HOCHSCHULE OSNABRÜCK

UNIVERSITY OF APPLIED SCIENCES

Entwicklung eines ROS Wrappers zur Nutzung von KI-Modellen auf Agrarrobotern

Wissenschaftliche Arbeit

vorgelegt von:

Simon Kösters

Studiengang: M.Sc. Informatik - Verteilte und Mobile Anwendungen

Arbeit betreut durch:

Dr. Stefan Stiene

Osnabrück, 22. Dezember 2024

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die moderne Landwirtschaft nimmt eine zentrale Rolle in der Forschung und Entwicklung des 21. Jahrhunderts ein. Durch die steigende Bevölkerungszahl, die bis 2050 auf bis zu 10 Milliarden Menschen geschätzt wird, werden zunehmend mehr Lebensmittel und Anbauflächen benötigt. Allerdings stehen durch die zunehmende globale Erwärmung und die damit verbundenen Extremwetterbedigungen, sowie durch die stetige Auswanderung von Bevölkerungsgruppen aus ländlichen Gegenden in Städte, immer weniger Agrarflächen zur Verfügung, um der steigenden Nachfrage nach Lebensmitteln gerecht zu werden. [1] Um diesen Trend entgegenzuwirken, stehen moderne Landwirtschaftskonzepte wie die Präzisionslandwirtschaft und Agriculture 4.0 immer mehr im Vordergrund. Präzisionslandwirtschaft beschreibt ein landwirtschaftliches Konzept, wobei beispielsweise nicht ein ganzes landwirtschaftliches Feld mit derselben Menge an Düngemitteln behandelt oder ein großer Tierbestand mit der gleichen Menge an Futtermitteln versorgt wird, sondern die unterschiedliche Gegebenheiten innerhalb eines Feldes gemessen und die Dünge- oder Erntestrategie entsprechend angepasst werden. Dies wird häufig durch den Einsatz digitaler Technologien zur Überwachung und Optimierung unterstützt. Die Präzisionslandwirtschaft dient vor Allem zur Steigerung der Quantität sowie der Qualität landwirtschaftlicher Erzeugnisse und zur Reduzierung der benötigten Ressourcen, mit dem Ziel die Kosten und Auswirkungen auf die Umwelt zu senkenr. [2]. Hinzu kommen die Entwicklungen im Bereich Ägriculture 4.0". Dieser Begriff wird als Erweiterung der Industrie 4.0 im Bezug auf die Landwirtschaft verwendet und beschreibt in erster Linie die Verwendung von Technologien wie Robotik, Internet of Things (IoT), Künstlicher Intelligenz (KI) und machine vision und dessen Kombination mit dem Fokus eine nachhaltigere Landwirtschaft entlang der gesamten Prozesskette zu schaffen. Die damit verbundene Nachfrage nach autonomen und intelligenten Robotiksystemen führte dabei zu einer stetigen Weiterentwicklung von Agrarrobotern in Verbindung mit künstlicher Intelligenz, die es erlaubt, dass die Roboter Sensordaten in Echtzeit auswerten und auf die Ergebnisse reagieren können. [3]

Die Nutzung von künstlicher Intelligenz in Verbindung mit autonomen Agrarrobotern bringt jedoch einige Herausforderungen mit sich. Die Roboter sind in der Regel mit zahlreichen Sensoren versehen, dessen Integration in KI-Modelle, häufig individuell an den Anwendungsfall angepasst werden muss und so wenig Raum für Flexibilität lässt. Darüberhinaus setzen die meisten Modelle voraus, dass die gesammelten Daten vor der Eingabe in das Modell vorpro-

zessiert, sowie die Ergebnisse aus dem Modell anschließend nachprozessiert werden müssen.
Diese Arbeit beschäftigt sich daher damit, ein Konzept zur flexiblen Integration von KI-Modellen auf Agrarrobotern mit Hilfe der Robot Operation System(ROS)-Middleware [4] zu entwickeln und umzusetzen. Das Ziel ist dabei einen Wrapper zu erstellen, der es ermöglicht Sensordaten über die ROS-Middleware entgegenzunehmen und diese in ein KI-Modell zu geben, um anschließend die Ausgabe als ROS-Topic zu veröffentlichen, auf das andere ROS Bausteine zugreifen können. Dabei steht die Flexibilität im Vordergrund möglichst einfach Modelle austauschen zu können und unterschiedliche Sensordaten entegegennehmen zu können, sowie die Möglichkeit zu bieten die individuellen Vor- und Nachprozessierungsschritte in den Wrapper zu integrieren, ohne Änderungen am Kern der Anwendung durchführen zu müssen. Dieser Wrapper soll in Zukunft im Agro-Technicum der Hochschule Osnabrück verwendet werden können, um flexibel KI-Modelle auf Roboter ausrollen zu können, die diverse Tätigkeit im Feld erledigen könnnen.

