PIPELINE ZUR ERKENNUNG UND SCHÄTZUNG DER 6-DOF-POSE BEKANNTER OBJEKTE

Dustin Spallek
Department of Computer Science
Hamburg University of Applied Sciences
Berliner Tor 7, 20099 Hamburg, Germany
dustin.spallek@haw-hamburg.de

7. Juli 2019

ABSTRACT

In diesem Projektbericht wird eine Ende-zu-Ende Pipeline zur Objekterkennung und 6DoF (degrees of freedom, deutsch Freiheitsgrade) Lagebestimmung mittels Machine Learning erläutert. Hierzu gehört die Erstellung einer Datenmenge mit Hilfe der Unreal Engine, in die 3D-Objekte geladen werden können, um über einen Algorithmus synthetische Daten in Form von Bildern sowie Metadaten über die Lageinformation zu erzeugen. Diese Daten dienen weiter als Grundlage für das Training eines Neuronalen Netzes. Ziel ist die Beschreibung einer grundlegenden vollständigen Pipeline, welche im Folgeprojekt zur Bestimmung der Position und Ausrichtung definierter Objekte in Bildern Anwendung findet.

***K*eywords** Augmented Reality · Deep Learning · Objektverfolgung · Objekterkennung · 6DoF · synthetische Datenerzeugung

1 Einleitung

Der Mensch scheint Objekte in seiner Umgebung recht problemlos und intuitiv zu erkennen, gleichzeitig findet dabei ein Prozess statt, in dem Informationen an das Gehirn gesendet werden, wo weiter Objekterkennungsprozesse ausgelöst werden. Der Computer hingegen muss die Objekte erst aus beispielsweise Eingabepixeln extrahieren, welche eine wichtige Informationsquelle für den Computer in der Objekterkennung darstellen. In diesem Zusammenhang entwickelten sich im Laufe der Zeit unterschiedliche Mechanismen zur Lösung des Problems der Objekterkennung, welche sich in der Informatik in dem Bereich der Computer Vision ansiedelten. Neben der Objekterkennung wird in der Literatur der Computer Vision von Objekt-Tracking (deutsch Objektverfolgung) geredet [1], welche sich ausschließlich auf die effiziente Verfolgung von bereits erkannten Objekten in einer Sequenz von Bildern spezialisiert. Aus diesem Grund werden Techniken zur Objektverfolgung im Verlauf dieser Ausarbeitung vorgestellt, jedoch nicht im Kontext der Pipeline verwendet, weil das neuronale Netz nur auf die Erkennung und Lagebestimmung von Objekten aus ganzen Bildern trainiert wird.

Passend zu den Aussichten aus der Bachelorarbeit von Marcell Klepper [2], in denen er erwähnt, dass eine Pipeline entwickelt werden könnte, um charakteristische Ansichten von vorhandenen 3D-Objekten zu generieren, wird im Rahmen dieser Arbeit eine Pipeline zur Erkennung und 6DoF-Lagebestimmung von Objekten vorgestellt. In der Pipeline wird zunächst eine synthetische Trainingsmenge mit Hilfe eines Plugins für die Unreal Engine 4 erstellt, welche weiter für das Training eines neuronalen Netzes genutzt werden kann. Somit ist das Ziel die Beschreibung einer funktionsfähigen Pipeline für die Annotation von Objekten in Bildern oder in einem Videolivestream, welche im Folgeprojekt zu dieser Arbeit implementiert und analysiert wird.

2 Problemanalyse

2.1 Objekterkennung

Die Objekterkennung (engl., Objectrecognition) beschreibt ein Verfahren zur Identifizierung von bekannten Objekten mittels optischer, akustischer oder anderer physikalischer Erkennungsverfahren [3]. So wird die Präsenz eines Objektes in einem Bild oder dessen Position und Lage über Methoden aus der Computer Vision beispielsweise mittels Bildverarbeitung bestimmt. Die Methoden zur Objekterkennung variieren hierbei je nach Anwendungsfall. So existiert derzeit noch keine Methode, die für jede Situation ideal ist, da bei der Wahl der Methode unterschiedliche Faktoren eine Rolle spielen. In diesem Zusammenhang ist bei der Wahl der Methode unter anderem Folgendes zu berücksichtigen:

- Was ist das Ziel der Objekterkennung? (z. B. Klassifizierung oder Standortbestimmung)
- Welche Randbedingungen existieren? (z. B. Umgebung, Anzahl der Objekte, Hardware, ...)
- Was für Information liefern die zu erkennenden Objekte (z. B. Form, Textur, Größe,...)

Allgemein sind für die Umsetzung von Objekterkennung Sensoren notwendig, die Informationen aus ihrer Umgebung liefern. Aktuell werden hierbei für Anwendungen beispielsweise 2D-Kameras oder auch RGB-D Tiefenkameras verwendet. So dienen im Fall von 2D-Kameras zweidimensionale Bilder und im Fall von RGB-D Tiefenkameras 3D-Punktewolken [4] als Berechnungsgrundlage.

2.1.1 Methoden zur Objekterkennung

Zur Erläuterung der Funktionsweise soll zunächst ein kurzes Gedankenexperiment dienen. Die einfachste Methode ein Objekt zu erkennen wäre das zu erkennende Objekt zu fotografieren und dieses Objektbild anschließend pixelweise mit einem unbekannten Bild, das das Objekt enthält, zu vergleichen. Diese Methode ist allerdings sehr fehleranfällig, da sie beispielsweise nicht robust gegenüber Größenänderungen des Objekts im Bild, Rotationen des Objektes oder Änderungen des Blickwinkels ist. Des Weiteren wäre sie sehr ineffizient, da zum Beispiel auch eckige Objekte verglichen werden würden, obwohl nach runden Objekten gesucht wird. So kommen zur Bestimmung eines Objektes sogenannte modellbasierte Methoden, erscheinungsbasierte Methoden als auch lernbasierte Methoden zum Einsatz [5].

Modellbasierte Methoden

Bei der Verwendung von modellbasierte Methoden etablierte sich im Laufe der Zeit, wie Yu-Wei Chao1, et al. [8] beschreiben, ein Ablauf in dem zunächst ein Bild vorverarbeitet wird, beispielsweise durch die Minimierung von Aufnahmestörungen und anschließend eine Extraktion der Merkmale stattfindet. Bei der Extraktion der Merkmale werden in der Regel zunächst Kanten detektiert, um in einem Bild Konturen von Objekten hervorzuheben. Hierbei werden starke Änderungen des Pixelwertes, beispielsweise mithilfe eines Sobel Filters, als Hinweis für den Übergang von einem Objekt zum nächsten genutzt. Anschließend findet bei den modellbasierten Methoden zur Extraktion von weiteren Merkmalen häufig eine Houghtransformation statt. Hierbei wird das Eingabe-Bild über einen Algorithmus durchlaufen, der nach Formen sucht. Formen können im einfachsten Fall Geraden oder Kreise sein, es können aber auch Ellipsen und andere geometrische Figuren detektiert werden [9]. Nachdem die Algorithmen durchgelaufen sind, werden die Merkmale mit Merkmalen aus einer Datenbank verglichen, um auf die bekannten Objekte aus der Datenbank zu schließen.

