HAW HAMBURG

BACHELOR THESIS
Justin Tran

Robuste Türerkennung mit Deep Learning in der bildbasierten Roboternavigation

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Justin Tran

Robuste Türerkennung mit Deep Learning in der bildbasierten Roboternavigation

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
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: 31. August 2022

Justin Tran

Thema der Arbeit

Robuste Türerkennung mit Deep Learning in der bildbasierten Roboternavigation

Stichworte

Autonom, Husky, Roboter, neuronales Netz, Objekterkennung, Deep Learning

Kurzzusammenfassung

Um die Robustheit der Erkennung von Türen durch neuronale Netzwerke zu verbessern, werden Bilddatensätze augmentiert. Das Verändern der Helligkeit, der Farbe, des Kontrastes und dem Löschen von Bildausschnitten, sowie das Nutzen eines Greenscreens sind die Image Data Augmentation Methoden, die vorgestellt werden. Im Ergebnis zeigt sich, dass das Verändern des Kontrastes gefolgt von dem Löschen von Bildausschnitten die Methoden mit den höchsten Ergebnissen sind. Das neuronale Netz, welches eine Kombination aller verwendeten Methoden als Trainingsdatensatz erhielt, konnte die höchsten Erkennungsraten unter unterschiedlichen Umgebungsbedingungen erzielen.

Justin Tran

Title of Thesis

Robust door detection using deep learning in visual based robot navigation

Keywords

Autonomous, Husky, Robot, neural Network, Object detection, Deep Learning

Abstract

To improve the robustness of door detection by neural networks, image data sets are augmented. Changing the brightness, color, contrast and deleting parts of the image, as well as using a green screen are the image data augmentation methods that are used. As a result, it can be seen that changing the contrast followed by deleting parts of the image are the methods with the highest results. The neural network that received a combination

of all the used methods as a training dataset was able to achieve the highest recognition rates under different environmental conditions.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

CNN - Convolutional Neural Network.

DD2 - Datensatz DeepDoors2 von Ramôa *et al.*[3].

FN - False Negative.

FP - False Positive.

HAW - Hochschule für Angewandte Wissenschaften.

HS - normaler unveränderter Hochschuldatensatz.

IoU - Intersection over Union.

KHS - Kombination der Bilder des Image Data Augmentation mit dem Hochschuldatensatz.

KHS2 - Kombination der Bilder des Image Data Augmentation mit dem Hochschuldatensatz, jedoch ohne die Bilder der Methode „Greenscreen“.

mAP - Mean Average Precision.

PR-Curve - Precision-Recall-Curve.

RGB - Rot-Grün-Blau(-Farbraum).

ROS - Robot Operating System.

TP - True Positive.

1 Einleitung

Roboter, die in Innenräumen sicher navigieren sollen, müssen Türen für ihre Orientierung erkennen können. Mit Hilfe von Deep Learning können neuronale Netze entwickelt und trainiert werden, die Türen auf Bildern in kurzer Zeit erkennen. Schaut man sich Abbildungen in Arbeiten über Türerkenner an [3][4][5], nimmt man Bilder von Türerkenner wahr, die die zu sehenden Türen korrekt erkennen. Sucht man im Internet verallgemeinert nach Bildern über „Object Detection“ wird man größtenteils korrekt klassifizierte Objekte mittels Bounding Boxes finden (siehe Abbildung 4.1). Dies kann dem Anwender ein verzerrtes Bild darüber mitteilen, dass jegliche Objekterkenner eine sehr hohe Erkennungsrate erreichen. Die Erwartungen des Anwenders können entsprechend hoch sein, dass ohne das Verändern der Architektur des neuronalen Netzes eine hohe Anzahl korrekter Klassifizierungen ausgegeben wird.

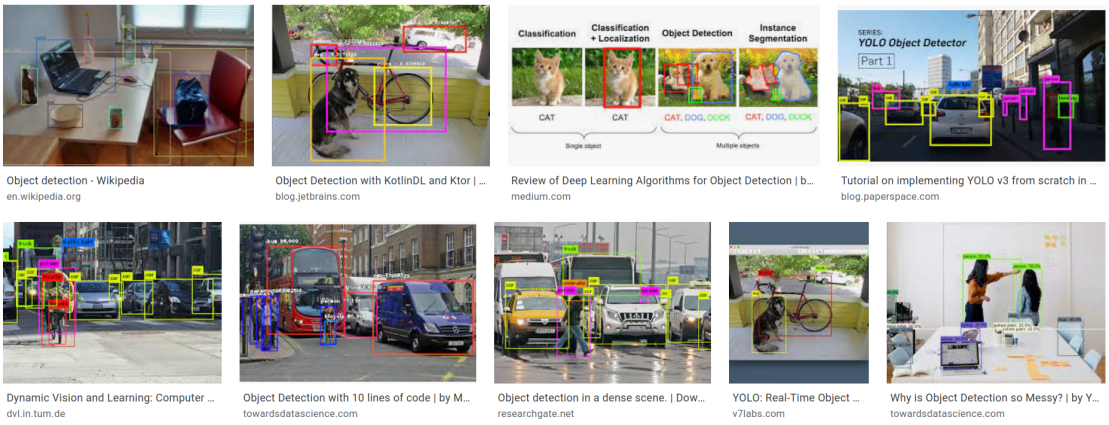


Abbildung 1.1: Google Bilder Suche des Begriffes „Object Detection“ [1].
Neun Suchergebnisse, die korrekt klassifizierte Objekte mittels Bounding Boxes aufzeigen.

Bei der Anwendung eines neuronalen Netzes auf Videos oder Kameraaufnahmen ist die Fehleranfälligkeit von Objekterkennern höher als bei der Erkennung auf Bildern. Unzu-

treffende Labels oder falsche Lokalisierungen von Objekten durch Bounding Boxes sind in solchen Fällen wahrzunehmen [6][7].

Ein Machine-Learning-Modell, welches auch bei Kameraaufnahmen konstant korrekte Erkennungen mit einer 100% sicheren Erkennungspunktzahl ausgibt, wäre ein ideales Modell für die Roboternavigation. Durch die hohe Korrekheit der Erkennung können Unfälle mit Objekten, Tieren und Menschen vermieden werden.

Daher ist das Bestreben des Entwicklers, dass der implementierte Objekterkenner in bekannter und unbekannter Umgebung, sowie bei Verdeckung von Objekten[5] und unter unterschiedlichen Lichtverhältnissen[8] die höchstmöglichen Erkennungsraten ausgibt.

1.1 Aufbau

1. Zu Beginn der Thesis wird im ersten Kapitel mit der Einleitung, dem Aufbau und der Aufgabenstellung begonnen.
2. Im zweiten Kapitel geht es um die Grundlagen, die belangvoll für die Thematik und das Verständnis sind.
3. Das Kapitel drei handelt über Arbeiten, die ähnliche oder gleiche Themen bearbeiten.
4. Implementation nennt sich das vierte Kapitel mit dem Inhalt, wie die Kreation der neuronalen Netze und die Versuchsaufbauten umgesetzt worden sind.
5. Das fünfte Kapitel beschreibt die Versuchsergebnisse, den Zusammenhang zur Aufgabenstellung und die Erfüllung der Robustheit.
6. Im Fazit werden die wichtigsten Erkenntnisse der Arbeit zusammenfassend bestimmt.

1.2 Aufgabenstellung

Das Ziel der Arbeit ist es ein robustes Machine-Learning-Modell für die Objekterkennung von Türen inklusive dessen Zustand (*offen*, *halbgeschlossen*, *geschlossen*) zu implementieren. Das Machine-Learning-Modell soll von dem Roboter namens Husky der Forschungsgruppe Autosys der HAW Hamburg für die Navigation innerhalb der Flure der HAW verwendet werden, da das Ziel von Autosys ist, dass sich der Husky autonom innerhalb des Hochschulgeländes fortbewegen kann [9].

Für die Navigation des Huskys ist es notwendig, dass dieser Türen und dessen Zustände erkennen kann, um Ein- bzw. Ausgänge zu erkennen und zu überprüfen, ob eine Durchfahrt möglich ist. Ebenso kann dieses Verfahren im Zusammenspiel mit der Sensorik des Roboters genutzt werden, um zu validieren, dass es sich korrekterweise um eine durchfahrbare Tür handelt.

Die Robustheit in dieser Thesis wird darüber definiert, dass ein neuronales Netz entwickelt wurde, welches höhere Erkennungsraten erreicht, als ein neuronales Netz, welches den unveränderten Trainingsdatensatz erhielt [8]. Um zu diesem Ziel zu gelangen, werden

die Techniken von Michaelis *et al.* [8] verwendet, da bei unterschiedlichen Datensätzen eine Steigerung der Erkennungsrate von 16% bis 41% bewirkt werden konnte. Es werden somit Trainingsbilddatensätze erschaffen und untersucht, aber die Architektur des neuronalen Netzes bleibt unberührt.

Eine ausführliche Definition des Begriffes der Robustheit innerhalb dieser Arbeit wird in Kapitel 4 beschrieben.

2 Grundlagen

Dieses Kapitel bietet Grundlagenwissen, welches hilfreich ist, um die Arbeit zu verstehen. Es werden wichtige Begriffe und Hintergrundwissen erklärt.

2.1 Evaluierungsmetriken

Um einen Objekterkenner zu evaluieren und mit anderen Objekterkennern vergleichen zu können, benötigt man **Evaluierungsmetriken**. Da es eine Vielzahl von Evaluierungsmetriken gibt, wurde sich lediglich auf die in dieser Thesis verwendeten Metriken fokussiert.

2.1.1 Intersection over Union

Bei der **Intersection over Union** berechnet man die Überlappung der vorhergesagten Bounding Box gegenüber der tatsächlichen Bounding Box. Die Größe der Fläche der Überlappung wird durch die Fläche der Vereinigung geteilt und als Parameter erhält man die Intersection over Union. Eine Vorhersage wird als True Positive eingestuft, wenn der Wert über einem festgelegten Schwellenwert liegt. Werte unter dem Schwellenwert werden als False Positive eingestuft. Je höher der Wert, desto besser ist die Erkennung.

2.1.2 Recall

Recall zeigt, wie viele True Positives in *allen* Vorhersagen enthalten sind. Man berechnet den Recall mit der Formel $TP/(TP + FN)$.

2.1.3 Precision

Die **Precision** zeigt, wie viele True Positives in *allen positiven* Vorhersagen enthalten sind. Dabei berechnet man $TP/(TP + FP)$ und erhält die Precision als Ergebnis.

2.1.4 PR-Curve

PR-Curve ist die Kurzschreibweise für **Precision-Recall-Curve**. Diese Kurve stellt man dar indem man den Recall auf die X-Achse und die Precision auf die Y-Achse setzt. Je größer die Fläche unter der Kurve ist, desto eher kann man der Annahme folgen, dass das Modell eine hohe Precision (*niedrige FP-Rate*) und einen hohen Recall (*niedrige FN-Rate*) haben [10].

2.1.5 Mean Average Precision

Eine häufig verwendete Metrik ist die Mean Average Precision, um die Korrektheit eines Objekterkenners anhand eines Datensatzes anzugeben. Für die Berechnung der mAP werden eine Vielzahl von Bildern an den Objekterkenner übergeben. Bei der Objekterkennung der Bilder werden die folgenden Eigenschaften berechnet:

- **Korrektheit** = *wahr*, wenn IoU >= IoU Schwellenwert;
  falsch, wenn IoU < IoU Schwellenwert
- **Precision** = *bk/tk*
  *bk := Anzahl **bisher** korrekter Vorhersagen;*
  *tk := Anzahl **bisher tatsächlich** korrekten Ergebnisse*
- **Recall** = *bk/ak*
  *ak = Anzahl **aller tatsächlich** korrekten Ergebnisse*

Anhand der Berechnungen wird eine PR-Curve erstellt dessen Fläche unterhalb der Kurve ermittelt wird. Bevor dies passiert, wird die Kurve geglättet indem in Schritten von 0.1 entlang der X-Achse (*Recall*) jeweils der rechte maximale Y-Achsen-Wert (*Precision*) an der X-Achsen-Position als neuer Y-Achsen-Wert festgesetzt wird. Durch die Glättung werden eventuelle unebene Muster im Diagramm entfernt. Die Fläche prozentual als Fließkommazahl dargestellt, repräsentiert die Average Precision (AP), wobei 1.0 das „perfekte Modell“ mit einer hohen Precision und einem hohen Recall charakterisiert. Die AP ist von dem IoU-Schwellenwert abhängig. Je niedriger der Schwellenwert, desto

höher ist die Precision. In der nachfolgenden Formel für die Berechnung der mAP ist n die Anzahl der Schwellenwerte und k die Anzahl der Klassen:

$$mAP = \frac{1}{n} \sum_{k=1}^{n} AP_k$$

Der Unterschied von der mAP zur AP ist, dass für die Berechnung der mAP die IoU-Schwellenwerte von 0.5 bis 0.95 in 0.05er-Schritte für jede vorhersagbare Objektklasse durchlaufen und die AP für alle Klassen berechnet wird. Anschließend berechnet man den Durchschnitt aller APs für die verschiedenen Schwellenwerte und erhält die **mAP** [11].

2.1.6 Loss

Der Loss wird über Loss Funktionen berechnet und zeigt an, wie fehlerhaft die Vorhersage eines Machine-Learning-Modell gegenüber den tatsächlichen Ergebnissen ist. Der Wert der Loss Funktion gibt an, wie weit entfernt das Vorhergesagte vom Tatsächlichen entfernt ist. Vor allem beim Training ist dieser Wert von Bedeutung, denn je größer der Loss ist, desto größer ist die Gewichtsveränderung der Neuronen [12].

3 Verwandte Arbeiten

Ein Überblick über verwandte Arbeiten zur Erkennung von Türen und robuster Objekterkennung.

3.1 Erkennung von Türen

Im Jahr 2014 haben **Wei Chen *et al.*** (2014) [4] die damalig neue Machine-Learning Methode Deep Learning mit einem CNN untersucht, um Eigenschaften einer Tür zu erkennen. Mit ihrem trainierten CNN konnte auf ihrem Bilddatensatz eine Erkennungsrate von $\approx 97\%$ erzielt werden. Diese Arbeit hatte das Ziel für die Navigation von Robotern verwendbar zu sein.

Burak Kakillioglu *et al.* (2016) [13] beschäftigten sich mit der Erkennung von Türen indem mit 3D-Punktwolken und RGB-Bildern validiert wurde, ob das aufgenommene Bild einer Tür entspricht. Darüber hinaus wurde auf eine Objekterkennung in Echtzeit gesetzt, da die Implementierung für die Navigation von Robotern genutzt werden soll.

Ein Framework in Form eines ROS Packages wurde von **Bersan *et al.*** (2016) [14] entwickelt. Dieses Framework ist für damalig häufig genutzte Roboter-Hardware konzipiert und verwendet einen Objekterkenner auf Basis eines CNNs, sowie eine 3D-Segmentierungstechnik, um verschiedene Objekte innerhalb einer Szene zu erkennen. Anhand des Beispiels der Türerkennung wird das Framework vorgestellt.

In dem Paper von **João Ramôa *et al.*** (2021) [3] geht es um drei verschiedene Methoden zur Klassifikation des Zustandes einer Tür mit dem Ziel die Roboternavigation im Innenraum zu verbessern. Dabei ist ebenso ein Ziel, dass die Methoden auf Geräten mit wenig Leistung, wie z.B. einem Raspberry Pi oder Jetson Nano, ausführbar sind und dass die Ausführung in Echtzeit geschieht. Die Methoden von Ramôa *et al.* umfassen dreidimensionale Objekterkennung anhand von aufgenommenen Punktwolken, semantischer Segmentierung und Farbbildern.

3.2 Robuste Objekterkennung

Eine Methode zur Identifizierung von Eigenschaften von Türen wurde von **Christopher Juenemann *et al.*** (unbekannt) [15] in MATLAB implementiert. Die Methode beruht auf dem „Canny Edge Detection"-Algorithmus, der Hough-Transformierung und der Fuzzy-Logik, um Türen möglichst zuverlässig zu identifizieren. Dabei ist die durchschnittliche Erkennungszeit der Methode mit 7 Sekunden angegeben.