2 Grundlagen

2.1 KI und Robotik in Landwirtschaft

Die voranschreitenden Entwicklungen im Bereich Precision Agriculture und Agriculture 4.0 sorgen vermehrt für die Nutzung verschiedener Robotiksysteme. Dazu wird immer mehr auf Robotiklösungen in Form von Drohnen oder fahrerlosen Bodenrfahrzeugen gesetzt, welche zur Navigation und zur Analyse beispielsweise Bilder oder Sensordaten zur Erkennung von Umwelt, Wetter oder Bodenbeschaffung sammeln. Dies kann je nach Anwendungsfall über verschiedene Sensoren abgebildet werden von Kameras über Lichtsensoren und LiDAR bis hin zu Ultraschall oder Sattelitendaten. [5]
Eine der wohl gängigsten Zusätze eines Agrarroboters ist eine Kamera, die je nach Anwendungsfall sowohl für Navigationszwecke als Hinderniserkennung genutzt werden kann, als auch für die Auswertung von Bilddaten zur Erkennung von Pflanzen und dessen Gesundeitszustand. Zahlreiche Beispiele verwenden Kameras zur Unterstützung im Navigations- und Analyseprozess wie die Auswertung von Bildern zur Wegefindung in Maisfeldern [6], die Erkennung von kranken Pflanzen [7] oder die Erkennung des Reifegrads bei Tomaten [8]. Alternativ zu normalen RGB Kameras lassen sich auch Kameras mit Tiefendaten nutzen, da diese die Möglichkeit bieten Pixeln eine Distanz zuzuordnen und einen dreidimensionales Bild vom Umfeld zu bekommen. Gu et al. [6] zeigen die Nutzung einer Microsoft Kinect Kamera zur Erkennung von Maispflanzenunterlagen und Quan et al. [9] nutzen ein YOLOv4-Modell zur Identifkation von Weizen mit Hilfe von Tiefenkameras. Allerdings werden Kamerabilder häufig durch äußere Einflüsse wie Wetterbedingungen und Lichteinfluss beeinflusst und eignen sich vorallem in Problemstellungen auf offenen Feldern nur bedingt oder mit zeitlichen Einschränkungen [10].
Alternativ wird daher andere Sensorik wie Radar- und Lidarsysteme genutzt, da diese unabhängiger von Wetter- und Lichteinflüssen und insbesondere bei der Hinderniserkennung sehr zuverlässig sind [5]. Diese Lidarsysteme werden in zahlreichen Anwendungen bereits in der Landwirtschaft genutzt. Beispielsweise bietet die Integration eines Lidars auf Bodenrobotern die Möglichkeit Baumstümpfe auf Obstplantagen zu erkennen und deren Höhe, Volumen und Masse zu bestimmen, wodurch sich Aussagen über den zu erwartenden Ertrag ableiten lassen [11]. Weiss et al. [12] zeigen hingegen die Nutzung eines Lidars zur Identfikation von Pflanzen im Boden und eine hohe Erkennungsrate selbst bei geringer Auflösung wie in Abbildung 2.1 gezeigt wird.
Diese gesammelten Sensordaten werden in der Präzisionslandwirtschaft ver-

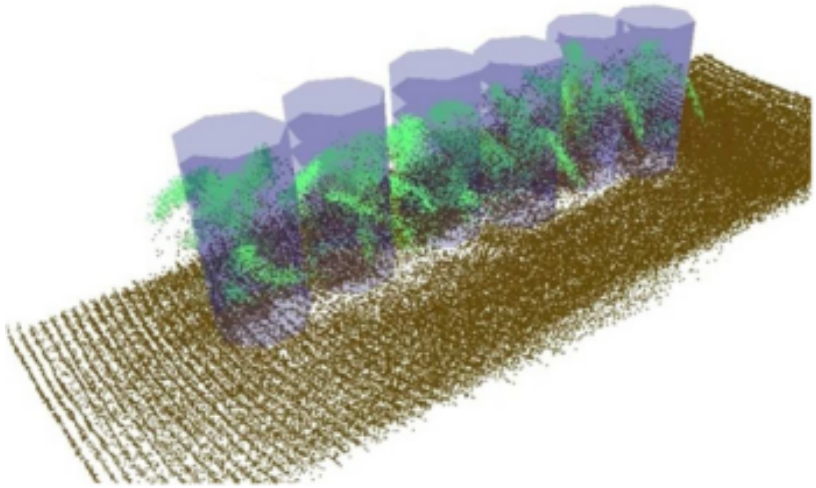

Abbildung 2.1: Erkennung von Maisplfanzen in einer Lidar Punktewolke [12]

mehrt durch künstliche Intelligenz in Form von Deep Learning-Algorithmen ausgewertet, die jedoch häufig mit zentralisierten Systemen abgebildet werden, bei denen die gesammelten Daten von IOT-Systemen an ein zentrales System gegeben werden, welches im Nachhinhein mit Hilfe des jeweiligen KI-Modells die Daten auswertet [13]. Insbesondere die Fortschritte mit gängigen Convolutional Neural Networks (CNN) aus den letzten Jahren in Form von weit verbreiteten Modellen wie AlexNet [14], VGG [15], GoogleNet [16] und ResNet [17], sorgen dafür, dass diese präzise Ergebnisse im Bereich der Bild- und Videoerkennung erzielen können. Beispielsweise zeigen Zhang et al. [18] anhand des GoogleNet models eine erhöhte Krankheitserkennungsrate bei Blättern von Maispflanzen, während Too et al. [19] mit Hilfe vom ResNet und DenseNet Netzen kranke Plfanzen mit einer Genauigkeit von bis zu 99.75% erkennen. Die Nutzung solcher CNNs ist in der Forschung im landwirtschaftlichen Bereich bereits weit verbreitet, basiert aber vorwiegende darauf, Daten mit Hilfe von Kameraaufnahmen zu sammeln, diese an einer zentralen Stelle zusammenzuführen, und anschließend diese auf einer zentralen Recheninstanz oder in einem Cloud Computing Umfeld auszuwerten.