Erscheinungsbasierte Methoden

Erscheinungsbasierte Methoden entwickelten sich aus der Frage heraus, ob die Form mit ihren geometrischen Eigenschaften eines Objektes zur Erkennung dessen ausreicht. Die oben genannten Ansätze haben bisher ausschließlich geometrische Merkmale wie Ecken, Kanten oder Flächeninhalte berücksichtigt, jedoch wurden bisher keine nicht-geometrischen Merkmale wie Farbe oder das Reflexionsvermögen berücksichtigt. Erscheinungsbasierte Methoden nutzen eben diese Merkmale, vernachlässigen dabei jedoch geometrischen Eigenschaften. Dies führt dazu, dass die Nachteile der modellbasierten Erkennung, wie Segmentierungsfehler und Informationsverlust aufgrund der Beschränkung auf geometrische Merkmale umgangen werden können [19]. Zu den erscheinungsbasierten Methoden gehören hierbei unter anderem Eigenspaces und Histogramme.

Die Idee hinter dem Eigensprace ist hierbei die komplette Folge von Bildern in einem Raum, um

ein Objekt herum (dem Eigenspace) zu transformieren, wobei Bilder die eine hohe Korrelation zu anderen Bildern aufweisen weggelassen werden. Hierdurch wird der Eigensprace im Prinzip zum kleinsten Raum, der das Objekt repräsentiert [21].

Bei den Histogrammen ist die Idee eine andere. So wird bei Histogrammen die Anzahl von Pixeln eines Bildes, die alle dieselbe Farbe aufweisen, gespeichert [22]. Da jedoch bei einer hohen Farbtiefe je Farbe umso mehr Einträge zu speichern sind, werden gleichartige Farben in Klassen, z. B. mittels des K-Means-Clustering [6] zusammengefasst. Auch hier werden die Daten anschließend mit Daten aus einer Datenbank zur Klassifizierung abgeglichen.

Lernbasierte Methoden

Zuletzt existieren zur Objekterkennung noch sogenannte lernbasierte Methoden. Hierzu zählen laut Yann LeCun, et al. [5] Nearest Neighbor Methoden (euklidische Distanz), Support Vector Maschinen und Convolutional Networks. Bei der Nearest Neigbor Methode (häufig als kNN bezeichnet, mit k für die Anzahl der gesuchten Nachbarn), wird die euklidische Distanz zu allen Nachbarn innerhalb eines klassifizierten Datensatzes als Grundlage genommen, um die euklidisch nächsten Nachbarn zu finden. So weisen Nachbarn die nah beieinander liegen, viele Ähnlichkeiten auf. Anschließend werden die nächsten Nachbarn zur Klassifizierung des Objektes herangezogen [25].

Anders als bei der Nearest Neigbor Methode unterteilt eine Support Vector Machine (kurz SVM) eine Menge von Objekten in Klassen, sodass ein möglichst breiter Bereich (eine Hyperebene) um die Klassengrenze herum frei von Objekten bleibt. Somit fungiert die Hyperebene als Trennfläche zwischen Klassen, wobei der Abstand der Vektoren, die der Hyperebene am nächsten liegen, maximiert wird. Bei SVM's handelt es sich um ein Verfahren des sogenannten überwachten Lernens, welches zur Klassifizierung eine Menge von Trainingsobjekten fordert, deren Klassifizierung bekannt ist. Aus der Kombination von bekannten Trainingsobjekten und einer Hyperebene mit maximaler Breite können selbst Objekte, derer Zuordnung nicht genau den Klassen aus den Trainingsobjekten entspricht, klassifiziert werden [26].

Faltungsnetzwerke (engl. Convolutional Neural Networks (CNN)) sind eine Spezialform von künstlichen neuronalen Netzen, deren Konzept bereits 1982 von Kunihiko Fukushima, et al. [27] eingeführt wurde. Im Gegensatz zu einem künstlichen neuronalen Netzwerk, welches zur Eingabe einen Vektor erwartet, werden bei CNNs zur Eingabe 3D-Blöcke erwartet, wobei die Höhe, Breite und Tiefe eines 3D-Blocks definierbar ist. Grundsätzlich baut sich ein CNN schichtweise auf und trainiert, wie ein neuronales Netz, bei der Erstellung künstliche Neuronen. Da allerdings bei CNN Bilder als 3D-Blöcke verarbeitet werden, können oben genannte Methoden zur Bildverarbeitung bei der Erzeugung eines CNNs angewendet werden. So werden mit jeder CNN-Schicht beliebig viele Faltungskerne auf das Eingabebild angewandt, wobei als Ergebnis beispielsweise ein durch das Sobel-Filtering gefiltertes Teilbild entsteht. Weiter werden von einer Eingabe schichtweise Merkmale extrahiert, bis beliebig viele Teilmengen der ursprünglichen Eingabe entstehen. Dies wiederholt sich während des Trainings eines CNNs pro Eingabe mehrfach und je nachdem, wie der Aufbau eines CNNs konfiguriert ist.

Faltungsnetzwerke wurden im Laufe der Zeit, wie unter anderem von Yann LeCun, et al. [30] verbessert. Mit der heutigen Leistungsfähigkeit von Computer Hardware und den verbesserten Algorithmen hinter der Erstellung von neuronalen Netzen, stellen nach Alex Krizhevsky, et al. [23] CNNs die State-Of-the-Art-Lösung für das Extrahieren von Merkmalen in RGB-Bildern dar, worauf tiefer im Kapitel 'CNN basierte Objekterkennung' eingegangen wird.

2.2 Object Tracking

Für die meisten Anwendungsfälle ist es laut David G. Lowe [28] nicht wünschenswert jedes Bild erkennen zu lassen, da die Erkennung von Objekten häufig recht rechenintensiv ist. Gleichzeitig werden viele Anwendungen in Zukunft wahrscheinlich auf Geräten laufen, die in ihrer Rechenleistung limitiert sind, was wie Ali Al-Shuwaili et al. [29] beschrieben, eine Auslagerung der Rechenlast fordert. Stattdessen kann nach der Erkennung eines Objektes die dadurch erzeugte Information des Standortes und der Erscheinung des Objektes genutzt werden, um aus zwei aufeinander folgenden Bildern die Bewegungsrichtung sowie die Geschwindigkeit des Objektes zu errechnen. Dabei wird die Suche des Objektes auf einen kleineren Bereich beschränkt, in dem das Objekt im nachfolgenden Bild vermutet wird. Dies hat zur Folge, dass weniger Bilddaten verarbeitet werden müssen, wodurch die benötigte Rechenleistung sinkt. In diesem Zusammenhang

wird in der Literatur allgemein von Object-Tracking (deutsch, Objektverfolgung) gesprochen, wenn die Position eines Objektes in einer aufeinander folgenden Reihe von Bildern, beispielsweise in einem Video, verfolgt wird [7].