Claudio Michaelis *et al.* (2019) [8] nutzen für den Vergleich der Robustheit neuronaler Netze ihr eigenes entwickeltes Framework und zwei selbst definierte Metriken namens Mean Performance Under Corruption (mPC) und Relative Performance Under Corruption (rPC). Das Ergebnis ihrer Arbeit ist, dass man für die Bestimmung des Grades der Robustheit die Erkennungsrate am Validierungsdatensatz des neuronalen Netzes als Maßstab nehmen kann, welches mit dem unveränderten Bilddatensatz trainiert worden ist. Das Cartoonisieren der Bilder steigerte die Erkennungsrate und auch Image Data Augmentation, wie zum Beispiel das Verändern des Kontrastes oder der Helligkeit, führten zu einer höheren Zuverlässigkeit. Durch die Anwendung der Techniken konnten Steigerungen der Erkennungsraten von 16% beim Datensatz PASCAL, 12% beim Datensatz COCO und 41% beim Datensatz Cityscapes erzielt werden.

Die Experimente von **Wang *et al.*** (2020) [5] zeigen, dass vortrainierte neuronale Netze, wie Faster R-CNN, „teilweise verdeckte Objekte nicht robust erkennen können". Compositional CNNs hingegen zeigen eine höhere Zuverlässigkeit, da die einzelnen Bestandteile eines Objektes als ein zusammengesetztes Objekt erkannt werden. Mit der Methode konnten Wang *et al.* die absoluten Erkennungsraten im Vergleich zum Faster R-CNN am Beispiel des Datensatzes PASCAL um 41% und beim Datensatz COCO um 35% steigern.

Um den Prozess einer Türöffnung von Kühlschränken, Türen und Schränken für einen Roboter so zuverlässig, wie möglich zu implementieren, haben **Arduengo *et al.***(2021) [16] ein CNN trainiert, welches Türen und Handgriffe anhand von Pointclouds erkennt. Die Autoren sagen, dass sie die Robustheit des Objekterkenners durch die verschiedenen Varianten der zu erkennenden Objekte im Datensatz erreicht haben. Beim Validieren ihres Objekterkenners mit ihrem eigenen Bilddatensatz wurde eine mindestens 10% niedrigere mAP festgestellt, als mit einem Datensatz, wie VOC 2012 oder COCO. Dennoch wurde die Erkennung der Türen und Klinken als robust beschrieben, da die Objekterkennung und der Greifmechanismus in den Experimenten erfolgreich war.

4 Implementation

Das Kapitel Implementation befasst sich mit der Implementierung des neuronalen Netzes sowie der Implementierung am Husky.

4.1 Robustheit

Das Nomen **Robustheit** bzw. das Adjektiv **robust** wird im Machine-Learning verwendet, um Modelle zu beschreiben, welche zu manipulierten oder fehlerhaften Inputdaten korrekte Outputdaten liefern[17]. Bezogen auf Objekterkenner werden diese als **robust** deklariert, wenn

- die **Erkennung von Objekten unter verschiedensten Wetterbedingungen** im Vergleich zu klaren Wetterverhältnissen genauso korrekt verläuft[8]. Im Allgemeinen wurde der unveränderte Datensatz als Maßstab verwendet, um zu messen, ob angewendete Methoden die Fehlerkennungen reduzieren und die Erkennungsrate, sowie die Robustheit des Objekterkenners steigern.
- das Modell **keine falsche Erkennung bei Nachahmungen von Objekten** ausgibt bzw. eine richtige Erkennung bei leicht veränderten oder verdeckten Merkmalen. Ein Beispiel für Nachahmungen sind Straßenschilder als Graffiti bei Straßenschilderkennern. Für leicht veränderte Merkmale wird das Beispiel einer getragenen Brille bei Gesichtserkennern genannt, wenn das Modell mit dem zu erkennenden Gesicht ohne Brille trainiert worden ist.[18]

Das Modell für die Türerkennung soll ebenfalls, wie im ersten Punkt der Liste beschrieben, unter verschiedenst starken Umwelteinflüssen, wie Beleuchtung, korrekte Ergebnisse liefern. Ebenso wird in dieser Arbeit der Vergleich zum unveränderten Datensatz gezogen, um zu zeigen, dass die Anwendung von den gewählten Methoden eine Steigerung der Erkennungsrate und eine Reduzierung der fehlerhaften Erkennungen mit sich bringt.

Auch die zweite Definition der obigen Auflistung passt zu dem Türszenario, da es auf den Fluren der HAW Türen gibt (*Stromkastentüren*), die erkenntlich nicht begehbar sind.

Das Modell des Türerkenners soll explizit folgende Eigenschaften erfüllen, um als robust anerkannt zu werden:

1. Bei der **Unterscheidung der Zustände einer Tür** sollen mindestens 57%[1] der Türen dem korrekten Zustand zugeordnet werden.

2. Da die Flure des Gebäudes HAW Berliner Tor 5 nur künstlich beleuchtet werden sollen mehr als 51%[2] der **Erkennungen in verschieden stark beleuchteten Umgebungen** so erfolgreich sein, wie die Erkennungen bei klaren Wetterhältnissen und alltäglicher Beleuchtungsstärke des Raumes.

3. Mehr als als 71%[3] der **Tür-ähnlichen Objekte**, wie erkenntlich nicht begehbare Türen oder Schränke, sollen korrekt als Nicht-Tür erkannt oder gar keine Erkennung vorhergesagt werden.

4. Der **Blickwinkel der Kamera** soll kein Beeinflussungsfaktor sein, denn es soll sowohl die untere, fest montierte als auch die frei bewegbare Tiefenkamera am Arm des Huskys für das Machine-Learning-Modell verwendbar sein.

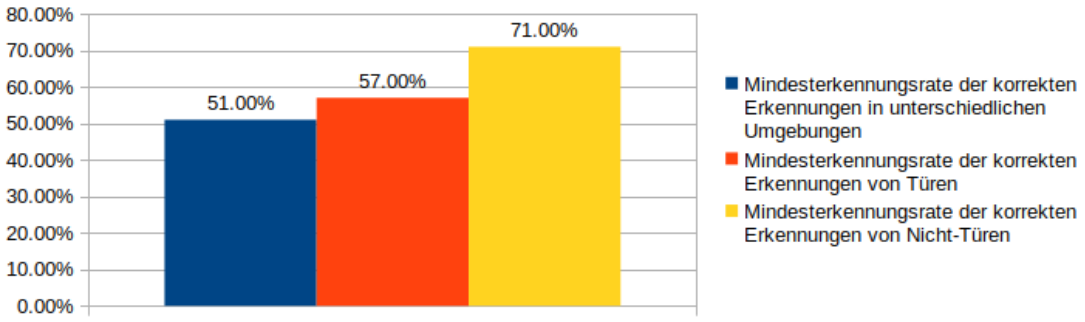


Abbildung 4.1: Korrekte Mindesterkennungsraten eines neuronalen Netzes, um dieses als robust definieren zu können.
Blau: In untersch. stark beleuchteten Umgebungen müssen mind. 51% der Türen korrekt erkannt werden. **Rot**: Insgesamt sollen mind. 57% der Türen korrekt erkannt werden. **Gelb**: Mindestens 71% der Nicht-Türen sollen keinem Label oder dem Label Nicht-Tür zugeordnet werden.

[1] Die Tabelle 5.5 zeigt, dass 43% der Erkennungen beim HS einem falschen Zustand der Tür zugeordnet waren.
[2] In Tabelle 5.7 waren beim HS 51% aller Erkennungen unter unterschiedlichen Lichtverhältnissen erfolgreich.
[3] Gemäß Tabelle 5.8 wurden beim HS 29% der Tür-ähnlichen Objekte fälschlicherweise als Tür erkannt.

Die Schwellenwerte für die Eigenschaften der Definition entstammen den Referenzwerten aus den Ergebnissen der durchgeführten Experimente für das neuronale Netz SSD Mobilenet V2 mit dem unveränderten Bilddatensatz HS (*siehe ab Kapitel 5*).

4.2 Datensatz

Es wird beschrieben, wie die Aufnahmen des HS zustande kamen und wie die Image Data Augmenation angewendet worden ist.

4.2.1 Aufnahme des Hochschuldatensatzes

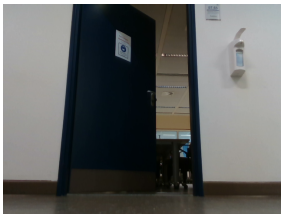
Für die Aufnahme der Bilder des HS wurde die Intel RealSense D435 verwendet, welche eine Auflösung bei Fotos von 640x480 Pixeln erlangt. Es wurde ein Pythonskript entwickelt (siehe Listing A.4), welches in benutzerdefinierten Intervallen Aufnahmen der Tiefenkamera macht und diese anschließend in einem Ordner speichert.

Die Anforderungen an die Aufnahmen des HS waren folgende:

- Enthält Fotos vom **unteren Bereich der Tür**, da der Husky eine Kamera in Bodennähe beistzt.
- Enthält Fotos vom **oberen Bereich der Tür**, da der Husky eine bewegbare Kamera am Greifarm besitzt.
- Mehr als die **Hälfte der Tür** muss **erkennbar** sein, da somit genug Merkmale einer Tür erkannt werden können und es zu geringeren Verwechslungen zu einem Fenster kommen kann.
- Ein **Mensch** muss auf dem Foto eine Tür bzw. Nicht-Tür **klassifizieren können**.

Insgesamt sind 113 Bilder in dem HS enthalten. In der nachfolgenden Abbildung 4.2 sind sechs Beispiele des Datensatzes zu sehen.

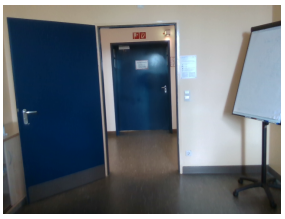
(a) Offene Tür



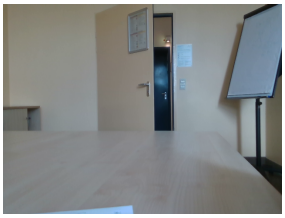
(b) Halbgeschlossene Tür

(c) Stromkastentür

(d) Geschlossene Tür



(e) Offene Tür

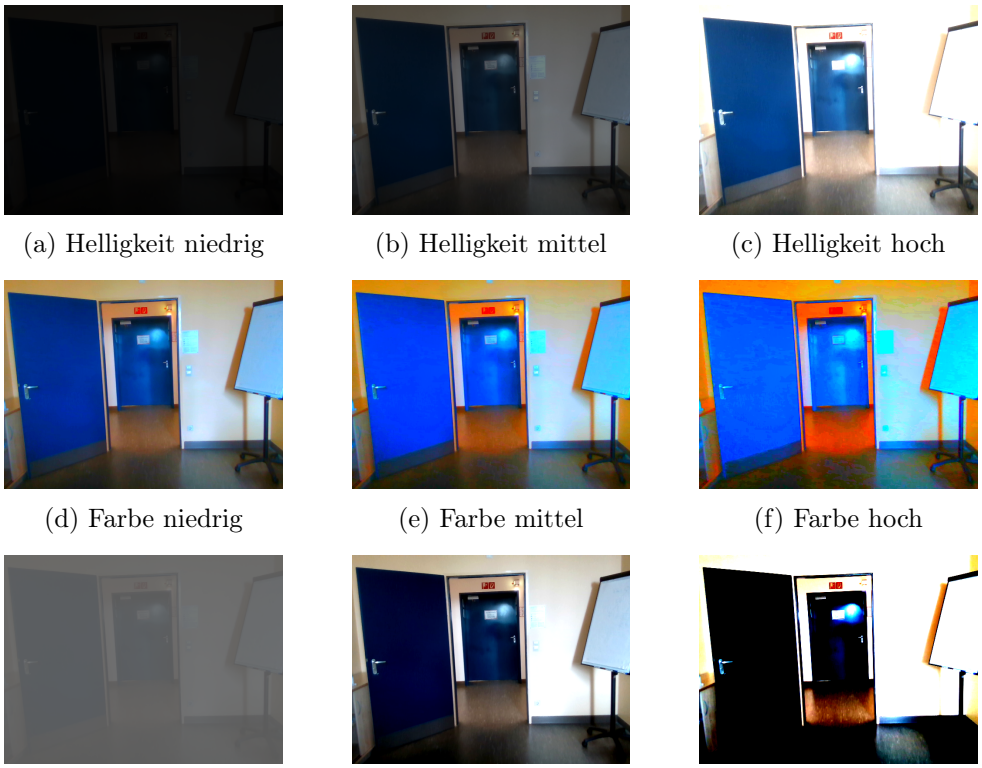


(f) Halbgeschlossene Tür

Abbildung 4.2: Beispielbilder des Hochschuldatensatzes (KHS).
(a) unterer Bereiche einer offenen Tür; (b) unterer Bereich einer halbgeschlossenen Tür; (c) Stromkastentür als Nicht-Tür; (d) oberer Bereich einer geschlossenen Tür; (e) offene Tür mit innerem Anschlag, (f) halbgeschlossene Tür mit innerem Anschlag, die zur Hälfte durch einen Tisch verdeckt ist

4.2.2 Image Data Augmentation

Mit dem Python Package Augmentor [2] ist es möglich Bilder zu generieren, die veränderte Eigenschaften gegenüber dem ursprünglichen Bild (*siehe Abbildung 4.5a*) haben. Für die Generierung wurde eine Augmentor Pipeline erstellt, die den Pfad zu dem Ordner mit den Originalbildern erhalten hat. Im weiteren Prozess wurden die Eigenschaften Helligkeit, Farbe und Kontrast einzeln verändert. Hierzu wurde jeweils der niedrigstmögliche, der mittelstmögliche und der höchstmögliche Wert genommen und die Wahrscheinlichkeit des Eintritts der Veränderung wurde auf 100% gesetzt. Dadurch enstand für jede Eigenschaft ein eigener Bilddatensatz (*siehe ab Abbildung 4.3a bis Abbildung 4.3i*).



(a) Helligkeit niedrig (b) Helligkeit mittel (c) Helligkeit hoch

(d) Farbe niedrig (e) Farbe mittel (f) Farbe hoch

(g) Kontrast niedrig (h) Kontrast mittel (i) Kontrast hoch

Abbildung 4.3: Beispielbilder der Methoden „Helligkeit“, „Farbe“ und „Kontrast“ des erweiterten Hochschuldatensatzes (KHS).
Je Reihe aufsteigender Faktor der Methode von links nach rechts.

Zusätzlich zu diesen Methoden wurde der Inhalt des Bildes verändert. Durch die Methode des Löschens wurden Bildausschnitte ausgeschnitten. Die Größe der Ausschnitte (*Rectangle Area*) wurde in Augmentor auf den Wert 0.1 gesetzt und die Proportionen von 50x65 Pixel wurden durch Augmentor vorgegeben (*siehe ab Abbildung 4.4a bis Abbildung 4.4c*).

(a) Löschen niedrig (b) Löschen mittel (c) Löschen hoch

Abbildung 4.4: Beispielbilder der Methode „Löschen“ des erweiterten Hochschuldatensatzes (KHS).
Aufsteigender Faktor der Methode von links nach rechts.

Die in dieser Arbeit entwickelte Methode, die der Recherche nach innerhalb von Türerkennungsprojekten noch keine Verwendung fand, funktioniert folgendermaßen: Bei Bildern auf denen eine halbgeschlossene oder offene Tür zu sehen ist, wurde der Boden und Hintergrund in dem offenen Bereich der Tür durch einen einfarbig grünen Farbbereich (*Greenscreen*) ersetzt. Dadurch ergab sich die Möglichkeit den grünen Bereich eines Bildes durch ein gewünschtes Bild zu ersetzen (*siehe Abbildung 4.5b und Abbildung 4.5c*).

(a) Original (b) Greenscreen (c) Greenscreen (2)

Abbildung 4.5: Beispielbilder der Methode „Greenscreen“ des erweiterten Hochschuldatensatzes (KHS).
Links: Originalbild, **Mitte**+**Rechts**: Anwendung der Greenscreen Methode.

