HAW HAMBURG

BACHELORTHESIS
Oleg Janke

Visuell geführtes Pick-and-Place auf einem autonomen mobilen Roboter mithilfe von Objekterkennung

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Oleg Janke

Visuell geführtes Pick-and-Place auf einem autonomen mobilen Roboter mithilfe von Objekterkennung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stephan Pareigis
Zweitgutachter: Prof. Dr. Tim Tiedemann

Eingereicht am: 26. Juli 2023

Oleg Janke

Thema der Arbeit

Visuell geführtes Pick-and-Place auf einem autonomen mobilen Roboter mithilfe von Objekterkennung

Stichworte

Roboter, Maschinelles Lernen, Computer Vision, Deep Learning, künstliche Intelligenz, Aufnehmen und Ablegen, autonom, Objekterkennung

Kurzzusammenfassung

Da Roboter immer mehr in unseren Alltag finden und uns Arbeit abnehmen, ist es wichtig Algorithmen zu finden, die es ihnen ermöglichen, in einem dynamischen Umfeld Objekte aufzunehmen und zu platzieren. Ein anderes aktuelles Forschungsthema ist Deep Learning. Dieser Bereich wird aktuell stark erforscht und hat in den letzten Jahren große Fortschritte gemacht.
In der vorliegen Arbeit wurde ein Ansatz entwickelt, der in einem vereinfachten Szenario, das Aufnehmen und Ablegen eines Pakets umsetzt und dabei einen Objekterkenner als visuelle Unterstützung verwendet. Als Fahrplattform kommt das Unmanned Ground Vehicle namens Husky mit einem UR5 Arm von Universal Robots zum Einsatz. Die Lösung wurde sowohl für die Simulation, als auch auf dem realen Roboter entwickelt.
Als Objekterkenner wurde ein Yolov5 Modell verwendet, welches mit Hilfe von Transfer Learning trainiert wurde. Dazu wird erläutert, welche Parameter und Bilder für das Training verwendet wurden. Die Steuerung der Roboter wurde mit Hilfe einer Finite State Machine realisiert. Dabei wird erklärt, wie diese aufgebaut ist und wie die Objekterkennung in den Prozess integriert wurde.
Diese Lösung wurde dann anhand der Kriterien Zuverlässigkeit, Genauigkeit, Flexibilität und Sicherheit evaluiert. Dabei wird zunächst auf die Unterschiede zwischen der Simulation und dem realen Roboter eingegangen. Anschließend werden die drei Hauptphasen Detect, Pick und Place einzeln bewertet. Zum Schluss werden Verbesserungspotentiale aufgeführt.
Ergebnis der vorliegenden Arbeit ist, dass die Objekterkennung erfolgreich funktioniert. Jedoch werden mehr Umgebungsdaten benötigt, um einen zuverlässigen Prozess zu gewährleisten. Außerdem ist die Sim-To-Real gap ein Problem, wodurch die Ergebnisse in der Simulation nicht ganz auf den realen Roboter übertragbar sind.

Oleg Janke

Title of Thesis

Visually guided Pick-and-Place on an autonomous mobile robot using object recognition

Keywords

Robot, Machine Learning, Computer Vision, Deep Learning, Artificial Intelligence, Pick-and-Place, autonomous, Objectdetection

Abstract

Robots are taking over more and more of our work in everyday life. That is why it is important to find algorithms that enable them to pick and place objects in a dynamic environment. Another current research topic is Deep Learning. Currently, this area is being heavily researched and has made great progress in recent years.
In this work, an approach has been developed which picks and places packages in a simplified scenario. An object detector was used for visual support. The used platform is the Unmanned Ground Vehicle called Husky with an UR5 arm from Universal Robots. The solution was developed for both simulation as well as for the real robot.
Yolov5 was used as a object detector. It was trained via transfer learning. Thus, it is shown which parameters and images were used for the training. The process control of the robots is realized with the help of a Finite State Machine (FSM). It is explained how the FSM is structured and how the object recognition was integrated into the process.
This solution is then evaluated in terms of reliability, accuracy, flexibility and safety. The differences between the simulation and the real robot are discussed. Then the three main phases Detect, Pick and Place are evaluated individually.
The result of the present work is that the object recognition works successfully, but more data from the enviroment is needed to ensure a reliable process. In addition, the sim-to-real gap is a problem, which makes the results in the simulation not completely transferable to the real robot.

Inhaltsverzeichnis

Abbildungsverzeichnis **vii**

Tabellenverzeichnis **viii**

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Problemstellung . . . 1
- 1.3 Aufbau der Arbeit . . . 2

2 Grundlagen **3**
- 2.1 Robotik . . . 3
  - 2.1.1 Autonomer mobiler Roboter . . . 3
  - 2.1.2 Sensorik . . . 4
- 2.2 Objekterkennung . . . 6
  - 2.2.1 Überblick der Verfahren . . . 6
  - 2.2.2 Deep Learning . . . 6
  - 2.2.3 Convolutional Neural Networks . . . 7
- 2.3 YOLO . . . 9
- 2.4 Setup . . . 10
- 2.5 Software . . . 12
  - 2.5.1 ROS . . . 12
  - 2.5.2 Gazebo . . . 13

3 Implementierung **15**
- 3.1 Prozessablauf . . . 15
- 3.2 Finite State Machine . . . 16
  - 3.2.1 Armbewegungen . . . 17
  - 3.2.2 Objekterkennung . . . 18
  - 3.2.3 Basissteuerung . . . 19

3.2.4 Grippersteuerung 20
3.3 Objekterkennung 20
3.3.1 Datenerfassung 20
3.3.2 Annotation und Vorverarbeitung 21
3.3.3 Training 23
3.4 Kriterien 26

4 Evaluation **28**
4.1 Testumfeld 28
4.2 Auswertung 30
4.2.1 Metriken 30
4.2.2 Versuche 32
4.2.3 Phase 1: Detect 32
4.2.4 Phase 2: Pick 33
4.2.5 Phase 3: Place 34
4.3 Bewertung und Potentiale 35
4.3.1 Phase 1: Detect 35
4.3.2 Phase 2: Pick 36
4.3.3 Phase 3: Place 36
4.3.4 Verbesserungspotentiale 36

5 Zusammenfassung und Ausblick **38**
5.1 Zusammenfassung 38
5.2 Ausblick 39

Literaturverzeichnis **40**

Selbstständigkeitserklärung **43**

Abbildungsverzeichnis

2.1 Das Prinzip der Stereokamera [13] 5
2.2 Klassische vs. Deep learning Verfahren[20] 7
2.3 Beispiel für Data Augmentation [17] 8
2.4 Aufbau von Convolutional Neural Network [2] 9
2.5 Modellvergleich zwischen den aktuellen Versionen YOLOv5 bis YOLOv8 [14] 9
2.6 YOLO v5 Architecture Overview [6] 10
2.7 Der verwendete Roboter Husky beim Öffnen einer Tür. [7] 11

3.1 Ein Ablaufdiagramm des geplanten Prozesses von dem Use Case 'visuell geführtes Pick-and-Place' 16
3.2 Finite State Machine zu dem Prozessablauf aus Kapitel 3.2. 17
3.3 Die Berechnung der Mittelpunkte des Objektes im Bild. 19
3.4 Beispielaufnahmen aus Gazebo 21
3.5 Beispielaufnahmen vom realen Husky 21
3.6 Beispielbild mit Bounding Box 22
3.7 Links ist die Originalaufnahme und rechts ist die Aufnahme transformiert und enthält sowohl eine negative Helligkeitsveränderung, als auch eine horizontale Drehung der Bounding box. 22
3.8 Beispiele für Overfitting (links) und Underfitting (rechts). [3] 25

4.1 Testumgebung aus der Sicht des Huskys, wobei in rechts die Simulation und links die Realität abgebildet ist. 30
4.2 Graphen der Metriken für die Simulation und für die realen Daten. 31
4.3 Graphen der Loss-Metriken für die Simulation und für die realen Daten. . 32
4.4 Versuchsdurchlauf mit Licht (a) und ohne Licht (b) 33
4.5 Der Husky beim Versuch das Paket seitlich zu greifen. 34

Tabellenverzeichnis

2.1 Specs Computer 11

3.1 Trainingsparameter 23

1 Einleitung

1.1 Motivation

In dem Science-Fiction Film „I, Robot“ können autonome Roboter verschiedenste Aufgaben, wie zum Beispiel mit dem Hund spazieren gehen oder Mülltonnen einsammeln, erfüllen und so das Leben in der Gesellschaft erleichtern. Die Besonderheit besteht darin, dass die Roboter problemlos mit der Umwelt interagieren können. Eine der häufigsten Aufgaben in diesem Zusammenhang ist das Aufnehmen und Ablegen von Objekten, Pick-and-Place genannt. Sei es die Aufnahme einer Hundeleine oder das Wegräumen einer Mülltonne.
In der Realität sind die Roboter noch nicht so weit, aber die Entwicklung geht in diese Richtung. In der Industrie hingegen ist Pick-and-Place bereits ein gängiges Thema. Der große Unterschied ist hierbei, dass die Roboter in einer kontrollierten Umgebung arbeiten. Jedoch erobern Roboter in den letzten Jahren auch den Alltag. Ein Beispiel hierfür ist der Staubsaugerroboter oder die Auslieferungsroboter. Diese Alltagsroboter können kleine Probleme bewältigen, aber stoßen sie auf komplexe Hindernisse, kann die Aufgabe nicht mehr fortgesetzt werden oder es wird ein menschlichen Helfer benötigt, der das Hindernis beseitigt.
Um diese Probleme zu lösen, müssen die Roboter in der Lage sein, ihre Umgebung wahrzunehmen und zu verstehen. Ein Ansatz dazu ist „Deep Learning”. Deep Learning ist ein Teilbereich des maschinellen Lernens und gewinnt seit 2011 zunehmend an Relevanz.[15] Mittlerweile können schon komplexe Probleme, wie zum Beispiel das Autofahren, autonomisiert werden. [18]

1.2 Problemstellung

Wenn man diese Entwicklungen verfolgt, ergibt sich die Problemstellung, ob ein Prozess für einen Roboter entwickelt werden kann, der es ihm ermöglicht, visuell geführtes Pick-

and-Place auf einem autonomen Roboter mithilfe von Objekterkennung zu realisieren. Ein Problem besteht darin, dass der Roboter in der Lage sein muss, Objekte aus verschiedenen Perspektiven und in verschiedenen Größen zu erkennen und korrekt zu greifen. Die sensorische Wahrnehmung und die präzise Steuerung des Roboterarms sind von entscheidender Bedeutung, um eine zuverlässige Handhabung der Objekte zu gewährleisten. In dieser Arbeit wird der vereinfachte Anwendungsfall „Aufnehmen und Ablegen eines Pakets" gewählt um solch einen Prozess exemplaraisch umzusetzten und somit durch herausgerarbeitete Verbesserungspotentiale eine Grundlage für zukünftige Forschungen zu schaffen.

1.3 Aufbau der Arbeit

Zunächst werden die Grundlagen für die Arbeit erläutert. Hierbei wird auf die verwendeten Technologien und das Setup eingegangen.
Im dritten Kapitel wird das Konzept, in Form eines Flussdiagrammes, und Implementierung beschrieben. Außerdem werden in diesem Teil auch relevante Bewertungskriterien herausgearbeitet.
Das vierte Kapitel beschäftigt sich mit der Evaluation der Umsetzung. Dabei werden die Ergebnisse in Bezug auf die *Sim-To-Real gap* , die Metriken der Objekerkennung und der Versuche ausgewertet.
Im letzten Kapitel werden die Ergebnisse noch einmal mit Hinblick auf die Problemstellung zusammengefasst und ein Ausblick auf weiterführende Arbeiten gegeben.