2.2 Robot Operating System

Das Steuern und Orchestrieren von Robotern und dessen Sensorik, insbesondere bei der Nutzung von autonomen Systemen, bringt einige Herausforderungen mit sich. Die benötigte Software ist sehr Komplex, da das System in Echtzeit auf dynamische Umweltänderungen reagieren muss. Hinzu kommen zahlreiche unterschiedliche Sensortypen und Antriebstypen. Der Aufwand eine solche

Anwendung inklusive der benötigten Treiber für die Hardwarekomponenten vollständig selbst zu entwickeln ist enorm, weshalb in den meisten Fällen Abstraktionsebenen in Form von Middleware genutzt werden. Eine Middleware stellt eine Softwareschicht dar, die zwischen dem Betriebssystem und der Benutzeranwendung agiert, und bietet die Möglichkeit vordefinierte Logik zu integrieren, die beispielsweise bei der Ansteuerung von Sensorik hilft. [20]
Das Robot Operating System (ROS) ist eine solche Middleware, die das Interagieren mit verschiedenen Komponenten eines Roboters in einer modularen Weise ermöglicht. ROS wird durch einzelne Nodes abgebildet, die ein Publish/Subscriber Modell nutzen, um untereinander zu kommunizieren. Darüberhinaus bietet ROS eine Vielzahl von verschiedenen Werkzeugen und Libraries zur Entwicklung und Einbindung von Robotikkomponenten. Für viele gängige Komponenten wie Kameras, Lidars oder Motorregler bietet ROS vorgefertigte Nodes, die die Interaktion ohne weitere Eigenentwicklung ermöglichen. Die einzelnen Komponenten publishen die vorliegenenden Daten der Hardware in sogenannte Topics, auf die wiederum andere Nodes über eine Subscription zugreifen können. Der Aufbau ermöglicht somit eine Modularisierung und Abstraktion der Hardwareinteraktion für Robotersysteme und eignet sich für diverse Anwendungsfälle wie die Robotersteuerung, Datenauswertung oder KI-Integration. [4]

2.3 Edge-Computing und Frameworks

Der Ansatz der zentralen Datenverarbeitung ist insbesondere in ländlicheren Gegenden, wo beispielsweise ein Roboter ein lokales Feld abfährt aufgrund von mangelnder Strom- und Internetverbindung problematisch, da die Daten nur mit großer Verzögerung an ein Auswertungssystem gesendet werden könnten [21]. Daher befassen sich andere Ansätze mit einem eher dezentralen Ansatz, bei dem das Ausführen und die Inferenz der KI-Modelle näher an die Datenerfassung herangebracht wird und beispielsweise direkt auf dem Endgerät am Roboter ausgeführt wird.
Das sogenannte Edge-Computing beschreibt den Ansatz die Rechenressourcen näher an den Endnutzer und die Datenquellen zu bringen, um die Probleme der Zentralisierung wie hohe Latenz oder unzureichende Bandbreite zu umgehen. Insbesondere im Bereich der Robotik, wo Sensoren ein sehr hohes Datenvolumen generieren, für die in vielen Anwendungsfällen nicht die Möglichkeit besteht, diese zeitnah an ein zentrales Auswertungssystem zu übermitteln, tritt dieses Problem vermehrt auf. Durch die Entwicklung von leistungsstärkeren Endgeräten in den letzten Jahren wird es möglich auch rechenintensivere Operationen wie die Inferenz von KI-Modellen auf diesen auszuführen [22]. Zahlreiche Sektoren nutzen diese Möglichkeit durch Anwendungen wie selbstfahrende Autos, autonome Roboter, KI-gesteuerte Haushaltsgeräte oder mobile Endgeräte wie Smartphones, wodurch diese sofort auf die Ergebnisse des jeweiligen

