergrund-Textur gewählt. Andernfalls geschieht zu diesem Zeitpunkt nichts. Im Fall eines SDR-Datensatzes wird im nächsten Schritt das virtuelle Wohnzimmer prozedural generiert. Für alle Datensätze wird als Drittes bestimmt, welche Distraktoren, virtuellen Zielobjekte und Lichtquellen wo platziert werden. Außerdem werden ihre Texturen, Farbwerte, Beleuchtungsintensitäten sowie weitere optische Eigenschaften der Objekte festgelegt. Da

im vorherigen Schritt die Objekte zunächst zufällig in der Umgebung positioniert wurden, kann es passieren, dass sich diese ineinander verschoben in der Umgebung befinden. Dies soll in den SDR-Datensätzen vermieden werden, da diese Kollision in der Realität nicht vorkommt. Die Vermeidung findet in den nächsten beiden Phasen der Anwendung statt. Dort wird überprüft, ob eine Kollision der Objekte untereinander auftritt und so lange die Position neu gewählt, bis keine Kollision mehr vorliegt. Nachdem die virtuelle Umgebung fertig gestaltet wurde, wird die Kamera darin platziert. Nun wird ein Bild der Umgebung angefertigt, das zugehörige Label erstellt und beides zusammen abgespeichert. Die Anwendung ist abgeschlossen, wenn eine statisch festgelegte, gewünschte Anzahl an Daten erreicht wurde. Die in Abb. 4.2 abgebildete State Machine wurde in dem Skript *StateMachine.cs* implementiert. Die Zustände dienen hier insbesondere der zeitlichen Koordination der vielen Echtzeit-Komponenten.

4.1.2 Implementierte Funktionalitäten zur Umgebungsgestaltung

Die entwickelte Anwendung beinhaltet alle notwendigen Funktionen, um Datensätze entsprechend den in Kap. 3.2 definierten Anforderungen erstellen zu können. Um die Umgebung dafür dynamisch zur Laufzeit zu gestalten, wurden die folgenden Funktionen implementiert:

- **Beleuchtung**
  - **Gerichtetes Licht**
    Hierbei handelt es sich um eine Beleuchtung, die stets auf den gemeinsamen Mittelpunkt aller virtuellen Zielobjekte gerichtet ist. Sie hat eine unendliche Reichweite und kann beispielsweise genutzt werden, um Sonnenlicht zu simulieren. Hier sind zur Laufzeit die Farbgebung, die Position und die Intensität der Lichtquelle modifizierbar.
  - **Punktuelles Licht**
    Punktuelle Lichtquellen strahlen, im Gegensatz zu gerichteten Lichtern, in jede Richtung mit einer begrenzten Reichweite. Auch bei ihnen lassen sich die Farbgebung, die Position und die Intensität einstellen. Zusätzlich ist die Anzahl der Lichter variierbar.

- **Hintergrund**
  Als Hintergrund der Umgebung kann eine 2D-Textur eingeblendet werden. Dafür können das sichtbare Bild sowie ein Farbstich eingestellt werden.

- **DR4-Boden**
  Für den Datensatz DR4 wird die Möglichkeit der Nutzung einer waagerechten 2D-Textur implementiert, die einen Boden simuliert, indem alle anderen Objekte aus Sicht der Kamera darüber positioniert werden. Vergleichbar zum Hintergrund sind auch hier das sichtbare Bild sowie ein Farbstich einstellbar.

- **Distraktoren**
  Als Distraktoren können unterschiedliche 3D-Objekte in der Umgebung eingeblendet werden. Zum einen wurde der Gebrauch von primitiven Formen wie Quadern, Sphären oder Zylindern implementiert. Zum anderen können alle Objekte in die Umgebung geladen werden, die in einem statisch festgelegten Ordner als vorbereitetes Prefab[1] bereitstehen. Neben ihrer jeweiligen Form können ihre Anzahl, ihre Position, ihre Rotation und ihre Größe dynamisch verändert werden. Zudem sind Textur und Farbgebung der primitiven Objekte variabel.

- **Zielobjekte**
  Auch die zu labelnden 3D-Objekte müssen als Prefab fertig konfiguriert vorliegen, damit sie zur Laufzeit flexibel in die Umgebung geladen werden können. Wie bei den Distraktoren kann man auch hier verändern, welche Objekte sich genau in der Umgebung befinden und welche Position, Rotation und Skalierung bei ihnen vorliegen.

- **SDR Wohnzimmer**
  Für die SDR-Daten wurde die prozedurale Generierung eines virtuellen Wohnzimmers implementiert. Dieses besteht aus einem Boden, vier Wänden, Fenstern, einem Tisch, Stühlen, einem Schachbrett, Deckenlampen und Stehlampen. Die 3D-Modelle werden zufällig aus mehreren Objekten der jeweiligen Kategorie ausgewählt. Auch dafür müssen die Objekte als fertiges Prefab in einem statisch festgelegten Ordner vorliegen. Variabilität ergibt sich dadurch, dass die Positionierung

[1] „Prefabs“ sind in der Unity3D Engine GameObjects, die fertig konfiguriert verwendet werden können. Sie beinhalten direkt alle relevanten Skripte, Texturen oder weitere GameObjects, die dazugehören.[33]
„GameObject“ bezieht sich auf die Einheit der Unity3D Engine. Prinzipiell ist jeder Bestandteil einer dort angelegten 3D-Umgebung ein GameObject, dessen Echtzeit-Verhalten durch dort angehängte C#-Skripte bestimmt wird. GameObjects können eine optische Repräsentation in der Umgebung haben, müssen sie jedoch nicht.[32]

vieler Objekte zufällig gestaltet wird und diese sich zudem an den zufällig gewählten Objekt-Modellen orientiert. Eine genauere Schilderung des Prozesses erfolgt in Kapitel 4.2.

4.1.3 Szenen Struktur

Alle Funktionen zur Gestaltung der Umgebung müssen für die in Kap. 3.2 geplanten Datensätze jeweils unterschiedlich eingesetzt werden. Damit diese Unterscheidung möglichst simpel stattfindet, wird jeder Datensatz in einer eigenen Unity Szene[2] angelegt. Auf diese Weise kann durch die Wahl der Szene der anvisierte Datensatz jederzeit gewechselt werden. Der Aufbau jeder Szene folgt einer ähnlichen Struktur. Alle benötigten GameObjects werden in der Hierarchie, die in Abb. 4.3 zu sehen ist, eingeordnet.

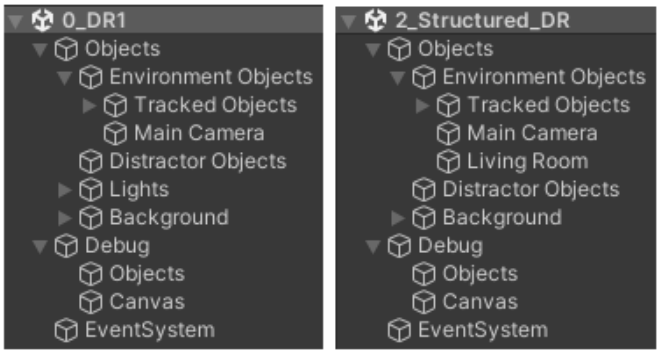

Abbildung 4.3: Hierarchien der Szenen DR1 und SDR1, Quelle: Eigene Abbildung

Die GameObjects werden in zwei Gruppen unterteilt. Die Gruppe *Debug* enthält GameObjects, die zum Debugging dynamisch ein- und ausgeblendet werden können. Dazu gehören zwei- und dreidimensionale Punkte sowie zweidimensionale Rechtecke. Mithilfe der Rechtecke können z.B. die kalkulierten Bounding Box Labels überprüft werden. Alle zweidimensionalen Debug-Objekte werden als Kinder des GameObjects *Debug* > *Objects* und alle dreidimensionalen als Kinder von *Debug* > *Canvas* angelegt.
In der Gruppe *Objects* befinden sich alle GameObjects, die gezielt in der Umgebung zur Gestaltung der Datensätze platziert werden. Sie werden wiederum in mehrere Untergruppen aufgeteilt. Die erste lautet *Background* und enthält den aus einer 2D-Textur

[2] Eine „Szene“ ist eine Einheit der Unity Engine, die über eine eigene, vollständige Umgebung mit allen dazugehörigen GameObjects und ihren Eigenschaften verfügt. So kann mittels eines Szenenwechsels zwischen verschiedenen Umgebungen gewählt werden.[34]

bestehenden Hintergrund und verwaltet diesen mithilfe des Skripts *BackgroundImageHandler.cs*. Die zweite Untergruppe lautet *Distractors*. Hier werden alle GameObjects, die Distraktoren beinhalten, zur Laufzeit dynamisch eingeordnet. Dies geschieht durch das Skript *DistractorObjectHandler.cs*. Die virtuellen Zielobjekte befinden sich in der Untergruppe *Tracked Objects* und werden durch das hier angehängte Skript *TrackedObjectHandler.cs* gehandhabt. Außerdem befindet sich in allen Szenen eine virtuelle Kamera namens *Main Camera* zur Aufnahme der Bilder. Wie die restlichen GameObjects genau eingeteilt werden, hängt davon ab, ob ein DR- oder ein SDR-Datensatz generiert wird. Bei SDR-Datensätzen findet die Gestaltung eines virtuellen Wohnzimmers durch das Skript *EnvironmentHandler.cs*, das zu der Gruppe *Environment Objects* gehört, statt. Die Objekte des Wohnzimmers werden dort der Untergruppe *Living Room* zugeordnet. Auch die Beleuchtung wird in Form von Lampen und Fenstern fest in das Wohnzimmer integriert. Im Gegensatz dazu wird bei DR-Datensätzen die Beleuchtung zufällig positioniert, ohne sich an anderen Objekten zu orientieren. Daher gibt es hier die Untergruppe *Lights* mit dem Skript *LightHandler.cs*, das die Lichter gesondert verwaltet.

4.1.4 Erstellung der Labels

Neben den genannten Skripten, die für die Umgebungsgestaltung verantwortlich sind, wurden weitere entwickelt, die der Anfertigung der Labels nachkommen. Wie in Kap. 3.4 definiert, ist das Ziel der zu generierenden Datensätze, die zweidimensionale Position und Größe der Zielobjekte auf einem Kamerabild festzustellen. In der Unity3D Engine sind die Position, Rotation und die dreidimensionale Bounding Box von Objekten im 3D-Raum direkt abrufbar, jedoch nicht die Bereiche, die sie auf einem Kamerabild einnehmen. Daher muss dies gesondert für jedes virtuelle Kamerabild berechnet werden, was im Skript *BoundingBoxLabel.cs* geschieht. Um zunächst den Bereich auf dem Bild einzugrenzen, in dem ein Objekt zu sehen ist, werden die Eckpunkte der dreidimensionalen Bounding Box des Objekts auf das Kamerabild projiziert und eine zweidimensionale Bounding Box so festgelegt, dass alle Punkte darin eingeschlossen sind. Ein Beispiel für eine solche Bounding Box ist in Abb. 4.4 zu sehen. Dieser Bereich kann jedoch je nach Blickwinkel sehr ungenau sein, da das Objekt ggf. nur einen kleinen Teil des Bereichs einnimmt, wie man ebenfalls der Abbildung entnehmen kann.