2 Grundlagen

In diesem Kapitel werden die Grundlagen vorgestellt, welche benötigt werden, um visuell gestütztes Pick-and-Place mit einem autonomen, mobilen Roboter zu lösen. Zunächst werden die Elemente aus der Robotik dargestellt. Das daraufhin folgende Unterkapitel behandelt die Grundlagen der visuellen Unterstützung mittels Objekterkennung. Abschließend wird das verwendete Setup einschließlich der Hardware und der Software vorgestellt.

2.1 Robotik

Die Robotik beschäftigt sich mit dem Aufbau und der Steuerung von Robotern. Somit wird nachfolgend zunächst definiert, was autonome, mobile Roboter sind. Danach werden die einzelnen Sensoren beschrieben, welche für Pick-and-Place relevant sind. Es folgt das Unterkapitel, welches Pick-and-Place erläutert.

2.1.1 Autonomer mobiler Roboter

Nach Intel ist der autonome mobile Roboter (AMR) wie folgt beschrieben: „[e]in autonomer mobiler Roboter ist eine Art Roboter, der seine Umgebung erfassen und sich unabhängig durch diese Umgebung bewegen kann." [11] Die zusätzlichen Attribute autonom und unabhängig verdeutlichen, dass der Roboter „ohne Fernsteuerung, insbesondere ohne physische Kabelverbindung für Daten- oder Energieübertragung agiert" [8]. Hieraus ergibt sich somit eine essenzielle Anforderung an die vorliegende Arbeit. Um dieser Anforderung gerecht zu werden, muss der Roboter seine Umgebung erfassen können. Dafür hat er eine Vielzahl an externen Sensoren, wie zum Beispiel Kameras und Laserscanner. Zusätzlich zur Wahrnehmung des äußeren Umfelds, besitzt er interne Sensoren, um seinen eigenen Zustand zu bestimmen. Die für diese Arbeit relevanten Sensoren werden im folgenden Unterkapitel Sensorik erläutert. Diese Sensordaten werden gesammelt und

mithilfe eines Computers und einer Vielzahl von Algorithmen ausgewertet, damit der Roboter dann seine Aufgabe ausführen kann.
Zusätzlich kann es zu unerwarteten Veränderungen in der aktuellen Umgebung kommen. In solchen Fällen sollte der MAR in der Lage sein, sein Verhalten selbstständig anzupassen und entsprechend zu reagieren. Veranschaulichen lässt sich dies, wenn der Roboter bei der Aufgabe von A nach B zu fahren eine störende Person nicht überfährt, sondern seine Geschwindigkeit drosselt und eine neue Route berechnet.
Anwendungsgebiete von autonomen mobilen Robotern sind sehr vielseitig und erstrecken sich unter anderem auf Logistik, Fertigung, Einzelhandel und auf das Gesundheitswesen. Ziele des Einsatzes von AMR sind Sicherheit, Flexibilität und Effizienz. Sicherheit für Menschen soll in der Hinsicht gegeben sein, wenn schwierige oder gefährliche Aufgaben wie Bombenentschärfungen durchgeführt werden können, ohne dass ein Mensch in der Nähe sein muss. Während Industrieroboter an ihre Arbeitsumgebung gebunden sind, bringt ein autonomer mobiler Roboter die Flexibilität mit, in unterschiedlichen Umgebungen zu arbeiten und angemessen auf Veränderungen zu reagieren. Mit dem Einsatz passender Roboter können Menschen Aufgaben abgenommen werden, welche sich dann auf ihre Kernaufgaben konzentrieren können, sodass Produktivität und Effizienz gesteigert werden können. [16]

2.1.2 Sensorik

RGB-Kamera

Die klassische RGB-Kamera kann auch als das „Sehen" des Roboters bezeichnet werden. Die Größe des Sichtfeldes hängt dann von der Anzahl der Pixelsensoren auf dem Bildsensor ab und durchläuft die folgenden Schritte [9]:
1) Bei Kameras mit Autofokus wird zunächst das Bild über die Entfernungsmessung „scharf" gestellt. Sonst muss dieser Schritt manuell erreicht werden.
2) Anschließend wird der Bildsensor mit einer definierten Belichtungszeit mit Umgebungslicht, welches durch eine Linse bzw. ein Linsensystem fokussiert wird, bestrahlt.
3) Der Bildpixel wandelt die Lichtintensitäten in elektrische Signale um, die durch einen Analog-Digital-Wandler quantisiert werden.
4) Die errechneten digitalen Signale entsprechen den Helligkeiten und werden im letzten Schritt von der Kamera an den Steuerrechner übertragen.

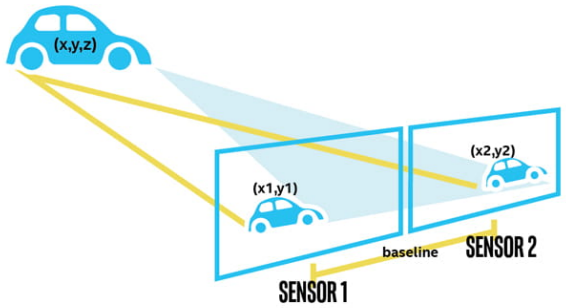


Abbildung 2.1: Das Prinzip der Stereokamera [13]

Tiefen-Kamera

Die RGB-Kamera liefert zwar ein 2D-Bild, doch benötigt man andere Verfahren, um Tiefeninformationen und somit eine 3D-Representation zu den 2D-Bildern zu bekommen. Hierzu gibt es viele gängige Verfahren wie zum Beispiel Time-of-flight-, Streifenlicht- oder Stereokameras. In dieser Arbeit soll lediglich auf die Stereokamera eingegangen werden, da dieser Tiefen-Kamera-Typ zusammen mit der RGB-Kamera verbaut ist und keine zusätzliche Hardware benötigt. Die Funktionsweise basiert, wie in Abbildung 2.1 zu sehen ist, auf zwei unterschiedlichen Sensoren, die in einem bestimmten, möglichst kleinen Abstand zueinander angebracht werden. Die Differenz d der korrespondierenden Bildpunkte (auch Disparität genannt) wird mit $d = x1 - x2$ berechnet. Kennt man nun den Abstand b zwischen den beiden Sensoren (mit baseline in der Abbildung gekennzeichnet) und den Abstand f zur Linse, kann man nach dem Strahlensatz die Distanz z wie folgt berechnen [13]:

$$\frac{z}{b} = \frac{f}{d} \Leftrightarrow z = \frac{f \cdot b}{d}$$

Diese Punkte werden als Punktwolke zusammengefügt und bilden die Ausgabe der Stereokamera.

2.2 Objekterkennung

2.2.1 Überblick der Verfahren

Damit der Roboter autonom agieren kann, muss er in der Lage, seine Zielobjekte erkennen zu können. Im Bereich der Bildverarbeitung erforscht die Objekterkennung genau diesen Anwendungsfall. Prinzipiell werden Bilder der Kamera entnommen, um dann aus ihnen weitere Informationen zu gewinnen.
Die Frage ist dann, ob sich Zielobjekte im Bild befinden und des Weiteren, welche Positionen und Orientierung die Objekte haben. Für die Objekterkennung gibt es eine Vielzahl von Vorgehensweisen, welche in drei verschiedene Kategorien gegliedert werden können[20]:
1. Traditionelle Verfahren
2. Klassische maschinelle Lernverfahren
3. Deep Learning Verfahren
Diese sind in Abbildung 2.2 schematisch dargestellt. Die beiden Verfahren a) und b) wenden im zweiten Schritt *Merkmalsextraktion* auf die Inputbilder an, um aus den gefundenen Merkmalen dann über Algorithmen (mit festen Algorithmen oder maschinellem Lernen, wie zum Beispiel Klassifikatoren) einen Output zu generieren. Das „Deep Learning“ Verfahren c) nutzt keinen festen Merkmalsextraktionsalgorithmus, sondern erlernt die Merkmale anhand seiner Trainingsdaten.

2.2.2 Deep Learning

Deep Learning (DL) ist ein Unterfeld des Machine Learnings. Die zugehörigen Neuronalen Netze werden auch als Deep Neural Networks (DNNs) bezeichnet und bestehen aus mindestens drei Schichten: Input-, Hidden- und Outputlayer. Die Anzahl dieser Schichten wird als Tiefe des Netzwerks bezeichnet, wobei es mehrere Hiddenlayer gibt. Diese Netzwerke versuchen, das menschliche Gehirn nachzuahmen und aus großen Datenmengen zu lernen. [10]
Wie schon im Vergleich zu den traditionellen Verfahren erwähnt, benötigt DL keine Merkmalsextraktion, sondern lernt diese automatisch aus den Trainingsdaten. Ein solches Trainingsdatenset ist zum Lernen essenziell und muss zunächst erstellt werden.
Dafür ist vorerst kein Expertenwissen nötig, sondern eine große Menge an Daten, die

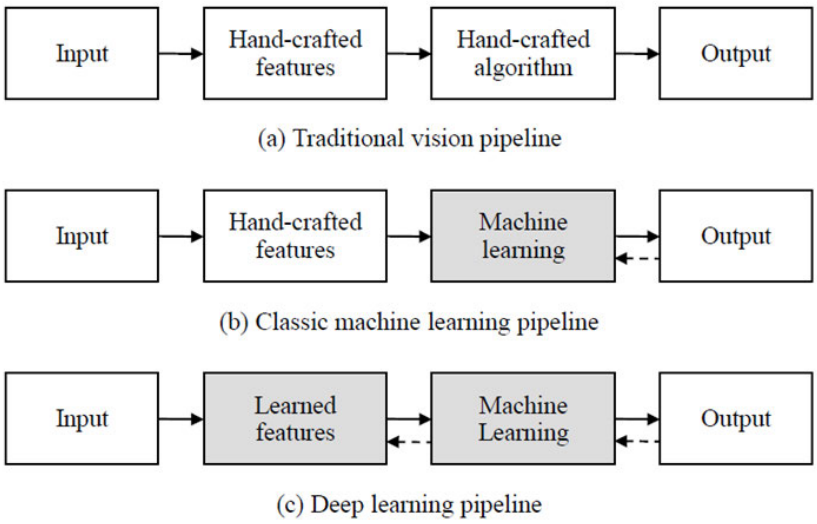


Abbildung 2.2: Klassische vs. Deep learning Verfahren[20]

das zu erkennende Objekt in verschiedenen Kontexten zeigt. Da das Erstellen von realen Trainingsdaten sehr aufwendig sein kann und die Datenmenge begrenzt ist, kann auf Data Augmentation zurückgegriffen werden. Dabei werden die vorhandenen Daten durch verschiedene Techniken verändert, um neue Daten zu generieren. In Abbildung 2.3 sind zum Beispiel sechs verschiedene Augmentationstechniken mit fünf Originalbildern dargestellt (von links nach rechts): Ohne Augmentation, Rotation, Verwischung, Kontraständerung, Skalierung, Helligkeitsveränderung und Projektion. Durch die Erhöhung der Datenmenge kann Overfitting, also eine Überanpassung an die vorhandenen Daten, vorgebeugt werden, da das Netzwerk mehr Daten zum Lernen hat. [1]
Für das Training selbst gibt es verschiedene Netzarchitekturen. Ein prototypisches Beispiel für Deep Learning ist das Convolutional Neural Network (CNN), welches im nächsten Abschnitt genauer erläutert wird.