KI-Modells reagieren können [23].
In der Landwirtschaft kommen vermehrt Lösungen auf, die ebenfalls Edge-Computing nutzen, um gesammelte Sensordaten direkt auf dem Gerät auswerten zu können. Die Anwendungsfälle reichen von der Präzisionlandwirtschaft, über Wettererkennung bis hin zur Krankheitserkennung bei Tieren [24]. Insbesondere die Verwendung von KI-Modellen wird durch die Fortschritte in Rechengeschwindigkeiten von Endgeräten und Optimierungen in KI-Modellen für den Anwendungsfall Landwirtschaft stetig interessanter und umfasst mittlerweile zahlreiche Themen wie Videoanalyse, Robotik, Erkennung von Pflanzenerkrankungen oder Identifikation von Pflanzenspezies [25].
Majeed et al. [26] zeigen anhand vom Anwendungsbeispiel vorkommende Erkrankungen von Tomatenpflanzen zu erkennen, wie performant verschiedene Edge Devices bei der Inferenz mit unterschiedlichen CNN-Modellen sind. Dazu werden ein Nvidia Jetson Nano Orin, ein Nvidia Jetson AGX Orin, sowie ein Rasperry PI mit und ohne KI-Hardwarebeschleuniger in Form eines Google Corals verglichen. Auf jedem dieser Geräte wurden verschiedene Modelle wie GoogleNet oder ResNet-18 ausgerollt und gemessen wie schnell diese die Klassifikation von Tomatenpflanzen durchführen können. Die Modelle werden dabei in ein einheitliches ONNX-Format überführt und auf dem jeweiligen Gerät in einem vordefinierten Docker-Container ausgeführt, um eine Vergleichbarkeit zwischen den Hardwarekomponenten zu schaffen.
Der Nvidia Jetson AGX Orin und Nano zeigen dabei eine deutlich bessere Leistung von 3,5 und 5,2 ms im Vergleich zum Rasperry PI ohne Hardwarebeschleuniger mit 15,3 ms und beim PI mit Google Coral von 10,5ms pro Klassifikation. Dieser Unterschied lässt sich insbesondere auf die bessere Leistung der Nvidia Geräte zurückführen, sowie deren Möglichkeit den eingebauten Grafikprozessor nutzen zu können. Allerdings bietet der Aufbau mit dem Rasperry PI in Verbindung mit einem Google Coral die Kosteneffizienteste Lösung im Bezug auf Kosten je Bild pro Sekunde. Demnach eignet sich ein Nvidia Jetson Gerät je nach Anwendungsfall eher bei benötigter niederiger Auswertungzeit, während ein Rasperry PI mit Hardwarebeschleuniger bereits eine geeignete Alternative für Anwendungen darstellt, bei der die Geschwindikeit eher vernachlässigt werden kann [26].
Das Sammeln der Sensordaten und Auswertung auf dem Endgerät geschieht dabei häufig mit Hilfe von Middlewares vorwiegend in Form von ROS. Bhavan et al. [27] demonstrieren die Nutzung eiens YOLOv5-Modells auf einem NVIDIA Jetson NX zur Pflanzenerkennung. Dabei wird das YOLOv5-Modell in einem ROS-Node ausgerollt, wobei das Modell nicht nur zur Inferenz genutzt, sondern auch das Training auf dem NVIDIA Jetson ausgeführt wird. Das Modell erreicht Genauigkeiten von bis zu 99% bei hinreichender Trainingsepochenanzahl, wobei das NVIDIA Gerät keine nenneswerte Verlangsamung des Trainingsprozesses zeigt [27].
Auch außerhalb des Anwendungsfalls der Inferenz spielt das ROS eine große Rolle. Insbesondere der ROS Navigation Stack, der zur Steuerung und Weg-

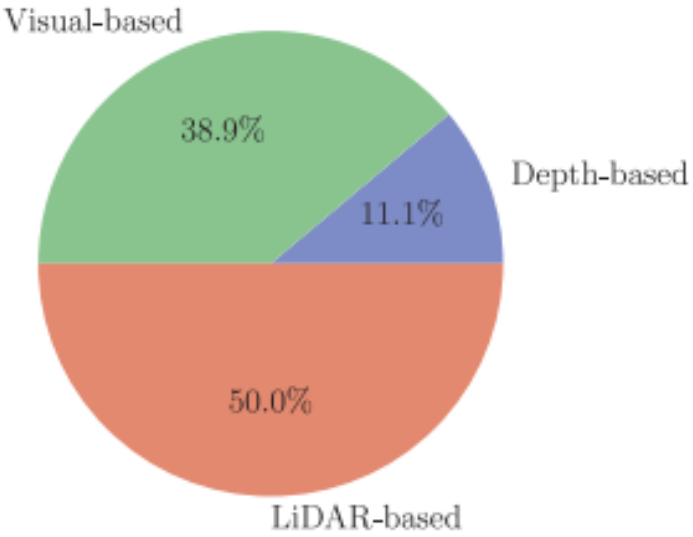

Abbildung 2.2: Darstellung der genutzten Sensorik für die Wegfindung von Agrarrobotern [28]

findung des Roboters genutzt wird, bietet viele Möglichkeiten für Anwendungsbeispiele in der Landwirtschaft. Mit Hilfe von Lidars, RGB-Kameras oder Tiefenkameras in Verbindung mit Klassifizierungsmodellen, lassen sich Hindernisse gezielt erkennen und die Routenfindung auf einem Feld oder in Gewächshäusern in Echtzeit anpassen [28]. Die von Diego et al. [28] angefertigte Auswertung von Projekten, die sich mit der Wegfindung von fahrerlosen Robotern in Feldern und Gewächshäusern beschäftigt, zeigt mit welcher Häufigkeit auf welche der Navigationsmöglichkeiten zurückgegriffen wird. Für diese Analyse wurden insgesamt 148 Artikel untersucht, die sich mit der Wegfindung von Agrarrobotern befassen. Die Verteilung in der Grafik 2.2 zeigt welche Sensorik primär für diese Aufgabe genutzt wird. Die Hälfte der untersuchten Arbeiten bildet die Navigation mit Hilfe eines LiDARs ab, da dieser vor Allem für die Hinderniserkennung am besten geeignet ist und unabhängiger von Umweltfaktoren funktioniert. 38.9% der Arbeiten nutzen stattdessen einen visuellen Ansatz mit Hilfe von gängigen Kameras und 11.1% nutzen Kameras mit Tiefensensor.

Die meisten der genannten Ansätze zur Ausführung von Machine Learning Modellen auf den Edge-Geräten, basiert auf konkreten Anwendungsfällen, bei denen das Überführen der Sensordaten in das jeweilige KI-Modell meist individuell entwickelt wird, ohne übergeordnete Standardisierung, die sich für ähnliche Projekte wieder verwenden ließe. Das liegt insbesondere daran, dass viele dieser Anwendungsbeispiele auf ein spezifisches Problem beschränkt sind und das Vor- und Nachprozessieren der Daten meist an den jeweiligen Fall angepasst werden muss, wodurch es keinen ersichtlichen Standard gibt, nach dem

die Sensordaten eines Roboters innerhalb von ROS in ein KI-Modell gegeben werden. Auch außerhalb des Agrarsektors finden sich kaum Anwendungen, bei denen ein standardisierter ROS-Node genutzt wird, der zentral die Inferenz eines Modells ausführt und an das System übergibt. Jedoch gibt es vereinzelnte Ansatz ein solches Framework zu erstellen.