Zum Zweck der genaueren Eingrenzung wurde ein Algorithmus implementiert, den Abb. 4.5 visualisiert. In diesem Algorithmus wird der zuvor abgesteckte Bildbereich von außen nach innen in einer festgelegten Schrittgröße so lange abgetastet, bis ein Schritt auf

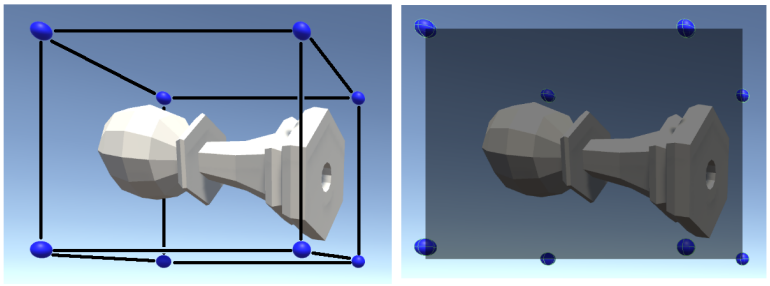

Abbildung 4.4: Bestimmung der maximalen Objekt Grenzen.
a) 3D Grenzen b) Maximale 2D Grenzen, Quelle: Eigene Abbildung

das gesuchte Objekt stößt. Um festzustellen, ob an einer Stelle das gesuchte Objekt zu sehen ist, müssen Raycasts verwendet werden. Ein Raycast ist eine native, relativ zeitaufwendige Operation der Unity3D Engine, die einen mathematischen Strahl von der Kamera durch den Bildpunkt in die 3D-Umgebung verwendet, um zu überprüfen, ob dieser ein bestimmtes Objekt in der 3D-Umgebung schneidet [35]. Durch die schrittweise Abtastung werden weniger Raycasts benötigt, was die Performanz verbessert, jedoch zum Preis von Genauigkeit beim Bestimmen der äußeren Objektgrenzen (siehe Abb. 4.5).

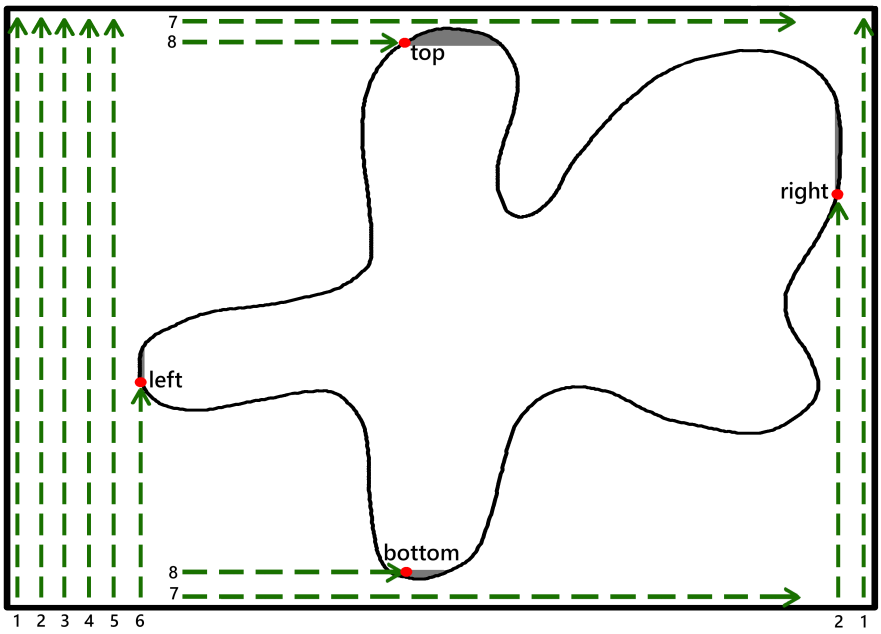

Abbildung 4.5: Algorithmus zur Bestimmung der 2D-Grenzen eines Objekts
Quelle: Eigene Abbildung

Wie in Kap. 3.4 geplant, dienen die generierten Daten als Basis für Trainings des neuralen Netzwerks YOLOv8. Damit die berechneten Bereiche dort als Label eingesetzt werden können, müssen sie in einer .txt-Datei in einem passenden Format abgespeichert werden. Die hier verwendete YOLOv8 Implementierung setzt voraus, dass für jedes gesuchte Objekt eine Zeile in der .txt-Datei steht. Diese besteht aus einer Ziffer, die die Objektklasse identifiziert, zwei Zahlen, die in normalisierter Form für die Position des Objekts auf dem Bild stehen und zwei weiteren Zahlen, die ebenfalls normalisiert die Breite und die Höhe des Bildbereichs definieren. Ein Beispiel für ein solches Label und das dazugehörige Bild sind in der Abb. 4.6 zu sehen. Die Koordination mit den anderen Teilen der Anwendung und die Abspeicherung der Labels in dem passenden Format erfolgen in dem Skript *LabelCreator.cs*.

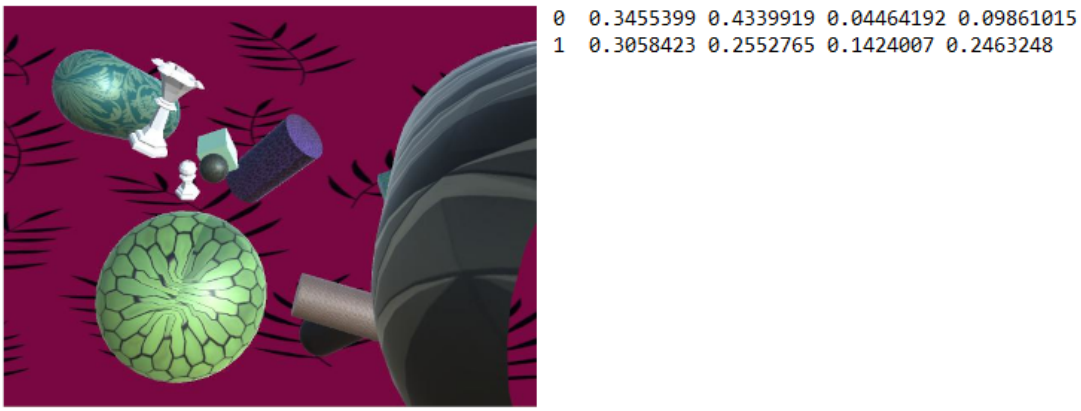

Abbildung 4.6: Label eines generierten DR-Bildes, Quelle: Eigene Abbildung

4.2 Generierte Datensätze

Mit der erläuterten Anwendung wurden sechs synthetische Datensätze generiert. Die Gestaltung orientiert sich an den in Kapitel 3.2 definierten Anforderungen. Für jeden Datensatz wird im Folgenden dargelegt, wie diese Anforderungen konkret umgesetzt wurden. Dafür wird darauf eingegangen, welche visuellen Gestaltungsmittel der Anwendung jeweils verwendet wurden, welche virtuellen Objekte und Texturen zum Einsatz kamen und wie viel Variabilität und visueller Realismus daraus entstanden.

Domain Randomization - DR1

Die Umgebungsgestaltung in DR1 läuft wie folgt ab: Zuerst wird entschieden, welche virtuellen Zielobjekte gezeigt werden. Diese werden in einem statisch festgelegten, begrenzten 3D-Raum vor der Kamera platziert und rotiert. Anschließend wird ausgewählt, wie viele Distraktoren verwendet werden und welche Form sie jeweils haben. Die Anzahl bewegt sich zwischen 10 und 20 Objekten, jeweils in der Form einer Kugel, eines Würfels, eines Zylinders oder einer Kapsel. Die Distraktoren werden in einem weitläufigeren Raum vor der Kamera platziert als die Zielobjekte, um Verdeckungen zeigen zu können. Anschließend werden alle Distraktoren skaliert und rotiert. Jeder Distraktor erhält eine Textur in einer einheitlichen Farbe oder eines von 33 gemusterten Bildern, das zusätzlich einen Farbstich bekommt. Dieselben Texturen stehen für die senkrechte Ebene zur Verfügung, die den Hintergrund bildet. Zum Schluss wird die Beleuchtung der Szene festgelegt. Sie besteht aus einem gerichteten Licht und ein bis drei Punktlichtern. Für alle Lichter wird eine Ausgangsposition und ein Farbton ausgesucht. Alle genannten Entscheidungen und Aktionen passieren zufällig. Dadurch entsteht die Variation der Bilder des Datensatzes. Realitätsnähe ist in diesem Datensatz, bis auf die Ansicht der Zielobjekte, nicht gegeben.

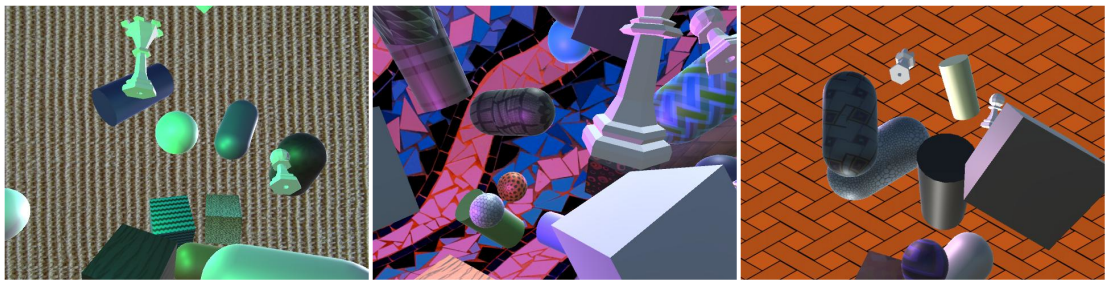

Abbildung 4.7: Beispiele aus dem Datensatz DR1, Quelle: Eigene Abbildung

Domain Randomization - DR2

Verglichen mit der Umgebung aus DR1 verändern sich hier nur die Texturen, die für den Hintergrund und die Distraktoren zur Auswahl stehen. Neben den 33 gemusterten Bildern stehen nun auch 10.000 zufällig ausgesuchte Bilder des Datensatzes *COCO* von Lin u. a. [18] zur Verfügung. Die Bilder haben unterschiedliche Inhalte und variieren stark untereinander. Durch diesen Umstand und die Vielzahl an Auswahlmöglichkeiten ist die visuelle Varianz des Datensatzes DR2 wesentlich größer als die von DR1.

Abbildung 4.8: Beispiele aus dem Datensatz DR2, Quelle: Eigene Abbildung

Domain Randomization - DR3

Auch der Datensatz DR3 steigert die Varianz gegenüber DR1. Jedoch werden hier auch die Formen der Distraktoren vielfältiger gestaltet. Dafür wurden 96 3D-Modelle aus 10 Objektklassen des Datensatzes *ShapeNet* von Chang u. a. [4] zufällig entnommen. Pro Kamerabild werden zwischen 30 und 50 dieser Modelle zufällig als Distraktoren platziert. Ihre Texturen wurden im Datensatz festgehalten und von der hier entwickelten Anwendung zusätzlich zufällig eingefärbt, um die Varianz zu erhöhen. Der restliche Aufbau der Umgebung entspricht dem in DR1.