Die Definition klingt zunächst einleuchtend, allerdings ist Object-Tracking beim maschinellem Lernen und in der Computer Vision ein breit gefächerter Begriff, der das gleiche Konzept beschreibt, technisch jedoch unterschiedlich umgesetzt wird.

Dr. Satya Mallick[1] beschreibt in diesem Zusammenhang folgende unterschiedliche Ideen zur Umsetzung von Object Tracking:

1. **Dense Optical Flow**
   Zur Abschätzung des sogenannten optischen Flusses (engl. Optical Flow) [10, 11], was die wahrnehmbaren Bewegungen von Strukturen in Form eines Richtungsvektors darstellt.

2. **Sparse Optical Flow** Diese Algorithmen, wie der Kanade-Lucas-Tomashi (KLT) [12, 13] nutzen markante Punkte (engl. Feature Points), zur Bestimmung von Bewegungsvektoren.

3. **Kalman Filtering**
   Ein Algorithmus zur Vorhersage des Standortes von sich bewegenden Objekten, basierend auf vorherigen Bewegungsinformationen [14].

4. **Meanshift (Camshift)**
   Ein Algorithmus, der in diesem Kontext etwas aus der Reihe tanzt, da die Verwendung dieses Algorithmus für Tracking-Zwecke ursprünglich nie beabsichtigt war, jedoch in dieser Rolle recht effektiv ist [16]. So nutzt dieser Algorithmus Dichtefunktionen (z. B. anhand von der Farbverteilung in Bildern) zur Berechnung des Bewegungsvektors [15].

5. **Single-Object-Tracker**
   Eine Klasse von Trackern, die darauf ausgelegt sind möglichst effizient einzelne Objekte in sequenziellen Bildern zu verfolgen. Häufig werden Single-Object-Tracker in Verbindung mit Mechanismen zur Objekterkennung gekoppelt [1].

6. **Multiple-Object-Tracker**
   Wie bereits Single-Object-Tracker in Kombination mit Mechanismen zur Objekterkennung gekoppelt sind, so sind dies auch Multiple-Object-Tracker. Hierbei besteht jedoch keine Begrenzung zur Verfolgung genau eines Objektes, stattdessen werden hierbei mehrere Objekte gleichzeitig verfolgt. Allerdings ist bei der Verwendung von Multiple-Object-Tracker die notwendige Rechenleistung zur Objekterkennung zu berücksichtigen, um die je nach Anwendungsfall definierte Performance einzuhalten [17].

2.3 Methoden zum Object-Tracking

Diese grundlegenden Algorithmen finden noch in den heutigen Tracking verfahren Anwendung, gleichzeitig hat bezüglich Object-Tracking eine Entwicklung, bedingt durch neue Einsatzmöglichkeiten mittels maschinellem Lernen, stattgefunden. In diesem Zusammenhang untersuchte Dr. Adrian Rosebrock[2] unterschiedliche Object-Tracking-Verfahren mittels OpenCV und bewertete deren Leistungsfähigkeit wie die Tabelle 1 zeigt. Weiter empfiehlt Rosebrock CSRT zu nutzen, wenn höhere Genauigkeit benötigt wird und geringere FPS tolerierbar sind. Für ein gutes Mittelmaß zwischen Performance und Genauigkeit ist der KCF Tracker zu empfehlen. Kann allerdings auf eine hohe Performance nicht verzichtet werden, soll stattdessen MOSSE verwendet werden.

Zusammengefasst lässt sich sagen, dass derzeit noch nicht die optimale Lösung gefunden ist, da stets zwischen Genauigkeit und Performance abgewogen werden muss. Hier bleibt demnach noch abzuwarten, wie tauglich die Anwendung von DeepLearning Methoden zum Object-Tracking auf unserer aktuellen Hardware sind.

[1] (https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/), 13. Februar. 2019
[2] (https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/), 16. Mai. 2019

Tabelle 1: Bewertungstabelle aktueller Tracker

BOOSTING	Langsam und für die heutige Anwendungen nicht zu empfehlen.
MIL	Höhere Genauigkeit als der BOOSTING Tracker allerdings schlecht bei Fehlerberichten
KCF	Schneller als BOOSTING und MIL, nicht praktikabel, wenn Objekte verdeckt werden
CSRT	Höhere Genauigkeit als der KCF Tracker dafür etwas langsamer
MedianFlow	Gut in Fehlerberichten
TLD	Recht anfällig für False-Positives, daher nicht empfehlenswert.
MOSSE	Sehr schnelle Funktionsweise, auf Kosten geringerer Genauigkeit als CSRT oder KCF
GOTURN	Deep Learning basierte Objekterkennung, Bewertung ausstehend

3 CNN basierte Objekterkennung

Wie oben erläutert wurden in den vergangenen Jahren bereits einige Verfahren zur Erkennung von Objekten entwickelt. Für die Extraktion von Merkmalen nutzt dabei ein Großteil dieser Verfahren RGB-Bilder. Mittlerweile existiert hierzu ein Trend in Richtung der Nutzung von Tiefenkameras [32]. So beschäftigen sich aktuell Forscher, wie unter anderem Saurabh Gupta, et al. [20] mit der Einbeziehung dieser 3D-Daten bei der Objekterkennung, zur Verbesserung der Qualität bei der Objekterkennung. Weiter wurde von Matan Atzmon, et al. [4] auf der SIGGRAPH[3] ein Konzept vorgestellt, mit dem es möglich ist neuronale Netze mittels Punktwolken in sogenannten PCNNs (Point Cloud Convolutional Neural Networks) anzutrainieren. Bei der Erkennung von Objekten aus 2D-Bildern stellen derzeit grundsätzlich Faltungsnetzwerke (engl. Convolutional Neural Networks (CNN)) die State-of-the-Art-Lösung dar [18]. Im weiteren Verlauf wird der aktuelle Stand zur Objekterkennung mittels CNNs erläutert.

3.1 Entwicklung von Convolutional Neural Networks

Bei der Erzeugung eines Convolutional Neural Networks werden auf einem Bild in einem Convolutional Layer mehrere Faltungskerne angewandt. Diese Faltungskerne repräsentieren hierbei unterschiedliche Filter, wie z. B. den oben genannten Sobel-Filter zur Kantenerkennung.