Das Projekt 'Edge Impulse' umfasst eine Cloud-basierte machine learning Plattform, die das Erstellen, Trainieren und Ausrollen auf dedizierte Edge-Devices von KI-Modellen ermöglicht. Edge Impulse setzt dabei den Fokus insbesondere auf das Konzept 'Tiny Machine Learning' [29]. Das Konzept 'TinyML' umfasst das in den letzten Jahren verstärkt aufkommende Feld der Entwicklung von Machine Learning Modellen für Mikrocontroller und ähnliche Geräte. Die Nutzung von Machine Learning Modellen auf solchen Geräten bringt einige Probleme mit sich, da Mikrocontroller sehr begrenzte Hardwareressourcen haben und die Optimierung der Modelle auf solchen häufig an das jeweilige Gerät angepasst werden muss[30]. Frameworks wie Edge Impulse sollen die Nutzung auf solchen Endgeräten erleichtern, in dem die Modelle vereinheitlicht und für die Nutzung auf ressourcenbegrenzte Endgeräte optimiert werden. Im Projekt Edge Impulse sind zahlreiche Funktionalitäten integriert, die den Weg von der Datensammlung über das Training bis hin zum Ausrollen des KI-Modells auf ein Endgerät erleichtern. Darunter die Möglichkeit das Modell in der Cloud Infrastruktur zu trainieren, eine Vorprozessierung der Daten zu ermöglichen und automatisierte Optimisierung des Modells zur Nutzung auf ressourcenbegrenzten Geräten durchzuführen [29]. Dabei bietet die Plattform zusätzlich die Möglichkeit die Komponenten in ROS zu integrieren [31] und vereinzelte Forschungsprojekte zeigen die Möglichkeit der Nutzung dieser Plattform im Bereich der Agrarwissenschaften, wie die Klassifizeriung von kranken Tomatenpflanzen auf mobilen Endgeräten [32] oder die Klassifizierung von Blättern der Sojabohnenpflanze [33]. Allerdings gehen die Beispiel meist nicht über Bildklassifizierung hinaus und komplexere Anwendungsfälle wie Segmentierungsmodelle sind zum aktuellen Stand nicht mit der Plattform umsetzbar. Der Fokus des Frameworks liegt eher auf mobilen Endgeräten und Mikrocontrollern und wird nur vereinzelt in komplexeren Anwendungsfällen wie autonomer Robotik genutzt.

Einen Fokus auf diesen Aspekt legt hingegen das Projekt 'OpenDR'. Dieses Projekt enstand aus dem Horizon 2020 Research and Innovation Programm der europäischen Union und befasst sich in erster Linie damit, ein standardisiertes Toolkit für die Nutzung von Machine Learning Modellen bereitzustellen. OpenDR umfasst eine Reihe vorgerfertigter Anwendungsbeispiele, trainierte Modelle und standardisierte Datentypen für den Umgang mit Sensorik, die typischerweise in automen Robotiksystemen verbaut ist. OpenDR bietet vorgefertigte Schnittstellen in Python und C++, die das Entgegennehmen von Sensordaten, das Vorprozessieren und die Inferenz durch Machine Learning Modelle ermöglicht. Der Aufbau wird in Abbildung 2.3 dargestellt und umfasst die verschiedenen Werkzeuge, die in OpenDR enthalten sind. Die darin enthaltenen

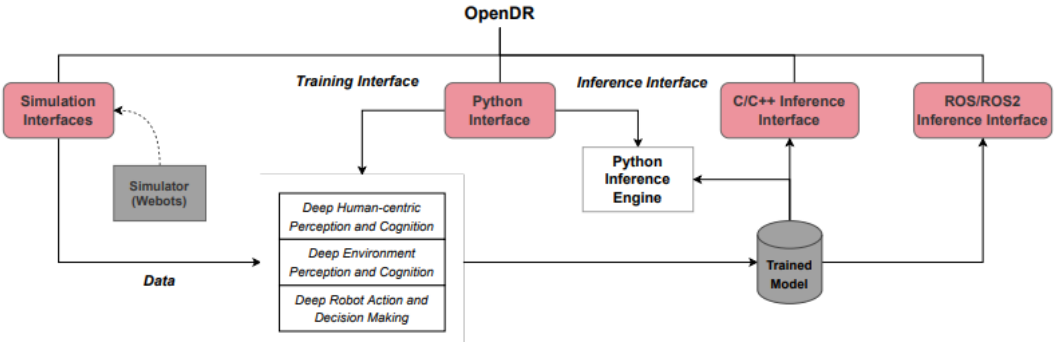

Abbildung 2.3: OpenDR Toolkit [34]