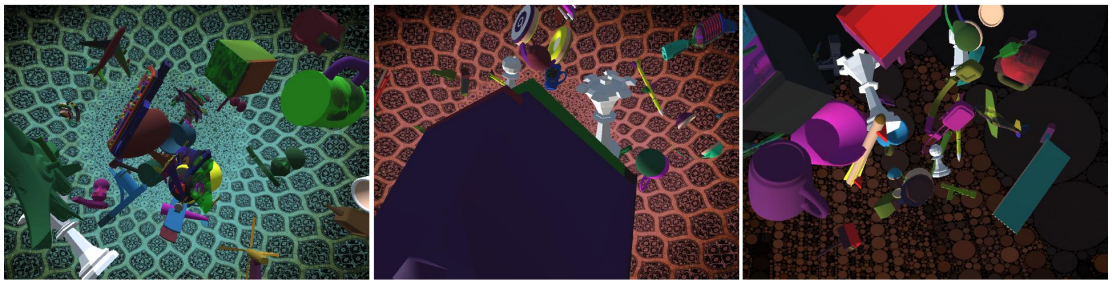

Abbildung 4.9: Beispiele aus dem Datensatz DR3, Quelle: Eigene Abbildung

Domain Randomization - DR4

DR4 setzt voraus, dass die Entwickler der Trainingsdaten die Anwendungsdomäne kennen. Mit diesem Wissen wird die Umgebung realistischer gestaltet, ohne den vollen Kontext einer 3D-Umgebung nachzubauen, wie es für die SDR-Daten getan wird. Dafür wird die Umgebung von DR1 zunächst um eine horizontale Fläche erweitert, die mit einem von 19 Schachbrett-Mustern versehen wird. Die Zielobjekte werden nun zufällig auf dieser Fläche platziert und nur noch um eine Achse rotiert. Auf diese Weise wird das reale Platzieren von Schachfiguren auf einem Schachbrett simuliert. Als Textur für den Hintergrund stehen nun 38 Bilder von aktiv benutzten Schachbrettern, 17 Bilder von Schachtischen, 45 Bilder von Wohnzimmern und 411 Bilder von Tischen zur Auswahl. Dafür wurden Bilder aus drei Open Source Datensätzen verwendet [20, 25, 26]. Die Distraktoren haben in DR4 dieselben geometrisch primitiven Formen wie in DR1 und erhalten ebenfalls einen zufälligen Farbstich. Jedoch werden zusätzlich zufällig Distraktoren bestimmt, die Bilder aus dem Datensatz COCO verwenden. Positioniert werden die Distraktoren in einem statisch festgelegten Bereich über dem Schachbrett. Zuletzt wird auch die virtuelle Kamera in einem statisch festgelegten Bereich über dem Schachbrett und über den Distraktoren zufällig positioniert und so ausgerichtet, dass sie auf das Schachbrett blickt.

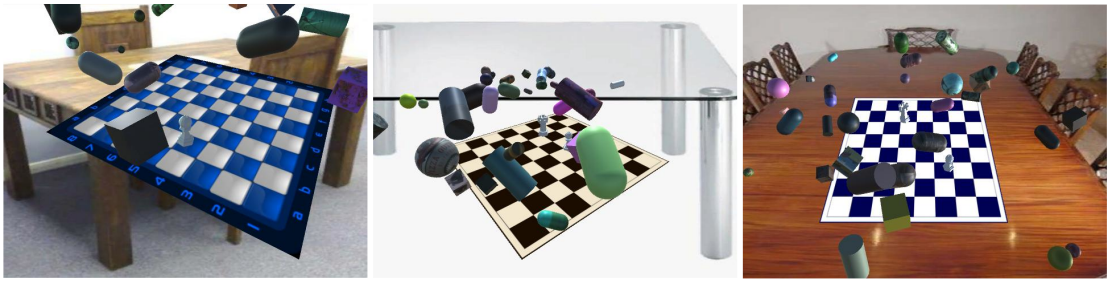

Abbildung 4.10: Beispiele aus dem Datensatz DR4, Quelle: Eigene Abbildung

Structured Domain Randomization - SDR1

Im Gegensatz zu den vorherigen Datensätzen wird für die SDR-Daten die komplette Umgebung, die zu der anvisierten Anwendungsdomäne gehört, nachgestellt. Dafür wurde in der in Kap. 4.1 vorgestellten Anwendung die prozedurale Generierung eines virtuellen Wohnzimmers implementiert.
Die Generierung läuft wie folgt ab: Zuerst wird eine waagerechte Fläche in einer zufälligen

Größe als Boden verwendet. Der Boden wird mit einem Material versehen, das entweder einen Parkett- oder einen Teppichboden zeigt. Anschließend werden die Wände des Raumes generiert. Dafür wird zuerst entschieden, welche Wände Fenster enthalten sollen. Die fensterlosen Wände bestehen aus senkrechten Flächen, die der Größe des Bodens angepasst und mit einem Tapeten-Material ausgestattet werden. Für die anderen Wände wird zunächst ein Fenster-Prefab ausgesucht. Dann wird kalkuliert, wie häufig dieses maximal in die Wand passt und eine Anzahl zwischen eins und der maximalen Zahl ausgewählt. Anschließend werden die Fenster entlang der geplanten Wand platziert und die Lücken mit senkrechten Flächen gefüllt. Im nächsten Schritt wird eine Sitzgruppe hinzugefügt. Dafür werden zuerst ein zufälliger Platz im Raum für den Tisch sowie ein Prefab für ihn ausgesucht. Nun werden Stühle um den Tisch gestellt. Ihre Anzahl ergibt sich aus der Größe des Tisch-Prefabs und der des ausgewählten Stuhl-Prefabs. Auf der Tischfläche wird ein Schachbrett positioniert, auf das später zufällig die Zielobjekte in Form von Spielfiguren gesetzt werden. Zuletzt werden in den vier Ecken des Raumes Stehlampen und zufällig an der Decke eine Deckenlampe platziert. Für alle genannten Objekte stehen mehrere Prefabs zur Auswahl, um diverse Wohnzimmer generieren zu können. Es wurde die Nutzung von 18 Stühlen, 15 Tischen, fünf Fenstern, neun Schachbrettern, zwölf Decken- und 15 Stehlampen vorgesehen. In jedem generierten Wohnzimmer geschieht die Wahl, welche Objekte verwendet werden, zufällig. Auch die Materialien von Boden und Wänden wurden zufallsbasiert gestaltet. Dabei fällt die Entscheidung zwischen acht Tapeten und 24 Teppich- bzw. Parkettböden. Zwei Beispiele für so generierte Wohnzimmer sind in Abb. 4.11 zu sehen.

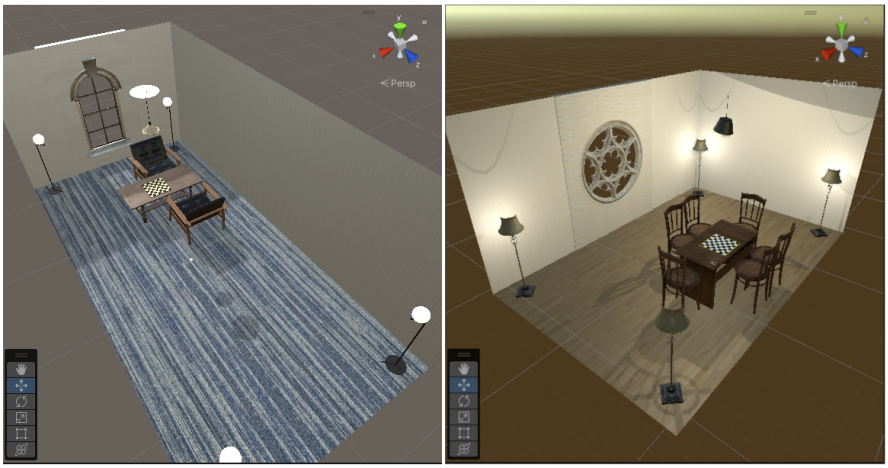

Abbildung 4.11: Zwei generierte Wohnzimmer, Quelle: Eigene Abbildung

Neben der Gestaltung des Raums variiert auch die Beleuchtung. Dabei orientiert sie sich an realistischen Lichtverhältnissen. Zu diesem Zweck wird für jedes Kamerabild entschieden, ob Tag oder Nacht simuliert werden soll. Tageslicht wird dadurch dargestellt, dass aus der Richtung aller Wände, die Fenster enthalten, je ein gerichtetes Licht in das Zimmer scheint. Zudem wird ein weit entferntes Punktlicht mit großer Strahlenlänge dafür verwendet, das Licht im Zimmer diffus zu machen. Auch die geworfenen Schatten werden relativ weich eingestellt. Die Lampen im Zimmer strahlen kein Licht aus. Zur Variation wird die Intensität des Tageslichts zufällig gestaltet, ohne dabei unrealistisch intensiv zu werden. Das linke Bild der Abb. 4.11 zeigt ein generiertes Wohnzimmer im simulierten Tageslicht. In Abb. 4.12 sind drei Beispiele für Bilder des SDR1-Datensatzes, in denen ebenfalls Tag ist, zu sehen.

Abbildung 4.12: Auszüge des SDR1-Datensatzes mit Tageslicht
Quelle: Eigene Abbildung

Bei Nacht strahlen nur noch Punktlichter von den Positionen der Lampen aus. Sie werfen härtere Schatten, als das simulierte Tageslicht (siehe Abb. 4.13 e) und f)). Auch die Intensität kann höhere Werte erreichen, um nachzustellen, dass in der Realität Objekte direkt angestrahlt und dadurch überbelichtet werden (siehe Abb. 4.13 c) und d)). Auch eine niedrige Intensität ist hier vorgesehen, um eher indirekt leuchtende, weit entfernte Lichtquellen nachzuahmen (siehe Abb. 4.13 a) und b)). Das rechte Bild von Abb. 4.11 zeigt ein Beispiel für ein komplettes generiertes Zimmer in der nächtlichen Variante.

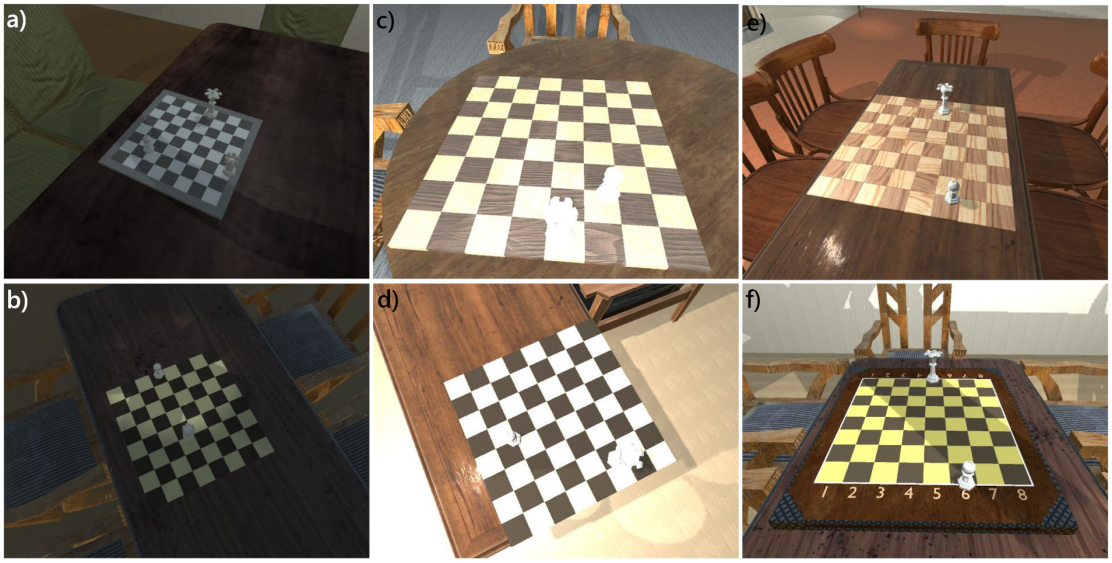

Abbildung 4.13: Auszüge des SDR1-Datensatzes mit nächtlicher Beleuchtung
Quelle: Eigene Abbildung

Structured Domain Randomization - SDR2

Für den Datensatz SDR2 wurde die Umgebung von SDR1 um fliegende Distraktoren erweitert. Diese haben ausschließlich geometrisch primitive Formen. Zufällig wird für jeden von ihnen bestimmt, ob sie einfarbig sind oder mit einer Textur des COCO-Datensatzes versehen werden. Ebenfalls zufällig wird bestimmt, ob die Distraktoren einen Farbstich bekommen (siehe Abb. 4.14 c)) oder nicht (siehe Abb. 4.14 a) und b)). Dadurch entsteht hinsichtlich der Texturen eine hohe Varianz.