Bilder mit halbgeschlossenen oder offenen Türen wurden händisch mit einem Bildbearbeitungsprogramm bearbeitet, so dass der sichtbare Bereich hinter der Tür ausgeschnitten und durch neongrüne Farbe ausgetauscht worden ist. Das Python Package OpenCV2 bietet die Möglichkeit die grünfarbigen Bereiche im Bild zu ersetzen. Das gewünschte Bild, welches statt dem grünen Bereich dargestellt werden soll, wurde eingelesen und auf die Größe von 640x480 Pixel angepasst. Anschließend wurde ein Farbbereich für die Farbe Grün definiert, welches nach RGB-Farbcode von [0, 100, 0] bis [120, 255, 100] geht. Die Pixel auf dem Bild, dessen Farbe in diesem Farbbereich enthalten sind, wurden einer Maske hinzugefügt. Anhand der Maske wurde der Hintergrund ausgeschnitten. Der ausgeschnittene Hintergrund und das Originalbild wurden dann zusammengefügt. In dem Bereich des Originalbildes, wo vorher grüne Farbe zu sehen war, wurde der ausgeschnittene Hintergrund drübergesetzt. Zum Schluss wurden die Bilder für die weitere Verwendung gespeichert. In dieser Arbeit wurde sich für weißes Rauschen für den Hintergrund entschieden.

4.3 Labeling

Damit der erweiterte Hochschuldatensatz (KHS), der Bilddatensatz DeepDoors2 (DD2) und der eigene aufgenommene Bilddatensatz Hochschuldatensatz (HS) noch zusätzlich für jedes Bild ein Label enthält, wurden die Bilder mit dem Programm LabelImg händisch gelabelt. Es wurde darauf geachtet, dass die Eigenschaften Türrahmen und Klinke, soweit möglich, innerhalb der gezeichneten Bounding Box enthalten sind und die Namensbezeichnung durchgängig konstant ist. Es wurden die Label *open_ door*, *semi_ door*, *closed_ door* und *no_ door* definiert.

Eine Tür gilt als geschlossen, wenn der Öffnungswinkel zwischen ungefähr 0° bis 30° liegt und man frontal keinen Einblick in den dahinterliegenden Raum hat. Eine Tür ist bei einem Winkel zwischen 30° und 75° halbgeschlossen, wenn man einen Bereich des Raumes hinter der Tür einsehen kann und zusätzlich noch die Tür. Der Zustand der offenen Tür wird darüber definiert, dass der Öffnungswinkel mehr als 75° beträgt, man die Tür gar nicht mehr oder nur zu einem geringen Anteil sieht. Daran gekoppelt ist, dass der Roboter eine Breite von $\approx 67cm$ hat und die Tür nur durchfahren kann, wenn dieser Abstand gegeben ist. Dies ist bei einem Öffnungswinkel von mehr als 75° der Fall.

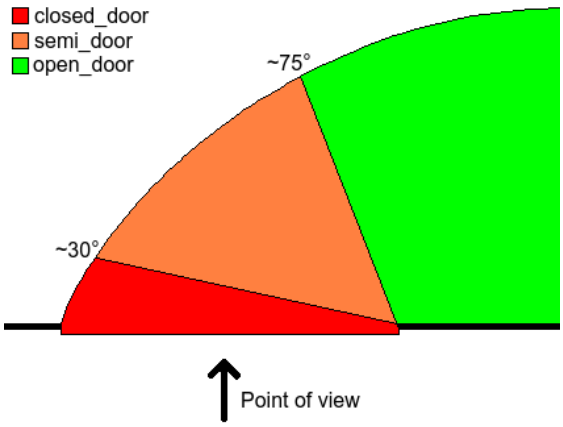


Abbildung 4.6: Zuordnung der Label durch Winkel anhand einer illustrierten Tür, die nach innen öffnet.
Roter Bereich: Tür wird als geschlossene gelabelt; **Orangener Bereich**: Tür wird als halbgeschlossene gelabelt; **Grüner Bereich**: Tür wird als offen gelabelt.

Ein Diskussionspunkt waren die offenen Türen bei denen man die Tür und zusätzlich den Türrahmen sah, da man bei ihnen lediglich den Türrahmen oder den Türrahmen *und* dazu die Tür in die Bounding Box aufnehmen kann. Es wurde sich dazu entschieden pro Bild beide Varianten aufzunehmen (*siehe Abbildung 4.7c*).

Im Stockwerk befinden sich Türen zu Stromkästen. Da es zu Missinterpretationen kommen kann, wurden diese Türen mit dem Label *no_door* gekennzeichnet.

(a) semi_door

(b) closed_door + no_door

(c) open_door überlappend

Abbildung 4.7: Illustrierte Beispiele für Bounding Boxes anhand von Bildern.
Links: Halbgeschlossene Tür mit einer roten Bounding Box;
Mitte: Geschlossene Tür und Stromkastentür mit einer roten und grünen Bounding Box;
Rechts: Offene Tür mit innerem Anschlag mit zwei Bounding Boxes, die sich überlappen

Um den Aufwand des Labeln zu reduzieren, wurden alle Bilder des HS händisch gelabelt und alle nachfolgend erstellten Datensätze aus Unterabschnitt 4.2.2 erhielten automatisiert die gleichen Labeldateien, da die Bounding Boxes stets die gleichen blieben.

4.4 Batch-Size

Eine Batch-Size wird beim Training für das neuronale Netz gebraucht, da dieser Wert anzeigt, wie viele Bilder des Trainingsdatensatzes pro Trainingsdurchgang verwendet werden. Nach jedem Durchgang werden die Gewichte der Neuronen des neuronalen Netzes aktualisiert [19]. Bei einer größeren Batch-Size wird mehr VRAM und RAM der Hardware verwendet.

Die Hardware ist daher ein limitierender Faktor beim Training. Um herauszufinden, in wie weit das volle Potenzial der Hardware genutzt werden kann, wurden die Batch-Sizes 2, 4, 8, 16, 32, 64 und 128 ausgewählt. Der Datensatz für das Experiment sollte gelabelt

sein und genügend Bilder enthalten, damit verwertbare Resultate entstehen. Dadurch, dass der Datensatz DD2[3] aufgrund der Schritte aus Abschnitt 4.3 bereits gelabelt ist und insgesamt 3000 Bilder enthält, wurde DD2 als Example verwendet. Dazu mussten die Bild- und Labeldatei mittels des Skriptes *generate_ tfrecord.py*[20] in das Tensorflow-Format *.record* konvertiert werden.

Für die Durchführung wurde das neuronale Netz SSD Mobilenet V2 eingesetzt. Für jede oben erwähnte Batchsize wurde ein eigenes Neuronales Netz implementiert und trainiert. Die Architektur des neuronalen Netzes blieb unverändert. Lediglich die Batch-Size wurde auf die jeweilige Zahl angepasst. Insgesamt hat jedes neuronale Netz 3000 Trainingsschritte durchgeführt. Mit Tensorflow wurde der Fortschritt anhand von Metriken schrittweise festgehalten und anschließend verglichen.

Da es relevant für die weiteren Abschnitte ist, ist vorgreifend zu sagen, dass der höchstmögliche Wert für die Batch-Size 16 ist. Für ein ausführliches Ergebnis siehe Abschnitt 5.1.

4.5 Training

Beim Training wird das jeweilige neuronale Netz mit Bildern trainiert, um Eigenschaften von Türen zu erlernen. Das Training bewirkt, dass die Eigenschaften auf bisher nicht gesehenen Türen ebenfalls erkannt werden und das neuronale Netz eine Erkennung eines Objektes ausgibt.

4.5.1 Erweiterter Hochschuldatensatz

Durch die Datensatzerweiterung aus Unterabschnitt 4.2.2 entstanden Bilddatensätze mit Labeln für die Methoden Helligkeit, Farbe, Kontrast, Löschen und Greenscreen. Es soll der unveränderte HS für das Training eines neuronalen Netzes verwendet werden und anschließend sollen neuronale Netze mit den erweiterten Bilddatensätzen trainiert werden. Die Metriken werden dann miteinander verglichen, um herauszufinden, ob die Methoden einen Mehrwert zur Steigerung der Robustheit des Türerkenners bieten und welche Methoden sich am besten dafür eignen.

Insgesamt wurde jeweils ein eigenes neuronales Netz mit jeweils einem der folgenden Datensätze trainiert:

- Helligkeit-Datensatz
- Farbe-Datensatz
- Kontrast-Datensatz
- Löschen-Datensatz
- Greenscreen-Datensatz

Dafür wurde das ausgewählte neuronale Netz SSD Mobilenet V2 verwendet und die Größe der Batch-Size in den Konfigurationseinstellung auf 16 gesetzt. Alle anderen Einstellungen blieben standardmäßig unverändert.

Der unveränderte HS wurde als Testdatensatz verwendet und enthält 113 Bilder. Die einzelnen erweiterten Datensätze enthalten ebenfalls 113 Bilder und wurden als Trainingsdatensätze verwendet. Auch hier hat das Skript *generate_tfrecord.py*[20] Verwendung gefunden, um die TFRecord-Dateien für Test- und Trainingsdatensatz zu erzeugen. Jedes neuronale Netz wurde mit 10000 Schritten trainiert, während Tensorflow die Metriken, wie beispielweise Loss oder mAP gemessen und aufgezeichnet hat.

4.5.2 Kombination von Datensätzen

Hinzukommend wurde untersucht, ob die Kombination von mehreren Datensätzen einen Vorteil bietet. Dazu wurden alle erweiterten Hochschuldatensätze vereint und es entstand der Kombi-Hochschuldatensatz (KHS) aus insgesamt 565 Bildern, wovon alle 565 Bilder dem Trainings- und die 113 Bilder des unveränderten HS dem Testdatensatz zugeordnet wurden.

Die TFRecord-Dateien wurden erzeugt und das neuronale Netz mit einer eingestellten Batch-Size von 16 durchlief 100000 Schritte.

Hintennach wurde der HS mit dem KHS vereinigt und derselbe Prozess mit ebenfalls 100000 Schritten Training durchgeführt.

Als letztes Experiment in diesem Abschnitt wurde der HS mit dem KHS und dem DD2 zusammengeführt und gleichermaßen mit 100000 Schritten trainiert.

Die erstellten Datensätze enthalten folgende Anzahl an Bildern, wie ebenfalls in Abbildung 4.8 als Diagramm dargestellt:

- **HS:** 113 Bilder
- **KHS:** 1657 Bilder
- **DD2:** 2700 Bilder
- **HS+KHS:** 1770 Bilder
- **HS+KHS+DD2:** 4470 Bilder

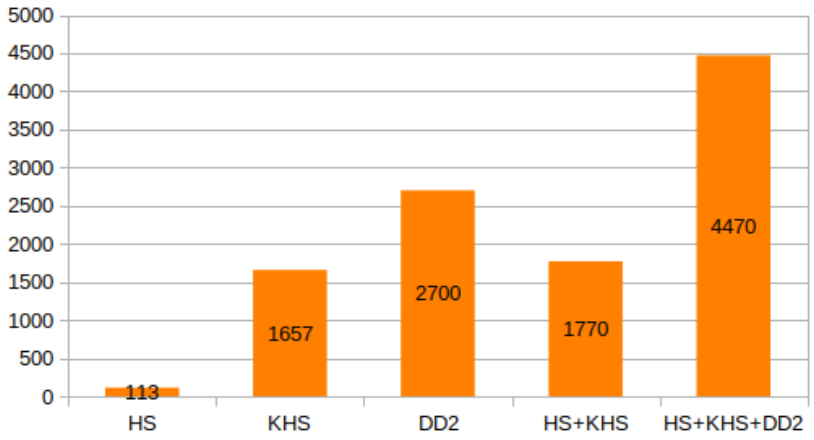


Abbildung 4.8: Anzahl der Bilder der Datensätze in einem Diagramm.
HS: 113 Bilder, KHS: 1657 Bilder, DD2: 2700 Bilder, HS+KHS: 1770 Bilder, HS+KHS+DD2: 4470 Bilder

Alle Experimente liefen unter dem Monitoring von Tensorflow. Während des Trainings wurden vom Framework nach jeden 100 Schritten Berechnungen der Metriken durchgeführt und gespeichert.

4.6 Einbettung Husky

Es wird auf den Programmablauf der Einbettung am Husky eingegangen.

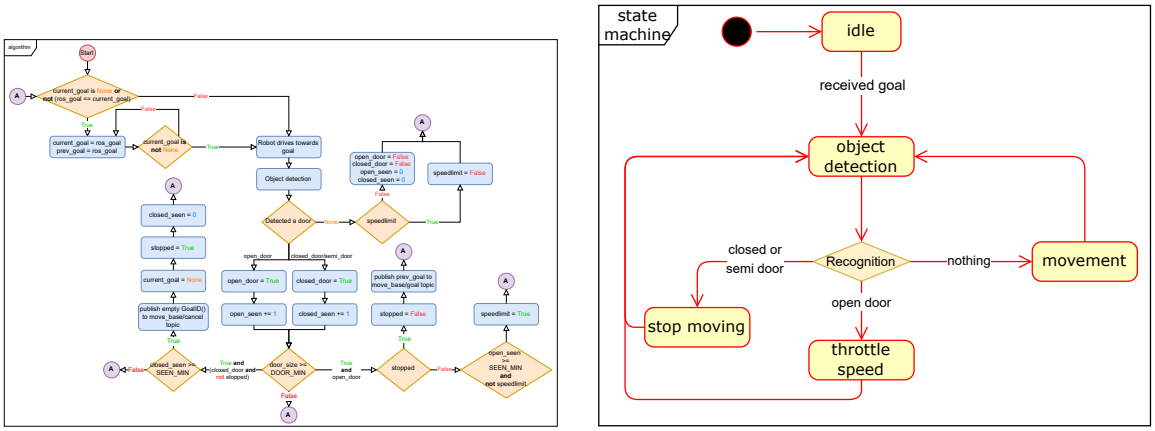
4.6.1 Programmablauf

In ROS wurde ein Node namens *door_detection* erstellt, welches die Bilder der Kamera mit Hilfe eines neuronalen Netzes und der Objekterkennung auswertet. Anhand dieser Objekterkennung wird ein spezifisches vordefiniertes Verhalten hervorgerufen.

Zu sehen ist eine Liste für ein besseres Verständnis der in Abbildung 4.9 vorkommenden Variablen und Begriffe:

- **current_goal : MoveBaseActionGoal** - derzeitiges zwischengespeichertes Ziel des Huskys
- **ros_goal : MoveBaseActionGoal** - derzeitiges Ziels des Huskys des ROS Topics *move_base/goal*
- **prev_goal : MoveBaseActionGoal** - vorheriges zwischengespeichertes Ziel des Huskys
- **speedlimit : Boolean** - gibt an, ob eine Geschwindigkeitbegrenzung gesetzt wurde
- **open_door : Boolean** - gibt an, ob eine offene Tür erkannt wurde
- **closed_door : Boolean** - gibt an, ob eine (halb-)geschlossene Tür erkannt wurde
- **open_seen : Integer** - gibt an, wie viele Male hintereinander eine offene Tür erkannt wurde
- **closed_seen : Integer** - gibt an, wie viele Male hintereinander eine (halb-)geschlossene Tür erkannt wurde
- **SEEN_MIN : Integer** - Mindestanzahl an Erkennungen hintereinander, bis eine Tür tatsächlich anerkannt wird
- **stopped : Boolean** - gibt an, ob der Husky sich im Stillstand befindet
- **door_size : Integer** - Größe der erkannten Tür
- **DOOR_MIN : Integer** - Mindestgröße einer Tür, damit sie als Tür anerkannt wird
- **move_base/goal** - ROS Topic auf dem die MoveBaseActionGoals für den Husky publiziert werden

- **move_base/cancel** - ROS Topic auf dem GoalIDs publiziert werden, um den Husky zu stoppen
- **GoalID** - ID eines MoveBaseActionGoals



(a) Flussdiagramm

(b) Zustandsdiagramm

Abbildung 4.9: Fluss- und Zustandsdiagramm der Türerkennung am Husky in ROS. Diagramme stellen den Ablauf der Türerkennung an dem Husky in ROS dar. Variablen und Begriffe werden in der Liste in Unterabschnitt 4.6.1 definiert.

