



Bericht zum praktischen Gruppenarbeit

Studiengang Edge Intelligence

Firma hydrophobic Inc.

Name	Michel Berg, Yannik Bickert, Joshua Born, Thomas Liebgott, Marvin Schulz
Studiengang	Edge Intelligence
Semester	Wintersemester 22/23
Firma	hydrophobic Inc.
Standort	Mannheim
Abteilung	Forschung und Entwicklung
Betreuer	Prof. Dr. Marcus Vetter

Zusammenfassung

Die Industrie schreitet immer weiter voran im Bereich der Automatisierung von Produktion, Verpackungsabläufen sowie Kurzstreckentransporte. Unter dem weitgreifenden Begriff der „Industrie 4.0“ steht einerseits die gefahrenlose enge Zusammenarbeit autonomer Maschinen und den Mitarbeiter:innen des Unternehmens sowie die Vernetzung dieser einzelnen Systemen zur Steigerung der Autonomie, Sicherheit sowie Effizienz der Produktionskette.

Das Start-Up *hydrophobic Inc.*, gegründet am 13. Oktober dieses Jahres, stellt hiermit den Prototypen eines Systems zur automatisierten Erkennung von Wasserpflützen mittels der Anwendung des Deeplearning Netzes YOLOv7 vor. Dieses System soll autonom fahrenden Industrierobotern ermöglichen Nässe oder Verschmutzungen auf ihrer Fahrbahn in Industriehallen zu erkennen, zu meiden sowie zu melden. Somit soll sichergestellt werden, dass der Bremsweg durch diese Art der Umwelteinflüsse nicht verlängert und die Gefahr einer Kollision erhöht. Zudem soll die Erkennung solcher Gefahrenstellen, umgehend einer zentralen Instanz gemeldet werden können.

Inhaltsverzeichnis

1 Einführung	3
2 Lösungsansatz	5
2.1 YOLOv7	5
2.2 Azure Kinect	5
2.3 Vergleich eines zentralen und dezentralen Konzept	6
2.4 MQTT	7
2.5 Auswahl der Hardware	9
3 Trainingsdatengenerierung	11
3.1 AI-Generierung	11
3.2 Generierung von virtuellen Daten	11
3.3 Blender	13
3.3.1 Die Render-Umgebung	13
3.3.2 Das Python-Script	14
3.4 Automatische Boundingbox-Generierung	15
3.4.1 Verbesserung des Binarisierung Thresholds	16
3.4.2 Anwendung des Skripts	17
4 Training und Evaluation	20
4.1 Training	20
4.2 Evaluation	20
5 Fazit	23
Literaturverzeichnis	24

Kapitel 1

Einführung

Im Gegensatz zum autonomen Fahren im Straßenverkehr, bei dem ein möglichst geringe Gefahr für das Leben von Verkehrsteilnehmer:innen akzeptiert wird, gilt in der Industrie eine Null-Risiko Politik. Eine mögliche Gefährdung der Mitarbeiter:innen auf dem Gelände muss absolut ausgeschlossen werden können. Ein wichtiger Bestandteil des Sicherheitskonzeptes im Autonomen fahren stellt die Erkennung von Objekten und Kollisionsvermeidung dar. Zur Objekterkennung kommt es bereits vermehrt zum Einsatz von Neuronalen Netzen. Die Erkennung von Schildern, Fahrbahnen, Ampeln und weiterer Objekte ist bereits weit fortgeschritten.

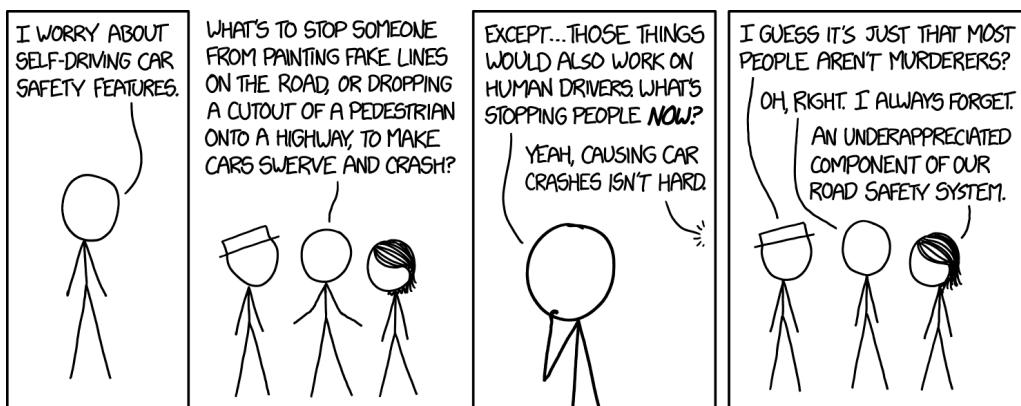


Abbildung 1.1: Self-Driving Issues [Quelle: <https://xkcd.com/1958/>]

Um eine verlässliche Kollisionsvermeidung sicherstellen zu können, ist die Bestimmung des Bremsverhaltens von hoher Bedeutung. Entsprechend der Bremsfähigkeit muss eine Maximalgeschwindigkeit bestimmt werden, da ein zu langer Bremsweg das Unfallrisiko erhöht. Der Bremsweg ist jedoch nicht nur von der Geschwindigkeit des Roboters abhängig. Im Abschnitt 3.1 Industrieroboter der Deutschen Gesetzlichen Unfallversicherung (DGUV) Information 209-074 werden zudem noch die Faktoren Last, Auslenkung, Temperatur der Bremsen, sowie deren Verschleiß und Verschmutzungsgrad genannt. [1] Die DGUV bezieht sich jedoch in ihrem Dokument allein auf statische Knickarmroboter. Somit

fehlen derzeit gesetzliche Vorgaben für autonom fahrende Roboter im Arbeitsumfeld. Die generelle mathematische Formel zur Berechnung des reinen Bremsweg eines auf Reifen fahrenden Objekts lautet folgendermaßen [6]:

$$d_B = vt_{RB} + \frac{v^2}{2\mu g} \quad (1.1)$$

Neben der Geschwindigkeit(v), der Reaktionszeit (vt), der Oberflächenbeschleunigung (g), stellt der Reibungskoeffizient (μ) einen stark variierende und schwer messbare Wert dar. Diese ebenfalls stark abhängig von Faktoren wie der Materialpaarung, Oberfläche, Schmierung, Temperatur, Feuchte, Verschleiß oder Normalkraft. [7] Da in der Praxis ein ständiges Monitoring des Reibungskoeffizienten nicht möglich ist, ist das Ziel dieses Projekts den Faktor Feuchtigkeit konstant zu halten.