Abbildung 4.14: Auszüge des SDR2-Datensatzes, Quelle: Eigene Abbildung

4.3 Performanz der Anwendung

Während der Generierung der Datensätze kam es hin und wieder zu Abstürzen der Unity3D Engine. Wann diese genau auftreten, ließ sich nicht genauer eingrenzen, da sich weder in den zuletzt generierten Daten noch in den Zeitpunkten Regelmäßigkeiten erkennen ließen. Zudem wurde keine Fehlermeldung der Engine ausgegeben und in der Übersicht der gebrauchten Ressourcen des PCs ließ sich ebenfalls keine zu hohe Auslastung ablesen.[3] Auch die Verwendung unterschiedlicher Versionen der Engine änderte den Umstand nicht.
Es gibt mehrere Erklärungsansätze für die möglichen Ursprünge. Einer liegt in der Verwendung der vielfältigen, virtuellen 3D-Objekte, die in diesem Projekt in einer relativ großen Anzahl verwendet und zur Laufzeit geladen werden. In dem Datensatz DR3 wurden beispielsweise 96 unterschiedliche Objekte als Distraktoren dynamisch geladen. Ebenso kamen für die SDR-Datensätze insgesamt 64 3D-Objekte zum Einsatz, um Variation in den Daten zu erreichen. Wenn man die benötigte Zeit der anderen DR-Datensätze mit derjenigen des DR3-Datensatzes vergleicht (siehe Tabelle 4.1), fällt die doppelt so hohe benötigte Zeit auf, die dadurch zu erklären ist. Die verwendeten Modelle enthalten mitunter viele Polygone, die in einer aufwändigen Vorverarbeitung reduziert werden könnten, was hier aus zeitlichen Gründen jedoch nicht geschehen ist. Auch die Operationen, die Unity3D nativ mitbringt, um Ressourcen dynamisch zu laden oder GameObjects wieder zu löschen, könnten ressourcenintensiv sein. Da jedoch bei der Generierung von DR3 nicht wesentlich mehr Abstürze zustande kamen als während den anderen DR-Datensätzen, ist unklar, ob das Laden der 3D-Objekte maßgeblich für die Abstürze verantwortlich ist.

	500 Daten	100.000 Daten
DR1	2.348min	ca. 7.83h
DR2	2.29min	ca. 7.63h
DR3	5.835min	ca. 19.45h
DR4	2.163min	ca. 7.21h

Tabelle 4.1: Gemessene und geschätzte Zeiten der DR-Datensatz-Generierung

Da sich die Abstürze bei der Generierung der DR-Datensätze nur nach ca. 500 bis 10.000 Daten abspielten, wurde die Generierung nach den Unterbrechungen fortgeführt und wie geplant beendet. Die Generierung der SDR-Datensätze wurde jedoch wesentlich häufiger

[3] Die Hardware-Komponenten des PCs, auf dem die Generierung durchgeführt wurde, sind dem Anhang A zu entnehmen.

unterbrochen. Dies geschah bereits nach 3 bis 50 generierten Daten. Aus diesem Grund wurden hier nur 700 Daten pro Datensatz generiert. Diese Zahl wurde angestrebt, da 560 Daten ausreichen, um eine Mischung der SDR-Daten mit den realen Daten in einem Verhältnis von 60% realen und 40% SDR-Daten durchzuführen, da ein realer Datensatz mit ca. 1400 Daten erzeugt wurde (vgl. Kap 5.1). Dieses Verhältnis wurde angestrebt, weil in Kap. 2.2 Tremblay u. a. [29] die besten Ergebnisse damit erzielten. Die 140 restlichen SDR-Daten werden zur Validierung der SDR-Modelle verwendet.

Zu Beginn der Arbeit wurde die Forschungsfrage „Wie viel Aufwand bei der Gestaltung der synthetischen Daten ist notwendig, um das Ergebnis realer Daten zu verbessern?“ gestellt. Gemeinsam mit der Evaluation der Trainingsergebnisse in Kap. **??** und der Betrachtung des Aufwands der Anwendungsentwicklung soll auch die Performanz der Anwendung als Grundlage dienen, um diese Frage in Kap. 6.1 zu beantworten. Dafür werden die Zeiten in Betracht gezogen, die die Anwendung bräuchte, um die Datensätze zu generieren, wenn sie nicht abstürzen würde. Für die Beurteilung der Performanz wurden jeweils 500 DR-Daten und fünf SDR-Daten mit 640x480 Pixel großen Bildern generiert und die dafür benötigte Zeit gemessen. Diese Werte sind in der ersten Spalte der Tabellen 4.1 und 4.2 zu sehen. Da implementiert wurde, dass für jedes Datum eine eigene virtuelle Umgebung nach demselben Prinzip erstellt wird, wächst der Zeitaufwand mit einem größeren Datensatz in etwa linear. Nur die kollisionsfreie Platzierung der Objekte kann eine Abweichung bringen, da die erneute Platzierung der Objekte zufällig stattfindet und somit ein unterschiedlich langer Prozess entstehen kann. Jedoch wurden die Anzahl und die Größe der zu platzierenden Objekte so gewählt, dass ein häufiges Auftreten von Kollisionen unwahrscheinlich ist, sodass sie den Prozess der Generierung nicht stark verlängern. Daher wird als Annäherung von einem linearen Zeitaufwand ausgegangen. Anhand dieser Annahme und den gemessenen Zeiten, wurde der Zeitaufwand für die Generierung von 100.000 Daten in der zweiten Spalte von Tabelle 4.1 und 4.2 geschätzt.

	5 Daten	100.000 Daten
SDR1	2.67s	ca. 14.83h
SDR2	2.87s	ca. 15.94h

Tabelle 4.2: Gemessene und geschätzte Zeiten der SDR-Datensatz-Generierung

5 Evaluation

Die zuvor generierten synthetischen Datensätze werden in diesem Kapitel evaluiert. Zu diesem Zweck wurden auch einige Trainings- und Testdaten in realen Umgebungen aufgenommen. Sie werden im folgenden Kapitel 5.1 vorgestellt. Anschließend werden in Kapitel 5.2 Metriken erläutert, mit denen die Genauigkeit der trainierten Funktion eines Modells gemessen wird. Kapitel 5.3 dokumentiert die Trainingsverläufe der Modelle mit synthetischen und real aufgenommenen Trainingsdaten sowie verschiedenen Mischungen daraus. In Kapitel 5.4 werden schließlich alle Modelle mit den Testdaten getestet und durch den Vergleich miteinander evaluiert.

5.1 Real aufgenommene Datensätze

Die erste Forschungsfrage, die diese Arbeit beantworten will, lautet „Lässt sich das Trainingsergebnis realer Daten durch den Einsatz synthetischer Daten verbessern?“ (siehe Kap. 1). Um diese Frage beantworten zu können, muss als Richtwert ein Modell mit einem Datensatz trainiert werden, der ausschließlich reale Umgebungen inklusive der 3D-gedruckten Zielobjekte repräsentiert. Dieser Datensatz wird im Folgenden *REAL* genannt. Durch den Vergleich der Performanz dieses Modells mit der eines Modells, das mit synthetischen Daten oder einer Mischung synthetischer und realer Daten trainiert wurde, kann der Einfluss der verschiedenen Datensätze auf das Ergebnis beurteilt werden. *REAL* wurde in zwei Ausführungen erstellt. *REAL1* enthält 1.435 real aufgenommene Bilder, während in *REAL2* die Bilder aus *REAL1* durch optische Veränderungen vervielfältigt und variiert wurden, sodass der Datensatz 4.305 Daten enthält. Als optische Veränderungen wurden Helligkeit, Sättigung, Belichtung und Verschwommenheit zufällig eingestellt. Die Augmentationen wurden mithilfe der Webseite *Roboflow*, auf der die originalen Daten ebenfalls manuell gelabelt und für die Datensätze sortiert wurden, automatisch umgesetzt (siehe Kap. 3.3). Die Bilder aus *REAL1* wurden an drei verschiedenen Orten aufgenommen. Zwei davon entsprechen im Aufbau dem Wohnzimmer Szenario, das in den

SDR-Datensätzen simuliert wurde (siehe Abb. 5.1 b) und c)). Der dritte Ort besteht aus einem Tisch mit digitalem Display, auf dem Bilder von unterschiedlichen Schachbrettern während der Aufnahmen angezeigt wurden (siehe Abb. 5.1 a)).

Abbildung 5.1: Orte, an denen der Datensatz REAL1 aufgenommen wurden
Quelle: Eigene Abbildung

Neben diesen Daten werden weitere benötigt, an denen die erreichte Präzision und Robustheit aller trainierten Modelle gleichermaßen getestet werden kann. Um damit auch evaluieren zu können, wie die mit *REAL* trainierten Modelle auf neue Umgebungen reagieren, wurden die Aufnahmen an drei separaten Orten aufgenommen, die ebenfalls den synthetisch generierten Wohnzimmern der SDR-Datensätze ähneln (siehe Abb. 5.2).