Von der Grundform der CNNs ausgehend, entwickelte sich die Objekterkennung jedoch weiter. So gelten derzeit sogenannte R-CNNs als erweiterte State-of-the-Art-Lösung, wobei Region Proposal Networks (RPN) mit CNNs kombiniert werden. Dies bewirkt, dass beim Training eines Netzes Hypothesen auf die Lage des Objektes mit in die Netzstruktur einfließen [24]. Gleichzeitig prägte sich der Begriff Fast R-CNN, bedingt durch eine 25-fache Beschleunigung sowie Faster R-CNN mit einer 250-fachen Beschleunigung bei der Erkennung von Objekten. Kaiming He, et al. [24] beschreiben hierbei die Unterschiede zwischen R-CNN, Fast R-CNN und Faster R-CNN, anhand der Netzarchitektur. 2017 stellten Kaiming He, et al. [31] eine weitere Netzarchitektur namens Mask R-CNN vor, die wiederum auf Faster R-CNNs aufbaut. Hierbei werden parallel während der Erstellung eines Models zu jedem Objekt Segmentierungen vorgenommen, wodurch das Model in der Lage ist die Silhouette eines erkannten Objektes wiederzugeben.

Unter Verwendung der Mask R-CNN Architektur entwickelten Yu Xiang, et. al [33] ein neuronales Netz, welches aus Bildern die 6DoF Position erkannter Objekte liefert. Für die Extraktion der 6DoF Position werden zwei Modelle trainiert (siehe Abbildung 1). Eines zur Segmentierung (mittels Mask R-CNN), basierend auf realen Bilddaten und eines zur Lagebestimmung, basierend auf synthetischen Daten (mittels PoseCNN) [34], die in einer Ende-zu-Ende Architektur zusammengefasst werden. Dabei wird die Ausgabe des Netzes zur Segmentierung als Eingabe für das Pose Interpreter Network verwendet. Weiter operiert das Pose Interpreter

[3] https://www.siggraph.org/, 08.10.2018

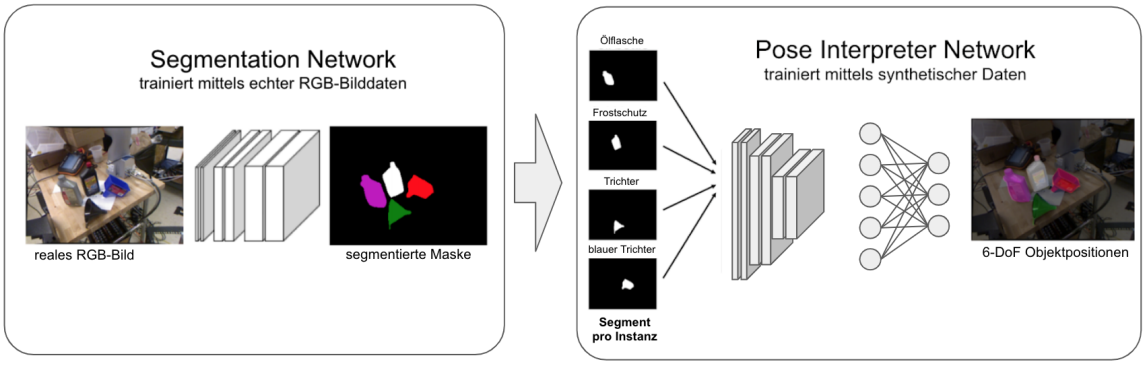

Abbildung 1: Ende zu Ende Architektur für die 6DoF Lage von Objekten in Bildern [34]

Network auf einzelnen Objektinstanzen bekannter Objektklassen aus der Segmentierung und wird rein auf Basis synthetischer Daten trainiert. In dem dargestellten System wird somit pro Bild einmal das Netz zur Segmentierung durchlaufen und anschließend wird für jede Objektinstanz das Pose Interpreter Netzwerk durchlaufen, um die Lage der Objekte zu bestimmen.

3.2 Erzeugung synthetischer Trainingsdaten

Die Erzeugung der notwendigen Menge an realen Bilddaten zum Training des Segmentation Networks von Yu Xiang, et. al [33] ist erheblich. Hinzu kommt, dass die Erzeugung der Daten mit einem hohen Aufwand verbunden ist, damit das Netz ein generalisierendes Lernverhalten aufweist.

So sind bei der Erzeugung der realen, als auch der synthetischen Trainingsmenge viele Einflüsse der Umgebung, wie unter anderem Lichtverhältnisse, Verdeckungsgrad und variierende Hintergründe zu berücksichtigen. Hierbei wird in der Literatur von einem Reality-Gap gesprochen, also dem vorhandenen Unterschied zwischen den Fotos der Aufnahmen aus der Umgebung bei zur Erzeugung der Trainingsmengen, und der realen Umgebung je nach Anwendungsfall.

Hierfür wurde von Nvidia[4] auf der Conference on Robot Learning (CoRL) 2018 eine Lösung vorgestellt, bei der eine realistische synthetische Datenmenge mit Hilfe der Unreal Engine 4 erzeugt wird. Jonathan Tremblay et al. [35, 36] zeigen, dass das Problem des Reality-Gaps mit der Kombination aus zufälligen digitalen Umgebungen und fotorealistischen Bildern erfolgreich übergangen werden kann. Diese Lösung verspricht, für das Training von tiefen neuronalen Netzen, die Erzeugung von erheblich großen vorgefertigten Trainingsdaten mit wenig Aufwand.

4 Aufbau der Pipeline

Die gezeigten Ansätze zur Erkennung und Schätzung der 6DoF-Lage bekannter Objekte unter Verwendung von einer neuartigen Architektur- und Datengenerierungspipeline werden im weiteren Verlauf dieser Arbeit detaillierter erklärt. Der Aufbau der Pipeline richtet sich hierbei grundsätzlich an den aufeinander aufbauenden Arbeiten von Jonathan Tremblay et al. [35] und Yu Xiang et al. [33] (siehe Abbildung 2).

4.1 Netzarchitektur

Das tiefe neuronale 'Poseinterpreter Network' von Jonathan Tremblay et al. [35] orientiert sich neben der Arbeit von Yu Xiang, et. al [33] zusätzlich an den von Shih-En Wei et al. [37] vorgestellten Convolutional Pose Machines (CPM), welche durch den Einsatz einer sogenannten 'intermediate supervision' (Zwischenbegutachtung), das Problem der verschwindenden Gradienten für Faltungsnetze erfolgreich vermindert. Shih-En Wei et al. [37] stellen hierbei die Entwicklung des Gradienten von der Ausgabe bis zur Eingabe eines Netzes in Histogrammen dar. In jeder Phase eines CPMs werden Bildmerkmale und die von der vorherigen Phase erstellten 'belief maps' (Glaubenskarten) verwendet. Der Begriff der Glaubenskarte beschreibt laut Shih-En Wei et al. [37] das Ergebnis der Ableitung eines Knotens zu einem vorherigen Zeitpunkt. Bei Modellen ohne

[4] `https://research.nvidia.com/publication/2018-09_Deep-Object-Pose`, 17. 05. 2019