Im folgenden Kapitel stellt hydrophobic Inc. ihren Lösungsansatz zur Erkennung von Nässe auf der Fahrbahn mittels des Neuronalen Netzes YOLOv7 zur Kamera basierten Objekterkennung vor. Nach einer Vorstellung der Trainingsdatengenerierung mittels der 3D-Animationssoftware Blender und openCV, werden die Trainings sowie Testergebnisse vorgestellt. Zudem wird das auf darauf aufbauende Warnsystem vorgestellt, welches sich noch in der Entwicklung befindet und auf Basis des Netzwerkprotokolls MQTT andere autonom fahrende Roboter und Mitarbeiter über eine erkannte Nässe der Fahrbahn warnen soll. Final werden die vorgestellten Ergebnisse zusammengefasst und beurteilt.

Kapitel 2

Lösungsansatz

2.1 YOLOv7

Bei YOLO handelt es sich um einen Algorithmus zur Erkennung von Objekten in einer Echtzeitumgebung. Der Begriff YOLO bedeutet „You Only Look Once“ und der Zusatz „v7“ steht für die siebte Version des Algorithmus. Aufgrund dessen, dass mit YOLOv7 auf dem Bereich der Objekterkennung der neuste Stand der Technik vorliegt, wurde dieser vom Auftraggeber Marcus Vetter empfohlen und folglich eingesetzt. Für YOLOv7 existieren verschiedene Basismodelle. Von Relevanz für die vorliegende Aufgabenstellung sind YOLOv7 (für gewöhnliches GPU-Computing optimiert) sowie YOLOv7-tiny (für Edge-GPUs optimiert).

2.2 Azure Kinect

Das Kamera-Modul, welches in diesem Projekt verwendet werden soll, ist eine Azure Kinect. Bei dieser Kamera handelt es sich um eine hochintegrierte Kamera für IOT-Zwecke, welche speziell für räumliches Computing, maschinelles Sehen und Sprachverarbeitung entwickelt wurde.

Relevante Spezifikationen

- 12-MP Videokamera (RGB)
- 1-MP Tiefensensor (NIR)
- Beschleunigungssensor und Gyroskop (IMU)
- USB-3.0 Typ C

RGB-Kamerasensorik

- Auflösung: 1280x720 - 3840x2160
- Seitenverhältnis: 4:3; 16:9
- Bildfrequenz: 0 - 30 FPS
- Sichtfeld: 90° x 59° - 90° x 74.3°
- Format: MJPEG (YUY2, NV12)
- Pixelformat: BGRA (Blue Green Red Alpha)

Zu Testzwecken wurde eine generische USB-Webcam als Capture-Module verwendet. Welche später dann durch Kinect Kameras ersetzt werden sollen.

2.3 Vergleich eines zentralen und dezentralen Konzept

Eine zu Beginn des Projekts bestehende Frage mit entscheidenden Auswirkungen auf die Auswahl der Hardware lautet, ob das die Bilderkennung zentral oder dezentral erfolgen soll.

Im Fall eines zentral auf einem leistungsstarken Edge-Computer, senden alle Roboter ihren Videostream per Real-time Transport Protocol (RTP) an diesen. Das auf dem Edge-Computer laufende YOLOv7-Netz sorgt in Verbindung mit einer leistungsfähigen Grafikkarte für die Pfützenerkennung für empfangenen Bildmaterials jedes sich verbundenen Roboters. Wird Wasser auf der Fahrbahn eines Roboters erkannt, wird eine Warnung mit den Positionsdaten des betroffenen Roboters an alle Teilnehmer gesendet. Der betroffene autonome Roboter kommt daraufhin zum stehen. Die weiteren Teilnehmer umfahren den markierten Bereich.

Dieser Lösungsansatz, zeichnet sich durch eine hohe Datenaustausch ab und ist daher stark von einer guten Netzausdehnung auf dem gesamten Einsatzgelände abhängig. Im Bereich der benötigten Hardware punktet dieses Konzept mit Blick auf die Anschaffungskosten, da die Roboter keine leistungsstarke Hardware mit Grafikprozessoren benötigen. Bei einer hohen Anzahl an Robotern im Betrieb, die selbst keine kostenintensive Hardware für diesen Usecase benötigen, rechnet sich die Investition in eine Leistungsstarke Grafikkarte für den zentralen Edge-Computer.

Ein dezentrales Aufbau erfolgt durch den Betrieb des YOLOv7-Netzes auf der Hardware jedes einzelnen Roboters. Die Mitteilung an alle Teilnehmer im Falle der Detektion einer Pfütze, erfolgt über das MQTT-Protokoll. Ein Edge-Computer übernimmt hierbei die Rolle des MQTT-Brokers.

Dieses Konzept ist aufgrund der geringen Datenstroms und des bewährt robusten MQTT-Protokolls weniger Anfällig gegenüber Netzwerkverbindungsproblemen. Zusätzlich ist die Mitteilung der Positionsdaten der Pfütze nicht mehr so stark zeitabhängig, da der unmittelbar betroffene Roboter die Pfütze selbst erkennt. Nachteile dieses Konzepts sind die erhöhten Leistungsansprüche an die Roboterhardware und dessen damit einhergehenden höheren Kosten.

Aufgrund der hohen Sicherheitsanforderungen muss eine Netzwerkunabhängige Pfützenerkennung mit kurzen Reaktionszeiten des betroffenen Rotobers sichergestellt werden können. Da das zentrale Konzept stark von einer einhundertporzentigen Net zabdeckung auf dem Einsatzgelände abhängig ist und eine solche nicht sichergestellt werden kann, verfolgt hydrophobic Inc. das dezentrale Konzept.

2.4 MQTT

Man wollte sich eine Kommunikation zwischen verschiedenen Wagen vorstellen, die sich in einer Fabrik befinden, um mitzuteilen, dass sich an einer bestimmten Stelle eine Pfütze befindet. Es wurde beschlossen, diese Kommunikation zwischen den verschiedenen Wagen über MQTT zu simulieren. Das MQTT-Protokoll mit seiner Publish/Subscribe-Implementierung würde es in unserem Fall ermöglichen, die parallele Kommunikation zwischen mehreren Wagen zu strukturieren. Da unser Entwicklungsboard durch die Nutzung unseres Erkennungsprogramms stark beansprucht werden wird, ermöglicht die Leichtigkeit des MQTT-Protokolls eine effiziente Implementierung des Kommunikationssystems. Um miteinander zu kommunizieren, benötigen die verschiedenen Wagen einen Broker. Dieser Broker wird es ermöglichen, die verschiedenen Nachrichten zwischen den Subscribers und Publishern auszutauschen. Es wurde entschieden, für das Kommunikationsbeispiel den OpenSource Eclipse Mosquitto Online Broker zu verwenden.[2] Da der Code für das Erkennungsprogramm in Python ist, wurde die Eclipse-Bibliothek Paho verwendet.[3] Für die Kommunikation hat er also einen Publisher und einen Subscriber.[5]

```

1 import paho.mqtt.client as mqtt
2 from random import randrange, uniform
3 import time
4
5 mqttBroker = "mqtt.eclipseprojects.io"
6 client = mqtt.Client("puddleDetection1")
7 client.connect(mqttBroker)
8
9 while True:
10     numberPuddle = uniform(1, 10)
11     client.publish("puddleDetection", numberPuddle)
12     print("Just published " + str(numberPuddle) + " to Topic PuddleDetection")
13     time.sleep(1)