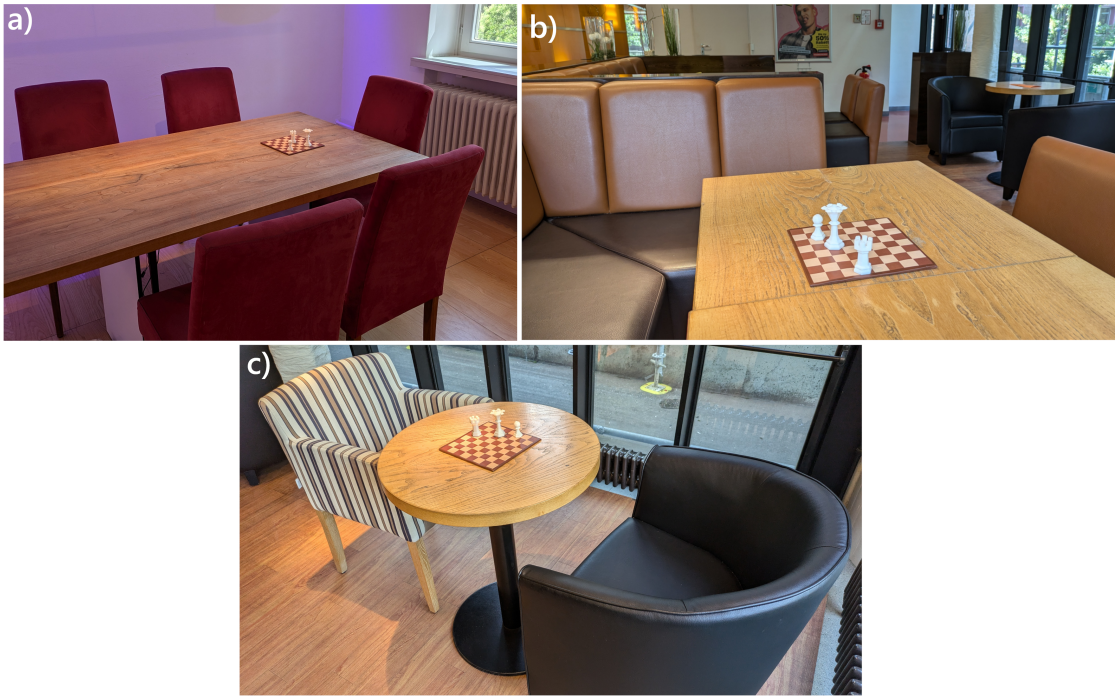

Abbildung 5.2: Orte, an denen die Testdaten aufgenommen wurden.
Quelle: Eigene Abbildung

Die in der Realität aufgenommenen Daten müssen alle Aspekte abbilden, hinsichtlich derer die trainierten Modelle getestet werden sollen. Einer dieser Aspekte ist die Robustheit gegenüber vielfältigen Orten, die einem Wohnzimmer, wie es für die SDR-Daten generiert wurde, gleichen. Die Ortefür die Aufnahmen der Datensätze *REAL* und der Testdaten, wurden entsprechend dieser Anforderung ausgesucht. Auch die Robustheit gegenüber Verschwommenheit der Kamerabilder, Verdeckungen der Zielobjekte und verschiedenen Beleuchtungen soll getestet werden. Die Bilder des Datensatzes REAL wurden mit den Kameras der Smartphones Samsung Galaxy S9, Google Pixel 3XL und iPhone 11 Pro Max gemacht. Die Testdaten wurden mit dem Google Pixel 7A angefertigt.

Damit die Evaluation der Robustheit gezielt hinsichtlich einzelner Bedingungen stattfinden kann, wurden die Testdaten in vier unterschiedliche Datensätze aufgeteilt. Der erste lautet *TEST-SIMPLE* und enthält Kamerabilder, auf denen ausschließlich der Ort, die sichtbaren Zielobjekte, der Blickwinkel der Kamera und die Platzierung der Zielobjekte verändert werden. Diese Eigenschaften werden auch in allen anderen Testdaten variiert. In dem zweiten Datensatz *TEST-VERDECKUNG* wurden die Zielobjekte so fotografiert, dass sie sich gegenseitig verdecken oder gerade von einer Hand versetzt werden und

dadurch nicht vollständig zu sehen sind. Für den Datensatz *TEST-BELEUCHTUNG* wurde die Helligkeit der Kamera so eingestellt, dass sie über- oder unterbelichtete Bilder produziert. TEST-SIMPLE umfasst 129 Daten, TEST-VERDECKUNG 105 Daten und TEST-BELEUCHTUNG 74 Daten.

Abbildung 5.3: Auszüge der TEST-Datensätze, Quelle: Eigene Abbildung

Auch in *REAL* sind alle genannten Eigenschaften vertreten und wurden auf dieselbe Art und Weise erzeugt wie in den Testdaten. Da sich jedoch der Ort der Aufnahme unterscheidet und die konkrete Ausführung der Eigenschaften zufällig geschah, zeigen die Daten vergleichbare Umstände, ohne visuell gleich zu sein. Durch die unterschiedlichen Orte in *REAL* und *TEST* wird simuliert, dass Entwickler des Modells von *REAL* den letztendlichen Einsatzort nicht kennen. Dadurch dass auf dieselbe Weise bei den Aufnahmen vorgegangen wurde, erwartet man, dass *REAL* Modelle liefert, die in der Evaluation mit den Testdaten sehr gute Ergebnisse bringen können. Dadurch soll der Vergleich zu den synthetischen Datensätzen besonders aussagekräftig sein.

5.2 Evaluationsmetriken

Die in dieser Arbeit verwendete Implementierung von YOLOv8 unterstützt die Verwendung einer Reihe von Metriken, um die erzielte Leistung eines trainierten Modells zu bewerten. Diese Metriken fanden auch bereits in den in Kap. 2.2 vorgestellten, verwandten Arbeiten Verwendung. Sie wurden von Jocher u. a. [15] zusammengefasst und hier zur Evaluation eingesetzt.

- **Intersection over Union (IoU)**
  In dieser Metrik wird auf Bildern die Überschneidung zwischen den Ground Truth Bounding Boxen und denen, die das Modell dort vermutet, gemessen. Dann findet eine Division zwischen der Schnittfläche und der vereinigten Fläche beider Boxen statt. Somit liefert der Wert eine Aussage darüber, wie groß der prozentuale Anteil der korrekt vom Modell bestimmten Fläche ist.[15]

- **Precision und Recall**
  Diese beiden Metriken bilden die rechnerische Grundlage einiger anderer Metriken. *Precision* bezieht sich auf das Verhältnis zwischen der Anzahl korrekt identifizierter Objekte und der Anzahl aller Fälle, bei denen das Modell dieses Objekt identifiziert hat, inklusive fälschlich vorgenommener Identifikationen. Somit ist die Precision eine Aussage darüber, wie zuverlässig es sich bei einem erkannten Objekt auch tatsächlich in der Realität um dieses Objekt handelt. *Recall* misst dagegen das Verhältnis der korrekt erkannten Objekte zu der Anzahl der tatsächlichen vorkommenden Objekte. Es wird also angegeben, wie viele der tatsächlich vorkommenden Objekte auch vom Modell erkannt wurden.[15]

- **mean Average Precision (mAP)**
  Die *Mean Average Precision* bietet die Möglichkeit, Precision und Recall für alle vorhandenen Objektklassen zu einem gemeinsamen Wert zusammenzufassen. Sie wird in mehreren Schritten berechnet. Zuerst werden die Eigenschaften Precision und Recall in der *Precision-Recall-Curve* zu einer Metrik vereint. Anschließend wird die *Average Precision* bestimmt, die aus der Fläche unter der Precision-Recall-Curve besteht. Für die Mean Average Precision wird schließlich der Mittelwert der Average Precisions aller zu erkennenden Objektklassen bestimmt. Oftmals wird die Metrik mAP gemeinsam mit einem IoU-Schwellenwert angegeben. Die Metrik

mAP50 bedeutet beispielsweise, dass nur die Erkennungen als korrekt betrachtet werden, die eine IoU von mindestens 50% erfüllen.[15]

- **F1-Score**
  Der *F1-Score* ist ein harmonisches Mittel zwischen Precision und Recall. Das bedeutet, dass der F1-Score nur dann hoch ist, wenn das sowohl auf Precision als auch auf Recall zutrifft.[15]

5.3 Training der Modelle

Alle zu untersuchenden Datensätze werden in diesem Kapitel genutzt, um je ein YOLOv8-Modell damit zu trainieren. Die verwendete Implementierung von YOLOv8 wurde mit COCO vortrainiert [14]. Im Rahmen dieser Experimente liefen die Trainings mit einer Batch Size von 32 und jeweils 500 Epochen, wobei sie frühzeitig unterbrochen wurden, wenn während der letzten 100 Epochen keine Verbesserung mittels der Validierung festgestellt werden konnte, um Overfitting[1] zu vermeiden. Zur Validierung wurden den kleineren Datensätzen SDR1, SDR2, REAL1 und REAL2 jeweils 20% des Datensatzes entnommen. Dies entsprach je 140 Daten bei den SDR-Datensätzen, 287 Daten für REAL1 und 861 Daten für REAL2. Für die umfangreichen DR-Datensätze wurden zusätzlich je 2.000 Daten zu diesem Zweck generiert. Der Rest wurde als Trainingsdaten verwendet. Zu den genannten Unterbrechungen kam es bei den Datensätzen DR1 nach 162 Epochen, bei DR4 nach 389 Epochen und bei REAL1 nach 477 Epochen. Die Tabelle 5.1 dokumentiert die erreichte Genauigkeit der Modelle.

	F1-Score	mAP50
REAL1	0.94	0.97
REAL2	0.84	0.82
DR1	0.85	0.81
DR2	0.97	0.98
DR3	0.92	0.93
DR4	0.98	0.98
SDR1	0.97	0.98
SDR2	0.91	0.93

Tabelle 5.1: Erreichte Genauigkeit der trainierten Modelle

[1] „Overfitting“ beschreibt die übermäßige Anpassung des Modells an die Trainingsdaten. Dies hat zur Folge, dass die erlernte Funktion nicht mehr generalisiert auf andere Daten angewendet werden kann. [12]

Alle Modelle bis auf REAL2 und DR1 haben eine sehr hohe Genauigkeit bei der Erkennung anderer, den Trainingsdaten ähnelnder Daten erzielt. Auch REAL2 und DR1 werden hier als gut genug betrachtet, dass sie für die Evaluation verwendet werden können.
In Kap. 2.2 kamen viele verwandte Arbeiten zu dem Schluss, dass sie mit einer Mischung realer und synthetischer Daten ein besseres Ergebnis erzielen als wenn die Datensätze nur eine Kategorie enthalten. Aus diesem Grund wurden in den Experimenten dieser Arbeit ebenfalls Datensätze getestet, in denen REAL1 mit jeweils 560 Daten der synthetischen Datensätze gemischt wurde. Das entspricht einem Mischverhältnis von 40% synthetischen und 60% realen Daten. Das Verhältnis wurde gewählt, da Tremblay u. a. [31] damit optimale Ergebnisse erzielten (siehe Kap. 2.2). Die Genauigkeit dieser Modelle bei der Validierung mit Daten derselben Kategorien wurde in Tabelle 5.2 festgehalten. Auch die gemischten Datensätze erzielten unter diesen Voraussetzungen sehr gute Zwischenergebnisse.

	F1-Score	mAP50
DR1 + REAL1	0.94	0.97
DR2 + REAL1	0.93	0.96
DR3 + REAL1	0.91	0.94
DR4 + REAL1	0.95	0.97
SDR1 + REAL1	0.96	0.98
SDR2 + REAL1	0.94	0.97

Tabelle 5.2: Erreichte Genauigkeit der gemischten Datensätze

5.4 Test der Modelle

Die trainierten Modelle werden nun mithilfe der Datensätze *TEST-SIMPLE*, *TEST-BELEUCHTUNG* und *TEST-VERDECKUNG* getestet und die Ergebnisse miteinander verglichen. Die jeweils gemessenen F1-Scores sind in der Tabelle 5.3 aufgeführt.

Wie in den verwandten Arbeiten erzielte auch hier der real angefertigte Datensatz REAL1 bessere Ergebnisse als die rein synthetisch generierten Versionen. Da Über- und Unterbelichtung sowie Verdeckungen sowohl in den Testdaten als auch in den Aufnahmen von REAL1 vorhanden sind, ist das Modell wie erwartet diesen Umständen gegenüber relativ robust. Trotzdem wertet es insbesondere extreme Beleuchtungsverhältnisse deutlich we-

	Simple	Beleuchtung	Verdeckung
REAL1	0.96	0.90	0.94
DR1	0.91	0.83	0.83
DR2	0.92	0.77	0.84
DR3	0.89	0.82	0.82
DR4	0.60	0.47	0.37
SDR1	0.59	0.46	0.44
SDR2	0.52	0.52	0.36
DR1 + REAL1	0.97	0.94	0.86
DR2 + REAL1	0.98	0.95	0.88
DR3 + REAL1	0.96	0.94	0.83
DR4 + REAL1	0.98	0.95	0.87
SDR1 + REAL1	0.98	0.97	0.88
SDR2 + REAL1	0.96	0.97	0.87

Tabelle 5.3: F1-Scores der getesteten Modelle

niger zuverlässig aus, als es das bei den anderen beiden Testdatensätzen tut. Dies trifft jedoch auf alle Datensätze zu, die nicht gemischt wurden.