2.2.3 Convolutional Neural Networks

Ein CNN besteht aus folgenden Schichten [10]:

Convolutional-Layer

Die Aufgabe des Convolutional-Layers ist es, die Eingabe zu filtern und die wichtigsten Informationen zu extrahieren. Dabei wird ein Filterkernel über die Eingabe

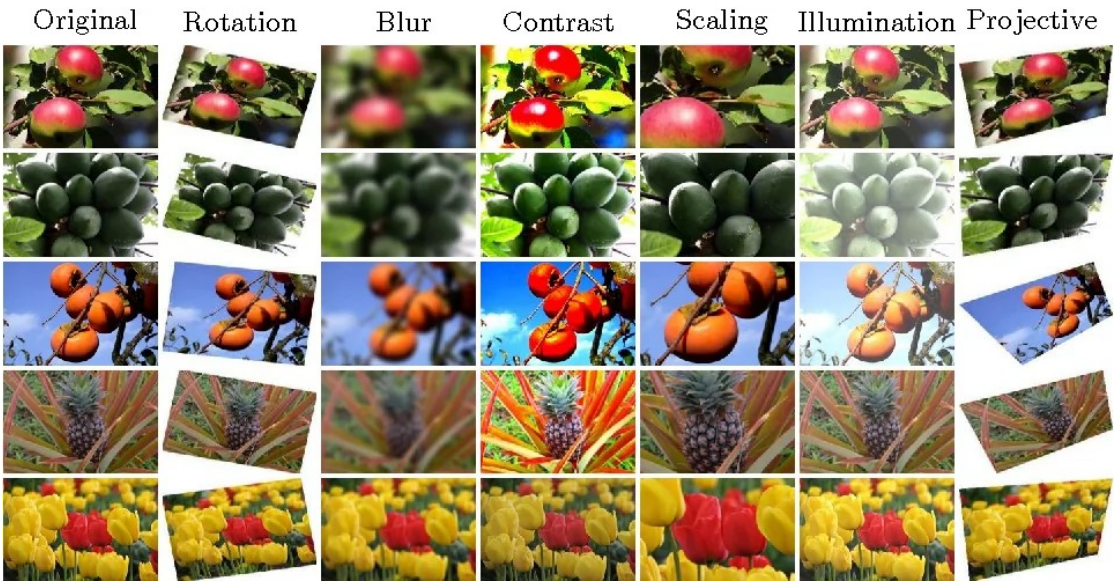


Abbildung 2.3: Beispiel für Data Augmentation [17]

geschoben und die Werte werden miteinander multipliziert und aufsummiert. Hierbei handelt es sich um die Faltungsoperation. Die Ergebnisse werden dann jeweils mit dem Relu (Rectified Linear Unit) aktiviert und in einer Activation Map gespeichert.

Pooling-Layer

Die Aufgabe des Pooling-Layers ist es, die Dimensionalität der Eingabe zu reduzieren (auch downsampling genannt), da nicht alle Informationen benötigt werden. Dafür wird ein Filterkernel über die Activation Map geschoben und die Werte werden zusammengefasst. Die Ergebnisse werden dann in einer neuen Activation Map gespeichert. Eine typische Funktion ist das Max Pooling, wobei nur der höchste Wert des Filters übernommen wird.

Fully-Connected-Layer

Die Aufgabe des Fully-Connected-Layers ist es, die Klassifikation durchzuführen. Dafür werden die Ergebnisse des vorangegangenen Layers in einen Vektor umgewandelt und mit einem Klassifikator verglichen.

Die Abbildung 2.4 zeigt einen beispielhaften Aufbau eines CNNs. Die Eingabe ist ein Bild. Dann folgen jeweils zweimal abwechselnd ein Convolutional- und Pooling-Layer.

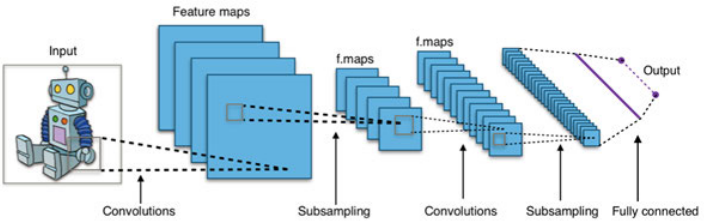


Abbildung 2.4: Aufbau von Convolutional Neural Network [2]

Zum Schluss folgt ein Fully-Connected-Layer, welcher die Klassifikation durchführt und einen Output generiert.

2.3 YOLO

Der Objekterkenner YOLO baut auf einem CNN auf und steht für „You only look once", was bedeutet, dass das Netzwerk nur einmal über das Bild schaut (auch Single Shot Detector genannt). Dieser liefert im Gegensatz zu einem Two Stage Detector, wie Fast RCNN oder Faster RCNN, eine deutlich höhere Performance, da die Objekterkennung und Klassifikation in einem Schritt durchgeführt wird. [4]
Die aktuellen Versionen von Yolo sind in der Abbildung 2.5 zusehen, wobei der linke Graph die Größe des Netzes durch die Anzahl der Parameter und der rechte Graph die Performance darstellt. Die einzelnen Punkte in den Graphen stellen die verschiedenen

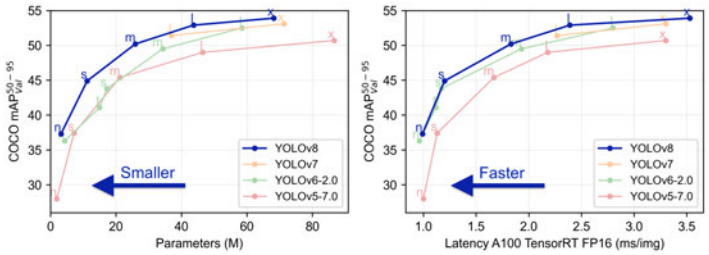


Abbildung 2.5: Modellvergleich zwischen den aktuellen Versionen YOLOv5 bis YOLOv8 [14]

Größen dar, so ist nano (n) die kleinste und xlarge (x) die größte Version. Der Aufbau von Yolo ist in Abbildung 2.6 dargestellt, wobei das Backbone ein CNN namens CSPDarknet ist, welches die Eingabe verarbeitet und die Feature Maps an das Neck weitergibt. Das

Neck bildet ein PANet und besteht aus vier Convolutional-Layern und zwei Pooling-Layern, die die Feature Maps weiterverarbeiten und an die Head-Layer weitergeben. Der Head besteht aus einer finalen Convolutional-Layer (mit YOLO-Layer bezeichnet) und generiert den Output.

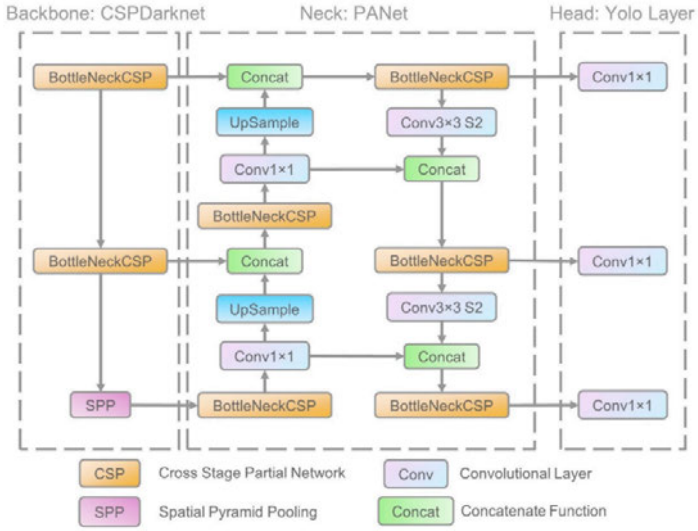


Abbildung 2.6: YOLO v5 Architecture Overview [6]

2.4 Setup

Basis dieser Arbeit ist die mittelgroße Roboterplattform namens „Husky" der Firma Clearpath, welcher in Bild 2.7 abgebildet ist. Der Husky wird im Folgenden mit dem Begriff „Roboter" referenziert, sofern nicht anders angemerkt.
Dieses „Unmanned Ground Vehicle" ist Teil des interdisziplinären Forschungsprojektes" Testfeld Intelligente Quartiersmobilität", welches diesen Roboter in solchen „Quartieren" zurechtfinden lassen soll. Quartiere können in diesem Rahmen beispielsweise eine Fußgängerzone, ein Uni-Campus oder ein Park mit einem Radius von bis zu drei Kilometern sein. [7] Der Roboter ist mit seinen vier Geländereifen für verschiedene Untergründe gewappnet und lässt sich durch seine Modularität mit vielen Aktoren und Sensoren erweitern, welche im Folgenden erläutert, werden.

Kern ist ein kleiner, austauschbarer Mini-Computer, welcher, wie in Tabelle 2.1 zu sehen, mit einer leistungsstarken GPU und CPU ausgestattet ist, die ihm dabei helfen,

Abbildung 2.7: Der verwendete Roboter Husky beim Öffnen einer Tür. [7]

auch ressourcenintensivere Algorithmen ausgeführt werden können. Außerdem sind eine WLAN-Karte, HDMI- und USB Anschlüsse verbaut, womit eine Steuerung von außen, zum Beispiel durch Tastatur und Bildschirm oder über eine Remoteverbindung, gewährleistet werden kann.

Tabelle 2.1: Specs Computer

CPU	Ryzen 5600G
GPU	Nvidea RTX 3060
WLAN	Wifi 6
Anschlüsse	6 USB, HDMI

Ebenso wichtig wie die Leistungsfähigkeit des Computers sind die Aktoren und Sensoren des Huskys. Der primäre Aktor für „Pick-and-Place" ist der Manipulator, welcher bei dem vorliegenden Setup aus einem Greifarm und einem Gripper besteht. Der Greifarm ist ein UR5 von Universal Robots und hat sechs Freiheitsgrade und eine Traglast von bis zu fünf Kilogramm. Der UR5-Arm kann sowohl über den verbundenen Computer, als auch über ein direkt verbundenes Tablet gesteuert werden. Der Gripper an der Spitze ist ein flexibler 2-Finger Greifer der Firma OnRobot, welcher das Greifen ausführt.

Kurz hinter dem Greifer ist eine Stereokamera „Intel Realsense d435" verbaut. Diese kann neben den RGB-Bildern auch Tiefenbilder und RGB-Punktwolken liefern. Die Tiefenbilder haben eine Auflösung von bis zu 1280 × 720 Pixel und die Bildwiederholrate für Tiefenbilder geht bis zu 90 fps. RGB-Bilder haben eine Auflösung von 1920 × 1080 Pixel und eine Bildwiederholrate von bis zu 30 fps. [12]

Weiterführend sind noch andere Sensoren wie, Lidar oder IMU verbaut, die aber nicht weiter erläutert werden, da sie nur eine passive Rolle für diese Arbeit spielen.

2.5 Software

In diesem Abschnitt wird die Software des Roboters erläutert, welche für die Ausführung der Algorithmen, der Steuerung und der Simulation des Roboters zuständig ist.

2.5.1 ROS

Der Husky arbeitet mit dem Open-Source Framework „Robot Operating System", kurz ROS, welches viele verschiedene Tools einschließlich Hardwareabstraktion, häufig verwendete Funktionen und die Kommunikation zwischen Prozessen für Roboter, zur Verfügung stellt. [5].
Ausgangspunkt eines ROS Systems ist der ROS Master, bei dem Prozesse (sogenannte Nodes) und Topics registriert werden. Die Topics können wie im Publish-Subscribe-Modell von den einzelnen Nodes abonniert werden. Gleichzeitig können Prozesse Nachrichten auf Topics publishen, welche dann vom Master an die jeweiligen Subscriber verteilt werden. Die jeweiligen Komponenten können dann in wiederverwendbaren Packages zusammengefasst werden. Außerdem ermöglicht eine solche Struktur die Verwendung von verschiedenen Programmiersprachen.
Von besonderer Wichtigkeit für diese Ausarbeitung sind folgende drei Pakete: move_base, MoveIt und Smach.

move_base

Das Paket move_base ist für die Navigation des Roboters zuständig. Hierfür implementiert jener einen Actionserver, welcher Ziele auf der Topic „move_base_simple/goal" Ziele als Pose bekommt und dann mit Hilfe von den Sensordaten und einer optionalen Karte, den Navigationstack des Roboters ansteuert, um das gegebene Ziel zu erreichen. Sensordaten können in diesem Fall beispielsweise Lidar, Tiefenkamera oder Laserscanner sein. Im Zusammenhang dieser Thesis wird keine globale Karte verwendet, sodass die Navigation allein auf der „local costmap" basiert, welche auf Basis der genannten Sensordaten

erzeugt wird. Nachteil ist hierbei, dass Ziele, die außerhalb der Reichweite des Roboters sind, nicht zuverlässig erreicht werden können.