```

Abbildung 2.1: pub.py

```

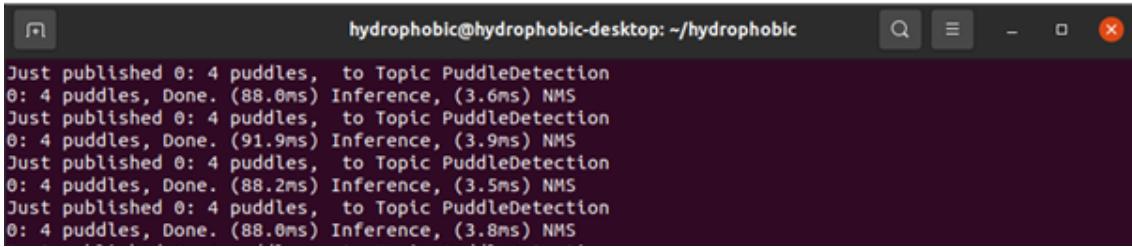
1 import paho.mqtt.client as mqtt
2 import time
3
4 def on_message(client, userdata, message):
5     print("Received message: ", str(message.payload.decode("utf-8")))
6
7 mqttBroker = "mqtt.eclipseprojects.io"
8 client = mqtt.Client("receiver")
9 client.connect(mqttBroker)
10
11 client.loop_start()
12 client.subscribe("puddleDetection")
13 client.on_message = on_message
14 time.sleep(10)
15 client.loop_end()

```

Abbildung 2.2: sub.py

Um die Kommunikation zwischen den beiden Wagen zu simulieren wurde eine Jetson Xavier (der Publisher) eingerichtet, die die Bilderkennung mithilfe der Kinect und des Programms detect.py durchführt. Und eine Nvidia Jetson Nano (der Subscriber), die nur die Informationen empfangen wird.

Der Code des Publishers wurde direkt in den detect.py-Code integriert (zu finden im Branch 11-vortrainiertes-yolov7-auf-nvidia-nano-board-laufen-lassen). Wenn nun also eine Pfütze entdeckt wird, veröffentlicht der detection-Code die Anzahl der entdeckten Pfützen auf dem Broker:



```
Just published 0: 4 puddles, to Topic PuddleDetection
0: 4 puddles, Done. (88.0ms) Inference, (3.6ms) NMS
Just published 0: 4 puddles, to Topic PuddleDetection
0: 4 puddles, Done. (91.9ms) Inference, (3.9ms) NMS
Just published 0: 4 puddles, to Topic PuddleDetection
0: 4 puddles, Done. (88.2ms) Inference, (3.5ms) NMS
Just published 0: 4 puddles, to Topic PuddleDetection
0: 4 puddles, Done. (88.0ms) Inference, (3.8ms) NMS
```

Abbildung 2.3: PuddleDetection MQTT Publish - Nvidia Xavier



```
:hydrophobic@nano02:~/testMqtt$ python sub.py
Received message: 0: 1 puddle,
Received message: 0: 1 puddle, age(client, userdata, message):
Received message: 0: 1 puddle, Received message: "", str(message.payload.decode("utf-8")))
Received message: 0: 1 puddle,
Received message: 0: 1 puddle, - "MQTT.eclipseprojects.io"
Traceback (most recent call last):
```

Abbildung 2.4: PuddleDetection MQTT subscribe - Nvidia® Nano

Dank dieser Kommunikation können wir weiter denken und uns vorstellen, Pfützen mit einer Lokalisierung zu melden, um Bereiche zu kartografieren, die von den Wagen gemieden werden sollten.

2.5 Auswahl der Hardware

Eine beträchtliche Herausforderung des dezentralen Konzepts dieses Projekts besteht in der Einsatz des Neuronalen Netz auf einem möglichst energieeffizienten Edge Device, welches Nvidia® stellt für die Anwendung Eingebettete Systeme mit der ihrer NVIDIA® Jetson™ Baureihe, eine System-on-Module (SOM) Plattform für eingebettete Anwendungen. [4] Sie ermöglichen ein den Einsatz leistungstarker Anwendung auf mobilen, energieeffizienten Systemen. Jeder Nvidia® Jetson™ SOM verfügt über ein Graphics Processing Unit (GPU). Dies ermöglicht es, komplexe Neuronale Netze auf dem Gerät selbst zu verwenden.

Zu Beginn des Projekts entschied sich hydrophobic Inc. für das kleinste Jetson™ Modul, dem Jetson™ Nano. Dessen geringen Kosten, kompakte Größe sowie Energieeffizienz erschienen Ideal für die Zielanwendung. Dieses erwies sich jedoch aufgrund seiner veralteten Python 3.6 Version problematisch. Ein nötiges Upgrades auf die Python-Versionen 3.8 und von CUDA-8 nach CUDA-11 auf dem Jetson eigenen Linux war nicht möglich. Daraufhin entschied man sich für die serienreife Jetson Xavier NX-Serie. Dessen aktuellere NVIDIA JetPack™ SDK Version, welches Jetson Linux, Entwickler-Tools, CUDA-X

beschleunigte Bibliotheken und andere NVIDIA Technologien bereitstellt, stellt Python 3.8 bereit.