Als nächstes werden die Modelle betrachtet, die ausschließlich mit synthetischen Daten trainiert wurden. Dabei fällt auf, dass DR1, DR2 und DR3 deutlich besser abschneiden als die realistischeren DR4, SDR1 und SDR2. Im Fall der SDR-Datensätze kommt mit einer hohen Wahrscheinlichkeit der Umstand zum Tragen, dass nur 700 anstatt der geplanten 100.000 Daten verwendet werden konnten. Insbesondere für DR4 ist jedoch ein weiterer möglicher Erklärungsansatz die, verglichen mit den besser bewerteten DR-Datensätzen, geringere Variation innerhalb der Daten. Aus ihr könnte eine geringere Anpassungsfähigkeit an die realen Bilder folgen.
DR2 ist am besten an die beiden Testdatensätze TEST-SIMPLE und TEST-VERDECKUNGEN angepasst. Trotzdem lassen die Verdeckungen die Genauigkeit um 8% sinken. Nur die Robustheit gegenüber Beleuchtungen ist bei DR2 verglichen mit denen von DR1 und DR3 auffällig schwach. Dies könnte damit zusammenhängen, dass bei den vielfältigen Texturen in DR2 die variierende Beleuchtung visuell nicht mehr so stark zum Tragen kommt wie in einer weniger detailreich gefärbten Umgebung. Dafür spricht, dass das Modell von DR1 deutlich robuster geworden ist.

Die Modelle haben durch eine Mischung von real aufgenommenen Bildern mit synthetischen Datensätzen durchweg bessere Ergebnisse bei den Tests durch TEST-SIMPLE und TEST-BELEUCHTUNG erreicht. Insbesondere die Robustheit von SDR-Mischungen ge-

genüber extremen Beleuchtungsverhältnissen fällt positiv auf. Das könnte der realistischen Beleuchtung in diesen Datensätzen zu verdanken sein. In der Probe durch TEST-VERDECKUNG schnitten die gemischten Modelle dagegen schwächer ab als bei der alleinigen Verwendung von REAL1. Ein weiteres, auffälliges Ergebnis betrifft die Mischung von DR4 und REAL1. Obwohl DR4 in einem alleinigen Datensatz eine wesentlich weniger genaue Objekterkennung liefert, kann die Mischung mit REAL1 mit den restlichen gemischten Datensätzen mithalten.

Die vorliegenden Messergebnisse sprechen dafür, dass auch die Verwendung ausschließlich synthetischer Daten schon relativ gute Ergebnisse bringen kann, wobei sie nicht so genau und robust sind wie die Verwendung real angefertigter Bilder. Sofern jedoch real angefertigte Daten vorliegen, kann sich die Mischung synthetischer Daten mit real angefertigten Daten lohnen. Hier wäre jedoch noch zu ergründen, warum sich die Modelle hinsichtlich Verdeckungen verschlechterten (vgl. Kap. 6.3).

6 Schluss

In diesem Kapitel werden zunächst die wichtigsten Erkenntnisse dieser Arbeit zusammengefasst. Diese Erkenntnisse werden verwendet, um anschließend die zu Beginn der Arbeit gestellten Forschungsfragen zu beantworten. Zum Abschluss wird ein Ausblick darauf gegeben, wie die Untersuchungen dieser Arbeit erweitert werden können.

6.1 Zusammenfassung der Ergebnisse

Ziel dieser Arbeit ist es, zu ergründen, wie synthetische Daten verwendet werden können, um genaue und robuste Modelle zur Objekterkennung zu erstellen.
Um sich diesem Thema zu nähern, wurden in Kap. 2 zunächst einige verwandte Arbeiten zurate gezogen. Die Arbeiten generierten unterschiedlich realistische und variierende Umgebungen, um damit neuronale Netze zur Objekterkennung zu trainieren. Manche Arbeiten generierten Umgebungen, die sich nicht an der Anwendungsdomäne orientierten und sowohl Zielobjekte als auch Distraktoren zufällig vor der Kamera verteilten. Diese vom Anwendungskontext losgelöste Strategie wird Domain Randomization genannt. Andere Arbeiten stellten den Kontext der Anwendungsdomäne zum Teil fotorealistisch, zum Teil abstrahiert nach und wurden der Structured Domain Randomization zugeordnet. Einzelne Arbeiten mit Domain Randomization näherten sich der Structured Domain Randomization an, indem die Zielobjekte auf einem gemeinsamen Boden platziert und einzelne Texturen an der Realität orientiert wurden.
Ein großes Problem der gemeinsamen Betrachtung der verwandten Arbeiten lag darin, dass diese unter wenig vergleichbaren und zum Teil intransparenten Bedingungen evaluiert wurden, weshalb die Ergebnisse nur bedingt vergleichbar sind. Dies lieferte die Motivation dieser Arbeit, selbst generierte, synthetische Datensätze durch Trainings desselben neuronalen Netzes und mit denselben Testdaten zu evaluieren. Auf diese Weise sollte für die Evaluation der eigens generierten Datensätze eine Vergleichbarkeit geschaffen werden, die den einzelnen verwandten Arbeiten fehlt.

Die fehlende Vergleichbarkeit der einzelnen Arbeiten hatte zur Folge, dass nicht verallgemeinert beurteilt werden konnte, welche Eigenschaften der Umgebungsgestaltung synthetischer Daten welche Änderungen in trainierten Modellen hervorrufen. Jedoch gab es einige Schlussfolgerungen aus den Experimenten, bei denen die Autoren sich einig waren. Zum einen erreichten alle Arbeiten deutlich bessere Ergebnisse, wenn sie real aufgenommene Bilder mit synthetischen Daten kombinierten, als wenn sie eine Kategorie alleine verwendeten. Die Kombination fand statt, indem entweder die Daten zu einem Trainingsdatensatz gemischt wurden oder indem zuerst ein Training mit synthetischen Daten ausgeführt und anschließend mit real aufgenommenen Daten feinjustiert wurde. Auch vergleichbare Kombinationen von DR-Daten und SDR-Daten führten zu guten Ergebnissen, die jedoch in keiner Arbeit mit Kombinationen synthetischer und real aufgenommener Daten direkt verglichen wurden.
Bei der Verwendung ausschließlich einer synthetischen Kategorie erzielten einige Arbeiten unzureichende Ergebnisse, während andere nur mit einem knappen Abstand schlechtere Ergebnisse erzielten als mit real angefertigten Daten. Jedoch erreichten alle Experimente mit nur einer synthetischen Kategorie schlechtere Resultate als mit einem Datensatz realer Aufnahmen, sofern die Testdaten und die real aufgenommenen Trainingsdaten sich ähnlich genug waren.

In Kap. 3 und 4 wurde die Generierung verschiedener synthetischer Datensätze geplant und implementiert. Zunächst entstanden vier Datensätze, die der Domain Randomization zugeordnet werden können. DR1 bot, verglichen mit den anderen DR-Datensätzen, wenig Varianz und den geringsten visuellen Realismus aller synthetischen Datensätze. DR2 wurde hinsichtlich der Texturen stark variabel gestaltet, während bei DR3 die Distraktoren-Formen vielfältig waren. DR4 legte mehr Wert auf Realismus, indem die gesuchten Schachfiguren auf einer Ebene mit Schachbrett-Optik platziert wurden und der Hintergrund Bilder von Orten zeigte, an denen typischerweise Schach gespielt wird. Es wurden zwei weitere Datensätze generiert, die noch stärker auf den realistischen Kontext setzten. Hierfür wurde die prozedurale Generierung eines Wohnzimmers implementiert, das sich für jedes entstandene Bild verändert. Durch den Anspruch an visuellen Realismus gab es verglichen mit den DR-Datensätzen weniger Varianz. In SDR1 wurden nur Kamerabilder dieser Umgebung verwendet, während bei SDR2 zusätzlich Distraktoren verwendet wurden, wie sie in DR4 zum Einsatz kamen. Entstanden sind schlussendlich vier DR-Datensätze mit jeweils 100.000 Daten. Die SDR-Datensätze enthalten nur 700 Daten aufgrund von Abstürzen der Unity3D Engine während der Generierung.

Die Trainings des neuronalen Netzes YOLOv8 mit den generierten Datensätzen und die Evaluation durch den Vergleich der Ergebnisse fand in Kap. 5 statt. Getestet wurden die Modelle hinsichtlich extremer Beleuchtungsverhältnisse, Verdeckungen der Zielobjekte und einfacher Umstände, in denen nur der Kamerablickwinkel, der Ort und die Positionen der Zielobjekte verändert wurden. Alle Modelle, die nur mit einem der zuvor generierten synthetischen Datensätze trainiert wurden, stellten sich als weniger genau und robust heraus, als die Verwendung real angefertigter Daten. Trotzdem boten gerade DR1 und DR2 eine gute Objekterkennung. Die Mischung von SDR-Daten mit real angefertigten Bildern ergab eine noch robustere Erkennung hinsichtlich der einfachen und der stark beleuchteten Testdaten. In diesen Feldern konnte der gemischte Datensatz selbst die real angefertigten Bilder übertreffen.

6.2 Beantwortung der Forschungsfragen

Das Thema dieser Arbeit - die Verwendung synthetischer Daten zur Erstellung genauer und robuster Modelle zur Objekterkennung - wurde in Kap. 1 in vier Forschungsfragen aufgeteilt. An dieser Stelle werden alle bisherigen Erkenntnisse genutzt, um Antworten auf diese Fragen zu geben.

Frage 1: Lässt sich das Trainingsergebnis realer Daten durch den zusätzlichen Einsatz synthetischer Daten verbessern?
Sowohl die verwandten Arbeiten in Kap. 2 als auch die eigenen Versuche zeigen, dass eine Mischung synthetischer Daten mit real angefertigten Daten bessere Ergebnisse erzielen kann, als die alleinige Verwendung real angefertigter Daten.

Frage 2: Reichen synthetische Daten für ein gutes Trainingsergebnis aus?
Alle betrachteten Arbeiten und die eigenen Versuche dieser Arbeit kommen zu unterschiedlichen Schlüssen. Cunha u. a. [5] und Tremblay u. a. [29] kamen mit ihren DR-Datensätzen zu unzureichenden Ergebnissen. Bei Damian u. a. [7] war es ebenso bei Verwendung ihrer SDR-Daten. Prakash u. a. [21] kamen wiederum mit SDR-Daten zu relativ guten Ergebnissen. Auch in den eigenen Experimenten in Kap. 5 erlangte die Verwendung von ausschließlich DR-Daten einen F1-Score von 0.92 bei einfachen Testdaten und 0.83 bzw. 0.84 unter erschwerten Bedingungen. Sowohl in der Arbeit von Prakash u. a. [21] als auch in der vorliegenden Arbeit wurden die Modelle jedoch nicht so gut wie bei einem Training mit real angefertigten Daten.