MoveIt

Die Steuerung des UR5-Arms wird durch das ROS Package MoveIt übernommen. Dabei übernimmt MoveIt zwei wesentliche Aufgaben, nämlich die Erstellung eines Motion Plans und die Ausführung dieses Plans. Die Berechnungen der Kinemetik und Inversenkinematik, einschließlich der Kollisionserkennung, sind bei der Planung und Ausführung automatisch integriert. Ziele können je nach Anwendungsfall entweder als Zielpose, als Kartesischer Pfad oder als Winkelangaben der einzelnen Joints angegeben werden.

Smach

Smach ist eine Python Bibliothek für die Erstellung von „hierarchical state machines", also hierarchische endliche Automaten, und wurde für die Erstellung von komplexem Roboterverhalten entwickelt. Mithilfe dieses Pakets werden die Abläufe der einzelnen Prozesse dieser Arbeit umgesetzt. Zusätzlich zu klassischen Zuständen sind in diesem Framework auch ROS-spezifische States integriert, wie zum Beispiel der MonitorState oder auch der SimpleActionState. Dadurch lassen sich Interaktionen mit beispielsweise dem Actionserver von move_base ohne viel Aufwand in die State Machine integrieren.

2.5.2 Gazebo

Zusätzlich zu der realen Hardware wurde die Simulationssoftware Gazebo verwendet. Diese schafft eine Umgebung, die sowohl die einzelnen Hardwarekomponenten des Huskys als auch Testobjekte, wie Tische und Pakete, virtuell darstellen kann.
In diesem Testszenario wurde ein vereinfachtes 3D-Modell des Huskys eingebunden, bei dem Aktoren und Sensoren grob abgebildet werden, aber beispielsweise einzelne Kabel oder die Einbuchtung, in welcher der Rechner liegt, fehlen. Hinzu kommt das Modell eines Tisches, auf dem die Objekte erhöht platziert werden können und das 3D-Modell eines Paketes, welches mit verschiedenen Größen vervielfacht wurde, um mehrere Szenarien erzeugen zu können. Die Algorithmen werden zunächst in Gazebo getestet, bevor sie auf der realen Hardware angewendet wurden. Verbunden wird diese Simulation mit dem

ROS-System des Huskys über das ROS-Paket „gazebo_ros_pkgs", welches die Kommunikation zwischen den beiden Programmen ermöglicht.

3 Implementierung

In diesem Kapitel soll zunächst der Ablauf von „visuell geführtem Pick-and-Place auf einem autonomen, mobilen Roboter" durch ein Flussdiagram modelliert werden. Auf Basis dieses Flussdiagramms wird eine FSM entwickelt, die die Steuerung des Prozessablaufs übernimmt. Im Anschluss wird die Objekterkennung erläutert, welche die visuelle Führung bereitstellen soll. Danach werden die einzelnen Nodes beschrieben, die durch die FSM angesteuert werden. Abschließend werden die Kriterien aufgestellt, anhand welcher die Implementierung in dem Kapitel 4 Evaluation bewertet wird.

3.1 Prozessablauf

Um den gesamten Prozess modellieren zu können, wurde dieser in einzelne Arbeitsschritte aufgeteilt. Diese lassen sich aus dem Flussidagramm in Abbildung 3.1 entnehmen.
Zunächst soll der Roboter in seine Ausgangslage versetzt werden, um vor jedem Durchlauf die gleichen Startbedingungen aufzuweisen. Nachdem diese Position erreicht wurde, geht der Prozess in die Schlüsselkomponente „Erkenne Objekt". Hier wird die Verarbeitung der Bilder der Realsense durch einen YOLOv5-Objekterkenner realisiert. Sollte aus der Ausgangssituation das Zielobjekt nicht erkannt worden sein, wird die Position des Roboters in der nächsten Aktivität verändert und dann erneut die Objekterkennung gestartet. Sollte ein Objekt erkannt worden sein, geht der Prozess weiter und berechnet mit Hilfe der Stereokamera die Distanz zum Objekt.
Im nächsten Schritt wird evaluiert, ob sich das Ziel in Griffweite befindet. Sollte dem nicht so sein, soll der Roboter sich in Richtung des Zielobjektes bewegen und die Distanz zum Ziel so verringern. In diesem Fall beginnt der Prozess erneut bei der Objekterkennung. Für den Fall, dass sich das Objekt in Griffweite des Arms befindet, soll erst der Greifer geöffnet werden und anschließend ein Plan zum Greifen des Zielobjekts erstellt werden. Da dieses Prozedere nicht immer erfolgreich durchgeführt werden kann, wird diese Aktivität bei Misserfolg wiederholt. Anschließend wird der Plan ausgeführt und

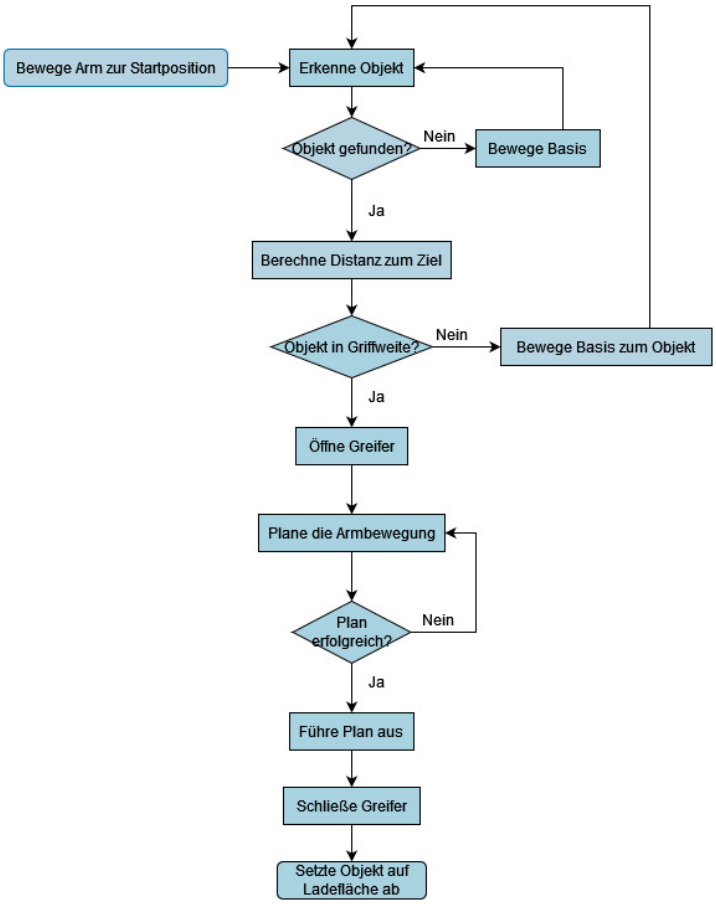


Abbildung 3.1: Ein Ablaufdiagramm des geplanten Prozesses von dem Use Case 'visuell geführtes Pick-and-Place'

der Greifer geschlossen. Im letzten Schritt soll das Objekt auf der Ladefläche des Huskys abgelegt werden.

3.2 Finite State Machine

Das Flussdiagramm aus dem vorhergehenden Kapitel bildet die Grundlage für die Entwicklung der dazugehörigen Finite State Machine (FSM). Von dem in Kapitel 2.5 beschriebenen Framework Smach ausgehend, wurden aus den Aktivitäten des Flussdiagrammes, Zustände für die FSM gebildet. Dementsprechend ist diese in Abbidlung 3.2 zu sehen. Der ganze Prozess ist in drei Phasen aufgeteilt, die jeweils einen Teil des Pro-

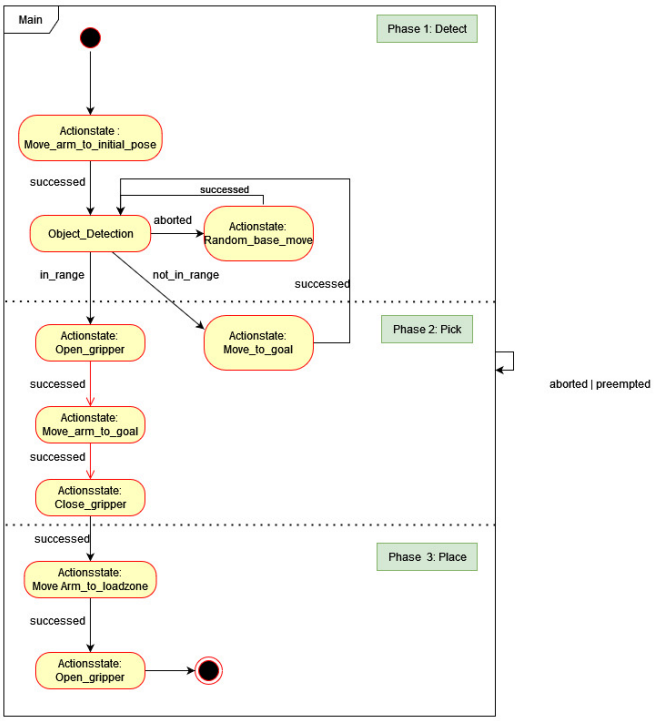


Abbildung 3.2: Finite State Machine zu dem Prozessablauf aus Kapitel 3.2.

zesses abdecken. Die erste Phase „Detect" ist für die Erkennung und Ansteuerung des Ziels zuständing, die zweite Phase „Pick" für das Greifen des Objekts und die dritte Phase„Place" für das Ablegen des Objekts zuständg.
Dabei wurden die Zustände, welche auf dem SimpleActionClient basieren, mit dem Präfix Actionstate markiert. Zu den drei grundlegenden Übergängen („successed", „aborted", „preempted") des SimpleActionClients wurden die beiden Übergänge („in_rage", „not_in_range") hinzugefügt. Ziel dieser Änderung ist es, die Ausgabe der Realsense, also Farbbild und Stereobild, an einem Punkt zu verarbeiten und somit die beiden Aktivitäten „Erkenne Objekt" und „Berechne Distanz zum Ziel" zu vereinen. In den nachfolgenden Unterkapiteln werden die einzelnen Zustände der FSM erläutert.

3.2.1 Armbewegungen

Für die Armbewegungen wurde ein Actionstate entwickelt, der je nach aktuellem Status eine andere Bewegung des UR5 auslöst. Dabei wird entweder ein Kommando oder eine Pose an den Actionserver gesendet. Im Status "Move_arm_to_initial_pose" wird

das Kommando *init* und im Status "Move_arm_to_loadzone" das Kommando *place* verwendet. Bei einem Kommando werden in der Armsteuerungsnode vordefinierte Gelenkwinkel an den moveit_commander gesendet, der daraufhin die Bewegung des Arms ausführt.
Wird eine Pose an den Actionserver übergeben, beginnt die Planung des kartesischen Pfades. Dafür ist vorausgesetzt, dass die Zielpose im Frame „base_link" vorliegt. Ist dies der Fall, wird die Pose zunächst der Planung hinzugefügt. Anschließend wird die eigentliche Planung ausgeführt. War die Planung erfolgreich, wird der Plan vom moveit_commander umgesetzt und der Wert „succeeded" wird zurückgegeben. Für den Fall, dass die Planung fehlschlägt, schlägt auch der Actionstate fehl.