Abstraktionenen von Learner-Klassen und gängigen Datenstrukturen wie Bildern oder Punktewolken, sowie die Integration von vorgefertigetn ROS-Nodes, werden im Rahmen des Projekts in unterschiedlichen Anwendungsbeispielen genutzt. [34]
Die Segmentierung von Punktewolken wird mit Hilfe der OpenDR Werkzeuge anhand vom KITTI-Datansatz gezeigt. Der KITTI-Datensatz beinhaltet eine Vielzahl an Aufnahmen aus dem Straßenverkehr inklusive Bilder und Lidar Punktewolken, welche mit bis zu 28 Klassen versehen ist, die gängige Objekte im Straßenverkehr wie Autos, Fahrradfahrer oder Fußgänger widerspiegeln. Der von Sirohi et al. [35] vorgestellte Ansatz behandelt die panoptische Segmentierung, welche eine Kombination aus der gängigen semantischen Segmentierung (Jedem Pixel eine Klasse zuzuweisen) und der Instanzsegmentierung (Jede Objekt Instanz zu erkennen und zu segmentieren) ist [36]. Das 'EfficientLPS' von Sirohi et al. [35] unterteilt die gesammelten Punktewolken aus dem Datensatz in die einzelnen Instanzen und färbt diese je nach Klassenzugehörigkeit in verschiedene Farben ein. Ein weiteres Beispiel nutzt den HANDS-Datensatz [37], welcher Aufnahmen von Handgesten, die mit einer RGB-Kamera mit Tiefenmessung aufgenommen wurden, enthält. Dabei werden die Daten mit Hilfe der OpenDR Schnittstelle verarbeitet und die Klassifizierung dieser anhand des zugrundeliegenden Machine Learning Modells durchgeführt [38]. OpenDR beinhaltet zahlreiche solcher Beispiele und veranschaulicht, wie verschiedene Sensoren wie Lidar, RGBD-Kameras, Audioaufnahmen und viele weitere über standardisierte Schnittstellen genutzt werden können [34]. OpenDR ermöglicht somit einen standardisierten Umgang mit Sensordaten und Machine Learning Modellen und erlaubt eine einfache Integration in Robotersysteme über die ROS Middleware. OpenDR stellt jedoch keine vollumfängliche Plattform dar, über die Modelle trainiert und ausgerollt werden können, sondern beschränkt sich auf die Standardisierung innerhalb der Verarbeitung. Das Erstellen und Trainieren des Modells, sowie das Ausrollen liegt weiterhin in der Hand des Entwicklers.
Eine solche Plattform bietet hingegen das 'Agri-Gaia' Projekt. Dieses ist spezifisch für die Landwirtschaftsbranche entwickelt und stellt ein Ökosystem zur

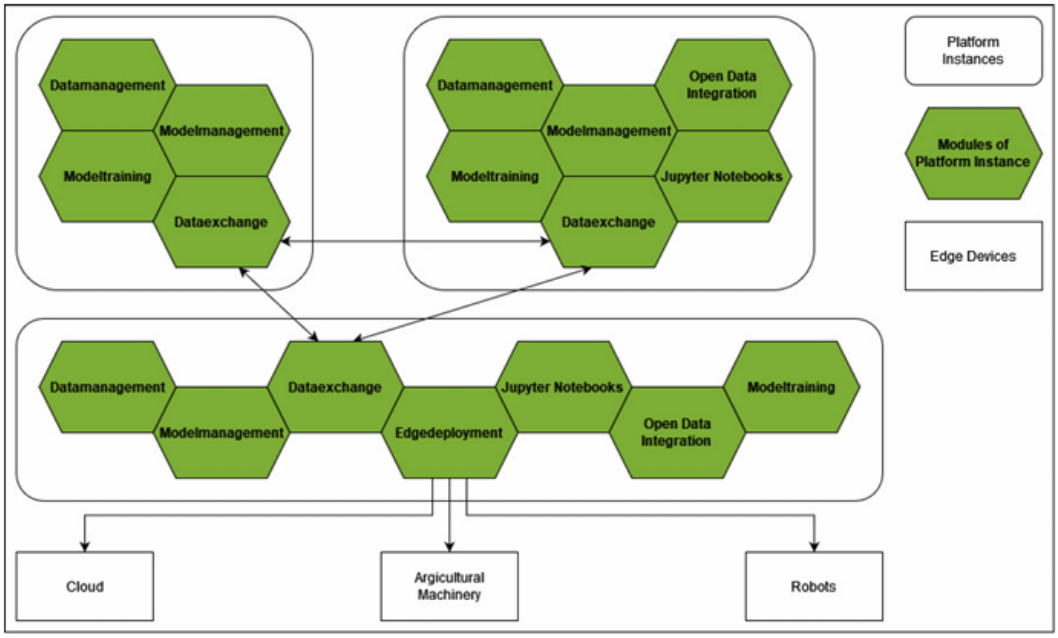

Abbildung 2.4: Darstellung des dezentralen Ökosystems des Agri-Gaia und dessen Komponenten [40]

Verfügung , über das branchenspezifische Daten und KI-Algorithmen über einen B2B-Marktplatz zwischen Firmen aus der Landwirtschaftsbranche ausgetausch werden können. Zusätzlich umfasst das Projekt eine KI-Trainingsplattform, sowie die Möglichkeit Modelle auf Edge-Devices auszurollen und sorgt für übergreifende Standards in der Datenerfasssung und Nutzung von KI-Modellen im Agrarsektor [39]. Das Agri-Gaia Ökosystem nutzt eine dezentrale systemarchitektur, die es den verschiedenen Interessenvertretern und Partnern erlaubt Daten untereinander auszutauschen und unabhängig von einer zentralen Instanz agieren zu können. Dabei können parallel mehrere Plattforminstanzen unabhängig voneinender laufen und die modularen Komponenten wie die Trainingsplattform oder das Datenmanagement bei Bedarf einbinden oder auslassen. Die Plattforminstanzen können sich untereinander vernetzen und so beispielsweise trainierte Modelle, Datensätze oder andere Informationen austauschen. [40] Die Abbildung 2.4 zeigt eine graphische Darstellung der einzelnen Komponenten und wie mehrere Plattform Instanzen untereinander vernetzt werden können.