In Bezug auf die Abbildung 4.9 und das Listing A.1 wird das Pythonskript beschrieben.

Durch das Starten des Pythonskriptes wird das ROS Node gestartet. Beim Initialisieren werden die Publisher- und Subscriber-Variablen erstellt auf denen man Funktionen aufrufen kann, um Nachrichten eines Topics zu empfangen oder Nachrichten an dieses zu senden. Zusätzlich werden das Modell des neuronalen Netzes und die Labelmap für die Zuordnung der Objekterkennung geladen.

Die aufgenommenen Bilder der Realsense werden auf dem Topic *realsense2/color/image_raw/compressed* veröffentlicht und da das Skript des ROS Nodes in einer Dauerschleife läuft, wird die Objekterkennung der Bilder des Realsense-Topics bis zum Beenden des Programmes vorgenommen. Wird kein Objekt erkannt, setzt der Husky seine geplanten Aktionen weiter fort.

```
1  (...)
2  else: # detected an object
3     if x_max > self.MIN_WIDTH and y_max > self.MIN_HEIGHT:
4     # check door size
5        if (detected_object == "closed_door" or detected_object ==
             "semi_door") and not self.stopped: # closed or semi
6           if self.closed_seen >= self.SEEN_MIN: # seen SEEN_MIN times
7              self.stop_husky()
8        elif detected_object == "open_door": # open
9           if self.stopped: # resume if stopped
10             self.resume_to_goal()
11             self.throttle_speed()
12          elif self.open_seen >= self.SEEN_MIN and not self.speedlimit: #
                seen SEEN_MIN times
13             self.throttle_speed()
14 (...)
```

Listing 4.1: Programmablauf der Objekterkennung in ROS/Python.
Codeschnipsel von der Else-Abzweigung, wenn ein Objekt erkannt worden ist.

In Zeile 2 der Listing 4.1 ist die Verzweigung der Else-Abzweigung zu sehen, wenn eine Tür erkannt wurde. In Zeile 3 wird überprüft, ob die Größe der erkannten Tür die geforderte Mindestgröße und -breite besitzt. Ist dem so, wird in Zeile 5 überprüft, ob es sich um eine (halb-)geschlossene Tür handelt. Halbgeschlossene und geschlossene Türen wurden zusammengefasst, da davon ausgegangen wird, dass der Roboter bei diesen Zuständen auf die Hilfe von Fremden angewiesen ist, um eine Tür so weit zu öffnen, dass der Husky keine Kollision bei der Durchfahrt verursacht.

Wenn die erkannte Tür eine (halb-)geschlossene Tür ist, wird in Zeile 6 überprüft, ob die Anzahl der hintereinander gesehenen (halb-)geschlossenen Tür größer gleich der Variable *SEEN_MIN* ist. Erst bei einem gleichem oder größeren Wert, wird das erkannte Objekt als Tür anerkannt und der Husky kommt in Zeile 7 zum Stillstand.

Wenn es keine (halb-)geschlossene Tür ist, wird in Zeile 8 überprüft, ob es sich um eine offene Tür handelt. Ist das erkannte Objekte eine offene Tür, dann wird in Zeile 9 überprüft, ob der Husky derzeit still steht. Steht der Husky still, dann soll der Husky seine Fahrt zu seinem vorherigen Ziel weiterführen und die Geschwindigkeitsbegrenzung wird gesetzt.

Befindet sich der Husky nicht im Stillstand, wird in Zeile 12 überprüft, ob die Anzahl der hintereinander gesehenen offenen Tür größer gleich der Variable *SEEN_MIN* ist. Ist die Überprüfung wahr, dann wird die Geschwindkeit des Huskys begrenzt und es kann weiter zu seinem Ziel fahren, da die Tür geöffnet ist.

Der Quellcode und alle weiteren zur Thesis dazugehörigen Dateien sind im internen GitLab der HAW und teilweise im Anhang dieser Arbeit zu finden.

5 Evaluation

Im vorletzten Kapitel Evaluation werden die Methoden Image Data Augmentation und die entstandenen Bilddatensätze anhand von Metriken und der Objekterkennung mit den neuronalen Netzen miteinander verglichen und die Einbettung am Husky untersucht.

Hinweis: In den nachfolgenden Tabellen dieses Kapitels wird das beste Ergebnis der jeweiligen Zeile farbig markiert.

5.1 Batch-Size

Beschreibung

Es wurden verschiedene Batch-Sizes miteinander verglichen, um das volle Potenzial der Hardware verwenden zu können. Als Datensatz wurde der Datensatz DeepDoors2 von Ramôa *et al.*[3] verwendet.

Insgesamt wurden acht neuronale Netzwerke erstellt, welche die Batch-Sizes 2, 4, 6, 8, 16, 32, 64 und 128 haben. Der Trainingsprozess mit einem neuronalen Netz, welches eine höhere Batch-Size als 16 hat, war aufgrund der Limitierung der Hardware nicht möglich. Beim Versuch eine höhere Batch-Size anzugeben, trat ein „Out of Memory"-Fehler auf. Somit waren lediglich die ersten vier neuronalen Netzwerke mit der Batch-Size 2, 4, 8 und 16 relevant für den Vergleich.

Um die Modelle gegeneinander zu messen, wurde die Metrik Total Loss gewählt, die durch Tensorflow aufgenommen und in Tensorboard visualisiert worden ist.

Beim Training für die Objekterkennung wurde für dieses und nachfolgende Experimente die folgende Hardware im JupyterLab[21] der HAW genutzt:

- **CPU:** 4x Intel Xeon Processor (Skylake, IBRS)
  (*Taktung und Kerne unbekannt*)

- **GPU:** Nvidia Tesla V100
- **VRAM:** 16 GB
- **RAM:** 16 GB
- **OS:** Linux Ubuntu 20.04.4 LTS

Ergebnisse

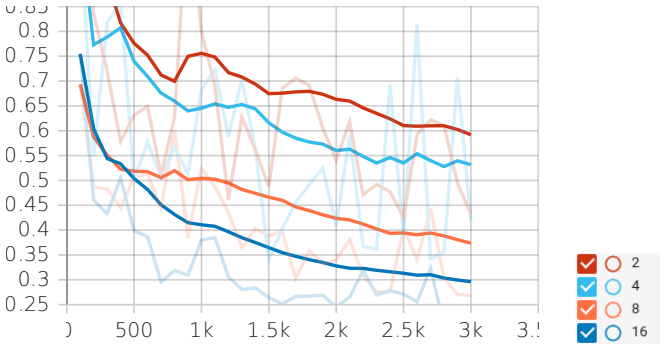


Abbildung 5.1: Abfallender Total Loss im Vergleich bei aufsteigenden Batch-Sizes. Die transparenten Graphen stellen die absoluten Werte dar und die farbigen Graphen stellen die Werte mit einer Glättung von 0.99 dar. **Rot**: Batch-Size 2; **Cyan**: Batch-Size 4; **Orange**: Batch-Size 8; **Blau**: Batch-Size 16.

Im Hintergrund des Diagrammes sind transparentfarbene Graphen zu sehen, die die tatsächlichen Werte des Total Loss darstellen. Für eine bessere Darstellung wurden alle Graphen in Tensorboard mit dem Faktor 0.99 geglättet.

Im Diagramm Abbildung 5.1 wird der Total Loss wiedergegeben, welcher bei der Batch-Size von 2 nach 3000 Schritten bei ≈ 0.59 lag. Die Batch-Size 4 erreichte den Wert ≈ 0.53. Im Vergleich zur Batch-Size 4 erfuhr der Total Loss mit einer Batch-Size von 8 eine Reduktion von $\approx 29\%$ auf ≈ 0.375. Mit der höchstmöglichen Batch-Size von 16 reduzierte sich der Wert auf ≈ 0.29 und somit um weitere $\approx 22\%$ im Gegensatz zur vorherigen Batch-Size.

Insgesamt erreichte die Batch-Size mit 16 den niedrigsten Total Loss Wert. Somit wurden im Verlauf der Experimente alle weiteren neuronalen Netze mit einer Batch-Size von 16 trainiert.

5.2 Metriken der Methoden

Beschreibung

Für die Steigerung der Robustheit wurden die Eigenschaften Helligkeit, Farbe und Kontrast des normale Hochschuldatensatzes (HS) jeweils seperat verändert und pro Methode ein eigener Datensatz erstellt. Das gleiche gilt für das Löschen von Bildausschnitten und dem Ersetzen des Greenscreens durch Bilder mit Rauschen. Außerdem wurde ein neuronales Netz trainiert, welches den normalen unveränderten HS als Trainingsdatensatz erhielt, um einen Referenzwert für die Metriken zu erhalten. Insgesamt hat jedes Netz 10000 Trainingsschritte durchlaufen und jede 1000 Schritte wurden die Metriken mAP mit einem Schwellenwert von 0.5, 0.75 und dem Durchschnitt aller mAP-Werte von 0.5 bis 0.95 in 0.05er-Schritten. Zusätzlich wurde der Recall bei einem Relevanzwert von 1, 10 und 100, sowie der Total Loss gemessen. All diese Metriken wurden nach dem Training nochmals mittels des Pythonskriptes *model_main_tf2.py*[22] berechnet und notiert.

Ergebnisse

	Brightness	**Color**	**Contrast**	**Erasing**	**Greenscreen**
mAP@0.50-0.95	-5,14%	-4,29%	**-1,39%**	-3,67%	-12,13%
mAP@0.50	-1,6%	-0,73%	**+2,42%**	+1,46%	-8,29%
mAP@0.75	-0,96%	-1,21%	**+3,02%**	+1,87%	-8,02%
Recall@1	-8,16%	-5,06%	**+3,68%**	-0,27%	-2,68%
Recall@10	-9,11%	-5,14%	**+1,64%**	+0,28%	-0,84%
Recall@100	-6,24%	-2,69%	**+1,33%**	+0,59%	-1,46%
Total Loss	-9,92%	+41,47%	-11,62%	**-11,68%**	+168,79%

Tabelle 5.1: Robustheitssteigernde Methoden im prozentualen Vergleich zum HS. Neuronales Netz mit Kontrast-Bilddatensatz erreichte höchsten positiven Unterschied im Vergleich zum HS.

Die Tabelle 5.1 zeigt, dass kein Datensatz bei der Metrik mAP@0.50-0.95 einen höheren Wert als das neuronale Netz mit dem unveränderten HS erreichen konnte. Bei allen anderen Metriken ist es anders, denn hier gibt es mindestens den Kontrast-Hochschuldatensatz, der einen höheren Wert der Metrik erzielte.

Der Kontrast-Hochschuldatensatz erreichte die höchsten Ergebnissen von $\approx -1,39\%$ bis $\approx +3,68\%$ in allen Metriken. Auch der Löschen-Hochschuldatensatz realisierte zwei Mal beim mAP und zwei Mal beim Recall einen positiveren Wert als der HS, aber jeweils niedrigere Werte, als der Kontrast-Hochschuldatensatz. Ausschließlich beim Total Loss hat der Löschen-Hochschuldatensatz einen um $\approx 0,06\%$ niedrigeren Total Loss als der Kontrast-Hochschuldatensatz.

	Brightness	**Color**	**Contrast**	**Erasing**	**Greenscreen**	**Normal**
mAP@0.50-0.95	0.7200	0.7265	0.7482	0.7312	0.6670	**0.7590**
mAP@0.50	0.7702	0.7770	**0.8017**	0.7942	0.7178	0.7827
mAP@0.75	0.7700	0.7680	**0.8009**	0.7920	0.7151	0.7774
Recall@1	0.7183	0.7426	**0.8109**	0.7800	0.7612	0.7821
Recall@10	0.7716	0.8053	**0.8629**	0.8513	0.8418	0.8489
Recall@100	0.8009	0.8313	**0.8656**	0.8593	0.8418	0.8542
Total L.	0.1481	0.2330	0.1453	**0.1452**	0.4419	0.1644

Tabelle 5.2: Robustheitssteigernde Methoden mit absoluten Werten im Vergleich. Neuronales Netz mit Kontrast-Bilddatensatz konnte bei fünf von sieben Metriken den Bestwert erwirtschaften.

Das Netz mit dem HS hat den besten Wert in Höhe von 0.759 für die mAP mit einem Schwellenwert von 0.5 bis 0.95 erreicht. Beim Kontrast-Hochschuldatensatz erzielte das Netz den besten Wert von 0.8017 für die mAP mit den Schwellenwert 0.5 und beim Schwellenwert 0.75 eine mAP von 0.8009.

Ebenfalls erreichte der Konstrast-Hochschuldatensatz beim Recall die höchsten Werte. Beim Faktor 1 betrug der Wert 0.8109, beim Faktor 10 den Wert 0.8629 und beim Faktor 100 den Wert 0.8629.

Insgesamt erreicht der Löschungs-Hochschuldatensatz den niedrigsten Total Loss von 0.1452 und war damit um 0.0001 besser als der Kontrast-Hochschuldatensatz, welcher einen Total Loss von 0.1453 erreichte.

Die Datensätze Helligkeit, Kontrast und Löschung erreichten einen niedrigeren Total Loss als der HS. Jedoch erreichte der Farb-Hochschuldatensatz einen $\approx 29\%$ höheren Wert. Ebenso erzielte der Greenscreen-Hochschuldatensatz einen um $\approx 63\%$ höheren Total Loss als der HS. Ein höherer Loss ist allerdings negativ ausgelegt.

Es entsteht folgende Platzierung, wenn man die neuronalen Netze mit den Datensätzen der Leistung nach sortiert:

1. Contrast
2. Erasing
3. Color
4. Brightness
5. Greenscreen

Resultierend kann man sagen, dass die Methode der Konstrast-Erhöhung und -Absenkung Maximalwerte für fünf von sieben Metriken vollbracht hat. Danach kommt das Löschen von Bildausschnitten auf Platz zwei, gefolgt von dem Verändern der Farbe, der Helligkeit und auf dem letzten Platz ist das Greenscreen-Verfahren.

5.3 Metriken der Datensätze

Beschreibung

Alle in diesem Experiment vorkommenden neuronalen Netze wurden mit 10000 Trainingsschritten trainiert. Nach dem Training wurden die Metriken mAP, Recall und Total Loss mittels des Pythonskriptes *model_main_tf2.py*[22] ermittelt und festgehalten.

Beispielsweise HS+KHS ist das neuronale Netz, welches die Datensätze HS und KHS kombiniert und als Trainingsdatensatz verwendet hat. Es wurde zusätzlich der KHS2 erstellt, welcher alle Image Data Augmentation Bilder des KHS enthält, bis auf die des Greenscreen-Verfahrens. Außerdem wurde der DD2 von Ramôa *et al.*[3] verwendet, um zu sehen, ob ein externer Datensatz in dem Hochschulszenario des Erkennens von Türen von Vorteil ist.

Ergebnisse

	HS	**KHS**	**KHS2**	**DD2**	**HS+KHS**	**HS+KHS+DD2**
mAP@0.50-0.95	0.7590	0.7735	0.7783	0.5641	**0.7826**	0.7779
mAP@0.50	0.7828	0.7798	0.7845	0.6065	0.7924	**0.7989**
mAP@0.75	0.7774	0.7795	0.7839	0.5699	0.7920	**0.7948**
Recall@1	0.7821	0.7818	0.7967	0.6214	0.7853	**0.7970**
Recall@10	0.8489	0.8198	**0.8787**	0.7022	0.8727	0.8331
Recall@100	0.8543	0.8465	0.8801	0.7189	**0.9121**	0.8797
Total L.	0.1644	0.0580	**0.0539**	0.2584	0.0541	0.0787

Tabelle 5.3: Neuronale Netze mit Datensätzen im Vergleich mit absoluten Werten. DD2 erzielte niedrigsten Werte. HS+KHS+DD2 hatte Höchstwerte in drei von sieben Metriken.