Kapitel 3

Trainingsdatengenerierung

3.1 AI-Generierung

Die Suche nach und die Vorverarbeitung von Trainingsdaten ist einer der wichtigsten Teile. Die Qualität der Daten und die Art und Weise, wie sie verarbeitet werden, wirken sich auf die Trainingsqualität des Algorithmus aus. Dieser Teil besteht aus der Erstellung eines DataSets. Die Trainingsdaten sollten einem Modell folgen, das zu 70% aus Trainings-, zu 10% aus Validierungs- und zu 20% aus Testdaten besteht. Wie bereits erwähnt, besteht das Ziel unseres Projekts darin, die Darstellung von Wasserpützen auf dem Boden zu erstellen. Die Datenbeschaffung kann in zwei Teile aufgeteilt werden:

- Verwendung von realen Daten aus Fotos von Situationen, in denen eine Wasserpütze vorhanden ist.
- Generierung von virtuellen Daten, die unsere Situation repräsentieren

Suche nach Daten auf Websites wie KaggleKaggle wird durchgeführt. Diese Webplattform, die zu Google gehört, bietet Zugang zu einer Vielzahl von Datenbanken, die nach verschiedenen Themen geordnet sind. Die Datenbestände, die man finden kann, entsprachen jedoch nicht der Situation, die uns gestellt wurde: Erkennung von Wasserpützen in Innenräumen. Es wird auch nach Datensätzen für selbstfahrende Autos gesucht. Diese DataSets ermöglichen es Wasserpützen bei Regenwetter zu sehen. Diese Datasets erfordern jedoch eine Extraktion der Teile, in denen Wasser vorhanden ist, und sind für unsere Situation ebenfalls nicht repräsentativ.

3.2 Generierung von virtuellen Daten

Die virtuelle Datengenerierung ermöglicht es, eine bestimmte Situation realitätsgenauer darzustellen. Es gibt verschiedene Programme zur Modellierung von Computeranima-

tionen. Nvidia® Issac und Blender sind Beispiele dafür. Hauptsächlich wird die freie Software Blender mit vielen Ressourcen verwendet. Die Blender-Software ermöglicht auch die Verwendung des künstlichen Intelligenzmodells Stable Diffusion. Dieses Modell der künstlichen Intelligenz ermöglicht es, aus einem Text ein Bild zu erstellen, das die Beziehungen zwischen Wörtern und Bildern versteht. Die Integration dieses Modells der künstlichen Intelligenz in Blender erfolgt über den „IA Render“. Dieser ermöglicht es, Bilder aus einem Prompt, einem Preset Style und anderen Parametern (Steps, Prompt Strength, Sampler) zu erstellen. Diese verschiedenen Parameter und der Prompt basieren auf einem 3D-Modell, das in Blender erstellt und bearbeitet wird. Basierend auf einem Modell einer Fabrikkulisse, wird versucht eine Wasserpfütze zu simulieren.



Abbildung 3.1: Beispiel für die Bildgenerierung mit IA Render in Blender

Es wurden daher verschiedene interessante und nutzbare Ergebnisse erhalten. Diese waren jedoch zu zufällig. Trotz verschiedener genauerer Prompts und der Manipulation verschiedener Parameter waren die erzeugten Bilder zu inkonsistent zueinander. Daher wurde die Entscheidung getroffen, die Pfützen ausschließlich mit Blender zu modellieren.

3.3 Blender

Die Wahl des Werkzeugs zur Erstellung einer großen Menge an Trainingsdaten fiel auf die 3D-Modellierungssoftware Blender. Blender bietet mehrere Vorteile gegenüber anderen 3D-Grafiksoftwares, wie etwa die durch eine stetige Weiterentwicklung über eine lange Existenz bedingte hohe Anzahl und Zuverlässigkeit der Funktionen, oder die riesige Nutzergemeinschaft, welche die Geschwindigkeit, mit welcher Antworten auf etwaige Fragen und Problem gefunden werden können, erhöht.

Der ausschlaggebende Grund für die Verwendung von Blender liegt jedoch in Blenders Unterstützung von Scripting: Jede mögliche Benutzer-Aktion in Blender besitzt einen korrespondierenden Befehl in der Programmiersprache Python. Dies bedeutet, dass mithilfe eines Python-Scripts jeder beliebige Vorgang innerhalb von Blender automatisiert werden kann, was die Erzeugung einer praktisch endlosen Menge von Bilddaten für das Training der KI mit einem Knopfdruck ermöglicht (mehr dazu in 3.3.2).

Die Erzeugung der Trainingsbilder in Blender verlief in zwei wesentlichen Schritten, welche im Folgenden erläutert werden: Die Erstellung einer 3D-Umgebung und die Entwicklung eines Python-Scripts, welches prozedural Bilder dieser Umgebung synthetisiert (dieser Vorgang wird im Folgenden als *Rendering* und die Ergebnisse als *Render* bezeichnet).

3.3.1 Die Render-Umgebung

Alle Render wurden in derselben 3D-Umgebung erstellt; Variation wird über andere Wege erreicht (siehe 3.3.2). Da der Arbeitsaufwand exponentiell mit der Komplexität der 3D-Umgebung steigt, und der Fokus auf den Pfützen (und damit dem Boden) liegt, fiel das 3D-Modell einfach aus: Ein Raum von quadratischer Grundfläche mit einigen wenigen Störobjekten, keinerlei Fenster oder Türen. Die Störobjekte haben die Form zweier Gabelstapler, einiger Metalltonnen sowie Aufstellschilder. Die Menge dieser Störobjekte ist bewusst niedrig gehalten, da zuviele Objekte die Pfützen zu stark verdecken würden.

Um möglichst realistische Reflexionen in den Wasserpfützen zu erreichen wurde diese Umgebung zusätzlich zu einer regulären Lichtquelle mit einem High Dynamic Range Image (HDRI) versehen. Die gewählte HDRI-Aufnahme stammt aus einer Reparaturwerkstatt und verleiht der Umgebung eine entsprechende Beleuchtung, sowie den Pfützen Reflexionen von industriell anmutenden Wänden und Decke samt Lampen.

Für den Boden wurden mehrere Texturen ausgewählt und teilweise händisch mit Markierungen versehen, wie sie üblich in industriellen Umgebungen sind. Diese verschiedenen Texturen lösen sich bei der Erstellung der Bilddaten ab (siehe 3.3.2).

Die Pfützen wurden durch den Einsatz einer sogenannten Musgrave-Textur realisiert: Eine Musgrave-Textur simuliert eine unebene Oberfläche, sie ist nicht tatsächlich dreidimensional, besitzt aber anders als reguläre Texturen unterschiedliche Höhenwerte für



Abbildung 3.2: Schnittdarstellung der Render-Umgebung

die Texturdarstellung in Blender. Positioniert man diese Musgrave-Textur innerhalb einer Ebene (in diesem Fall dem Boden), so schiebt sich die Musgrave-Textur an verschiedenen Stellen über die Textur dieser Ebene, was pfützenförmige Überlagerungen einer zweiten Textur über der ersten erzeugt. Größe, Form, Komplexität der Ränder, und weitere Eigenschaften dieser „Pfützen“ sind parametrierbar, sodass verschiedenste Flüssigkeiten realistisch darstellbar sind. Um Wasserpfützen zu simulieren wird die Musgrave-Textur mit einer Glossy-Textur zusammengeführt, welche teilweise die Umgebung reflektiert und teilweise die unterliegende Textur durchblicken lässt.