Jedoch müssen sich die Entwickler selbst um die Integration der jeweiligen Sensordaten in das fertige Modell kümmern, indem sie diese beispielsweise in einen ROS-Wrapper integrieren.

Ein vollumfängliches Projekt, welches das standardisierte Verarbeiten von Sensordaten auf Edge-Geräten ermöglicht und es erlaubt Modelle zu Entwickeln, zu Trainieren und auf das jeweilige Gerät auszurollen fehlt jedoch. Der Großteil der oben genannten Projekte beschäftigt sich in der Regel mit einem Anwen-

dungsfall und implementiert Lösungen, die auf das jeweilige Problem und die genutzte Hardware zugeschnitten sind. Im Rahmen dieser Arbeit soll daher an konkreten Beispielen ein Standard geschaffen werden, der die Nutzung von Machine Learning Modellen mit verschiedenen Sensordaten in einem ROS-Node ermöglicht.

3 Umsetzung

Der zu erstellende Baustein zur Entgegennahme von Sensordaten und Auswertung mit Hilfe von KI-Modellen besteht bereits im Arbeitskreis des Agro-Technicums an der Hochschule Osnabrück in Form eines proof-of-concept Prototypen. Hierbei wurde ein ROS2 Node entwickelt, der ausschließlich RGB-Kameradaten entgegennimmt und die erhaltenen Bilder in ein YOLOv5 Modell gibt. Das verwendete Modell wurde vorher auf die Erkennung von Weizen- und Maispflanzen in Bildern trainiert und gibt bei der Erkennung eine Klassifizierung, sowie Boundingboxen aus, die die erkannte Pflanze umranden. Die Ergebnisse werden anschließend vom Wrapper entgegengenommen und in die ROS Middleware veröffentlicht, sodass zukünftige Bausteine diese auslesen können und beispielsweise die Auswertung auf einem Leitstand anzeigen oder an weitere Feldroboter senden können.
In der Abbildung 3.1 wird die Architektur des Prototypen genauer erläutert.
Die Parameter des erstellten ROS2 Nodes wie die Subscriber und Publisher Topics, sowie der Pfad zum KI-Modell, lassen sich über eine Konfigurationsdatei einstellen. Beim Start des Nodes werden diese Ausgelesen und die jeweiligen Subscriptions erstellt. Sobald beispielsweise eine angeschlossene Kamera Daten sendet, werden diese an das Modell angepasst und eingegeben. Das Modell liegt dabei im gängigen ONNX-Format [42] vor und wird mit Hilfe des 'ONNX-Runtime'-Pakets ausgeführt. Anschließend werden die Ergebnisse ausgewertet und die Ergebnisse formatiert, sodass diese anschließend wieder ausgegeben werden können.
Dieses Konzept ist hinreichend für die Nutzung eines einzelnen Anwendungsfall wie das Erkennen von Pflanzenarten, bietet aber wenig Flexibilität für alternative Anwendungsfälle mit anderen Modellen oder mit anderen Eingabesensordaten. Einerseits bietet der vorhandene ROS2 Node ausschließlich die Möglichkeit RGB-Kameradaten entgegenzunehmen und ermöglicht somit nicht die Nutzung anderer Sensorik wie LiDAR- oder Tiefenkameradaten. Andererseits umfasst das aktuelle Konzept vordefinierte Vor- und Nachprozessierung der Kamerabilder, welche für alternative Anwendungsfälle nur bedingt nutzbar sind. Um demnach andere deep-learning Modelle nutzen zu können, die nicht auf Objekterkennung ausgelegt sind braucht es ein flexibleres Konzept, mit dem die Vor- und Nachprozessierung der Daten an den jeweiligen Anwendungsfall angepasst werden können.