	KHS	KHS2	DD2	HS+KHS	HS+KHS+DD2
mAP@0.50-0.95	+1.91%	+2.54%	-25.68%	**+3.11%**	+2.49%
mAP@0.50	-0.38%	+0.22%	-22.14%	+1.23%	**+2.06%**
mAP@0.75	+0.27%	+0.84%	-26.69%	+1.88%	**+2.24%**
Recall@1	-0.04%	+1.87%	-20.55%	+0.41%	**+1.91%**
Recall@10	-3.43%	**+3.51%**	-17.28%	+2.80%	-1.86%
Recall@100	-0.91%	+3.02%	-15.85%	**+6.77%**	+2.97%
Total L.	-64.72%	**-67.21%**	+57.17%	-67.09%	-52,13%

Tabelle 5.4: Neuronale Netze mit Datensätze im prozentualen Vergleich zum HS DD2 erzielte niedrigsten Werte. HS+KHS+DD2 hatte Höchstwerte in drei von sieben Metriken.HS+KHS errung höchsten prozentualen Unterschied im Vergleich zum HS.

Den Tabellen 5.3 und 5.4 kann entnommen werden, dass HS+KHS den höchsten Wert für die mAP@0.50-0.95 hat und dabei einen $\approx 3.11\%$ höheren Wert als der HS besitzt. Der HS+KHS+DD2 hat bei der Metrik mAP@0.50 den höchsten Wert mit einer Steigerung von $\approx 2.06\%$ und auch beim mAP@0.75 gibt es eine Erhöhung von $\approx 2.24\%$ gegenüber dem unveränderten HS.

Gleichermaßen beim Recall@1 erzielte der HS+KHS+DD2 den größten Wert und ist damit $\approx 1.91\%$ größer als der Wert des HS. Der KHS2 realisierte beim Recall@10 einen um $\approx 3.51\%$ höheren Wert als der Vergleichswert. Der Maximalwert für die Metrik Recall@100 kommt vom HS+KHS und liegt $\approx 6.77\%$ höher als der Wert des HS.

Abschließend beim Total Loss bewirkte KHS2 eine Reduktion von $\approx 67.21\%$ im Vergleich zum Total Loss des HS.

In toto realisierte das Modell mit dem DD2 die niedrigsten Werte und das neuronale Netz HS+KHS+DD2 schuf mit drei höchsten Werten die meisten Bestplatzierungen. Hingegen erzielte der HS+KHS insgesamt prozentual den höchsten positiven Unterschied im Vergleich zum HS.

5.4 Erkennung von Türen

Beschreibung

Das Testen des Türerkenners in der echten Welt unter realen Bedingungen ist ein Bestandteil der Überprüfung auf Robustheit. Dazu wurde eine Intel RealSense D435 an ein Lenovo Ideapad 5 14ARE05 angeschlossen, welcher folgende Systemkonfiguration enthält:

- **CPU:** AMD Ryzen 5 4500 U
- **GPU:** AMD Radeon Graphics Vega 6
- **VRAM:** 2 GB
- **RAM:** 16 GB
- **OS:** Linux Ubuntu 20.04.4 LTS

(a) Lenovo Ideapad 14ARE05[23] (b) Intel RealSense D435[24]

Abbildung 5.2: Verwendete Hardware für das Experiment zur Erkennung von Türen. **Links:** verwendeter Laptop, **Rechts:** verwendete Tiefenkamera

Die Kamera wurde nacheinander auf die Positionen 1 bis 7 aus der Abbildung 5.3 aufgestellt und die Linse auf die Tür gerichtet, wodurch die komplette Tür von der Kamera erfasst wurde. Dabei wurde jeweils eine Entfernung von 2 Metern zur Tür eingehalten. Danach wurden die aufgenommenen Bilder gespeichert, damit jedes neuronale Netz die gleichen Bilder erhält. Folgend wurde ein Python-Skript gestartet, welches dem jeweiligen neuronalen Netz (*Varianten siehe Tabelle 5.5*) die Bilder als Input sequenziell übergibt und dieses Netz eine Vorhersage über das erkannte Objekt im Bild ausgibt.

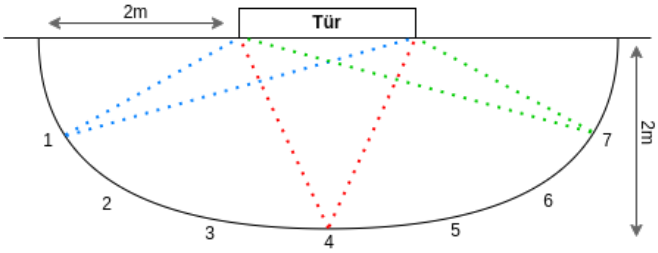


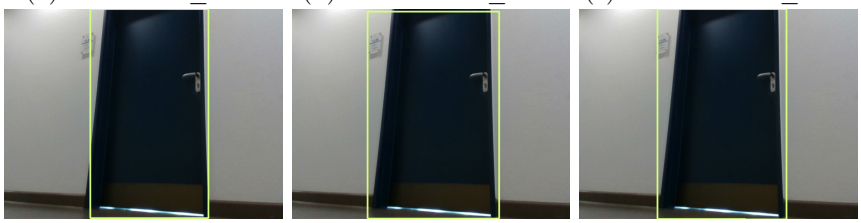
Abbildung 5.3: Betrachtungswinkel und -positionen der Kamera auf eine Tür mit einer Entfernung von zwei Metern.
Blaue Striche zeigen Betrachtungswinkel von Position 1. Rote Striche zeigen Betrachtungswinkel von Position 4. Grüne Striche zeigen Betrachtungswinkel von Position 7. Von den jeweiligen Betrachtungspositionen aus sind zwei Meter in direkter Luftlinie zur Tür eingehalten worden.

Die Kamera wurde am Boden platziert, um den Betrachtungswinkel der unteren Kamera des Huskys zu imitieren. Anschließend wurde die Kamera in einer ungefähren Höhe von 1,5 Metern gehalten, um den Betrachtungswinkel der bewegbaren Kamera am Arm des Huskys zu imitieren.

Somit wurden bei jedem Datensatz pro Zustand jeweils vier Bilder aus den beiden beschriebenen Betrachtungswinkeln aufgenommen und das Ergebnis wurde notiert. Insgesamt wurden pro Zustand 56 Bilder aufgenommen und dem jeweiligen neuronalen Netz zur Vorhersage übergeben.

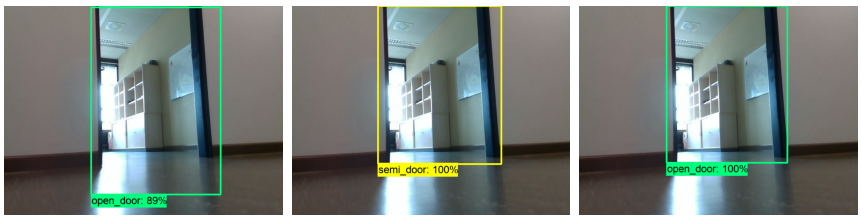
Ergebnisse

(a) HS: closed_door (b) KHS: closed_door (c) KHS2: closed_door

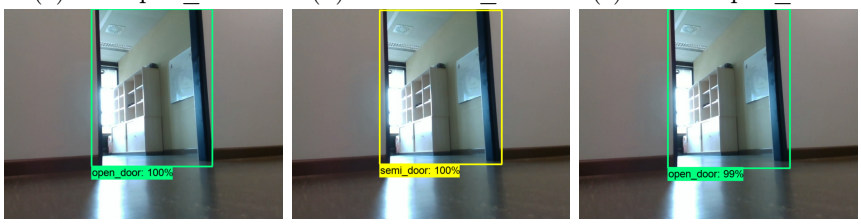


(d) DD2: closed_door (e) HS+KHS: closed_door (f) HS+KHS+DD2: closed_door

Abbildung 5.4: Einheitlich positive Erkennung einer geschlossenen Tür aller neuronalen Netze.
Sechs Bilder von Erkennungen von sechs neuronalen Netzwerken, die alle eine geschlossene Tür korrekt erkennen.



(a) HS: open_door (b) KHS: semi_door (c) KHS2: open_door



(d) DD2: open_door (e) HS+KHS: semi_door (f) HS+KHS+DD2: open_door

Abbildung 5.5: Unterschiedliche Erkennungen der neuronalen Netze von einer offenen Tür.
Sechs Bilder von Erkennungen von sechs neuronalen Netzwerken bei denen vier Erkennungen eine offene Tür korrekt erkennen und zwei neuronale Netze erkennen zu 100% eine halbgeschlossene Tür.

Alle neuronalen Netze in Abbildung 5.4 haben korrekterweise eine geschlossene Tür erkannt. Hingegen in Abbildung 5.5 haben die neuronalen Netze KHS und HS+KHS eine offene Tür als halbgeschlossene Tür vorhergesagt.

	HS	**KHS**	**KHS2**	**DD2**	**HS+KHS**	**HS+KHS+DD2**
Open	42%	**50%**	21%	21%	**50%**	28%
Semi	57%	78%	**85%**	57%	64%	78%
Closed	71%	71%	50%	64%	78%	**92%**
Total	57%	**66%**	52%	47%	64%	**66%**

Tabelle 5.5: Korrekte Erkennungen von Türen je Modell in Prozent.
KHS und HS+KHS+DD2 realisierten mit 66% die höchsten korrekten Erkennungsraten aller sechs neuronalen Netze.

Die besten Ergebnisse beim offenen Zustand haben der KHS und HS+KHS mit einem Prozentsatz von 50% erreichen können. Bei den halbgeschlossenen Türen erlangte das neuronale Netz KHS2 den höchsten Wert mit 85%. Mit dem Maximalwert von 92% bei den geschlossenen Türen realisierte der HS+KHS+DD2 alleine die höchste positive Erkennungsrate.

Folgend kann man sagen, dass der KHS und der HS+KHS+DD2 die besten Ergebnisse erreicht haben, da diese beiden Datensätze in der Gesamtauswertung mit 66% dasselbe höchste Ergebnis erreicht haben. Ebenfalls wurden die Punkte 1 und 2 der Definition der Robustheit unter Abschnitt 4.1 erlangt.

Vergleicht man die Ergebnisse der beiden neuronalen Netze, erkennt man, dass der KHS eine 22% höhere Erkennungsrate bei offenen Türen hat, als das neuronale Netz mit dem HS+KHS+DD2. Einen identischen Wert von 78% erkannt man bei den halbgeschlossenen Türen. Wiederum bei den geschlossenen Türen hat HS+KHS+DD2 eine 21% höhere Erkennungsrate als KHS.

Vergleicht man HS+KHS+DD2 und KHS einzeln mit HS, dann lässt sich feststellen, dass KHS als einziger der beiden Datensätze eine im Schnitt $\approx 9.6\%$ höhere Erkennungsrate bei allen Zuständen hat als HS. Während HS+KHS+DD2 bei offenen Türen eine 14% niedrigere Erkennungsrate hat als HS.

Ergebnisse (Frontansicht)

	HS	KHS	KHS2	DD2	HS+KHS	HS+KHS+DD2
Open	50%	**66%**	41%	50%	33%	33%
Semi	50%	**83%**	71%	50%	66%	**83%**
Closed	83%	**100%**	**100%**	**100%**	**100%**	**100%**
Total	61%	**83%**	70%	66%	66%	72%

Tabelle 5.6: Korrekte Erkennungen von Türen je Modell in Prozent (*nur Frontansicht*). Bei der Frontansicht konnte KHS 83% bei der Erkennungsrate erwirken.

Aufgrund dessen, dass der Husky bei der Verwendung jeglichen Wegfindungsalgorithmuses vor dem Durchfahren einer Tür sich frontseitig gerade vor einer Tür platziert, um diese auch möglichst gerade mit wenig Bewegung nach links und rechts zu durchfahren, wurde die Tabelle 5.6 erstellt. Beim Experiment wurden entsprechend lediglich die Betrachtungswinkel 3, 4 und 5 der Abbildung 5.3 betrachtet, um die Frontansicht zu simulieren. Der Husky kann sich somit vor dem tatsächlichen Durchfahren einer Tür frontal vor die Tür platzieren und die Türerkennung starten, um sich zu vergewissern, welchen Zustand die Tür hat, bevor diese gewiss durchfahren wird.

Laut den Ergebnissen ist der Zustand der geschlossenen Tür am fehlerfreisten zu erkennen, da hier bei allen Varianten der Datensätze eine Erkennungsrate von 83% bis 100% erreicht wurde. Bei den halbgeschlossenen Türen wurde eine Erkennungsrate von 50% bis 83% erreicht. Offene Türen wurden mit einer Erkennungsrate von 33% bis 66% korrekt erkannt, womit die Fehleranfälligkeit bei Erkennungen von offenen Türen am höchsten ist.

Der KHS hat bei den offenen Türen den höchsten Wert mit 66% korrekt klassifizierten Türen erreicht. Bei den halbgeschlossenen Türen erzielten der KHS und HS+KHS+DD2 mit 83% dieselbe höchste Erkennungsrate. Bis auf den HS haben alle anderen neuronalen Netze alle geschlossenen Türen richtig klassifiziert und daher den Wert 100% erreichen können.

Durch die der Image Data Augmentation und die Kombination der entstandenen Bilder war es möglich den Trainingsdatensatz KHS zu erstellen, der die insgesamt höchste Erkennungsrate verwirklichte. Dabei wurde eine Wachstum von 22% im Vergleich zum HS festgestellt, welcher die unveränderten Bilder besaß. Dadurch kann man folgern, dass

Image Data Augmentation einen positiven Einfluss auf die Erkennungsrate im Vergleich zum Modell mit dem unveränderten Datensatz hat.

Den Erwartungen entgegen brachte das Entfernen der Greenscreen-Methode im KHS2 keinen Vorteil. Die durchschnittliche Erkennungsrate in Tabelle 5.5 und Tabelle 5.6 war hierbei um 14% und 13% niedriger als im Vergleich zum KHS, welcher die Greenscreen-Methode enthielt. Im Korrelat zum HS hat das Netz mit dem KHS2 jedoch eine Steigung von 9% verzeichnen können.

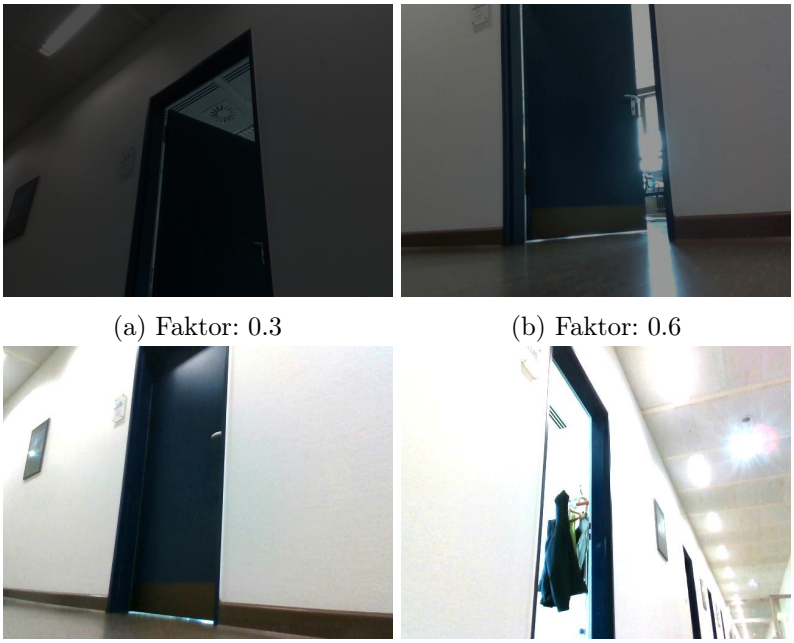
Das Zusammenfügen des unveränderten Datensatzes (HS) und dem kombinierten Image Data Augmenation Bildern (KHS) brachte ingesamt eine Verbesserung von 5% bei der Erkennung. Eine gleiche 5%ige Erhöhung zum HS hat das neuronale Netz mit dem DD2 erzielen können. Der HS+KHS+DD2 enthielt zusätzlich den Datensatz DD2 und mit der Fusionierung mit diesem externen Datensatz konnte ein Anstieg von 11% entstehen im Vergleich zum HS.