3.3.2 Das Python-Script

Das Python-Script erstellt zunächst Variablen für sämtliche 3D-Objekte und Shader-Elemente (Konfigurationswerkzeuge für das Aussehen der Oberflächen). Anschließend führt das Script eine for-Schleife aus, in welcher jede Iteration die folgenden Schritte durchläuft:

1. Die Pfützen verändern sich: Die Musgrave-Textur wird bewegt, sodass die Pfützen andere Positionen und Formen besitzen. Zudem wird die simulierte Tiefe der Pfützen um einen zufällig veränderten Faktor verändert, was ihre Sichtbarkeit und Reflektivität beeinflusst.
2. Die Helligkeit der Lichtquellen wird um einen zufälligen Faktor verändert, dies dient der besseren Erkennung von Pfützen in verschiedenen Lichtbedingungen.
3. Die Kameraperspektive verändert sich, indem die Kamera basierend auf einem zufällig generierten Zahlenwert innerhalb der xy-Ebene bewegt und um die z-Achse rotiert wird. Hierbei sind diese beiden Bewegungen in einem Verhältnis zueinander

gesetzt, welches garantiert, dass die Kamera stets zur Mitte des Raumes ausgerichtet ist, um perspektivisch unbrauchbare Render-Ergebnisse zu vermeiden.

4. Blender erstellt einen Render, macht also einen Schnappschuss der Szene aus der Perspektive der Kamera in der Auflösung 640x640 Pixel und speichert diesen als JPG-Datei.
5. Um die automatische Eintragung der Boundingboxen mit OpenCV zu ermöglichen (siehe 3.4) wird jede Konfiguration der Umgebung ein zweites Mal gerendert; diesmal mit einem möglichst hohem Kontrast zwischen Pfützen und der restlichen Umgebung. Zu diesem Zweck werden die Texturen aller Objekte durch ein nicht reflektierendes Schwarz ersetzt. Die Pfützen wiederum werden von Reflektierend-Durchsichtig zu einem möglichst hellen Weiß geändert, zudem werden ihre Ränder auf maximale Härte gesetzt, um möglichst klare Kanten zu erzielen.

Jede zweite Iteration wird die Szene ohne Pfützen gerendert, um den Datensatz für das Training des neuronalen Netzes zu diversifizieren. Zusätzlich wird jeweils nach 15/40/70% der Iterationen die Textur des Bodens ausgetauscht, um Bilddaten von Pfützen vor verschiedenen Hintergründen zu schaffen.

3.4 Automatische Boundingbox-Generierung

Für die ersten Tests wurden die Bounding-Boxen händisch gezeichnet. Da dies mit steigender Zahl von Bild Datensätzen zeitlich nicht realisierbar ist, musste ein Skript zur Bildvorverarbeitung geschrieben werden.

Um die Datensätze für den KI-Algorithmus (YOLOv7) aufzubereiten, wurde ein Python-Skript geschrieben, welches mittels der Open Camera Vision Library (OpenCV) eine einfache Vorverarbeitung vornimmt. Der Ablauf des Algorithmus ist in Abbildung 3.3 zu sehen, welcher in Zwischenzuständen der Verarbeitungskette unterteilt ist.

Der erste Punkt „Initial“ stellt die durch Blender automatisch generierten Rohdaten dar. Diese werden jeweils eingelesen und als RGB-Bild abgespeichert. Um die resultierenden Bounding-Boxen, der sich im Bild befindenden Pfützen besser zu detektieren, wurde das Bild in Graustufen umgewandelt und anschließend Binarisiert. Was beim Binarisieren zu beachten war, ist, dass ein sinnvoller Threshold (Schwellwert) gesetzt werden muss, damit die einzelnen Bildpunkte schwarz oder weiß abgebildet werden können. Ziel ist es, die Pfützen auf dem Boden in Vorder- und Hintergrund zu unterteilen.

Für den weiteren Segmentierung Schritt wurden alle Konturen ermittelt und aufgelistet. Anschließend wurde ebenfalls mit Hilfe eines weiteren Thresholds die minimale Größe einer Pfütze festgelegt, um zu herauszufiltern, dass nicht jeder kleinste Tropfen als vollwertige Pfütze detektiert wird. Am Ende bleibt eine Liste von Koordinaten, der Konturen, um die im Bild eine Bounding-Box der Pfütze aufzufinden ist. Diese Koordinaten werden

dann für jedes Bild in ein spezielles YOLOv7 Format gebracht, damit der Algorithmus mit den Werten arbeiten kann.

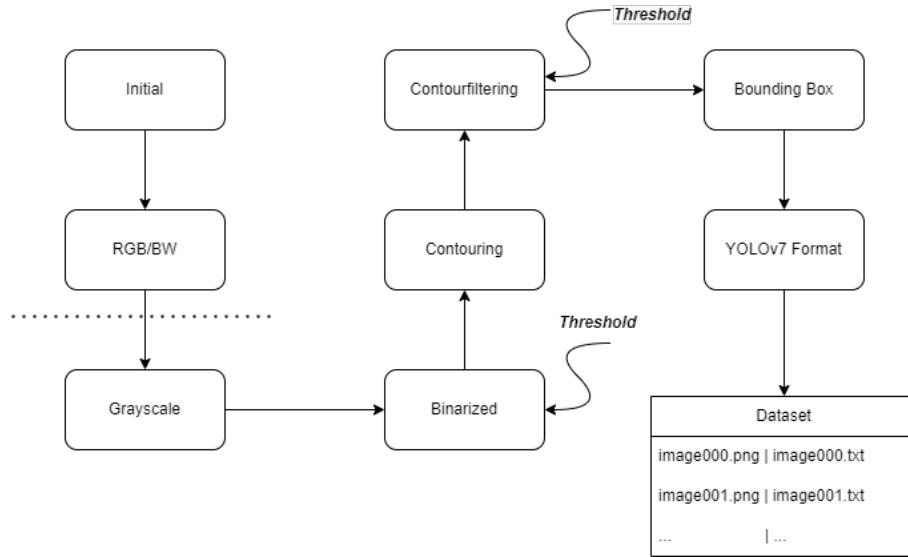


Abbildung 3.3: Ablaufdiagramm zur Trainingsbild-Vorverarbeitung.

Im letzten Abschnitt von Abbildung 3.3 ist ein Beispiel zu sehen, in dem das Bild mit „image000.png“ und die dazu passenden Bounding-Box Koordinaten jeweils paarweise in der „image000.txt“. Dies gilt dann für alle weiteren Bilder des Datensatzes.