Tremblay u. a. [31] trainierten jedoch mit einer Mischung von DR- und SDR-Daten ein Modell, das auch die Erkennung durch real angefertigte Bilder übertraf. Prakash u. a. [21] und Cunha u. a. [5] wiesen allerdings darauf hin, dass real angefertigte Daten nur unter der Voraussetzung, dass sie den Testdaten genügend ähneln, wirklich zuverlässige Modelle liefern. Somit könnten ausschließlich synthetische Datensätze eine gute Option sein, wenn keine ausreichend qualitativ hochwertigen, real angefertigten Daten vorliegen. Die negativen Eindrücke von Cunha u. a. [5], Tremblay u. a. [29] und Damian u. a. [7] können ein Zeichen dafür sein, dass diese Strategie nicht immer funktioniert. Jedoch könnten ihre Ergebnisse auch andere Ursprünge haben, wie z.B. einen nicht ausreichenden Umfang der Datensätze oder eine zu große Abweichung der visuellen Eigenschaften der Trainingsdaten von denen der Testdaten. Dies ist im Nachhinein schwer nachzuvollziehen, da die Testdaten der Arbeiten nicht mehr zugänglich sind.

Frage 3: Wie stark hängen der visuelle Realismus und die Varianz der Trainingsdaten mit dem Trainingsergebnis zusammen?

Wie bereits erwähnt wurde, beobachteten Prakash u. a. [21] und Cunha u. a. [5], dass selbst real aufgenommene Bilder nur dann ein gutes Ergebnis liefern, wenn die Trainingsdaten den Testdaten ähnlich genug sind. Selbst, wenn der grobe Anwendungskontext derselbe ist, kann es sein, dass das trainierte Modell nicht ausreichend robust ist gegenüber Umständen, die nicht in den Trainingsdaten widergespiegelt werden. Dass in den eigenen Experimenten die Datensätze DR1, DR2 und DR3 besser abschnitten als der visuell realistischere Datensatz DR4, ist ein Zeichen dafür, dass Realitätsnähe nicht der alleinige ausschlaggebende Faktor ist. Trotzdem ist Realitätsnähe in dem Sinne relevant, dass die visuellen Eigenschaften des Testdatensatzes abgedeckt sein müssen. Dies zeigt sich in den eigenen Experimenten beispielsweise dadurch, dass die Mischung realer Daten mit den SDR-Daten aufgrund der realistisch simulierten extremen Beleuchtungsverhältnisse, hinsichtlich dieser Verhältnisse besonders robust ist. Auch eine hohe Varianz kann dabei helfen, die Eigenschaften der Testdatensätze in den Trainingsdatensätze aufzugreifen. Falls die Varianz allerdings weniger an der Anwendungsdomäne orientiert ist, könnte sie eine geringere maximale Robustheit erreichen als eine Varianz, die sich an den realen Umständen der Testdaten orientiert. So erzielten in der Arbeit von Prakash u. a. [21] beispielsweise die variablen SDR-Daten bessere Ergebnisse als die weniger variablen, aber realistischeren Datensätze GTAV und Virtual KITTI aber auch als der variable, aber weniger realistische DR-Datensatz von Tremblay u. a. [29].

Frage 4: Wie viel Aufwand ist bei der Gestaltung der synthetischen Daten notwendig, um das Trainingsergebnis gegenüber der Verwendung realer Daten zu verbessern?

In den eigenen Experimenten war DR1 der Datensatz, dessen Generierung mit dem geringsten Aufwand implementierbar war. Für ihn wurden kaum zusätzliche Texturen und 3D-Modelle benötigt und die Platzierung aller Objekte fand rein zufällig statt. Gleichzeitig hat sein Training unter den ausprobierten, rein synthetischen Datensätzen mit am besten abgeschnitten. Dies ist ein Beispiel dafür, dass schon mit relativ geringem Aufwand viele Trainingsdaten erzeugt werden können, die zu einem guten Ergebnis führen. In Kombination mit real aufgenommenen Bilddaten führte dies zu einer deutlichen Verbesserung des Modells gegenüber der alleinigen Verwendung der realen Daten. Die Erzeugung der SDR-Daten war mit einem größeren Aufwand verbunden, da hier die Generierung eines Wohnzimmers implementiert werden musste. Doch auch hier zahlte sich in der Kombination mit realen Daten insbesondere die realistische Beleuchtung aus. Dies könnte ein Beispiels dafür sein, dass für Testdaten mit Eigenschaften, die in den realen Daten nicht vorhanden sind, sich eine gezielte Implementierung in Form einer realistischen Simulation lohnen kann.

6.3 Ausblick

Die in dieser Arbeit dokumentierten Experimente können auf unterschiedliche Weise fortgeführt werden.

Die in dieser Arbeit entwickelte Anwendung wurde in der Unity3D Engine entwickelt. Leider kam es bei der Generierung der Daten wiederholt zu Abstürzen der Engine. Daher wäre der Vergleich mit einer äquivalenten Anwendung der Unreal Engine interessant, um festzustellen, ob das Phänomen dort genauso auftritt oder ob die Engine z.B. Funktionen mitbringt, die besser mit der dynamischen Importierung von Objekten zur Laufzeit zurecht kommen. Außerdem wären weitere Schritte zur Evaluationen nötig, um festzustellen, warum die hier probierten gemischten Datensätze Verdeckungen weniger zuverlässig erkannten als extreme Beleuchtungen. Hier könnte beispielsweise die Strategie verfolgt werden, die Trainings- und Testdatensätze dahingehend genauer zu vergleichen, welche Verdeckungen auftreten, um festzustellen, ob sich die Daten in dieser Eigenschaft ähnlich genug sind.

Außerdem besteht die Möglichkeit, die Evaluation auszubauen, um die generierten Datensätze genauer zu analysieren. In Kap. 2.2 wurde gezeigt, dass verwandte Arbeiten beispielsweise testeten, wie umfangreich die synthetischen Datensätze sein müssen, um eine Sättigung der Trainings zu erreichen, indem unterschiedlich große Datensätze für die Trainings benutzt wurden. Zudem wurden Ablationsstudien zu den einzelnen Datensätzen durchgeführt, die zeigten, welchen Einfluss einzelne Gestaltungselemente der synthetischen Umgebung auf das trainierte Modell hatten. Auch die Frage, wie groß die Auswirkungen des hier verwendeten Vortrainings auf die gemessenen Ergebnisse sind, kann untersucht werden, indem die Trainings ohne das Vortraining wiederholt werden. Das gemeinsame Training von SDR- und DR-Daten, wie Tremblay u. a. [31] es durchgeführt haben, könnte zusätzlich genutzt werden, um weiter zu beurteilen, inwiefern auch rein synthetische Datensätze optimale Ergebnisse liefern können. Zuletzt könnten mit den generierten Trainingsdaten auch andere neuronale Netze trainiert werden, um die Auswirkungen auf das Trainingsergebnis zu vergleichen.

Eine weitere Option ist die Erweiterung der synthetischen Datensätze. Beispielsweise könnte die Robustheit hinsichtlich Bewegungsunschärfe oder verschiedener Kameraeinstellungen in den Datensätzen berücksichtigt werden, wenn diese Eigenschaften zusätzlich in der virtuellen Umgebung implementiert werden. Die entwickelte Anwendung zur Generierung der Daten könnte außerdem um die automatische Erstellung anderer Labels ergänzt werden, z.B. Zwecks der Posenerkennung von Objekten. Dies ist bereits in der Architektur der Anwendung vorgesehen (siehe A.1). Zuletzt könnte die Einstellbarkeit von visuellen Eigenschaften in dem Trainingsdatensatz implementiert werden, um beispielsweise aussuchen zu können, wie stark Verdeckungen im Trainingsdatensatz vorhanden sein sollen. Auf diese Weise ließe sich genauer untersuchen, wie ähnlich diese Eigenschaften dem Testdatensatz sein müssen, um gute Ergebnisse zu erzielen.

Die letzte Erweiterungsmöglichkeit bezieht sich auf die in dieser Arbeit gewählte Anwendungsdomäne. Hier wurde ein Anwendungsfall verfolgt, bei dem die Zielobjekte bereits als 3D-Modell vorlagen. In der Objekterkennung ist allerdings oftmals die Ausgangslage, dass das genaue gesuchte Objekt gar nicht im Voraus bekannt ist. Dann ist es das Ziel, alle Objekte einer bestimmten Kategorie zu suchen. Die in dieser Arbeit entwickelte Anwendung könnte man z.B. so erweitern, dass pro Klasse der Zielobjekte ein Datensatz an möglichen 3D-Modellen zur Verfügung steht. Alternativ müsste ein prozeduraler Generator für verschiedene Objektmodelle einer Klasse gebaut werden, um diese zu erzeugen, falls keine passenden Datensätze vorliegen. So könnten in der hier verfolgten Anwen-

dungsdomäne zum Beispiele jeweils mehrere Ausführungen der einzelnen Schachfiguren hinterlegt werden.

Literaturverzeichnis

[1] Borkman, Steve ; Crespi, Adam ; Dhakad, Saurav ; Ganguly, Sujoy ; Hogins, Jonathan ; Jhang, You-Cyuan ; Kamalzadeh, Mohsen ; Li, Bowen ; Leal, Steven ; Parisi, Pete ; Romero, Cesar ; Smith, Wesley ; Thaman, Alex ; Warren, Samuel ; Yadav, Nupur: *Unity Perception: Generate Synthetic Data for Computer Vision.* 2021

[2] Bousmalis, Konstantinos ; Irpan, Alex ; Wohlhart, Paul ; Bai, Yunfei ; Kelcey, Matthew ; Kalakrishnan, Mrinal ; Downs, Laura ; Ibarz, Julian ; Pastor, Peter ; Konolige, Kurt ; Levine, Sergey ; Vanhoucke, Vincent: Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, S. 4243–4250

[3] Calli, Berk ; Singh, Arjun ; Walsman, Aaron ; Srinivasa, Siddhartha ; Abbeel, Pieter ; Dollar, Aaron: The YCB Object and Model Set: Towards Common Benchmarks for Manipulation Research. In: IEEE (Hrsg.): *Proceedings of IEEE International Conference on Advanced Robotics (ICAR '15)*, July 2015, S. 510 – 517

[4] Chang, Angel X. ; Funkhouser, Thomas ; Guibas, Leonidas ; Hanrahan, Pat ; Huang, Qixing ; Li, Zimo ; Savarese, Silvio ; Savva, Manolis ; Song, Shuran ; Su, Hao ; Xiao, Jianxiong ; Yi, Li ; Yu, Fisher: ShapeNet: An Information-Rich 3D Model Repository / Stanford University, Princeton University, Toyota Technological Institute at Chicago. 2015 (arXiv:1512.03012 [cs.GR]). – Forschungsbericht

[5] Cunha, Kelvin B. d. ; Brito, Caio ; Valença, Luas ; Simões, Francisco ; Teichrieb, Veronica: A Study on the Impact of Domain Randomization for Monocular Deep 6DoF Pose Estimation. In: *2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*, 2020, S. 332–339

[6] Dai, Jifeng ; Li, Yi ; He, Kaiming ; Sun, Jian: R-fcn: Object detection via region-based fully convolutional networks, 05 2016

[7] Damian, Alexandru ; Filip, Claudiu ; Nistor, Anamaria ; Petrariu, Irina ; Mariuc, Cătălin ; Stratan, Valentin: Experimental Results on Synthetic Data Generation in Unreal Engine 5 for Real-World Object Detection. In: *2023 17th International Conference on Engineering of Modern Electric Systems (EMES)*, 2023, S. 1–4