3.2.2 Objekterkennung

In diesem Zustand werden die beiden Topics der Realsense entnommen und sobald ein Farbbild vorliegt, wird dieses an das neuronale Netz übergeben. Nach dem die Inferenz abgeschlossen ist, wird überprüft, ob ein Objekt erkannt wurde. Sollte keines erkannt worden sein, wird diese Node beendet und ohne Ziel an die FSM übergeben. Ist ein Objekt erkannt worden, wird anhand der Koordinaten der Bounding Box der Mittelpunkt des Objektes im Bild berechnet und die Distanz zusammen mit dem Stereobild ermittelt. Zusätzlich werden die beiden Punkte P_r und P_l berechnet, um die Breite des Objektes ermitteln zu können, wie im Abbildung 3.3 zu sehen ist. Die berechneten Koordinaten entsprechenden Pixelkoordinaten des Objektes im Bild. Um diese in den 3D-Raum zu transformieren, werden die Koordinaten zunächst in den Frame „base_link" transformiert. Dafür wird die bereits bestehende Funktion „rs2_deproject_pixel_to_point" des Realsense-Frameworks verwendet. Anzumerken ist, dass sich hierbei die Koordinaten durch die Frametransformation wie folgt ändern: x = z, y= -x und z = y. Für die Berechnung der Distanz zweier Punkte im 3D-Raum ist wie folgt definiert [19]:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

- (x_1, y_1, z_1) repräsentiert die Koordinaten des ersten 3D Punktes.
- (x_2, y_2, z_2) repräsentiert die Koordinaten des zweiten 3D Punktes.

Nun wird die Distanz der beiden Mittelpunkte P_r und P_l berechnet und somit die Breite der Bounding Box des Objekts bestimmt. Zusammenfassend gibt dieser Status bei Er-

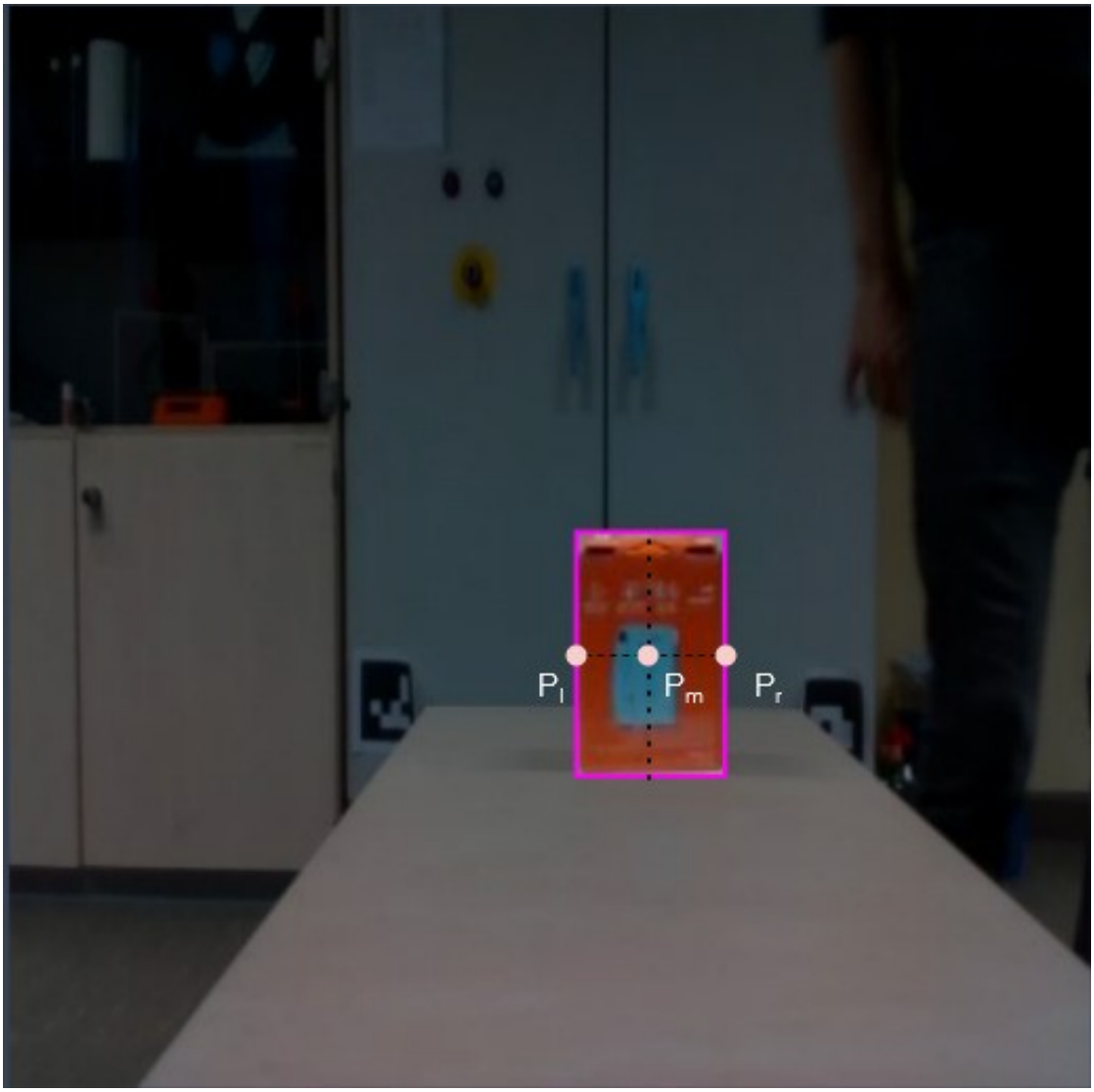


Abbildung 3.3: Die Berechnung der Mittelpunkte des Objektes im Bild.

folg die Breite der Bounding Box und den Mittelpunkt des Ziels im „base_link" Frame wieder.

3.2.3 Basissteuerung

Die Steuerung wird über die „move_base"-Node geregelt. Im Status „Move_base_to_-goal" wird das Ziel aus der Objekterkennung in den „odom"-Frame umgewandelt. Dabei

wird die Konstante *grip_ distance* von der X-Koordinate abgezogen, um einen Abstand zum Ziel zu halten, damit der Manipulator Bewegungsraum hat. „Move_base" erwartet eine Transformation in den „odom"-Frame, so werden die Zielkoordinaten über den Ros-Transformlistener transformiert und dann übergeben. Im Status „Random_base_movement" wird eine zufällige Drehung entlang der y-Achse ausgeführt.

3.2.4 Grippersteuerung

Die Gripperzustände erwarten ein Grippergoal. Dieses ist im Frame „odom" und hat einen Parameter für die Breite in cm. So wird zum Schließen des Grippers ein Ziel mit 0 cm und zum Öffnen des Grippers die maximale Breite übergeben. Im Greifprozess wird dem Grippergoal die Breite des Ziels minus einer Druckkonstante, beispielsweise 1 cm, übergeben. Die Druckkonstante wird verwendet, damit das Objekt nicht wegrutscht.

3.3 Objekterkennung

Wie in der FSM zu sehen ist, ist die Objekterkennung ein essentieller Bestandteil des Prozesses, da sie die visuelle Führung übernimmt. Um diese Aufgabe zu meistern, wurden hierfür verschiedene Datensätze erstellt und mit Hilfe dieser neurale Netzte trainiert. Die einzelnen Arbeitsschritte für die Erstellung der verwendeten Objekterkenner werden in den folgenden Unterkapiteln erläutert.

3.3.1 Datenerfassung

Bevor man mit dem Training beginnen kann, benötigt man zunächst ein Datenset. Da Pakete in dieser Arbeit das Zielobjekt sind, wurde eine Vielzahl an Bildern von den, für das Greifen ausgewählten Paketen, sowohl aus der Simulation als auch aus der realen Welt erstellt. Für die Simulation wurde ein Modell eines klassischen Kartons verwendet. In der Realität wurde ein orangefarbener Karton einer Instax-Kamera ausgewählt, da er sich, im Gegensatz zum klassischen braunen Karton, gut vom Hintergrund des HAW-Labors abhebt.
Für die Aufnahme der Bilder wurde der UR5-Arm in die Ausgangslage gesetzt und die Basis des Huskys manuell durch den Raum bewegt. Währenddessen wurde die Topic der Realsense für Farbbilder abonniert und ankommende Bilder gespeichert. Beispielhaft

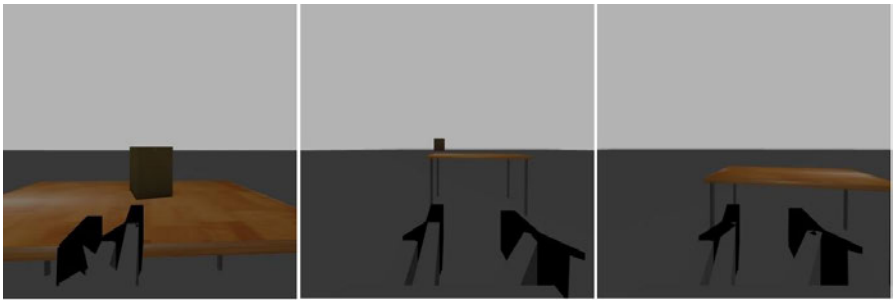


Abbildung 3.4: Beispielaufnahmen aus Gazebo

Abbildung 3.5: Beispielaufnahmen vom realen Husky

sind einige Aufnahmen in Abbildung 3.4 für die Simulation und in Abbildung 3.5 für die Aufnahmen aus der Realität zu sehen. Bilder wurden in verschiedenen Perspektiven aufgenommen, wobei das Ziel nicht immer zu sehen ist (vgl. Abbildung 3.4 rechts und Bild 3.5 rechts). In der Mitte von Abbildung 3.4 und 3.5 ist die Distanz zum Paket so gering, dass man das Paket nicht mehr komplett sehen, aber noch erkennen kann. Auf der Rechten Seite von Abbildung 3.4 bzw. 3.5 ist das Zielpaket nicht da bzw. nur noch über die Farbe zu erahnen. Insgesamt wurden 587 Aufnahmen in der Simulation und 1686 Aufnahmen von dem echten Husky gemacht.

3.3.2 Annotation und Vorverarbeitung

Das vollständige Datenset wurde zunächst ausgedünnt, da identische oder auch unbrauchbare Bilder aussortiert wurden, um die Qualität des Objekterkenners nicht zu gefährden. Außerdem wurde die Sammlung der Bilder in drei Teilmengen aufgeteilt: Training („train), Testen („test") und Validierung („validate "). Zusätzlich wurden die Bilder mithilfe des Tools Roboflow gelabelt und vorverarbeitet. Beim Labeln wurden Bounding Boxen um das Zielobjekt herum gezeichnet. Die Koordinaten der Eckpunkte dieser Bo-

xen wurden dann in einem zum neuralen Netz passenden Format, wie txt oder json, gespeichert. Im Rahmen der Vorverarbeitung wurde die Größe der Bilder durch Roboflow

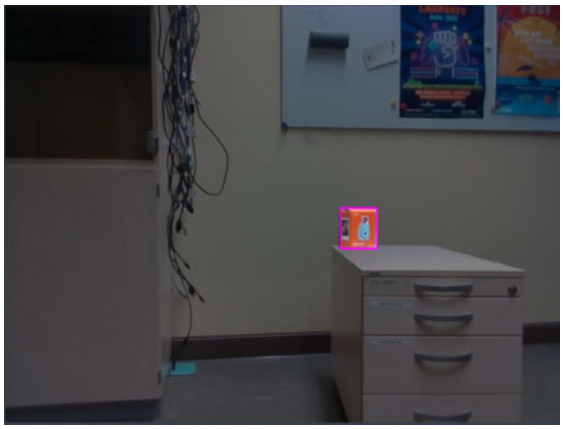


Abbildung 3.6: Beispielbild mit Bounding Box

auf 416px mal 416px geändert. Außerdem wurden verschiedene Bildaugmentationen auf die einzelnen Bilder angewendet, um im Vorfeld ein größeres und variableres Datenset zu generieren, wie zum Beispiel an Abbildung 3.7 zu sehen ist. Hier ist das erste Bild die originale Aufnahme und das zweite Bild zeigt dieselbe Aufnahme mit angewendeter Augmentation. Folgende Transformationen wurden auf die jeweiligen Abbildungen der

Abbildung 3.7: Links ist die Originalaufnahme und rechts ist die Aufnahme transformiert und enthält sowohl eine negative Helligkeitsveränderung, als auch eine horizontale Drehung der Bounding box.