3.4.1 Verbesserung des Binarisierung Thresholds

Für die einfach synthetisierten Pfützenbilder waren erste Ansätze der Graustufen Bildung und anschließender Binarisierung leicht umzusetzen. Durch neue, detailliertere Bilder ist ein Threshold jedoch nicht mehr einfach bzw. später kaum umzusetzen. Somit ist eine Auf trennung von Vorder- und Hintergrund nicht mehr möglich. Dadurch, dass die Histogramme kein bimodales Aussehen mehr haben, ist es auch nicht mehr sicher, den richtigen Threshold zu ermitteln 3.4a. Der neue Ansatz, parallel zu den Farbbildern ein schwarzweiß Bild in Blender zu generieren (siehe 3.3.2), vereinfacht die Wahl des Thresholds deutlich, da hier eine eindeutige Unterteilung vorliegt (3.4b).

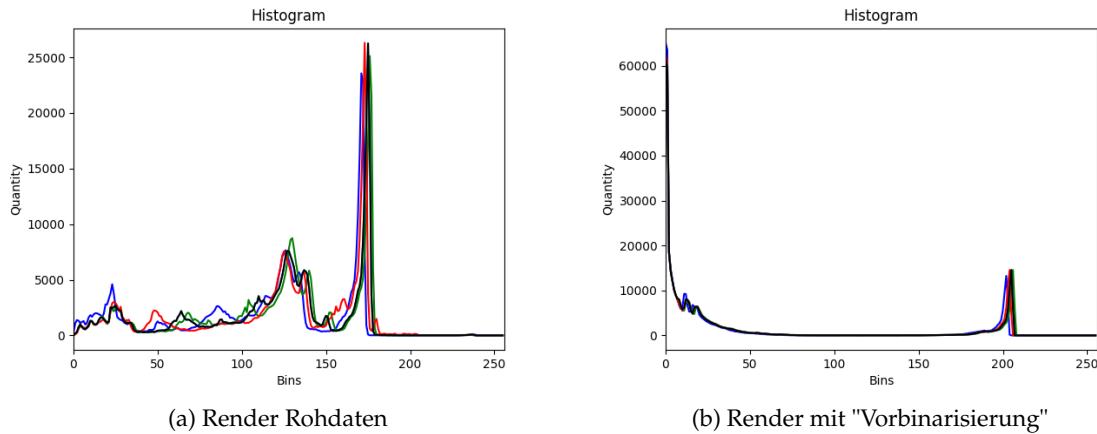


Abbildung 3.4: Vergleich Render Histogramme.

3.4.2 Anwendung des Skripts

Abbildung 3.5a zeigt ein Beispielbild, einer mit Blender generierten Industrieumgebung, in der ein Gabelstapler, Fässer und Schilder zu sehen sind. Bilder aus den neuesten Datensätzen besitzen sogar die Reflexionseigenschaften der Wasserpflüze, im Gegensatz zu den ersten Datensätzen, welche lediglich aus einfachen Pfützen auf Industrieböden bestanden.

Um die Wasserpflüze besser zu lokalisieren, wurde wie erwähnt, zu jedem Farbbild ein passendes pseudo-schwarz-weiß Bild generiert (Abbildung 3.5b), welches die Pfütze direkt vom Hintergrund unterscheidbar macht. Ein solches schwarz-weiß Bild, welches immer noch im RGB-Format existiert, wird in ein Graustufenbild umgewandelt und anschließend mit einem gegebenen Threshold binarisiert (Abbildung 3.5c).

Sobald das binäre Bild vorliegt, werden die Konturen und damit die Koordinaten der jeweiligen Bounding Boxen ermittelt. Zur Visualisierung zeigt Abbildung 3.5d ein Referenzbild für die gefundenen Bounding Boxen, welche um die gefundenen Pfütze platziert wurden. In diesem Fall wurden 6 Pfütze detektiert. Diese sind einfach mit dem Binärbild abzugleichen. Jedoch ist dabei zu beachten, dass nicht jede weiße Kontur als Pfütze definiert wird, da auch hier ein Threshold vorliegt, der die minimale Pfütze-Größe vorgibt. (Siehe oben Links beim Gabelstapler oder oben rechts neben Schild und Fass.)

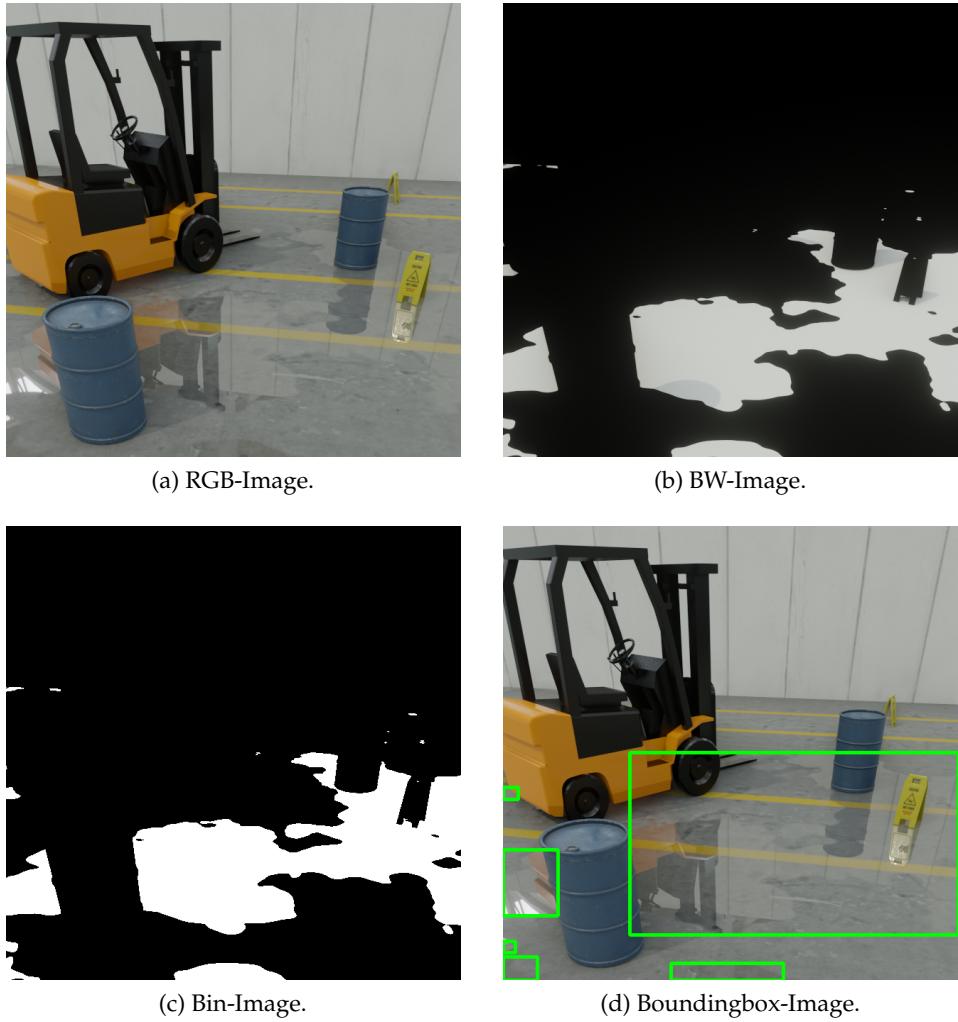


Abbildung 3.5: Verarbeitungszustände.