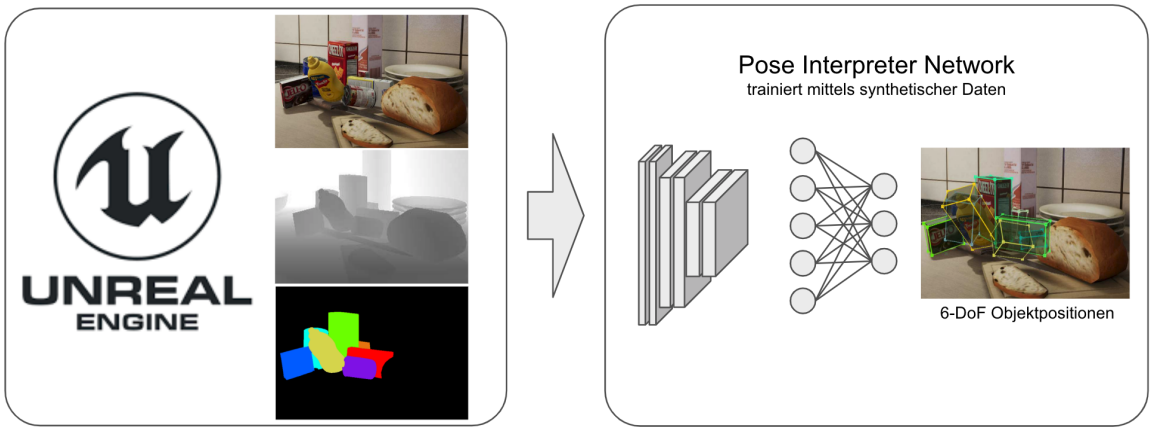

Abbildung 2: Architektur- und Datengenerierungspipeline [36]

Zwischenbegutachtung konnten sie beobachten, dass die Gradientenverteilung aufgrund von verschwindenden Gradienten um null herum eng zugespitzt ist. Modelle mit Zwischenbegutachtung weisen hingegen eine viel größere Varianz über alle Ebenen auf, was darauf hindeutet, dass die Lernfähigkeit des Netzes weiter erhalten bleibt. Angeregt von CPMs erkennt das vollständig konvolutionäre und tiefe 'Poseinterpreter Network' somit Schlüsselpunkte unter Verwendung einer mehrstufigen Architektur. Das Feedforward-Netzwerk nimmt als Input ein RGB Bild der Größe Breite * Höhe * 3 (Breite = 640, Höhe = 480, vgl. [35]) und verzweigt sich, um zwei verschiedene Ausgaben zu erzeugen, nämlich Glaubenskarten und Vektorfelder. Es gibt neun Glaubenskarten, eine für jeden der projizierten 8 Knoten der 3D-Begrenzungsrahmen und eine für die Zentroide. Ebenso gibt es acht Vektorfelder, die die Richtung von jedem der 8 Knoten zum entsprechenden Schwerpunkt angeben, ähnlich wie bei [33], um die Erkennung mehrerer Instanzen des gleichen Objekttyps zu ermöglichen. Das Netzwerk arbeitet in Phasen, wobei jede Phase nicht nur die Bildmerkmale, sondern auch die Ausgänge der unmittelbar vorhergehenden Phase berücksichtigt. Diese Eigenschaft ermöglicht es dem Netzwerk, Unklarheiten im Frühstadium aufgrund kleiner Empfangsfelder zu beheben, indem es in späteren Phase immer größere Mengen an Kontext einbezieht.

Die Extraktion von Merkmalen geschieht in den ersten zehn Schichten aus dem vom ImageNet vor trainierten VGG-19 Datensatz [38], gefolgt von zwei 3 * 3- Faltungsschichten, um die Merkmalsdimension von 512 auf 256 und von 256 auf 128 zu reduzieren. Diese 128-dimensionalen Merkmale werden der ersten Phase zugeführt, die aus drei 3*3*128 Schichten und einer 1*1*512 Schicht besteht, gefolgt von entweder einer 1*1*9 (Glaubenskarten) oder einer 1*1*16 (Neigungsvektoren) Schicht. Die restlichen fünf Phasen sind identisch mit der ersten Phase, mit der Ausnahme, dass sie einen 153- dimensionalen Eingang (128 + 16 + 9 = 153) erhalten und aus fünf 7*7*128 Schichten und einer 1*1*128 Schicht vor der 1*1*9 oder 1*1*16 Schicht bestehen. Alle Phasen sind in den Größen Breite/8 und Höhe/8 ausgeführt, wobei die ReLU-Aktivierungsfunktionen durchgängig verschachtelt sind (siehe Abbildung 3).

4.2 Erkennung und Schätzung der Pose

Nachdem das Netzwerk ein Bild verarbeitet hat, ist es notwendig, die einzelnen Objekte aus den Glaubenskarten zu extrahieren. Im Gegensatz zu anderen Ansätzen, bei denen Verfahren zur Individualisierung der Objekte oder komplexe Architekturen erforderlich sind [39, 40], beruht der Ansatz von [35] auf einem Nachbearbeitungsschritt, der in den Glaubenskarten oberhalb eines Schwellenwertes nach lokalen Spitzenwerten sucht, gefolgt von einem Zuweisungsalgorithmus, der projizierte Knoten mit den erkannten Mittelpunkten verknüpft. Für jeden Knoten vergleicht dieser letztgenannte Schritt die am Knoten ausgewerteten Neigungsvektoren mit der Richtung vom Knoten zu jedem Schwerpunkt, wobei der Knoten dem nächstgelegenen Schwerpunkt innerhalb eines Winkelschwellenwerts des Vektors zugeordnet wird. Nachdem die Eckpunkte jeder Objektinstanz bestimmt wurden, wird ein PnP-Algorithmus (Perspective-n-Point) [41] verwendet, um die Pose des Objekts zu schätzen, ähnlich wie bei [39, 40]. Der PnP-Algorithmus verwendet hierbei die erfassten projizierten Eckpunkte des Begrenzungsrahmens, die Eigenschaften der verwendeten Kamera und die Objektabmessungen, um die endgültige Translation und Rotation des Objekts in Bezug auf die Kamera wiederherzustellen. Alle erfassten projizierten Knoten werden verwendet, solange mindestens die minimale Anzahl von vier Knoten erkannt wird.