Realität und der Simulation angewendet:
- eine 50-prozentige Wahrscheinlichkeit zum horizontalen Drehen der Bounding box
- eine 50-prozentige Wahrscheinlichkeit zum vertikalen Drehen der Bounding box
- zufällige Helligkeitsveränderung zwischen -15 und +15 Prozent
- ein Zoom zwischen 0 und 15 Prozent in das Bild hinein

3.3.3 Training

Trainiert wurde das Modell YOLOv5 jeweils mit zwei verschiedenen Datensätzen mittels Transferlearning, das heißt es wurde ein bereits trainiertes Modell verwendet und man trainiert nur die Featureextraktionsschicht neu. Dies geschah einmal mit dem Datenset aus der Simulation und einmal mit dem Datenset aus der Realität. Die Trainingsparameter sind in Tabelle 3.1 zu sehen. Dabei hat das das Framework Pytorch verschiedene Metriken automatisch während des Trainingsprozesses erstellt. Die für diese Arbeit relevanten Metriken werden in den folgenden Unterkapiteln erläutert.

Tabelle 3.1: Trainingsparameter

Parameter	Simulation	Realität
Epochen	20	40
batch size	10	10
Lernrate	0.01	0.01
Bildgröße	416p x 416px	416p x 416px
Anzahl der Klassen	1	1
Anzahl der gesamten Bilder	1433	1144
Anzahl der Testbilder	60	68
Anzahl der Validierungsbilder	120	134

Precision

Die *Precision* ist eine Metrik, die angibt, wie viele der vorhergesagten Objekte einer Klasse tatsächlich das Zielobjekt zeigen. Sie wird berechnet mit der Formel [2]:

$$Precision = \frac{TP}{TP + FP}$$

- TP steht für True Positive und gibt die Anzahl der richtig vorhergesagten Objekte an.
- FP steht für False Positive und gibt die Anzahl der falsch vorhergesagten Objekte an.

Recall

Der *Recall*, auch *Sensitivity* genannt, ist eine Metrik, die angibt, wie viele der tatsächlichen Objekte richtig vorhergesagt wurden. Sie wird berechnet mit der Formel [2]:

$$Recall = \frac{TP}{TP + FN}$$

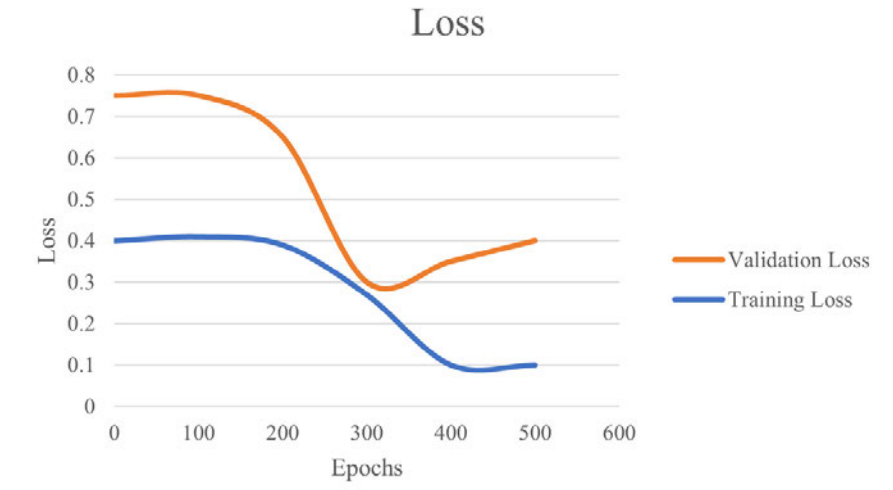
- TP steht für True Positive und gibt die Anzahl der richtig vorhergesagten Objekte an.
- FN steht für False Negative und gibt die Anzahl der falsch vorhergesagten Objekte an.

Loss

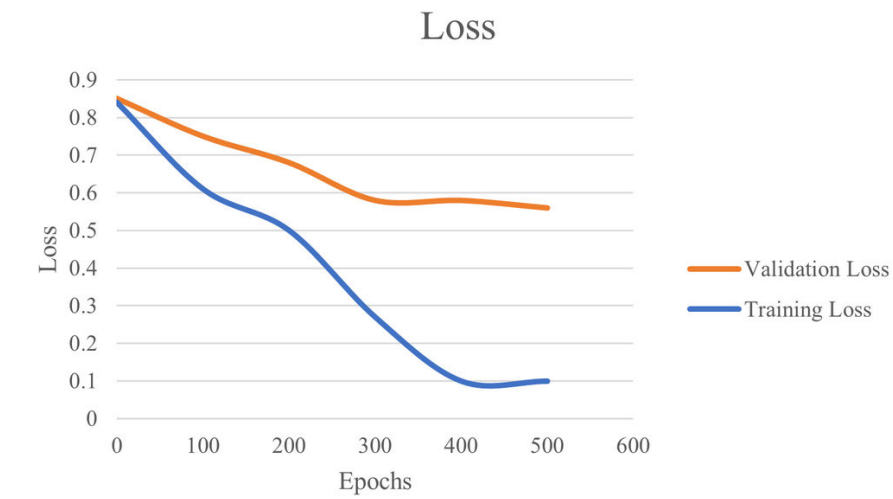
Die beiden Loss-Metriken *Training Loss* und *Validation Loss* geben an, wie gut sich das Modell an die Trainingsdaten für den Training Loss und an die Validierungsdaten für den Validation Loss angepasst hat. Diese werden bei pytorch automatisch berechnet und werden nochmal in den Boxloss für die Bounding Box und den Objektloss für die Objekterkennung unterteilt. Wenn man den Training Loss und den Validation Loss in einem Graphen zusammenfügt kann man die Over- und Underfitting-Phasen erkennen.
Overfitting heißt, dass das Netz die Trainingsdaten bereits in dem Maß gelernt hat, sodass es nicht mehr generalisieren kann.
Underfitting heißt, dass das Netz nicht gut genug trainiert hat. In der Abbildung 3.8 sind jeweils ein Beispiel für einen Graphen mit Overfitting und einen mit Underfitting zu sehen. Bei Underfitting stagniert der Validation Loss viel früher als der Training Loss. Bei Overfitting ist der Validation Loss viel höher als der Training Loss. [3]

mean Average Precision

Die *mean Average Precision* (mAP) und ist eine Metrik, die angibt, wie gut das Modell Objekte erkennt. Dabei wird die *Precision* für einen oder mehrere Schwellenwerte berechnet und über diese dann ein arithmetischer Mittelwert gebildet. Ein Schwellenwert basiert auf der *Intersection over Union* (IoU) Metrik. Somit ist der Schwellwert 0.5 bei der Metrik mAP0.5. Bei dem typischen Metrik mAP0.5:0.95 des Coco-Datensatzes startet der Schwellenwert bei 0.5 und wird in 0.05er Schritten bis 0.95 erhöht. Die Berechnung



(a)



(b)

Abbildung 3.8: Beispiele für Overfitting (links) und Underfitting (rechts). [3]

der mAP erfolgt mit der Formel [21]:

$$mAP = \frac{1}{n} \sum_{i=1}^{n} Precision_i$$

3.4 Kriterien

Um die Implementierung im folgenden Kapitel bewerten zu können, werden folgende Bewertungskriterien definiert.

Zuverlässigkeit

Die Zuverlässigkeit ist das wichtigste Kriterium für diese Implementierung, da in erster Linie die Funktionalität dieses Prototypen entscheidend ist. Ziel ist es, dass der Prozess, über alle Phasen hinweg konsistent funktioniert und dabei autonom agiert. Dabei sollen jedoch nur Szenarien betrachtet werden, die vorher definiert wurden. So soll der Roboter beispielsweise nur jeweils ein Objekt erkennen, das im Trainingssatz enthalten ist und sich auf Tischhöhe befindet. Für Szenarien mit mehreren Objekten oder solchen Objekten, die sich auf dem Boden befinden, ist das System nicht ausgelegt.

Genauigkeit

Unter Genauigkeit versteht sich, dass der Roboter möglichst genau und ohne viele Fehlgriffe das Ziel ergreifen kann.

Flexibilität

Ein flexibler Prozess soll in diesem Zusammenhang bedeuten, dass das Zielobjekt aus verschiedenen Positionen und Perspektiven erkannt und folglich auch ergriffen werden kann. Die Erkennung von Paketen in verschiedenen Formen und Größen wird als Kann-Kriterium angesehen, da diese stark vom verfügbaren Datensatz für das Training des neuronalen Netzes abhängt.

Sicherheit

Sicherheit ist für diesen Prozess ein wichtiges Kriterium, da der Roboter keine Personen oder Gegenstände beschädigen soll. Eingeschränkt wird diese Anforderung dahingehend, dass der Roboter nicht flexibel auf seine Umgebung reagieren soll, sondern nur in einem vorher definierten, statischen Szenario agiert.

Geschwindigkeit

Die Geschwindigkeit der Implementierung kann zwar relevant sein, jedoch stehen Zuverlässigkeit und Genauigkeit im Vordergrund. Daher wird Geschwindigkeit nur als optional angesehen und erstmal nicht betrachtet.

4 Evaluation

In diesem Kapitel wird aufgeführt, inwieweit die Implementierung aus dem vorhergehenden Abschnitt einen Prozess für „visuell geführtes Pick-and-Place auf einem mobilen Roboter mit Objekterkennung" umsetzt. Dafür wird zunächst das Testumfeld mit Hinblick auf die Unterschiede zwischen Simulation und Realität beschrieben.
Im Anschluss werden zu den einzelnen Phasen aus dem Kapitel 3.3, *detect*, *pick* und *place*, verschiedene Bewertungskriterien aufgestellt. Auf Basis dieser Kriterien werden die einzelnen Phasen ausgewertet, sodass zum Schluss Verbesserungspotentiale für diese aufgezeigt werden können.

4.1 Testumfeld

Das Testumfeld wurde einerseits in der Simulationssoftware Gazebo, andererseits in dem Labor der Hochschule für angewandte Wissenschaften Hamburg aufgebaut. Dabei wurde die Simulationsumgebung so weit wie möglich an die Realität angepasst, unterscheidet sich aber dennoch in einigen Punkten. Die Gemeinsamkeiten und Unterschiede werden folgend aufgeführt.

Raum

Während in der Simulation der Raum nur aus einem Tisch, einem Paket und dem Husky besteht, ist der Raum in der Realität ein Labor mit sehr vielen Störfaktoren, wie Stühlen, Tischen, Schränken, Fenstern und anderen Gegenständen. Auch variiert die Beleuchtung in der Realität, wenn die Sonne durch die Fenster scheint und sich so die Lichtverhältnisse im Raum verändern.

Tisch

Der Tisch in der Simulation kann frei manipuliert werden, im Labor hingegen haben die Tische eine feste Größe und der Platz ist begrenzt. Dementsprechend wird in

der Realität mit ähnlichen Gegenständen gearbeitet, um verschiedene Höhen zu simulieren. Zum Beispiel wird ein Rollcontainer hinzugezogen um niedrige Tische zu simulieren.

Paket

Pakete in der Simulation lassen sich einfach durch die Veränderung des Modells anpassen. In der Realität hingegen muss das Paket erst physikalisch erstellt/gekauft werden. Als Vereinfachung wurde hier ein orangenes Instax Paket gewählt, welches die Maße 8,5cm x 5,4cm x 5,5cm hat und sich eindeutig von der Umgebung abhebt.

Husky

Auch der Husky unterscheidet sich in der Realität von der Simulation. Die Interaktion mit der Umgebung in der Simulation ist viel genauer und weniger fehleranfällig. So kann zum Beispiel die Basis des Huskys in der Simulation zentimetergenau bewegt werden, wohingegen in der Realität Abweichungen durch die verwendete Panzersteuerung der Räder entstehen. Dazu kommen Vereinfachungen im Aufbau des Huskys, wie zum Beispiel das fehlende Tablet für den UR5-Arm, fehlende Kabel und die fehlende Kuhle auf der Ladefläche. In der Realität führen diese Faktoren vereinzelt zu Störungen oder ungewollten Kollisionen.