Wie zuvor in dem Ablaufdiagramm dargestellt, werden auch hier die Koordinaten der Boundingboxen in dem passenden Format abgespeichert. Tabelle 3.1 zeigt die Resultierenden Werte, der 6 in Pfützen, welche in Abbildung 3.5d zu sehen sind. Die Daten werden mit passendem Namen zum Bild als „.txt“ abgespeichert. In diesem Fall ist der erste Parameter das Label. Da hier nur Pfützen detektiert werden, sind keine weiteren Label-Nummern vergeben worden. Die Koordinaten werden hierbei relativ zur Bildgröße angegeben.

Label	x-Pos	y-Pos	Height	Width
0	0.492188	0.981250	0.246875	0.037500
0	0.037500	0.974219	0.075000	0.051562
0	0.013281	0.926562	0.026562	0.025000
0	0.060156	0.785156	0.120313	0.145313
0	0.016406	0.588281	0.032813	0.026562
0	0.639062	0.699219	0.721875	0.401562

Tabelle 3.1: Formatierte Boundingbox-Daten.

Kapitel 4

Training und Evaluation

4.1 Training

Für das Training des Algorithmus steht ein Datensatz mit 1000 Bildern und zugehörigen Labels zur Verfügung. Die eine Hälfte der Bilder enthält in irgendeiner Weise Pfützen, während die andere Hälfte keinerlei Pfützen beinhaltet. Anhand der durch die Labels beschriebenen Bounding-Box-Regionen wird es dem Algorithmus ermöglicht, eine Klassifizierung der Objektklasse Pfütze zu erlernen. Es erfolgt eine Aufspaltung der Bilder in die Kategorien Training, Validierung und Test. Zu diesem Zweck wird eine zufällige Auswahl von 700 Bildern dem Training, 200 Bildern der Validierung und 100 Bildern dem Test zugeordnet.

Das im Anschluss an das Training vorliegende Modell wird daraufhin auf die bisher ungesehenen Bilder der Kategorie Test angewendet. Die daraus resultierenden Ergebnisse werden in Abbildung 4.1 mithilfe der Konfusionsmatrix (engl. confusion matrix) vorgestellt. Eine Bewertung der Ergebnisse findet im abschließenden Kapitel 5 statt.

4.2 Evaluation

Bei der Konfusionsmatrix handelt es sich um eine Qualitätsmessung der Vorhersagen eines Klassifikationsmodells. Zur Bewertung wird hierbei eine Tabelle verwendet, in der die richtigen und falschen Vorhersagen zusammengefasst und nach den Klassen aufgeschlüsselt werden. Der Quadrant links oben in Abbildung 4.1 gibt die Anzahl der korrekten positiven Klassifizierungen (engl. True Positives, TP) wieder. Demnach wird die Anzahl der Pfützen aufgeführt, die vom Algorithmus als solche erkannt wurden. Es kann daraus abgeleitet werden, dass der Algorithmus eine tatsächlich vorhandene Pfütze mit einer Wahrscheinlichkeit von 43% erkennt. Der Quadrant links unten gibt die Anzahl der positiven Klassifizierungen an, die als negativ klassifiziert wurden (engl. False Negatives, FN). Es wird die Anzahl von Hintergründen nachgebildet, die vom Algorithmus

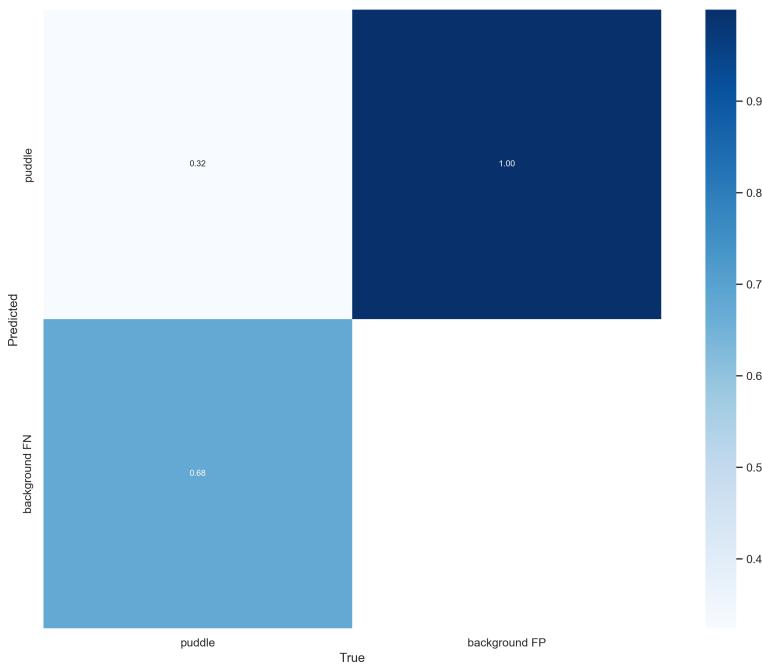


Abbildung 4.1: Konfusionsmatrix des Tests.

fälschlicherweise als Pfützen erkannt wurde. Man kann daraus schließen, dass in 57% aller Fälle, in denen der Algorithmus eine Pfütze erkennt, es sich in Wirklichkeit nicht um eine Pfütze handelt. Der Quadrant rechts oben veranschaulicht die Anzahl der falschen positiven Klassifizierungen (engl. False Positives, FP), während der Quadrant rechts unten die Anzahl der korrekten Ablehnungen (engl. True Negatives, TN) zeigt. Diese Werte werden im vorliegenden Fall ignoriert, da lediglich mit einer Klasse gearbeitet wird.