5.5 Lichtverhältnisse

Beschreibung

Es wurden dieselben Bilder, wie im Abschnitt 5.4 verwendet. Die Helligkeit jeden Bildes wurde verändert, so dass vier Exemplare zu jedem Originalbild entstanden. Dabei wurden mittels Augmentor zwei niedrigere Helligkeits- und zwei höhere Helligkeitsgrade verwendet. Der Grad der Helligkeit wurde realitätsnah angepasst, damit es zu keinen unrealistisch hell oder dunklen Bildern kommt. Als Referenzwerte wurden die Mindest- und Höchsthelligkeit der Büroräume in der HAW angenommen bei denen die Flure mindestens mit dimmendem Notfalllicht oder zusätzlich durch Sonnenstrahlen beleuchtet wurden.

Es wurde sich für die Überprüfung des Umweltverhältnisses Helligkeit entschieden, da dies eine Eigenschaft der Umgebung ist, die schwer zu verändern ist. Sonneneinstrahlungen lassen sich in gewisserweise dämmen, jedoch ist die Beeinflussung des Notfalllichtes ohne administrative Rechte nicht möglich.



(a) Faktor: 0.3 (b) Faktor: 0.6

(c) Faktor: 1.5 (d) Faktor: 2.0

Abbildung 5.6: Beispiele für Validierungsbilder von Türen mit verschiedenen Helligkeitsfaktoren.
Zu Erstellung wurd das Python Package Augmentor [2] verwendet und die Faktoren 0.3, 0.6, 1.5 und 2.0 wurden festgelegt.

Ergebnisse

	HS	**KHS**	**KHS2**	**DD2**	**HS+KHS**	**HS+KHS+DD2**
Open	58%	21%	21%	**60%**	33%	25%
Semi	28%	82%	**83%**	62%	60%	**83%**
Closed	**66%**	65%	57%	28%	51%	38%
Total	51%	**56%**	54%	50%	48%	49%

Tabelle 5.7: Korrekte Erkennungen von Türen unter unterschiedlich starken Lichtverhältnissen in Prozent

Wie in der Tabelle 5.7 festzustellen, hat das Modell mit dem DD2 den positivsten Wert bei der Erkennung von offenen Türen erzielt. Den höchsten Wert mit 83% bei der Erkennung von halbgeschlossenen Türen erreichten KHS2 und HS+KHS+DD2. Zuletzt bei den

geschlossenen Türen hat der unveränderte HS den größten Wert mit 66% Erkennungsrate erreichen können.

Bei der Berechnung der durchschnittlichen Erkennungsrate realisierte der KHS den Bestwert mit 56% und erlangte einen 5% höheren Wert als HS. Somit ist auch in diesem Helligkeits-Experiment das neuronale Netz mit dem KHS das Netz, welches die korrekteste Objekterkennungsrate besitzt.

Die Modelle DD2, HS+KHS und HS+KHS+DD2 erfuhren sogar einen 1%, 3% und 2% niedrigeren Wert als HS.

5.6 Zustandsänderungen

Beschreibung

In diesem Video Door State Detection wird das neuronale Netz mit dem Datensatz KHS verwendet, um den Zustand einer Tür zu erkennen, da dieses Netz in Abschnitt 5.4 und Abschnitt 5.5 die Bestwerte erreichen konnte.

Zuerst zeichnet die Kamera ein Video von einer Tür von links, dann frontal und anschließend von rechts auf, während die Objekterkennung über ein Python-Skript läuft und den Zustand der Tür mittels einer Bounding Box anzeigt. Während jeder Aufnahme aus der jeweiligen Perspektive durchläuft eine Person die Tür, um den Zustand der Tür zu verändern und um eine teilweise Verdeckung der Tür darzustellen.

Für diesen Versuch wurden der Laptop Lenovo Ideapad 5 14ARE05 (*Spezifikationen siehe Abschnitt 5.4*) und die Tiefenkamera Intel Realsense D435 verwendet.

Ergebnisse

Abbildung 5.7: Schnappschuss des Videos Door State Detection.
Eine Person durchläuft eine halbgeschlossene Tür während der Türerkenner eine Bounding Box vorhersagt, die die Tür korrekt vorhersagt.

Die Aufnahme von links ist konstant robust und erkennt ein einziges Mal bei Sekunde 16 die offene Tür inkorrekt als halbgeschlossen Tür. Dazu wird zweitweise kein erkanntes Objekt angezeigt.

Beim Zeitstempel 0:24 zeigt die Erkennung aus der frontalen Sicht kurzzeitig an, dass die halbgeschlossene Tür eine offene Tür ist, als eine Person durch die Tür geht. Abgesehen von diesem Fehler ist die Erkennung jedoch robust, wenn ein Objekt die Tür verdeckt.

Aus der rechten Ansicht erkennt das neuronale Netz bereits zu Anfang die offene Tür als eine halbgeschlossene Tür, obwohl dies nicht korrekt ist. Erst als eine Person durch die Tür geht, zeigt die Erkennung an, dass die Tür geöffnet ist. Während des Schließvorganges der Tür wird für insgesamt eine Sekunde kein Zustand der Tür angezeigt.

Bei der Änderung der Zustände gibt es zwar auch fehlerhafte oder gar keine Erkennungen, aber zum Großteil der Zeit ist die Erkennung akkurat und auch bei der Verdeckung der Tür durch eine Person wird der Zustand der Tür zuverlässig erkannt.

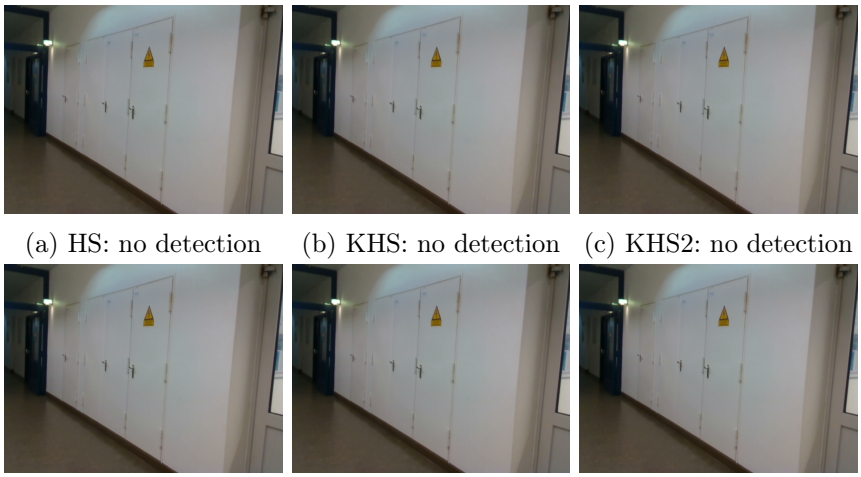
5.7 Erkennung von Nicht-Türen

Beschreibung

Es wurde der gleiche Laptop, die gleiche Tiefenkamera und das gleiche Pythonskript, wie in Abschnitt 5.4 verwendet. Ausschließlich die Ausführung unterscheidet sich gegenüber dem vorherigen Experiment, denn es wurden Aufnahmen von Stromkästen, die als Nicht-Türen definiert sind, aus verschiedenen Betrachtungswinkeln aufgenommen und den Türerkennern als Input übergeben.

Für das ganze Experimente wurden jedem Datensatz 24 Bilder von Nicht-Türen übermittelt und das Ergebnis der Auswertung für die Ergebnisse notiert.

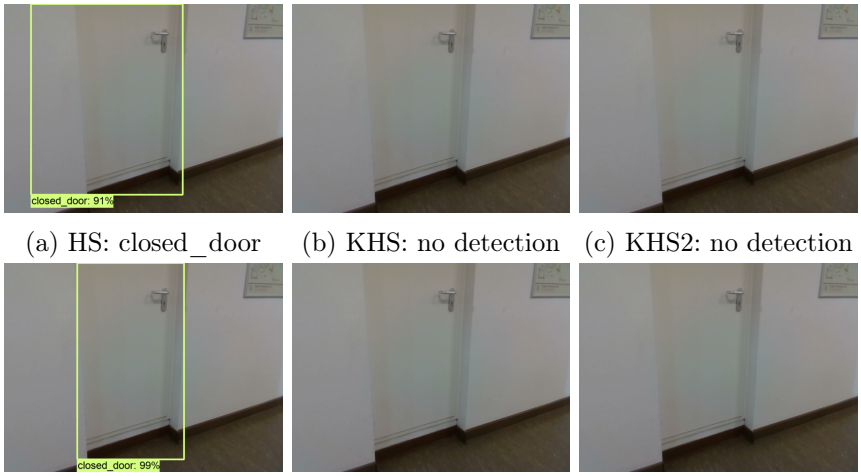
Ergebnisse



(a) HS: no detection (b) KHS: no detection (c) KHS2: no detection

(d) DD2: no detection (e) HS+KHS: no detection (f) HS+KHS+DD2: no detection

Abbildung 5.8: Keine falschen Erkennungen einer Nicht-Tür aller Varianten der neuronalen Netze.
Sechs Bilder von einer Stromkasten Tür, auf denen sechs neuronalen Netze keine Erkennung vorhersagen.



(a) HS: closed_door (b) KHS: no detection (c) KHS2: no detection

(d) DD2: closed_door (e) HS+KHS: no detection (f) HS+KHS+DD2: no detection

Abbildung 5.9: Unterschiedliche Erkennungen einer Nicht-Tür aller Varianten der neuronalen Netze.
Sechs Bilder einer Stromkastentür von sechs neuronalen Netzen bei denen zwei neuronale Netze fälschlicherweise eine geschlossene Tür erkannt haben.

In der Abbildung 5.8 sieht man, dass alle neuronalen Netze bei Sicht einer Stromkastentür korrekterweise keine Erkennung vorhersagen. Im Gegensatz dazu werden in Abbildung 5.9 beim HS und DD2 jeweils eine geschlossene Tür erkannt, wobei KHS, KHS2, HS+KHS und HS+KHS+DD2 keine Erkennung vorhersagen.

	HS	**KHS**	**KHS2**	**DD2**	**HS+KHS**	**HS+KHS+DD2**
No door	29%	**0%**	**0%**	62%	**0%**	**0%**

Tabelle 5.8: Falsche Erkennungen von Nicht-Türen je Modell in Prozent.
Außer HS und DD2 haben alle anderen neuronalen Netze 0% der Nicht-Türen falsch erkannt.

Eine Falscherkennungsrate von 0% konnten die Datensätze KHS, KHS2, HS+KHS, HS+KHS+DD2 erzielen und somit wurde der Punkt 3 aus der Definition der Robustheit unter Abschnitt 4.1 erreicht. Der Datensatz HS hat 29% der Nicht-Türen als eine offene, halbgeschlossene oder geschlossene Tür erkannt und der Datensatz DD2 hat im Vergleich dazu 62% der Nicht-Türen einem falschen Zustand zugeordnet. Das sind 113% mehr inkorrekt klassifizierte Nicht-Türen als beim HS.

Es wurde kein einziges Mal eine Nicht-Tür vorhergesagt, sondern es gab keine Erkennung jeglichen Objektes. Weshalb dies der Fall war, ließ sich nicht beantworten. Somit lässt sich auch nicht klar sagen, dass das Erweitern des Trainingsdatensatzes durch ähnliche Objekte die Robustheit eines Objekterkenners steigert.

5.8 Husky

Bevor Implementierungen am echten Husky Roboter getestet werden, werden neue Skripte o.ä. in der Simulation Gazebo getestet, in welcher der simulierte Husky Roboter mit einer hohen Übereinstimmung zum Original hard- und softwaretechnisch konfiguriert ist.

Beschreibung

Eine Umgebung wurde in Gazebo erstellt, die ein mögliches Einsatzgebiet für den Husky wiederspiegelt. Dabei sind drei Türen mit dem Zustand offen, halbgeschlossen und geschlossen mit Abständen von 8 Metern in einer Holzwand eingebaut. Da die Tür dunkelgrau und der Hintergrund holzfarben ist, ist die Tür zumindest für den Mensch optisch vom Hintergrund auseinanderzuhalten.

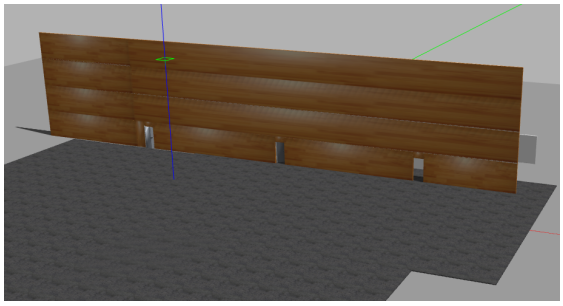


Abbildung 5.10: Umgebung in Gazebo.
Eine lange, hohe Holzwand enthält drei Türen, die die Zustände offen, halbgeschlossen und geschlossen angenommen haben. Der Untergrund der Umgebung ist Grau.

Der Husky besitzt eine fest montierte Tiefenkamera an der Plattform (*siehe „Intel RealSense“ in Abbildung 5.11*), die in der Tabelle 5.9 als RS bezeichnet wird und zusätzlich eine frei bewegbare Tiefenkamera am Greifarm (*siehe „Intel RealSense 2“ in Abbildung 5.11*), welche in der Tabelle 5.9 als RS2 angegeben wird.

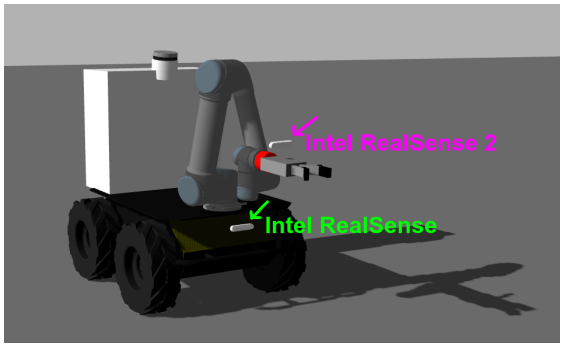


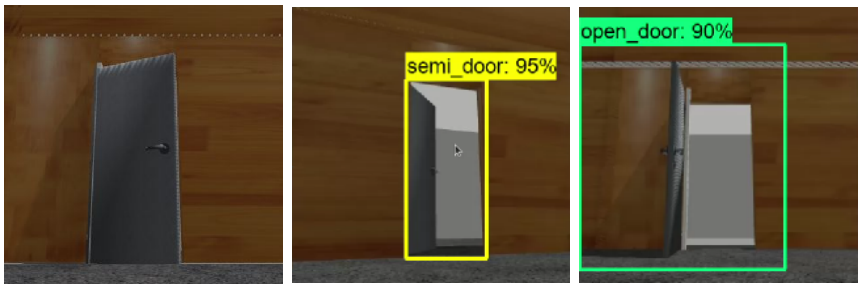
Abbildung 5.11: Intel RealSense Kameras am Husky Roboter in der Simulation Gazebo. Die Intel RealSense ist an der Fahrplattform fest montiert und in Bodennähe. Die Intel RealSense 2 ist am Greifarm montiert und durch den Arm frei bewegbar.

Für den Versuch wurden die Betrachtungswinkel 3, 4 und 5 aus Abbildung 5.3 verwendet und der Husky wurde an die jeweilige Stelle positioniert. Je Betrachtungswinkel wurden zehn Durchgänge durchgeführt. Entsprechend wurden je Tür und Zustand insgesamt 30 Durchgänge durchgeführt und ausgewertet. Für die Nicht-Tür wurde ein Bereich der Holzwand aufgenommen bei dem keine anderen Objekte im Winkel der Kamera zu sehen waren. Das Ganze wurde jeweils für die Kamera RS und RS2 umgesetzt, wobei bei der Kamera RS2 der Greifarm bewegbar ist und der Arm solange vertikal von oben nach unten bewegt worden ist, bis eine Erkennung zustande kam oder es zu keiner Erkennung kam und der Arm nicht weiter nach unten bewegt werden konnte.

Eine Tür gilt als korrekt erkannt, wenn der Türerkenner die Tür länger als zwei Sekunden ohne Unterbrechung korrekt erkennt. Der Schwellenwert von zwei Sekunden wurde gewählt, da beim Ausprobieren in Gazebo der Türerkenner eine Erkennung höchstens drei Sekunden kostant anzeigte. Nach dieser Zeit wurde entweder keine Bounding Box angezeigt oder es wurde ein anderer Zustand der Tür angezeigt, wodurch es zu einer Unterbrechung kam.