Gripper

Der Gripper ist in der Simulation vereinfacht dargestellt und kann nur das Paket aufnehmen und ablegen. In der Realität hingegen ist der Gripper mit zusätzlichen Sensoren ausgestattet, die beispielsweise einen Notstopp auslösen, wenn das innere des Grippers mit einem Objekt kollidiert. Auch die Reibung bei Kontakt mit einem Objekt ist in der Simulation vereinfacht dargestellt und kann individuell eingestellt werden, wohingegen in der Realität die Reibung physikalische Abhängigkeiten hat. Die Bewegung zum Öffnen und Schließen des Grippers ist in der Simulation linear von rechts nach link. In der Realität hingegen bewegt sich der Gripper in einem leichten Bogen nach vorne.

Software

Neben den Unterschieden in der echten und simulierten Hardware, gibt es auch Unterschiede in der Software. So nutzt die simulierte Realsense die Topic *realsense/rgb*, die echte hingegen *camera/rgb*. Auch der Maßstab der Ausgabe der Realsense unterscheidet sich von der echten Kamera. Die Rückgabe für Distanzen oder Koordinaten der echten Kamera erfolgt in Millimetern, wohingegen die Ausgabe in

der Simulation in Metern ist. Ein anderer Unterschied zeigt sich bei der Geschwindigkeit der Bewegungen des Roboters, da der reale Husky deutlich schneller agiert als der simulierte Husky.

Beide Testumgebungen sind in Abbildung 4.1 aus der Sicht des Huskys dargestellt.

Abbildung 4.1: Testumgebung aus der Sicht des Huskys, wobei in rechts die Simulation und links die Realität abgebildet ist.

4.2 Auswertung

Anhand der Anforderungen aus in Kapitel 3.3 wird die Implementierung bewertet. Dabei wird jede Phase der Implementierung einzeln betrachtet und evaluiert.

4.2.1 Metriken

In den Graphen aus Abbildung 4.2 sind Precision, Recall, mAP0.5 und mAP0.5:0.95 für die jeweiligen YOLOv5 Modelle für die Simulation und die realen Daten abgebildet. Die Precision und der Recall für die Simulation und für die realen Daten liegen bei etwa 1. Demnach weisen beide Modelle eine sehr hohe Genauigkeit auf, die Pakete aus den Trainingsdaten zu erkennen und zu lokalisieren. Die Kennzahl mAP0.5 liegt bei beiden

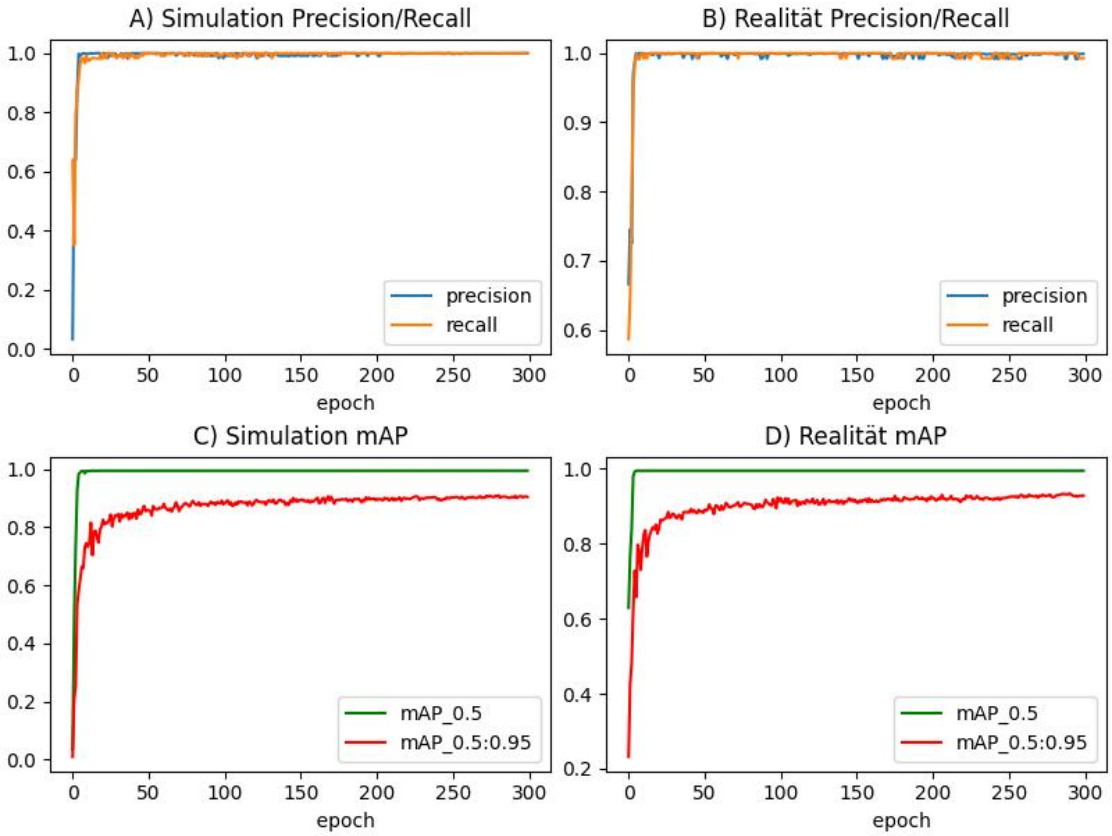


Abbildung 4.2: Graphen der Metriken für die Simulation und für die realen Daten.

Modellen bei 0.995. Bei der Kennzahl mAP0.5:0.95 liegt der höchste Wert für die Simulation bei 0.908 und für die realen Daten bei 0.938.
Da alle genannten Kennzahlen nah am Höchstwert 1 liegen, kann man sehr gute Ergebnisse des Objekterkenners erwarten. Außerdem wurde überprüft, ob ein Overfitting bzw. Underfitting vorliegt, indem die Kennzahlen für die Trainings- und Validierungsdaten verglichen wurden. In Abbildung 4.3 ist zu sehen, wie der Trainingsloss (Kurven mit dem Präfix 'train/') und der Validierungsloss (Kurven mit dem Präfix 'val/') zusammen gegen 0 laufen. Dabei beinhaltet der Graph A) die Daten der Simulation und Graph B) die Daten aus der Realität.

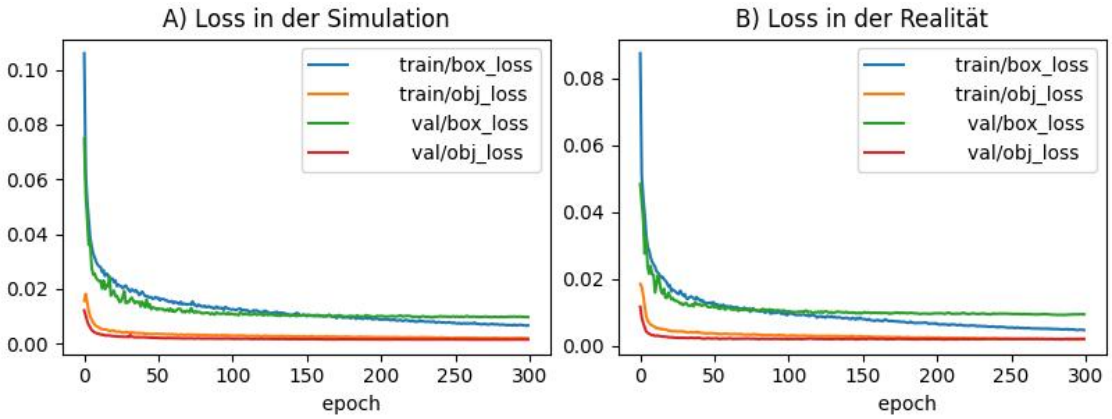


Abbildung 4.3: Graphen der Loss-Metriken für die Simulation und für die realen Daten.

4.2.2 Versuche

Die Versuche wurden in der Simulation und in der Realität durchgeführt. Einzelne Phasen weisen unterschiedliche relevante Szenarien auf, da die Ausgangslage je nach Phase für Roboter und Paket unterschiedlich ist. Die einzelnen Szenarien werden in den folgenden Teilabschnitten beschrieben.

4.2.3 Phase 1: Detect

In der Phase Detect wird zwischen drei unterschiedlichen Startszenarien unterschieden:
- Paket ist sichtbar und in Griffweite
- Paket ist sichtbar außerhalb der Griffweite
- Paket ist nicht sichtbar

Erfolgreich ist ein Versuch, wenn das Paket in Griffweite erkannt wurde und die Position des Pakets in die nächste Phase übergeben wurde.
Die Versuche zeigten, dass die Objekterkennung in der Simulation und in der Realität auch zuverlässig funktioniert hat. Sogar bei einer vorher nicht trainierten Situation ohne Licht, konnte das Paket zuverlässig erkannt werden (siehe Abbildung 4.4).
Die Fehlerequellen für die nicht erfolgreichen Durchläufe waren komplexe Basis-Positionierungen des Huskys, in der Nähe des Randes. In diesen Fällen kam es zu Kollisionen mit dem Tisch bzw. dem Rollcontainer.

(a) (b)

Abbildung 4.4: Versuchsdurchlauf mit Licht (a) und ohne Licht (b)

4.2.4 Phase 2: Pick

Voraussetzung für diese Phase ist, dass der Husky in Griffweite des Pakets ist und die Position des Pakets bekannt ist. Sollte dies in der vorhergehenden Phase nicht erfolgt sein, wird die Position des Huskys manuell angepasst und die Position durch die Objekterkennung neu ermittelt. Erfolgreich ist diese Phase, wenn das Paket vom Gripper umschlossen wurde. In der Phase Pick sind zwei Szenarien relevant:
- Paket steht frontal zum Gripper
- Paket steht seitlich zum Gripper

Mit seitlich ist eine Position gemeint, bei der nicht nur die Frontseite des Pakets sichtbar ist, sondern auch die Seitenfläche.
Der Erfolg dieser Phase hängt sowohl von der Position des UR5 Arms, als auch von der Position und der Orientierung des Pakets ab. Sowohl in der Realität, als auch in der Simulation ist es vorgekommen, dass der UR5 Arm keinen validen Pfad zum Paket gefunden hat.
Eine andere Fehlerquelle entsteht, sobald des Pakets nicht mehr frontal zum Gripper steht. Dann entspricht der Mittelpunkt der Bounding Box nicht mehr dem Mittelpunkt des Pakets und schiebt das Paket aus dem Gripper, da ein Finger des Grippers das Paket mitschleift.
Steht das Paket mit der Ecke zu dem Gripper, muss dieser das Paket über die gesamte Diagonale des Pakets greifen, wie auf Abbildung 4.5. Auch hier schiebt der Gripper das Paket vor sich her, ohne es greifen zu können oder könnte sogar in der Realität den Notstopp auslösen. Dies ist aber nicht vorgekommen ist.

Abbildung 4.5: Der Husky beim Versuch das Paket seitlich zu greifen.