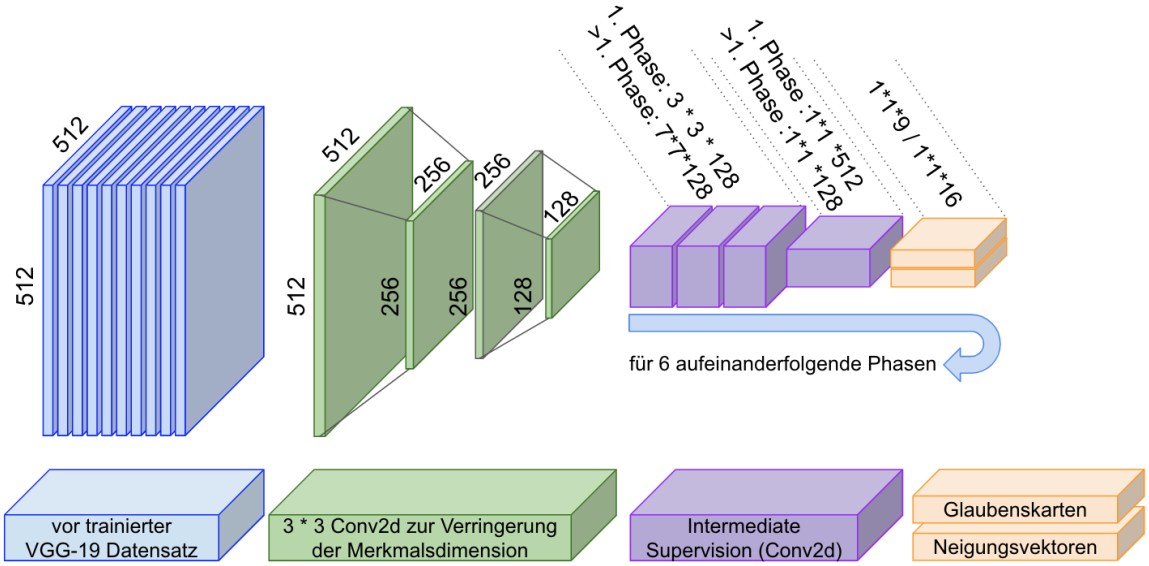

Abbildung 3: Veranschaulichung der Netzarchitektur [35]

4.3 Datenerzeugung

Bei der Erstellung der Trainingsmenge stellt sich die Frage, wie möglichst effektive Trainingsdaten erzeugt werden können. Beschriftete Daten für die 3D-Objekterkennung lassen sich manuell kaum erzeugen, da hierfür erhebliche Aufwände, wie Messungen des Abstandes zur Kamera oder der Rotation der Objekte aus verschiedenen Kameraperspektiven, nötig sind. Selbst halb automatische Beschriftungen mit Werkzeugen, wie z. B. LabelFusion [42] sind arbeitsintensiv, wenn Trainingsdaten mit ausreichender Variation zu generieren sind. So sind beispielsweise keine realen Trainingsdaten für 6DoF-Positionsschätzungen bekannt, die extreme Lichtverhältnisse oder die Posen von Objekten beinhalten. Um diese Einschränkungen der realen Daten zu überwinden, greifen [35] auf synthetisch generierte Daten zurück. Besonders bei diesen synthetischen Daten ist die Kombination aus nicht-fotorealistischen Umgebungsdaten (Domain Randomized) und fotorealistischen Daten zur Datenaugmentierung, weil hiermit die Stärken von beiden Bereichen mit in die Traininsmenge einfließen, was einer Überanpassung (englisch overfitting) entgegenwirkt. Weiter haben synthetische Daten zusätzlichen Vorteil, dass sie sich nicht an eine bestimmte Datensatzverteilung anpassen und so ein Netzwerk bilden, das robust gegenüber Lichtveränderungen, Kameravariationen und unterschiedlichen Hintergründen ist. Alle Daten wurden mit einem benutzerdefinierten Plugin generiert, das Nvidia für die Unreal Engine 4 (UE4) namens NDDS [43] entwickelt hat. Das Plugin erzeugt mit einer Rate von 50-100 Hz Daten, was deutlich schneller ist als beispielsweise die standardmäßige UE4-Screenshot-Funktion. Abbildung 4 zeigt Beispiele von Bildern, die mit dem Plugin erzeugt wurden und die die Vielfalt und Vielzahl sowohl von randomisierten als auch von fotorealistischen Daten veranschaulichen.

Weiter wurde die von [35] verwendete Trainingsmenge, im Vergleich zu anderen Datenmengen angefertigt und in [36] näher beschrieben. Insgesamt bietet der Datensatz folgende Informationen:

1. Tiefeninformation zur Bestimmung der Entfernung der Objekte
2. stereoskopische Ansichten, zur Simulation von RGBD-Kameras
3. 3D Pose von Objekten
4. Rotationseigenschaften zur Bestimmung der Ausrichtung von Objekten
5. Information zu teilweise verdeckten Objekten für bessere Objekterkennung
6. variierende Lichtverhältnisse für fotorealistische Bildeigenschaften
7. Segmentation von Objekten
8. 3D Rahmen-Koordinaten für alle Objekte

Abbildung 4: Beispiele zu randomisierten (links) und fotorealistischen (rechts) Bildern, die für das Training genutzt werden [35]

4.3.1 Domain Randomization

Bei der Erstellung der randomisierten Bilder wurde der Hintergrund mit zufälligen Bildern ausgetauscht, um eine Augmentierung des Datensatzes zu bewirken. Weiter wurden neben den zu erkennenden Objekten mehrere variierende Ablenkungsobjekte zufällig in der Szene platziert, überlagernde Texturen sowie unterschiedliche Beleuchtungen und Rauschen verwendet, um eine Überanpassung zu vermeiden und den Reality-Gap zu überbrücken. Genauer gesagt, wurden Anzahl und Arten von Ablenkungsobjekten aus einer Reihe von 3D-Modellen (Kegel, Pyramiden, Kugeln, Zylinder, etc.) ausgewählt, die Textur der Objekte zufällig ausgetauscht und der Hintergrund aus Fotografien (aus einer Teilmenge von 10.000 Bildern aus dem COCO-Datensatz [45]) ausgetauscht oder prozedural mit zufälligem mehrfarbigem Gittermuster erzeugt. Weiter wurden die 3D-Posen von Objekten und Ablenkungsobjekten gleichmäßig verteilt, das Licht mit unterschiedlicher Intensität eingefärbt und zufällig ausgerichtet. Insgesamt wurde sich bei der Erzeugung der zufälligen Umgebung an dem Artikel [44] orientiert.

4.3.2 fotorealistische Bilder

Die fotorealistischen Bilder wurden dadurch erzeugt, indem die zu erkennenden Objekte innerhalb einer 3D-Szene unter physikalisch korrekten Beleuchtungsbedingungen platziert wurden. Als 3D-Szenen wurden von [35] drei verschiedene Umgebungen (Wald, Küche und Sonnentempel) ausgewählt, welche aus der UE4-Umgebung stammen. Diese Umgebungen wurden aufgrund ihrer hochgenauen Modellierung und Qualität sowie der Vielfalt der Innen- und Außenaufnahmen ausgewählt. Für die Erstellung des Datensatzes wurden hierbei die gleichen Haushaltsobjekte aus dem YCB Datensatz, wie in [33] verwendet. Hierbei wurden die Objekte über definierte Flächen innerhalb der 3D-Szenen in der Luft erzeugt und unter Last der simulierten Schwerkraft in die Szene fallen gelassen, wobei die Kollision der Objekte untereinander und mit den Objekten aus der 3D-Szene berechnet wurden. Somit interagieren die Objekte auf physikalisch plausible Weise und erhöhen somit den Zufall bei der Platzierung der Objekte. Während die Objekte fielen, wurde das virtuelle Kamerasystem schnell auf zufällige Winkel, Höhen und Entfernungen in Bezug auf einen Fixierungspunkt teleportiert, um Daten zu sammeln. Der dabei erzeugte Datensatz trägt den Namen 'Falling Things (FAT)' und wurde in [36] öffentlich zugänglich gemacht.