Als neuronales Netz wurde das Modell KHS genutzt, da es in den vorherigen Experimente die Höchstwerte aller Modelle erzielen konnte.

Ergebnisse



(a) Inkorrekt (b) Korrekt (c) Semi-korrekt

Abbildung 5.12: Beispielbilder der Türerkennung in Gazebo mit dem neuronalen Netz KHS.
Links: Geschlossene Tür wurde nicht erkannt; **Mitte:** Halbgeschlossene Tür wurde mit 95% korrekt erkannt; **Rechts:** Offene Tür wurde erkannt, aber die Bounding Box ist größer als die Tür.

	Open	**Semi**	**Closed**	**No door**
Korrekt (RS)	13%	16%	26%	**86%**
Korrekt (RS2)	**20%**	**26%**	**40%**	80%

Tabelle 5.9: Korrekte Ergebnisse in Gazebo am Husky mit dem neuronalen Netz KHS

Der Tabelle 5.9 ist zu entnehmen, dass die Türerkennung mit der RS2 bei der offenen Tür ein 7% korrekteres Ergebnis als die Türerkennung mit der RS lieferte. Ebenso bei der halbgeschlossenen Tür war die Erkennung um 10% korrekter und auch bei der geschlossenen Tür ist die Erkennungsrate um 14% höher.

Lediglich bei den Nicht-Türen war die Kameraposition der RS im Vorteil, da die Erkennungsrate um 6% höher als bei der Kameraposition der RS2 war.

Laut eigener Definition der Robustheit unter Abschnitt 4.1 kann der Türerkenner KHS unter den Bedingungen der Simulation nicht als robust bezeichnet werden. Das neuronale Netz hat KHS mit der Tiefenkamera RS im Durchschnitt nur $\approx 18\%$ und mit der Tiefenkamera RS2 nur $\approx 28\%$ der Türen korrekt klassifizieren können, obwohl mindestens 57% gefordert sind. Der Blickwinkel ist laut den Ergebnissen der Tabelle 5.9 ein Beeinflussungsfaktor, da sich die Erkennungsraten um 6% bis 14% unterscheiden.

Einzig der Punkt, dass mehr als 71% der Nicht-Türen korrekt als Nicht-Türen oder gar nicht erkannt werden soll, wurde mit 80% und 86% um 9% bzw. 15% überboten.

Aufgrund der resultierenden Ergebnisse kann man daraus ableiten, dass die Position der RS2 einen Vorzug gegenüber der Position der RS hat. Dies hängt wohl damit zusammen, da die Bilder des Trainingsdatensatzes aus einer höheren Lage aufgenommen wurden und sich somit mehr mit den aufgenommenen Bildern aus höheren Position, wie die Bilder der RS, ähneln. Weshalb die RS die RS2 bei Nicht-Türen übertrumpfen konnte, ließ sich nicht beantworten.

Die Erkennung von Türen in verschieden stark beleuchteten Umgebungen innerhalb der Simulation wurde nicht behandelt und wurde daher nicht beantwortet.

6 Fazit

Folgend werden die Erkenntnisse der Thesis zusammengefasst und mögliche Erweiterungen der Arbeit besprochen.

6.1 Zusammenfassung

In dieser Thesis wurde untersucht, wie man ein neuronales Netz, welches für die Objekterkennung implementiert wurde, robuster machen kann mit der Hilfe von Image Data Augmentation und ohne die Architektur des Netzes anzupassen. Es wurde der Begriff der Robustheit definiert und anschließend wurden mehrere eigene angepasste neuronale Netzwerke implementiert, die miteinander verglichen wurden, um zu überprüfen, ob der Türerkenner robuster geworden sind. Zusätzlich wurde der robusteste Türerkenner in der Simulation Gazebo an dem Roboter Husky implementiert und untersucht.

Beim Vergleich der einzelnen Image Data Augmentation Methoden hat sich das Verändern des Kontrastes als die Methode mit den höchsten Ergebnissen aller verwendeten Methode herauskristallisiert. Die Methode mit den zweithöchsten Ergebnissen ist das Löschen von Bildausschnitten. Die für die Thesis entwickelte Greenscreen-Methode ist die Methode mit den niedrigsten gemessenen Werten geworden.

Mittels der Image Data Augmentation war es möglich einen Trainingsdatensatz zu erstellen und ein neuronales Netz namens KHS zu trainieren, welches eine hohe Robustheit unter realen Bedingungen vorweist. Dies äußert sich dadurch, dass das Modell höhere Werte in den Experimente erreichte als der unveränderte HS, sowie höhere Werte als die anderen Varianten der Modelle.

In einer dunkleren bzw. helleren Umgebung konnte das Netz KHS eine Steigerung gegenüber allen anderen neuronalen Netzen erzielen und auch bei der fortlaufenden Änderung

des Zustandes einer Tür, inklusive der Verdeckung der Tür durch ein Objekt, konnte die gesteigerte Robustheit des Modells bestätigt werden.

Das Simulieren des Huskys in Gazebo zeigte, dass die Objekterkennung in der Simulation keine ausreichend guten Ergebnisse erzielen konnte, so dass der Objekterkenner KHS innerhalb der Simulation als nicht robust deklariert wurde.

6.2 Ausblick

Es werden Möglichkeiten genannt, um das bestehende Projekt zu erweitern, zu verbessern oder wie man gewisse Punkte anders hätte umsetzen können.

6.2.1 Punktwolke

Da der Datensatz DD2[3] bereits Tiefenbilder mit Punktwolken bietet, wäre es möglich anhand dieser Daten ein neuronales Netz zu entwickeln, welches die Farb- und Tiefenbilder verwendet, um zu lernen, wie eine Tür dreidimensional strukturiert ist.

6.2.2 Semantische Segmentierung

Auch hierfür bietet der Datensatz DeepDoors2[3] Bilddateien für die semantische Segmentierung. Genauso könnte man ein neuronales Netz entwickeln, welches die Trainingsdaten verwendet, um Bilder semantisch segmentieren zu können.

6.2.3 Heatmap

Um die Entscheidungsfindung des neuronalen Netzes nachvollziehen zu können, wäre es möglich ein neuronales Netz in Keras zu entwickeln und eine Heatmap zu erstellen. Dadurch kann für den Menschen erkennbar gemacht werden, auf welche Merkmale sich das neuronale Netz konzentriert und weshalb es zu einer gewissen Vorhersage des Zustandes gekommen ist.

6.2.4 Interaktion mit dem Mensch

Falls ein Roboter vor einer halb-/geschlossenen Tür steht und keine Möglichkeit besitzt die Tür zu öffnen, gäb es die Möglichkeit die Architektur und Implementierung von Herrn Awab Abdelkarim auf dem Roboter zu implementieren. In der Bachelorarbeit geht es darum mit visuellen Leuchtmitteln ein Interface für die Interaktion zwischen Mensch und Maschine zu schaffen. Mit Hilfe von LED-Streifen werden dem Menschen Signale mitgeteilt, die dem Menschen zum Handeln auffordern. Für den Roboter der vor einer Tür steht gäbe es die Option mittels der Leucht- oder Tonsignale auf sich aufmerksam zu machen, damit ein Mensch die geschlossene Tür für den Roboter öffnet.

Literaturverzeichnis

[1] Google LLC. object detection - Google Suche. URL https://tinyurl.com/googlesearchobjdet. (30.08.2022, 12:29 Uhr).

[2] Marcus D. Bloice. Augmentor - Augmentor 0.2.9 Documentation, 2020. URL https://augmentor.readthedocs.io/en/master/. (30.06.2022, 16:58 Uhr).

[3] João Ramôa et al. Real-time 2D–3D door detection and state classification on a low-power device. *SN Applied Sciences*, 3, 05 2021. doi: 10.1007/s42452-021-04588-3. URL http://dx.doi.org/10.1007/s42452-021-04588-3. (30.05.2022, 13:45 Uhr).

[4] Wei Chen et al. Door recognition and deep learning algorithm for visual based robot navigation. 2014. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7090595. (23.05.2022, 12:03 Uhr).

[5] Wang et al. Robust Object Detection under Occlusion with Context-Aware CompositionalNets. 2020. URL https://openaccess.thecvf.com/content_CVPR_2020/papers/Wang_Robust_Object_Detection_Under_Occlusion_With_Context-Aware_CompositionalNets_CVPR_2020_paper.pdf. (21.05.2022, 15:43 Uhr).

[6] Siraj Raval. YOLO Object Detection - YouTube. URL https://www.youtube.com/watch?v=4eIBisqx9_g. (30.08.2022, 12:33 Uhr).

[7] khomson kocento. Simple vehicle counting using OpenCV YOLO3 Python 3.5 and Sort - YouTube. URL https://www.youtube.com/watch?v=MHVNzIfsLVU. (30.08.2022, 12:33 Uhr).

[8] Claudio Michaelis et al. Benchmarking Robustness in Object Detection: Autonomous Driving when Winter is Coming. 2019. URL https://ml4ad.github.io/files/papers/Benchmarking%20Robustness%

20in%20Object~Detection:%20Autonomous%20Driving%20when%20Winter%20is%20Coming.pdf. ().

[9] Stephan Pareigis, Tim Tiedeman, Maximilian A. De Muirier. Test Area Intelligent Quartier Mobility (TIQ). 2021. URL https://autosys.informatik.haw-hamburg.de/project/smartmobility/. (30.05.2022, 14:28 Uhr).

[10] Kendrick Boyd et al. Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals. 2013. URL https://link.springer.com/content/pdf/10.1007/978-3-642-40994-3_29.pdf. (30.08.2022, 16:08 Uhr).

[11] Jacob Solawetz. Mean Average Precision (mAP) in Object Detection, 2020. URL https://blog.roboflow.com/mean-average-precision/. (06.06.2022, 11:51 Uhr).

[12] Unknown. Loss Function Definition - DeepAI. URL https://deepai.org/machine-learning-glossary-and-terms/loss-function. (28.08.2022, 12:08 Uhr).

[13] Burak Kakillioglu et al. Doorway detection for autonomous indoor navigation of unmanned vehicles. 2016. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7533078. (20.05.2022, 09:03 Uhr).

[14] Dhiego Bersan et al. Semantic Map Augmentation for Robot Navigation: A Learning Approach based on Visual and Depth Data. 2018. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8588525. (29.05.2022, 15:59 Uhr).

[15] Christopher Juenemann et al. Robust door detection. unknown. URL https://stacks.stanford.edu/file/druid:rz261ds9725/Corbin_Li_Juenemann_RobustDoorDetection.pdf. (29.05.2022, 16:31 Uhr).

[16] Miguel Arduengo, Carme Torras, and Luis Sentis. Robust and adaptive door operation with a mobile robot. *Intelligent Service Robotics*, May 2021. ISSN 1861-2784. doi: 10.1007/s11370-021-00366-7. URL http://dx.doi.org/10.1007/s11370-021-00366-7. (13.06.2022, 17:05 Uhr).

[17] Prof. Dr. Stephan Günnemann. Robust Machine Learning, 2022. URL https://www.cs.cit.tum.de/daml/forschung/robust-machine-learning/. (11.07.2022, 17:25 Uhr).

[18] Eric Wong, J. Zico Kolter. Learning perturbation sets for robust machine learning. 2020. URL https://arxiv.org/pdf/2007.08450.pdf. (14.07.2022, 17:30 Uhr).

[19] Andrew Murphy. Batch size (machine learning) | Radiopaedia.org, 2019. URL https://radiopaedia.org/articles/batch-size-machine-learning. (30.08.2022, 20:15 Uhr).

[20] Dati Tran. Github.com - raccoon_dataset/generate_tfrecord.py, 2018. URL https://github.com/datitran/raccoon_dataset/blob/master/generate_tfrecord.py. (04.07.2022, 17:44 Uhr).

[21] HAW Hamburg. JupyterHub (HAW), 2022. URL https://jupyterhub.informatik.haw-hamburg.de/hub/spawn. (02.05.2022, 13:17 Uhr).

[22] Yilei et al. models/model_main_tf2.py - github. URL https://github.com/tensorflow/models/blob/master/research/object_detection/model_main_tf2.py. (29.08.2022, 17:50 Uhr).

[23] Lenovo Ideapad 14ARE05 - Notebookcheck.com. URL https://www.notebookcheck.com/Lenovo-IdeaPad-5-14ARE05.486352.0.html. (29.08.2022, 14:54 Uhr).

[24] Intel RealSense D435 - Idealo. URL https://www.idealo.de/preisvergleich/OffersOfProduct/200233111_-realsense-depth-camera-d435-intel.html. (29.08.2022, 14:54 Uhr).

A Anhang

```
#!/usr/bin/env python
from __future__ import print_function

import roslib
# roslib.load_manifest('my_package')
import sys
import rospy
import cv2
from std_msgs.msg import String
from sensor_msgs.msg import Image
from actionlib_msgs.msg import GoalID
from move_base_msgs.msg import MoveBaseActionGoal
from cv_bridge import CvBridge, CvBridgeError
import numpy as np
import tensorflow as tf
import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from sensor_msgs.msg import CompressedImage
import csv


class door_detection:
    detect_fn = None
    category_index = None
    currentHandleBox = None
    foundHandle = 0
    current_goal = None
    prev_goal = None
    open_door = None
    closed_door = None
    stopped = False
    prev_obj = None
    speedlimit = False
    closed_seen = 0
```

```
36      open_seen = 0
37      counter = 0
38      SEEN_MIN = 10
39      MIN_WIDTH = 400
40      MIN_HEIGHT = 900
41
42      def __init__(self):
43          # Load saved model and build the detection function
44          self.detect_fn = tf.saved_model.load(
45              "/home/justin/catkin_ws/src/haw_husky/haw_husky_door/scripts/saved_models/saved_model
46          self.category_index =
                label_map_util.create_category_index_from_labelmap(
47              "/home/justin/catkin_ws/src/haw_husky/haw_husky_door/scripts/model_configs/config_kom
48              use_display_name=True)
49          self.image_pub = rospy.Publisher("door_detection", Image,
                queue_size=100)
50          self.cancel_pub = rospy.Publisher("move_base/cancel", GoalID,
                queue_size=100)
51          self.goal_pub = rospy.Publisher("/move_base/goal", MoveBaseActionGoal,
                queue_size=100)
52          self.bridge = CvBridge()
53          self.image_sub =
                rospy.Subscriber("/realsense2/color/image_raw/compressed",
                CompressedImage, self.callbackColor,
54                                  queue_size=1)
55          self.goal_sub = rospy.Subscriber("/move_base/goal",
                MoveBaseActionGoal, self.callbackGoal,
56                                  queue_size=1)
57
58      def callbackGoal(self, ros_goal):
59          print(f"{self.counter} callbackGoal")
60          if (self.current_goal is None) or (ros_goal != self.current_goal):
61              self.current_goal = ros_goal
62              self.prev_goal = ros_goal
63
64      def stop_husky(self):
65          cancel_msg = GoalID()
66          self.cancel_pub.publish(cancel_msg)
67          self.current_goal = None
68          self.stopped = True
69          self.closed_seen = 0
70
71
72      def resume_to_goal(self):
```

```
73          self.stopped = False
74          self.goal_pub.publish(self.prev_goal)
75
76
77
78      def throttle_speed(self):
79          self.speedlimit = True
80          # TODO: set maximum speed for husky roboter
81
82
83      def callbackColor(self, ros_data):
84          self.counter += 1
85          try:
86              np_arr = np.frombuffer(ros_data.data, np.uint8)
87              colorImage = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
88          except CvBridgeError as e:
89              print(e)
90
91          image_np = colorImage[80:400, 160:480]
92          input_tensor = tf.convert_to_tensor(image_np)
93          # The model expects a batch of images, so add an axis with
                'tf.newaxis'.
94          input_tensor = input_tensor[tf.newaxis, ...]
95
96          detections = self.detect_fn(input_tensor)
97
98          # All outputs are batches tensors.
99          # Convert to numpy arrays, and take index [0] to remove the batch
                dimension.
100         # We're only interested in the first num_detections.
101         num_detections = int(detections.pop('num_detections'))
102         detections = {key: value[0, :num_detections].numpy()
103                       for key, value in detections.items()}
104         detections['num_detections'] = num_detections
105
106         # detection_classes should be ints.
107         detections['detection_classes'] =
                detections['detection_classes'].astype(np.int64)
108
109         image_np_with_detections = image_np.copy()
110
111         viz_utils.visualize_boxes_and_labels_on_image_array(
112             image_np_with_detections,
113             detections['detection_boxes'],
```

```
114             detections['detection_classes'],
115             detections['detection_scores'],
116             self.category_index,
117             use_normalized_coordinates=True,
118             max_boxes_to_draw=1,
119             min_score_thresh=.89,
120             agnostic_mode=False)
121
122         # Object detection output saved to variables
123         if detections["detection_scores"][0] >= 0.90:
124             detected_box = detections['detection_boxes'][0]
125             detected_class = detections['detection_classes'][0]
126             detected_object =
                    self.category_index.get(detected_class).get("name")
127             detected_score = detections['detection_scores'][0]
128             x_max = detected_box[3]
129             y_max = detected_box[2]
130
131             if detected_object == "open_door" or detected_object == "semi_door":
132                 self.open_door = True
133                 self.open_seen += 1
134             elif detected_object == "closed_door":
135                 self.closed_door = True
136                 self.closed_seen += 1
137
138         # START ALGORITHM
139         if "detected_object" not in locals(): # no object detected
140             if self.speedlimit:
141                 self.speedlimit = False
142             else:
143                 self.open_door = False
144                 self.closed_door = False
145                 self.closed_seen = 0
146                 self.open_seen = 0
147         else: # variable "detected_object" exists
148             if x_max > self.MIN_WIDTH and y_max > self.MIN_HEIGHT: # an object
                    is detected
149                 if detected_object == "closed_door" and not self.stopped:
150                     if self.closed_seen >= self.SEEN_MIN: # if closed_door seen
                            multiple times
151                         self.stop_husky()
152                 elif detected_object == "open_door" or detected_object ==
                        "semi_door": # open_door or semi_door
153                     if self.stopped:
```

```
154                     self.resume_to_goal()
155                     self.throttle_speed()
156                 elif self.open_seen >= self.SEEN_MIN and not self.speedlimit:
                        # if open_door seen multiple times
157                     self.throttle_speed()
158         # END ALGORITHM
159
160
161         if num_detections > 0:
162             self.currentHandleBox = detections['detection_boxes'][0]
163             self.foundHandle = 1
164
165         try:
166             self.image_pub.publish(self.bridge.cv2_to_imgmsg(image_np_with_detections,
                    "bgr8"))
167         except CvBridgeError as e:
168             print(e)
169
170
171 def main(args):
172     ic = door_detection()
173     rospy.init_node('door_detection', anonymous=True)
174     try:
175         rospy.spin()
176     except KeyboardInterrupt:
177         print("Shutting down")
178     cv2.destroyAllWindows()
179
180
181 if __name__ == '__main__':
182     main(sys.argv)
```