4.2.5 Phase 3: Place

Voraussetzung für diese Phase ist, dass der Gripper am Paket anliegt. Sollte dies in der vorhergehenden Phase nicht erfolgt sein, wird die Position des UR5 Arms und des Grippers manuell angepasst, sodass der Gripper das Paket umschließt. Erfolgreich ist diese Phase, wenn das Paket kontrolliert auf die Ladefläche des Huskys abgesetzt wurde. In

dieser Phase ist nur das Szenario relevant, dass das Paket vom Gripper umschlossen wird. In der Simulation sind alle Versuche fehlgeschlagen, da der UR5 Arm nicht in der Lage war, das Paket festzuhalten. Fehlerquelle hierfür ist, dass der UR5 Arm in der Simulation trotz vorhandener Reibung nicht halten konnte und fallen lassen hat. Eine andere Fehlerquelle ist, dass der Roboterarm zu nah an dem Tisch war und so bei der Bewegung nach unten mit dem Tisch kollidiert ist. Auf dem realen Husky war die Erfolgsquote 100% für diese Phase. Jedoch war die Bewegung sehr schnell und ein wenig zu Hoch im Vergleich zur Simulation

4.3 Bewertung und Potentiale

In diesem Abschnitt werden die einzelnen Phasen anhand der Kriterien aus Kapitel 3.5 bewertet. Die Geschwindigkeit wird dabei nur bei der Phase 1 Detect bewertet, da sie in den anderen Phasen keine hohe Relevanz hat.

4.3.1 Phase 1: Detect

Die Objekterkennung hat das Paket in allen Szenarien erkannt und die Position des Pakets wurde korrekt an die nächste Phase übergeben. Somit werden die Kriterien *Zuverlässigkeit* und *Genauigkeit* erfüllt.
Im Punkt *Sicherheit* kann durch den hohen Recall Wert davon ausgegangen werden, dass kein falsches Objekt gegriffen wird. Auch wird in move_base eine lokale Kostenkarte verwendet, die es dem Husky ermöglicht, Hindernisse zu erkennen und diesen auszuweichen. Bei flexibelen Hindernissen, wie zum Beispiel Menschen, kann es jedoch zu Kollisionen kommen.
Das Kriterium Flexibilität wird nur teilweise erfüllt, da die Objekterkennung nur für spezifische Pakete trainiert wurde. Diese kann das Modell jedoch aus verschiedenen Perspektiven und Lichtverhältnissen erkennen.

4.3.2 Phase 2: Pick

Die Phase 2 Pick ist nicht so zuverlässig wie die Phase 1 Detect. Die Erfolgsquote liegt bei 55% in der Simulation und bei 40% in der Realität. Dies liegt daran, dass die *Flexibilität* nur teilweise erfüllt wird. So kann das Paket nur frontal zuverlässig gegriffen werden. Die *Sicherheit* ist auch hier nur in der Planungsphase gegeben, jedoch nicht während der Ausführung.

4.3.3 Phase 3: Place

Die Phase 3 Place ist in der Simulation nicht *zuverlässig*, aufgrund von dem mangeldem Grip des UR5 Arms. In der Realität hingegen wurde dieser Punkt erfüllt. Die Bewegungen wurden *genau* ausgeführt, da feste Winkel statt relativer Positionen verwendet wurden. Dementsprechend leidet die *Flexibilität* und *Sicherheit* darunter, dass die Positionen nicht dynamisch angepasst werden können.

4.3.4 Verbesserungspotentiale

Ein Verbesserungspotential für die erste Phase besteht darin, dass nicht nur die Position und Bounding-Box des Pakets an die zweite Phase übergeben werden, sondern auch die Orientierung des Pakets. Weiterführend können auch die Positionen und Orientierungen von anderen relevanten Objekten, wie zum Beispiel dem Tisch, an die nächste Phase übergeben werden, um so die Zuverlässigkeit der folgenden Phasen zu verbessern.
Ein weiteres Verbesserungspotential besteht darin, dass die Objekterkennung nicht nur für ein spezifisches Paket trainiert wird, sondern für eine Kategorie von Paketen. So kann die Objekterkennung auch für andere Pakete verwendet werden.
Die statischen Positionen der einzelnen Phasen können durch dynamische Positionen ersetzt werden, die anhand der Position des Pakets berechnet werden, um so die Flexibilität des Arms bei verschiedenen Situationen zu erhöhen.
Hinzu kommt, dass die Sicherheit erhöht werden kann, indem die Bewegungen des Arms und des Huskys dynamisch an die Umgebung angepasst werden. Dabei kann zum Beipiel der Notstopp ausgelöst werden, bevor der Arm oder der Husky mit einem Objekt kollidiert.

Abschließend kann die Simulation noch weiter an die Realität angepasst werden, um so die Ergebnisse der Simulation besser auf die Realität übertragen zu können und nicht gesonderte Anpassungen für die Realität zu benötigen.

5 Zusammenfassung und Ausblick

5.1 Zusammenfassung

Zusammenfassend lässt sich sagen, dass das Ziel eines visuell geführtem Pick-and-Place auf einem autonomen, mobilen Roboter konzeptionell erfolgreich war. Zunächst wurde ein Prozess mit Hilfe eines Flussdiagramms entwickelt, welches den Ablauf des Systems modelliert. Anschließend wurde eine FSM konzipiert und mit Hilfe des ROS Pakets Smach implementiert. Dieses vereint die Ansteuerung der einzelnen Actionserver von move_base, MoveIt und dem Gripper mit der Objekterkennung. Für den Objekterkenner wurden zwei neuronale Netze, basierend auf dem YOLOv5, verwendet. Diese wurden mittels Transfer Learning einmal mit dem Datensatz aus der Simulation und ein weiteres Mal mit dem Datensatz aus der Realität trainiert.
In der Evaluation wurde zunächst die Sim-To-Real gap untersucht. Dabei wurde festgestellt, dass die Unterschiede zwischen Simulation und Realität gravierend sind, da sowohl die Hardware, als auch die Softwareunterschiede aufweisen und spezifische Anpassungen in der Prozesssteuerung benötigen.
Danach wurden die einzelnen Metriken des Objekterkenners ausgewertet, welche mit einem Wert von 0.908 für die Simulation bzw. 0.938 für die mAP0.5:0.95 einen sehr guten Wert aufweisen.
Es folgten die Untersuchungen der einzelnen Phasen. Dabei wurde festgestellt, dass die Objekterkennung in der ersten Phase wie erwartet gut funktioniert. Die Ansteuerung der Fahrplattform in der ersten bzw. die Ansteuerung des Armes in der zweiten und dritten Phase weisen hingegen noch Probleme bei komplexen Szenarien auf. Dementsprechend ist das Gesamtziel eines zuverlässigen und genauen Prozesses nur teilweise erreicht. Im letzten Schritt wurden Verbesserungspotentiale aufgezeigt, welche im nächsten Abschnitt noch einmal erläutert werden.

5.2 Ausblick

Für zukünftige Arbeiten gibt es mehrere Ansätze, um die vorhandenen Ergebnisse zu verbessern. Zum einen kann die Sim-To-Real gap verringert werden, indem man die Simulation an den realen Husky anpasst. So sollte zum Beispiel die Realsense in der Simulation auf die gleiche Topic publishen wie die reale Kamera. Außerdem sollte die Kamera in der Simulation auf die gleiche Höhe wie die reale Kamera gesetzt werden, um die gleiche Perspektive zu erhalten und so die Objekterkennung auf beide Umgebungen anwenden zu können.
Ein weiterer Ansatz ist die Erweiterung des Datensatzes auf mehre Klassen und Paketarten. So können dann im Anschluss mehrere Pakete erkannt und verarbeitet werden. Erkennt man den Tisch und andere Gegenstände, kann der Husky sicherer agieren und sogar den Anwendungsfall auf weitere Objekte ausweiten. Abschließend kann der MoveIt Planer noch weiter optimiert werden, um die Bewegungen des Armes so zu verbessern, dass er die Bewegungen ohne Abbrüche ausführt. Optimalerweise sollte auch eine Kollisionserkennung während der Ausführung der Bewegung implementiert werden, um die Sicherheit der Gegenstände und Personen zu erhöhen.

Literaturverzeichnis

[1] Alexander Thamm GmbH: *Data Auggmentation.* – URL https://www.alexanderthamm.com/de/data-science-glossar/data-augmentation/. – Zugriffsdatum: 2023-06-27

[2] Alzubaidi, L. ; Zhang, A.J. et a.: Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. (2021). – URL https://doi.org/10.1186/s40537-021-00444-8

[3] Baeldung: *Training and Validation Loss in Deep Learning.* 2019. – URL https://www.baeldung.com/cs/training-validation-loss-deep-learning. – Zugriffsdatum: 2023-07-20

[4] Daglioglu, M: *Object-Detection mit You Only Look Once (YOLO) : Einführung in die Objekterkennung mit YOLO sowie die Weiterentwicklung in den Versionen v2-v4.* 2021

[5] Dattalo, A.: *What is ROS?.* – URL https://wiki.ros.org/ROS/Introduction. – Zugriffsdatum: 2023-06-27

[6] Dong, B. ; Li, Q. ; Wang, J. ; Huang, W. ; Dai, P. ; Wang, S.: An End-to-End Abnormal Fastener Detection Method Based on Data Synthesis. In: *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*, 2019, S. 149–156

[7] HAW Hamburg : *Industrieroboter „Husky“ für die Autonomone Quartiersmobilität.* 2021. – URL https://www.haw-hamburg.de/hochschule/technik-und-informatik/departments/maschinenbau-und-produktion/forschung/forschungsgruppen/elektrische-mobilitaet/tiq-fahrplattform-husky/. – Zugriffsdatum: 2023-06-27

[8] Hertzberg, J. und Lingemann, K. und Nüchter, A.: *Mobile Roboter. Eine Einführung aus Sicht der Informatik.* Springer Berlin, 2012. – 4 S. – URL https://link.springer.com/book/10.1007/978-3-642-01726-1. – ISBN 9783642017261

[9] Hertzberg, J. und Lingemann, K. und Nüchter, A.: *Mobile Roboter. Eine Einführung aus Sicht der Informatik.* Springer Berlin, 2012. – 51 S. – URL https://link.springer.com/book/10.1007/978-3-642-01726-1. – ISBN 9783642017261

[10] IBM: *What is deep learning?*. – URL https://www.ibm.com/topics/deep-learning. – Zugriffsdatum: 2023-06-27

[11] Intel: *Autonome mobile Robotertechnik und Anwendungsfälle.* – URL https://www.intel.de/content/www/de/de/robotics/autonomous-mobile-robots/overview.html. – Zugriffsdatum: 2023-06-27

[12] Intel: *Depth Camera D435.* – URL https://www.intelrealsense.com/depth-camera-d435/. – Zugriffsdatum: 2023-06-27

[13] Intel: *Beginner's guide to depth.* 2019. – URL https://www.intelrealsense.com/beginners-guide-to-depth/. – Zugriffsdatum: 2023-06-27

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

[15] Kader, L: Eine kurze Geschichte der künstlichen Intelligenz. Von Höhen und Tiefen. In: Wie Maschienen Lernen. Künstliche Intelligenz verständlich erklärt. Springer Wiesbaden, 2019, S. 140. – URL https://link.springer.com/book/10.1007/978-3-658-26763-6. – ISBN 9783658267629

[16] Nehmzow, U.: *Mobile Robotik. Eine praktische Einführung.* Springer Berlin, 2002. – 7–8 S. – URL https://link.springer.com/book/10.1007/978-3-642-55942-6. – ISBN 9783-642559426

[17] Pawara, P. ; Okafor, E. ; Schomaker, L. ; Wiering, M.: Data Augmentation for Plant Classification, 09 2017. – ISBN 978-3-319-70352-7

[18] Stockburger, Christoph: *Die Denksportwagen.* 2017. – URL https://www.spiegel.de/auto/aktuell/kuenstliche-intelligenz-wie-autos-

durch-neuronale-netze-das-fahren-lernen-a-1132759.html. – Zugriffsdatum: 2023-07-25

[19] StudioFix: *StudioFix*. – URL https://studyflix.de/mathematik/abstand-zweier-punkte-2005. – Zugriffsdatum: 2023-06-27

[20] Szeliski, Richard: *Computer Vision: Algorithms and Applications*. Springer, 2021. – 238 S. – URL https://szeliski.org/Book

[21] Tan, Ren J.: *Breaking Down Mean Average Precision (mAP)*. 2019. – URL https://towardsdatascience.com/breaking-down-mean-average-precision-map-ae462f623a52. – Zugriffsdatum: 2023-06-27

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