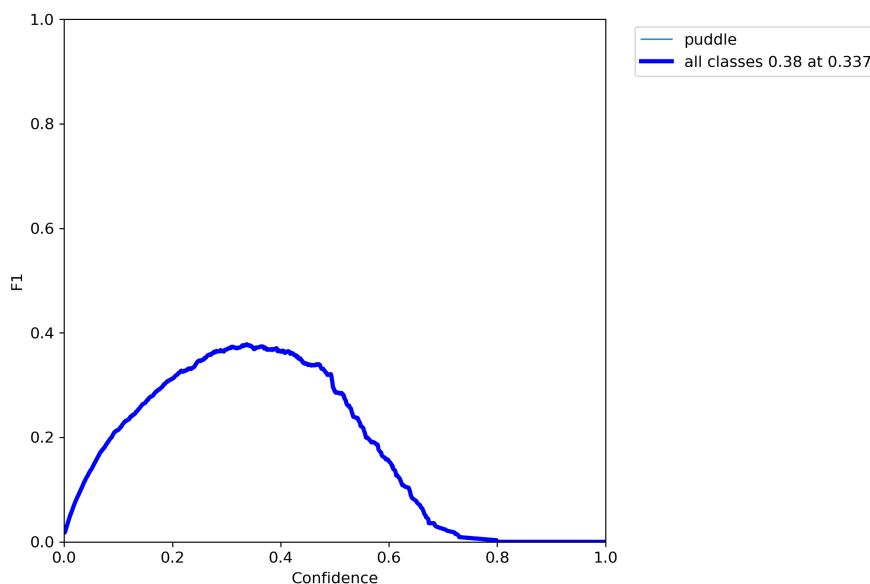


Abbildung 4.2: F1-Kurve des Tests.

Anhand der in Abbildung 4.2 dargestellten F1-Kurve kann der Zusammenhang zwischen Precision und Recall über dem Konfidenzwert visualisiert werden. Der Konfidenzwert drückt eine Schätzung des Modells (auf einer Skala von 0 bis 1) bezüglich der Sicherheit der vorhergesagten Werte aus. Aus Abbildung 4.2 geht hervor, dass Precision und Recall bei einem Konfidenzwert von 0.337 optimiert werden, da an dieser Stelle im Diagramm der F1-Wert mit 0.38 maximal ist.



Abbildung 4.3: Detektion von Pfützen.

Der Konfidenzwert ist für die spätere Nutzung des Modells sehr wichtig, da ansonsten keine optimalen Ergebnisse erzielt werden können. In der praktischen Anwendung wird er in Form eines Parameters dem Programm übergeben, das mithilfe des Modells die Detektion der Pfützen durchführt. Abbildung 4.3 zeigt beispielhaft das Resultat einer solchen Detektion. Es ist zu sehen, dass der Algorithmus zwei Pfützen mit einer Sicherheit von 68% bzw. 67% feststellt. Die Erkennung der linken Pfütze funktioniert hervorragend, indem die dargestellte Bounding-Box die Pfütze nahezu perfekt umschließt. Im Gegensatz dazu sind bei der Erkennung der rechten Pfütze Fehler vorhanden. Zum einen ist die Höhe der Bounding-Box zu groß, da sowohl ein Teil des Gabelstaplers unterhalb der Pfütze als auch ein Teil des Fußbodens oberhalb der Pfütze eingeschlossen wird. Zum anderen ist die Bounding-Box zu breit, da das linke Drittel der Box keine Pfütze beinhaltet.

Kapitel 5

Fazit

Es hat sich gezeigt, dass mithilfe der Software Anaconda und dem offiziellen GitHub-Repository von YOLOv7 bereits in kurzer Zeit gute Ergebnisse erzielt werden können. Das trainierte Modell ist in vielen der Trainingsbeispielen erfolgreich, indem es die Pfützen in den Bildern richtig erkennt. Allerdings gibt es bei der Detektion noch Verbesserungspotential. Hierbei sei insbesondere die Sicherheit genannt, mit der das Modell eine Pfütze erkennt. Diese Angabe liegt in den meisten Fällen unter 70%, weshalb das Modell einerseits keine hochverlässlichen Aussagen trifft und andererseits Pfützen aufgrund eines zu niedrigen Wertes nicht als solche erkannt werden. Bei der Betrachtung der Resultate fällt zudem auf, dass der TN-Wert sehr hoch ausfällt. Eine Häufigkeit von über 50%, mit welcher der Algorithmus eine Pfütze erkennt, es sich jedoch nicht um eine Pfütze handelt, ist nicht industrietauglich.

Dies ist zu einem großen Teil auf zu geringe Variation und Realitätsnähe der Trainingsdaten zurückzuführen. Es erfordert einen hohen Aufwand die Komplexität der erschaffenen 3D-Umgebung zu erhöhen, zudem fehlt es an notwendiger Erfahrung mit Grafiksoftwares um die Qualität der gerenderten Bilder in den Bereich des Fotorealismus zu bringen, was den Nutzen der Bilder für das Training signifikant verringert.

Zukünftige Arbeiten könnten sich daher auf die Verbesserung der generierten Bilder konzentrieren, sodass eine zielführendere Trainingsgrundlage für das Modell geschaffen werden kann. Insgesamt liefert der Klassifikator in einfachen Fällen dennoch gute sowie zuverlässige Ergebnisse und kann deshalb für die Zielanwendung von Nutzen sein.

Um in Zukunft noch bessere Sicherheit mit den Messwerten/Daten zu haben, könnte man auf eine Sensor-Fusion setzen. Dies wäre vor allem bei der Kinect praktisch, da diese neben dem Kamera-Modul (RGB) auch eine spezielle Tiefenkamera mit "Near Infrared" (NIR) besitzt.

Literaturverzeichnis

- [1] Deutsche Gesetzliche Unfallversicherung e.V. (DGUV). *DGUV Information 209-074 - Industrieroboter*. Deutsche Gesetzliche Unfallversicherung e.V. (DGUV), Glinkastrasse 40, 10117 Berlin, Germany, 01 2015.
- [2] Eclipse Foundation. Mqtt broker service. <https://mqtt.eclipseprojects.io/>. [letzter Zugriff: 24. Dez. 2022].
- [3] Python Software Foundation. Python package index. <https://pypi.org/>. [letzter Zugriff: 24. Dez. 2022].
- [4] NVIDIA. Eingebettete systeme mit jetson. <https://www.nvidia.com/de-de/autonomous-machines/>. [letzter Zugriff: 11. Jan. 2023].
- [5] Techletters. Mqtt beginners guide. <https://medium.com/python-point/mqtt-basics-with-python-examples-7c758e605d4>. [letzter Zugriff: 24. Dez. 2022].
- [6] Chris Urmson. Driving beyond stopping distance constraints. *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1189–1194, 2006.
- [7] Wikipedia. Reibungskoeffizient. <https://de.m.wikipedia.org/wiki/Reibungskoeffizient>. [letzter Zugriff: 17. Dez. 2022].