Listing A.1: Programmablauf der Husky Objekterkennung in ROS/Python

```
import glob
import os

os.environ['CUDA_VISIBLE_DEVICES'] = "-1"

import cv2
import tensorflow as tf
import xlsxwriter

from object_detection.builders import model_builder
from object_detection.utils import config_util
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from realsense_depth import *

BATCH_SIZE = 16

CLOSED_COUNTER = 0
SEMI_COUNTER = 0
OPEN_COUNTER = 0

OPEN_CORRECT = 0
SEMI_CORRECT = 0
CLOSED_CORRECT = 0

CUSTOM_MODEL_NAME = f'aug_kombiAugBest_{BATCH_SIZE}' # TODO: change when
    using other neural net
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8'
PRETRAINED_MODEL_URL =
    'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnl
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'

paths = {
   'WORKSPACE_PATH': os.path.join('../../Tensorflow', 'workspace'),
   'SCRIPTS_PATH': os.path.join('../../Tensorflow', 'scripts'),
   'APIMODEL_PATH': os.path.join('../../Tensorflow', 'models'),
   'ANNOTATION_PATH': os.path.join('../../Tensorflow', 'workspace',
       'annotations'),
   'IMAGE_PATH': os.path.join('../../Tensorflow', 'workspace', 'images'),
   'MODEL_PATH': os.path.join('../../Tensorflow', 'workspace', 'models'),
   'PRETRAINED_MODEL_PATH': os.path.join('../../Tensorflow', 'workspace',
       'pre-trained-models'),
```

```
40      'CHECKPOINT_PATH': os.path.join('../../Tensorflow', 'workspace',
            'models', CUSTOM_MODEL_NAME),
41      'OUTPUT_PATH': os.path.join('../../Tensorflow', 'workspace', 'models',
            CUSTOM_MODEL_NAME, 'export'),
42      'TFJS_PATH': os.path.join('../../Tensorflow', 'workspace', 'models',
            CUSTOM_MODEL_NAME, 'tfjsexport'),
43      'TFLITE_PATH': os.path.join('../../Tensorflow', 'workspace', 'models',
            CUSTOM_MODEL_NAME, 'tfliteexport'),
44      'PROTOC_PATH': os.path.join('../../Tensorflow', 'protoc')
45  }
46
47  files = {
48      'PIPELINE_CONFIG': os.path.join('../../Tensorflow', 'workspace',
            'models', CUSTOM_MODEL_NAME, 'pipeline.config'),
49      'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'],
            TF_RECORD_SCRIPT_NAME),
50      'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
51  }
52
53  category_index =
        label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
54
55  # Load pipeline config and build a detection model
56  configs =
        config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
57  detection_model = model_builder.build(model_config=configs['model'],
        is_training=False)
58
59  # Restore checkpoint
60  ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
61  ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'],
        'ckpt-101')).expect_partial() # TODO: change when using other neural net
62
63
64  @tf.function
65  def detect_fn(image):
66      image, shapes = detection_model.preprocess(image)
67      prediction_dict = detection_model.predict(image, shapes)
68      detections = detection_model.postprocess(prediction_dict, shapes)
69      return detections
70
71
72  def get_state_from_string(string):
73      global CLOSED_COUNTER, SEMI_COUNTER, OPEN_COUNTER
```

```
74      if ("closedDoor" in string) or ("closed_door" in string):
75          CLOSED_COUNTER += 1
76          return "closed_door"
77      elif ("semiDoor" in string) or ("semi_door" in string):
78          SEMI_COUNTER += 1
79          return "semi_door"
80      elif ("openDoor" in string) or ("open_door" in string):
81          OPEN_COUNTER += 1
82          return "open_door"
83
84
85  def increcase_correct(state):
86      global CLOSED_CORRECT, SEMI_CORRECT, OPEN_CORRECT
87      if (state == "closedDoor") or (state == "closed_door"):
88          CLOSED_CORRECT += 1
89      elif (state == "semiDoor") or (state == "semi_door"):
90          SEMI_CORRECT += 1
91      elif (state == "openDoor") or (state == "open_door"):
92          OPEN_CORRECT += 1
93
94
95  FILES = f"/media/jtrvz/LinuxFiles/door_reallife_testdata/uni_nolabel/all"
96
97
98  counter = 0
99  for file in sorted(glob.glob(FILES + "/*.jpg")):
100     im = cv2.imread(str(file))
101     image_np = im
102
103     door_state = get_state_from_string(file)
104
105     input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0),
            dtype=tf.float32)
106     detections = detect_fn(input_tensor)
107     num_detections = int(detections.pop('num_detections'))
108     detections = {key: value[0, :num_detections].numpy()
109                   for key, value in detections.items()}
110     detections['num_detections'] = num_detections
111
112     # detection_classes should be ints.
113     detections['detection_classes'] =
            detections['detection_classes'].astype(np.int64)
114
115     label_id_offset = 1
```

```
116     image_np_with_detections = image_np.copy()
117
118     viz_utils.visualize_boxes_and_labels_on_image_array(
119         image_np_with_detections,
120         detections['detection_boxes'],
121         detections['detection_classes'] + label_id_offset,
122         detections['detection_scores'],
123         category_index,
124         use_normalized_coordinates=True,
125         max_boxes_to_draw=1,
126         min_score_thresh=.01,
127         agnostic_mode=False)
128
129     # get detected state
130     detected_state = category_index[detections['detection_classes'][0] +
            label_id_offset].get("name")
131
132     if door_state == detected_state:
133         increcase_correct(door_state)
134
135     print("_____")
136     print(file.replace(FILES + "/", ''))
137     print(category_index[detections['detection_classes'][0] +
            label_id_offset].get("name"))
138     print(detections['detection_scores'][0])
139
140     img = cv2.resize(image_np_with_detections, (800, 600))
141     # cv2.imshow('object detection', img)
142     cv2.imwrite(f"/home/jtrvz/Git/door_aug/output/normal/{CUSTOM_MODEL_NAME}_{door_state}_{coun
            img)
143     counter += 1
144
145     if cv2.waitKey(0) & 0xFF == ord('q'):
146         cv2.destroyAllWindows()
147         break
148
149 print("_____")
150 print(f"DETECTION RESULTS FOR {CUSTOM_MODEL_NAME}")
151 print(f"OPEN: {OPEN_CORRECT}/{OPEN_COUNTER} = {OPEN_CORRECT/OPEN_COUNTER}")
152 print(f"SEMI: {SEMI_CORRECT}/{SEMI_COUNTER} = {SEMI_CORRECT/SEMI_COUNTER}")
153 print(f"CLOSED: {CLOSED_CORRECT}/{CLOSED_COUNTER} =
        {CLOSED_CORRECT/SEMI_COUNTER}")
```

Listing A.2: Statistik-Pythonskript für die Objekterkennung am Laptop

```
import cv2
import numpy as np
import os

import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
from realsense_depth import *

BATCH_SIZE = 16

CUSTOM_MODEL_NAME = f'aug_kombiAug_{BATCH_SIZE}'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_640x640_coco17_tpu-8'
PRETRAINED_MODEL_URL =
    'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fpnl
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'

paths = {
    'WORKSPACE_PATH': os.path.join('../../Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('../../Tensorflow', 'scripts'),
    'APIMODEL_PATH': os.path.join('../../Tensorflow', 'models'),
    'ANNOTATION_PATH': os.path.join('../../Tensorflow', 'workspace',
        'annotations'),
    'IMAGE_PATH': os.path.join('../../Tensorflow', 'workspace', 'images'),
    'MODEL_PATH': os.path.join('../../Tensorflow', 'workspace', 'models'),
    'PRETRAINED_MODEL_PATH': os.path.join('../../Tensorflow', 'workspace',
        'pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('../../Tensorflow', 'workspace',
        'models', CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('../../Tensorflow', 'workspace', 'models',
        CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH': os.path.join('../../Tensorflow', 'workspace', 'models',
        CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH': os.path.join('../../Tensorflow', 'workspace', 'models',
        CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH': os.path.join('../../Tensorflow', 'protoc')
}

files = {
```

```
36      'PIPELINE_CONFIG': os.path.join('../../Tensorflow', 'workspace',
            'models', CUSTOM_MODEL_NAME, 'pipeline.config'),
37      'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'],
            TF_RECORD_SCRIPT_NAME),
38      'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
39  }
40
41
42  category_index =
        label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
43
44  # Load pipeline config and build a detection model
45  configs =
        config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
46  detection_model = model_builder.build(model_config=configs['model'],
        is_training=False)
47
48  # Restore checkpoint
49  ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
50  ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'],
        'ckpt-101')).expect_partial()
51
52
53  @tf.function
54  def detect_fn(image):
55      image, shapes = detection_model.preprocess(image)
56      prediction_dict = detection_model.predict(image, shapes)
57      detections = detection_model.postprocess(prediction_dict, shapes)
58      return detections
59
60
61  dc = DepthCamera()
62
63  while True:
64      #ret, frame = cap.read()
65      ret, depth_frame, color_frame = dc.get_frame()
66      image_np = np.array(color_frame)
67
68      input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0),
            dtype=tf.float32)
69      detections = detect_fn(input_tensor)
70
71      num_detections = int(detections.pop('num_detections'))
72      detections = {key: value[0, :num_detections].numpy()
```

```
73                for key, value in detections.items()}
74    detections['num_detections'] = num_detections
75
76    # detection_classes should be ints.
77    detections['detection_classes'] =
          detections['detection_classes'].astype(np.int64)
78
79    label_id_offset = 1
80    image_np_with_detections = image_np.copy()
81
82    viz_utils.visualize_boxes_and_labels_on_image_array(
83       image_np_with_detections,
84       detections['detection_boxes'],
85       detections['detection_classes'] + label_id_offset,
86       detections['detection_scores'],
87       category_index,
88       use_normalized_coordinates=True,
89       max_boxes_to_draw=5,
90       min_score_thresh=.8,
91       agnostic_mode=False)
92
93    cv2.imshow('object detection', cv2.resize(image_np_with_detections, (800,
          600)))
94
95    if cv2.waitKey(10) & 0xFF == ord('q'):
96       dc.release()
97       cv2.destroyAllWindows()
98       break
```

Listing A.3: Pythonskript zur Objekterkennung mit einer angeschlossenen Tiefenkamera Intel RealSense D435

```python
import cv2
import time

from realsense_depth import *

dc = DepthCamera()

counter = 0
while True:
    ret, depth_frame, color_frame = dc.get_frame()
    img_np = np.array(color_frame)

    img_name = f"openDoor_{counter}.jpg"
    cv2.imwrite("/home/jtrvz/Git/door_aug/utils/objectDetection/img_depth_cam/"
        + img_name, img_np)
    print(f"\nSAVE: Saved image as {img_name}!")
    counter += 1

    cv2.imshow('object detection', cv2.resize(img_np, (800, 600)))

    time.sleep(1)

    if cv2.waitKey(1000) & 0xFF == ord('q'):
        dc.release()
        cv2.destroyAllWindows()
        break
```

Listing A.4: Pythonskript zum automatischen Speichern von Bildern in Intervallen von der Tiefenkamera Intel RealSense D435

Glossar

Augmentor - Python Package zum Erstellen von Bildern mit Image Date Augmentation.

Bounding Box - Koordinatenbereich, der bei Objekterkennern die Lage des Objektes innerhalb eines Bildes anzeigt.

Erkennungspunktzahl - (*Engl. detection score*) Punktzahl in Prozent (oft in Verbindung mit einer Bounding Box), die darüber mitteilt, wie hoch die Übereinstimmung des erkannten Objektes mit einem bereits gesehenen Objekt aus dem Trainingsdatensatz ist.

False Negative - Einstufung einer Objekterkennung, wobei kein Label und keine Bounding Box, obwohl Objekt auf- findbar.

False Positive - Einstufung einer Objekterkennung, wobei Bounding Box oder Label inkorrekt.

Gazebo - Open Source Simulator für die Robotik.

HAW Hamburg - Die HAW Hamburg ist die vormalige Fachhochschule am Berliner Tor.

Husky - Name des Roboters der Autosys Gruppe.

Intersection over Union - Schnittmenge über Vereinigung.

Loss - Zeigt an, wie fehlerhaft die Vorhersage eines Machine-Learning-Modell gegenüber den tatsächlichen Ergebnissen ist.

Mean Average Precision - Durchschnittlicher Mittelwert der Metrik Precision für den Schwellenwertebereich 0.50 bis 0.95.

neuronales Netz - *Synonyme:* Machine-Learning-Modell, Modell, Netz, neuronales Netzwerk, Objekterkenner, Erkenner, Türerkenner.

Precision - Metrik, die die Höhe des Anteils der positiven Erkennungen anzeigt, die tatsächlich korrekt sind.

Punktwolke - Dreidimensionales Modell eines Objektes, Lebewesen oder einer Umgebung.

Recall - Metrik, die anzeigt, wie viele Erkennungen von allen Erkennungen tatsächlich korrekt sind.

Tiefenkamera - Erfasst ein Bild, wie eine normale Kamera und zusätzlich für jeden Pixel die relative Entfernung zur Kamera.

True Positive - Einstufung einer Objekterkennung, wobei Label korrekt und Bounding Box korrekt.

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