[8] Everingham, M. ; Van Gool, L. ; Williams, C. K. I. ; Winn, J. ; Zisserman, A.: *The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results*. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html. 2012

[9] Ezzeddini, L. ; Ktari, J. ; Frikha, T. ; Alsharabi, N. ; Alayba, A. ; Alzahrani, J. ; Jadi, A. ; Alkholidi, A. ; Hamam, H.: Analysis of the performance of Faster R-CNN and YOLOv8 in detecting fishing vessels and fishes in real time. In: *PeerJ Computer Science 10:e2033* (2024)

[10] Gaidon, Adrien ; Wang, Qiao ; Cabon, Yohann ; Vig, Eleonora: Virtual Worlds as Proxy for Multi-Object Tracking Analysis. In: *CoRR* abs/1605.06457 (2016). – URL http://arxiv.org/abs/1605.06457

[11] Geiger, Andreas ; Lenz, P ; Stiller, Christoph ; Urtasun, Raquel: Vision meets robotics: the KITTI dataset. In: *The International Journal of Robotics Research* 32 (2013), 09, S. 1231–1237

[12] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning. Das umfassende Handbuch : Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze*. mitp Verlags GmbH & Co. KG, 2018. – ISBN 978-3-95845-701-0

[13] Hintze, Nadia: *Erzeugung von Trainingsdaten zur Bestimmung der 3D-Pose von Schachfiguren*. 2023. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2022-proj/hintze_hp.pdf

[14] Jocher, Glenn ; Chaurasia, Ayush ; Qiu, Jing: *Ultralytics YOLOv8*. 2023. – URL https://github.com/ultralytics/ultralytics

[15] Jocher, Glenn ; Munawar, Muhammad R. ; Vina, Abirami: *Performance Metrics Deep Dive*. 2024. – URL https://docs.ultralytics.com/guides/yolo-performance-metrics

[16] Johnson-Roberson, Matthew ; Barto, Charles ; Mehta, Rounak ; Sridhar, Sharath N. ; Rosaen, Karl ; Vasudevan, Ram: *Driving in the Matrix: Can Virtual*

Worlds Replace Human-Generated Annotations for Real World Tasks? 2017. – URL https://arxiv.org/abs/1610.01983

[17] Klasson, Marcus ; Zhang, Cheng ; Kjellström, Hedvig: *A Hierarchical Grocery Store Image Dataset with Visual and Semantic Labels.* 2019

[18] Lin, Tsung-Yi ; Maire, Michael ; Belongie, Serge ; Bourdev, Lubomir ; Girshick, Ross ; Hays, James ; Perona, Pietro ; Ramanan, Deva ; Zitnick, C. L. ; Dollár, Piotr: *Microsoft COCO: Common Objects in Context.* 2015. – URL https://arxiv.org/abs/1405.0312

[19] Liu, Wei ; Anguelov, Dragomir ; Erhan, Dumitru ; Szegedy, Christian ; Reed, Scott ; Fu, Cheng-Yang ; Berg, Alexander: SSD: Single Shot MultiBox Detector, 10 2016, S. 21–37. – ISBN 978-3-319-46447-3

[20] Nazarbayev University: *Tables Dataset.* may 2023. – URL https://universe.roboflow.com/nazarbayev-university-fdv4b/tables-4ahcc. – visited on 2024-07-30

[21] Prakash, Aayush ; Boochoon, Shaad ; Brophy, Mark ; Acuna, David ; Cameracci, Eric ; State, Gavriel ; Shapira, Omer ; Birchfield, Stan: Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data. In: *2019 International Conference on Robotics and Automation (ICRA)*, 2019, S. 7249–7255

[22] Ren, Shaoqing ; He, Kaiming ; Girshick, Ross ; Sun, Jian: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: Cortes, C. (Hrsg.) ; Lawrence, N. (Hrsg.) ; Lee, D. (Hrsg.) ; Sugiyama, M. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 28, Curran Associates, Inc., 2015

[23] Shorten, Connor ; Khoshgoftaar, Taghi M.: A survey on image data augmentation for deep learning. In: *Journal of big data* (2019)

[24] Solawetz, Jacob ; Inc., Roboflow: *What is YOLOv8? The Ultimate Guide. [2024].* Jan 2023. – URL https://blog.roboflow.com/whats-new-in-yolov8/

[25] Soongsil University: *Table Detection3 Dataset.* may 2023. – URL https://universe.roboflow.com/soongsil-univ-fcszy/table_detection3. – visited on 2024-07-30

[26] SVA: *living room Dataset*. may 2023. – URL https://universe.roboflow.com/sva-gldvc/living-room-pg2pg. – visited on 2024-07-30

[27] Tekin, Bugra ; Sinha, Sudipta N. ; Fua, Pascal: Real-Time Seamless Single Shot 6D Object Pose Prediction. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, S. 292–301

[28] Tobin, Josh ; Fong, Rachel ; Ray, Alex ; Schneider, Jonas ; Zaremba, Wojciech ; Abbeel, Pieter: Domain randomization for transferring deep neural networks from simulation to the real world. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, S. 23–30

[29] Tremblay, Jonathan ; Prakash, Aayush ; Acuna, David ; Brophy, Mark ; Jampani, Varun ; Anil, Cem ; To, Thang ; Cameracci, Eric ; Boochoon, Shaad ; Birchfield, Stan: Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, S. 1082–10828

[30] Tremblay, Jonathan ; To, Thang ; Birchfield, Stan: Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018, S. 2119–21193

[31] Tremblay, Jonathan ; To, Thang ; Sundaralingam, Balakumar ; Xiang, Yu ; Fox, Dieter ; Birchfield, Stan: Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. In: Billard, Aude (Hrsg.) ; Dragan, Anca (Hrsg.) ; Peters, Jan (Hrsg.) ; Morimoto, Jun (Hrsg.): *Proceedings of The 2nd Conference on Robot Learning* Bd. 87, PMLR, 29–31 Oct 2018, S. 306–316. – URL https://proceedings.mlr.press/v87/tremblay18a.html

[32] Unity Technologies: *Unity - Manual: GameObject*. 2024. – URL https://docs.unity3d.com/Manual/class-GameObject.html. – Zugriffsdatum: 2024-04-25

[33] Unity Technologies: *Unity - Manual: Prefabs*. 2024. – URL https://docs.unity3d.com/Manual/Prefabs.html. – Zugriffsdatum: 2024-04-30

[34] Unity Technologies: *Unity - Manual: Scenes*. 2024. – URL https://docs.unity3d.com/Manual/CreatingScenes.html. – Zugriffsdatum: 2024-04-25

[35] Unity Technologies: *Unity - Scripting API: Physics.Raycast*. 2024. – URL https://docs.unity3d.com/ScriptReference/Physics.Raycast.html. – Zugriffsdatum: 2024-07-26

[36] Westerski, Adam ; Teck, Fong W.: Synthetic Data for Object Detection with Neural Networks: State of the Art Survey of Domain Randomisation Techniques. In: *ACM Trans. Multimedia Comput. Commun. Appl.* (2023), dec. – URL https://doi.org/10.1145/3637064. – ISSN 1551-6857

[37] Xiang, Yu ; Schmidt, Tanner ; Narayanan, Venkatraman ; Fox, Dieter: PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes, 06 2018

A Anhang

Komponenten des PCs, auf dem die Datensätze generiert wurden

- 32GB RAM
- SSD Force MP600 NVMe Speicher
- AMD Ryzen 9 3900X 12-Core Prozessor mit 3.79 GHz
- NVIDIA GeForce RTX 3080

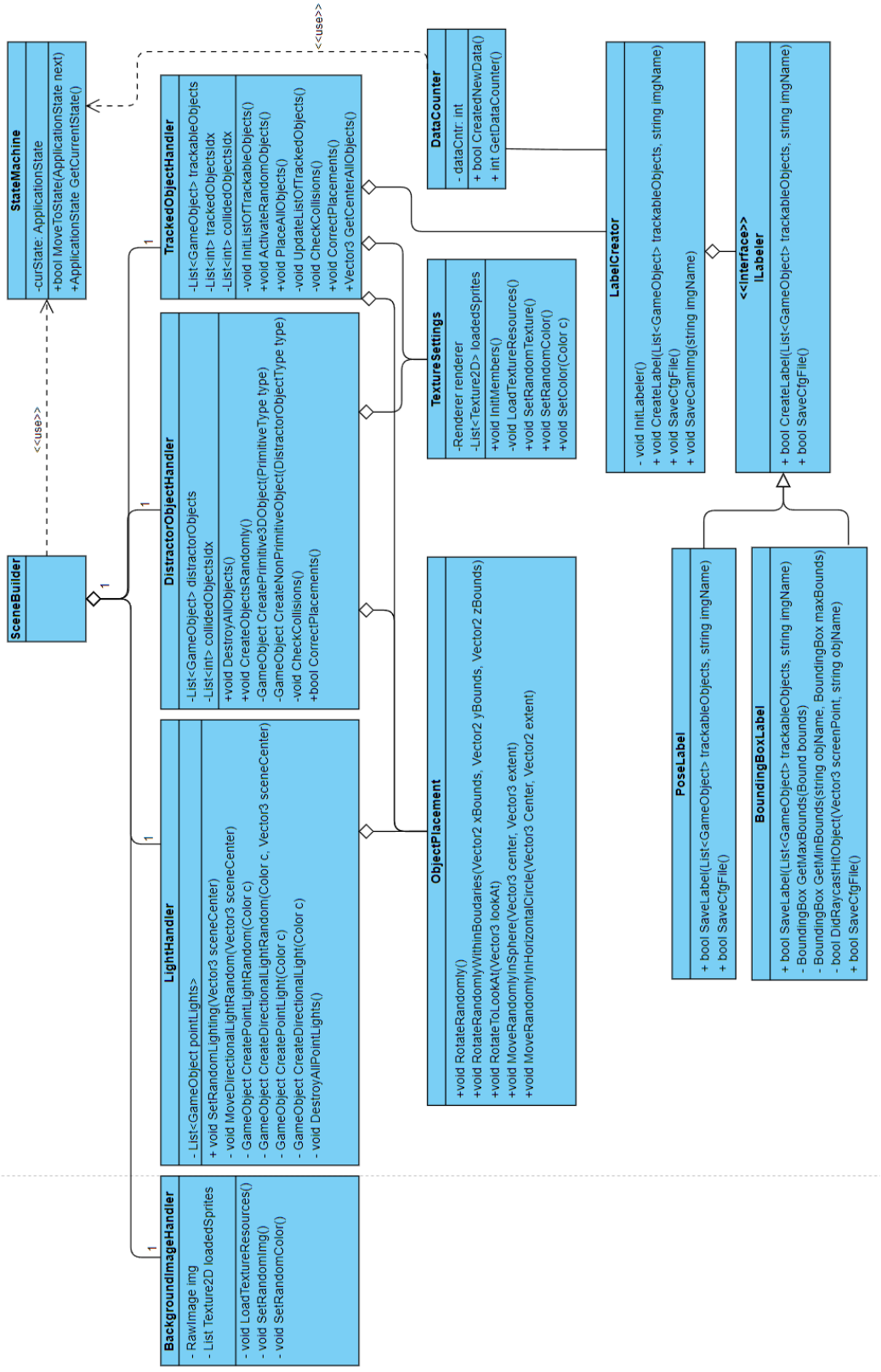

Abbildung A.1: Klassen Diagramm, Quelle: Eigene Abbildung

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original