4.4 Training

Für das Training des Pose Interpreter Networks verwendete [35] ca. 60.000 domain-randomisierte Bilder gemischt mit ca. 60.000 fotorealistischen Bildern. Weiter wurden für das PoseCNN [33] die öffentlich verfügbaren Gewichte verwendet, die mit einem nicht bekannten synthetischen Datensatz trainiert wurden. Um

das Problem der verschwindenden Gradienten in dem Netzwerk zu vermeiden, wurden wie bereits erwähnt CPMs [37] genutzt, welche am Ausgang jeder Phase ein Verlust berechnen, der die mittlere quadratische Abweichung für die Glaubenskarten und Neigungsvektoren verwendet. Das Netzwerk wurde mit PyTorch v0.4 implementiert. Die VGG-19 Feature-Extraktionen wurden aus öffentlich zugänglichen Trainingsgewichten in offenen Modellen entnommen und das Netzwerk wurden für 60 Epochen mit einer Batchgröße von 128 trainiert. Als Optimierer wurde Adam verwendet, wobei die Lernrate auf 0,0001 eingestellt wurde. Das System wurde auf einer NVIDIA DGX-Workstation (mit 4 NVIDIA P100- oder acht V100-GPUs) trainiert, und die Tests wurden mit einer NVIDIA Titan X durchgeführt.

Auf Nachfrage in der Forendiskussion zum Deep_Object_Pose-Projekt[5] äußerte sich Jonathan Tremblay darüber hinaus zum Training. Dabei schrieb er, dass die originalen Gewichte auf vier P100, oder acht V100 über mehr als 24 Stunden, bei einer Batchgröße von 128, bzw. 224 auf 220.000 Bildern trainiert wurden und es ihn überraschen würde, wenn eine einzelne GPU über Nacht ansatzweise erfolgreiche Ergebnisse erzielen würde. Weiter schrieb Tremblay, dass er vermutet, dass eine erste vorzeigbare Performance mit ca. 200.000 Bildern aus dem FAT-Datensatz [36] nach 10 Epochen mit einer Batchgröße von 96 auf 4 GPUs erreicht werden könnte.

5 Fazit

Beschrieben wurde eine Pipeline zur Erkennung und Schätzung der 6DoF-Pose bekannter Objekte unter Verwendung einer neuartigen Netzarchitektur sowie Methoden zur synthetischen Datengenerierung. Das tiefe neuronale Pose Interpreter Netzwerk verwendet mehrere Stufen zur Verfeinerung und Schätzungen der Positionen im dreidimensionalen Raum. Dabei wird eine zweidimensionale Position eines Objektes auf einem Bild in einen dreidimensionalen Raum projiziert und um 3D-Rahmen-Koordinaten erweitert. Diese Punkte werden weiter verwendet, um die endgültige Pose vorherzusagen, wobei von bekannten Kameraeigenschaften ausgegangen wird. Das tiefe neuronale Pose Interpreter Netzwerk wird ausschließlich mit synthetischen Daten trainiert und kann dennoch hochmoderne Leistungen erbringen und die Posen von Objekten in ausreichender Genauigkeit für diverse Anwendungsfälle berechnen. Für die Erstellung der synthetischen Datenmenge dient ein Plugin für die Unreal Engine 4, welches eine Augmentierung der Trainingsdaten durch die Einbindung von Objekten in randomisierte und fotorealistische Umgebungen ermöglicht.

6 Ausblick

Für umfangreiche Analysen zu dieser Pipeline sind weitere Entwicklungsarbeiten notwendig, die im Folgeprojekt zu dieser Arbeit vorgenommen werden. Für diesen Zweck soll das tiefe Pose Interpreter Netzwerk auf der Renderfarm der HAW Hamburg trainiert werden, welche für anspruchsvolle GPU-basierte Berechnungen sowie Maschine- und DeepLearning Aufgaben aufgesetzt wurde. Die Hardware der Renderfarm wurde von Matthias Nitsche und Stephan Halbritter [46] analysiert und beschrieben. Die Leistungsfähigkeit der Renderfarm ist vergleichbar mit der in [35] verwendeten Hardware und sollte somit für das Training des Netzwerkes ausreichen. Allerdings sind explorative Anpassungen der Trainingsparameter für das Training auf der Renderfarm und letztendlich der erfolgreichen Erkennung und Lagebestimmung von Objekten notwendig. Analysen hierzu sind jedoch ausstehend und somit Teil des Folgeprojektes.

Weiter ist ausstehend, inwieweit die beschriebene Pipeline für die Erkennung und Lagebestimmung von selbst definierten Objekten nutzbar ist. Hierzu gehört die Erzeugung einer eigenen Datenmenge für das Training des Netzes mithilfe des Plug-ins für die Unreal Engine und unter Berücksichtigung einer sinnvollen Augmentation der Daten (Domain Randomisierung/ fotorealistische Bilder). Interessant wäre hierbei die Verwendung der Pipeline für einzelne Bauteile von Windpark-Anlagen zur Unterstützung aktuell laufender Projekte an der HAW Hamburg. In diesem Zusammenhang könnte das langfristige Ziel eine Systemlandschaft für die Erkennung und Lagebestimmung von Objekten auf beliebigen Geräten für Augmented Reality Anwendungen sein.

Für die Verwendung eines fertig trainierten Modelles aus der Pipeline ist derzeit eine leistungsfähige Hardware notwendig. Aus diesem Grund wird in [35] eine Nvidia Titan GPU verwendet, deren Anschaffung relativ kostenintensiv ist. Hierzu liegt eine mögliche Verbesserung der Performance in der Entwicklung neuer Object-Tracking Verfahren mittels maschinellem Lernen. Diese gilt es jedoch noch zu erforschen.

[5] `https://github.com/NVlabs/Deep_Object_Pose/issues/13`, 29.05.2019

Literatur

[1] Yilmaz, Alper, Omar Javed, and Mubarak Shah. 'Object tracking: A survey.' Acm computing surveys (CSUR) 38.4 (2006): 13.

[2] Marchell Klepper 'Objekterkennung im Kontext einer Augmented-Reality- Anwendung'. Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg), 2019.

[3] Belongie, Serge, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. California Univ San Diego LA Jolla dept of computer science and engineering, 2002.

[4] Atzmon, Matan, Haggai Maron, and Yaron Lipman. 'Point Convolutional Neural Networks by Extension Operators.' arXiv preprint arXiv:1803.10091 (2018).

[5] LeCun, Yann, Fu Jie Huang, and Leon Bottou. 'Learning methods for generic object recognition with invariance to pose and lighting.' Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on. Vol. 2. IEEE, 2004.

[6] Wagstaff, Kiri, et al. 'Constrained k-means clustering with background knowledge.' ICML. Vol. 1. 2001.

[7] Yilmaz, Alper, Omar Javed, and Mubarak Shah. 'Object tracking - A survey.' Acm computing surveys (CSUR) 38.4 (2006) 13.

[8] Chao, Yu-Wei, et al. 'Learning to Detect Human-Object Interactions.' 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018.

[9] Duda, Richard O., and Peter E. Hart. Use of the Hough transformation to detect lines and curves in pictures. No. SRI-TN-36. SRI international Menlo Park CA Artificial Intelligence Center, 1971.

[10] Alvarez, Luis, Joachim Weickert, and Javier Sánchez. 'Reliable estimation of dense optical flow fields with large displacements.' International Journal of Computer Vision 39.1 (2000): 41-56.

[11] Horn, Berthold KP, and Brian G. Schunck. 'Determining optical flow.' Artificial intelligence 17.1-3 (1981): 185-203.

[12] Beauchemin, Steven S., and John L. Barron. 'The computation of optical flow.' ACM computing surveys (CSUR) 27.3 (1995): 433-466.

[13] Tomasi, Carlo, and Takeo Kanade. 'Detection and tracking of point features.' (1991).

[14] Grewal, Mohinder S. 'Kalman filtering.' International Encyclopedia of Statistical Science. Springer, Berlin, Heidelberg, 2011. 705-708.

[15] Allen, John G., Richard YD Xu, and Jesse S. Jin. 'Object tracking using camshift algorithm and multiple quantized feature spaces.' Proceedings of the Pan-Sydney area workshop on Visual information processing. Australian Computer Society, Inc., 2004.

[16] Bradski, Gary R. 'Computer vision face tracking for use in a perceptual user interface.' (1998).

[17] Bernardin, Keni, and Rainer Stiefelhagen. 'Evaluating multiple object tracking performance: the CLEAR MOT metrics.' Journal on Image and Video Processing 2008 (2008): 1.

[18] Schmidhuber, Jürgen. 'Deep learning in neural networks: An overview.' Neural networks 61 (2015): 85-117.

[19] Mayer, Stefan. 'Automatisierte Objekterkennung zur Interpretation hochauflösender Bilddaten in der Erdfernerkundung.' Humboldt-Universität zu Berlin, 2004.

[20] Gupta, Saurabh, et al. 'Learning rich features from RGB-D images for object detection and segmentation.' European Conference on Computer Vision. Springer, Cham, 2014.

[21] Murase, Hiroshi, and Rie Sakai. 'Moving object recognition in eigenspace representation: gait analysis and lip reading.' Pattern recognition letters 17.2 (1996): 155-162.

[22] Mignotte, Max. 'Segmentation by fusion of histogram-based k-means clusters in different color spaces' IEEE Transactions on image processing 17.5 (2008): 780-787.

[23] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. 'Imagenet classification with deep convolutional neural networks.' Advances in neural information processing systems. 2012.

[24] Ren, Shaoqing, et al. 'Faster R-CNN: towards real-time object detection with region proposal networks.' IEEE Transactions on Pattern Analysis & Machine Intelligence 6 (2017): 1137-1149.

[25] Beis, Jeffrey S., and David G. Lowe. 'Shape indexing using approximate nearest-neighbour search in high-dimensional spaces.' cvpr. IEEE, 1997.

[26] Hearst, Marti A., et al. 'Support vector machines.' IEEE Intelligent Systems and their applications 13.4 (1998): 18-28.

[27] Fukushima, Kunihiko, and Sei Miyake. 'Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition.' Competition and cooperation in neural nets. Springer, Berlin, Heidelberg, 1982. 267-285.

[28] Lowe, David G. 'Object recognition from local scale-invariant features.' iccv. Vol. 99. No. 2. 1999.

[29] Al-Shuwaili, Ali, and Osvaldo Simeone. 'Energy-efficient resource allocation for mobile edge computing-based augmented reality applications.' IEEE Wireless Communications Letters 6.3 (2017): 398-401.

[30] LeCun, Yann, et al. 'Gradient-based learning applied to document recognition.' Proceedings of the IEEE 86.11 (1998): 2278-2324.

[31] He, Kaiming, et al. 'Mask r-cnn.' Computer Vision (ICCV), 2017 IEEE International Conference on. IEEE, 2017.

[32] Maturana, Daniel, and Sebastian Scherer. 'Voxnet: A 3d convolutional neural network for real-time object recognition.' Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on. IEEE, 2015.

[33] Xiang, Yu, et al. 'Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes.' arXiv preprint arXiv:1711.00199 (2017).

[34] Wu, Jimmy; Zhou, Bolei; Russell, Rebecca; Kee, Vincent; Wagner, Syler; Hebert, Mitchell; Torralba, Antonio; Johnson, David: Real-Time Object Pose Estimation with Pose Interpreter Networks. In: arXiv preprint arXiv:1808.01099 (2018)

[35] Tremblay, Jonathan, et al. 'Deep object pose estimation for semantic robotic grasping of household objects.' arXiv preprint arXiv:1809.10790 (2018).

[36] Tremblay, Jonathan, Thang To, and Stan Birchfield. 'Falling Things: A Synthetic Dataset for 3D Object Detection and Pose Estimation.' Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2018.

[37] Wei, Shih-En, et al. 'Convolutional pose machines.' Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[38] Simonyan, Karen, and Andrew Zisserman. 'Very deep convolutional networks for large-scale image recognition.' arXiv preprint arXiv:1409.1556 (2014).

[39] Rad, Mahdi, and Vincent Lepetit. 'BB8: a scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth.' Proceedings of the IEEE International Conference on Computer Vision. 2017.

[40] Tekin, Bugra, Sudipta N. Sinha, and Pascal Fua. 'Real-time seamless single shot 6d object pose prediction.' Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.

[41] Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua. 'Epnp: An accurate o (n) solution to the pnp problem.' International journal of computer vision 81.2 (2009): 155.

[42] Marion, Pat, et al. 'Label fusion: A pipeline for generating ground truth labels for real rgbd data of cluttered scenes.' 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018.

[43] T. To, J. Tremblay, D. McKay, Y. Yamaguchi, K. Leung, A. Balanon, J. Cheng, and S. Birchfield. NDDS: NVIDIA deep learning dataset synthesizer, 2018. https://github.com/NVIDIA/Dataset_Synthesizer.

[44] Tremblay, Jonathan, et al. 'Training deep networks with synthetic data: Bridging the reality gap by domain randomization.' Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2018.

[45] Lin, Tsung-Yi, et al. 'Microsoft coco: Common objects in context.' European conference on computer vision. Springer, Cham, 2014.

[46] Matthias Nitsche, Stephan Halbritter 'Development of an End-to-End Deep Learning Pipeline'. Hochschule für Angewandte Wissenschaften Hamburg (HAW Hamburg), 2019.