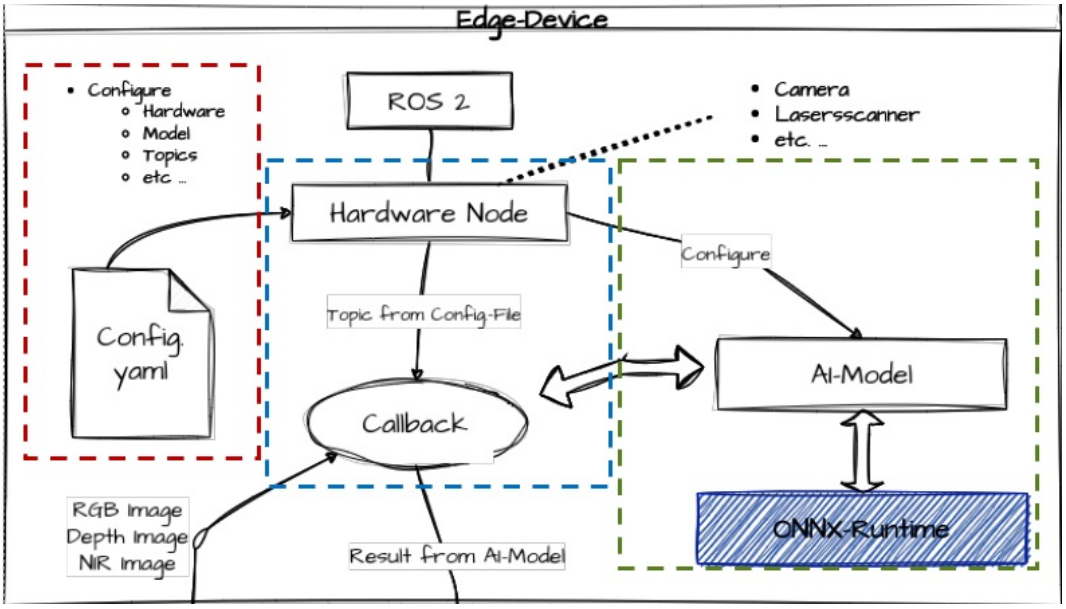

Abbildung 3.1: Konzept des existierenden Demo ROS2 Wrappers [41]

3.1 Konzeption und Planung

3.1.1 Deep Learning mit ROS

Die ROS2-Middleware enthält vordefinierte Datentypen die zur Kommunikation von Ergebnissen genutzter KI-Modelle dienen, sowie definierte Typen, die die unterschiedlichen Arten von Sensordaten abbilden. Wie in Kapitel 2 beschrieben, sind die wohl verbreitetsten Sensoren für autonome Agrarroboter RGB-Kameras, LiDAR-Sensoren und Tiefenkameras.
Sowohl RGB- als auch Tiefenkameradaten lassen sich im ROS2 über das 'Image' Objekt abbilden, welches Information über jeden Bildpixel speichert. Die Daten eines LiDAR-Scanners werden in der Regel über eine Punktewolke abgebildet, und durch die x,y und z Koordinate in einem 3-Dimensionalen Koordinatensystem abgebildet. Zusätzlich liefert der LiDAR die Intensität des Punktes bei der Messung, welche die Stärke des gemessen Impulses abbildet und somit beispielsweise Rückschlüsse über die Art des Objektmaterials zulässt. Dabei lassen sich auch Daten von beispielsweise 2D Laserscannern oder Tiefenkameras in Punktewolken überführen, falls diese für den Anwendungsfall benötigt werden.
Die Ergebnisse eines deep learning Modells hängen stark vom Anwendungsfall ab. Wie in Kapitel 2 bereits angerissen, gibt es drei Anwendungsfälle, die größtenteils bei der Inferenz mit Robotersensorik genutzt werden:

- *Klassifizierungsmodelle*, die die gesamte Eingabe einer Klasse zuordnen und in der Regel die Wahrscheinlichkeit der einzelnen Klassenzugehörigkeiten ausgibt

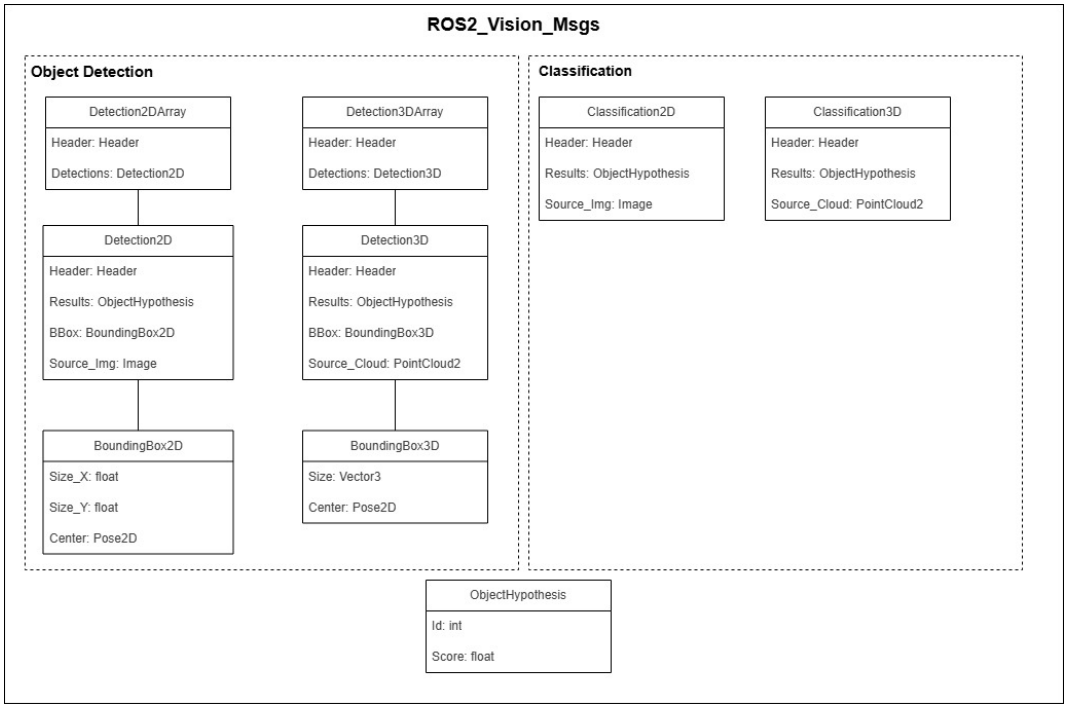

Abbildung 3.2: Eigene Darstellung des ROS2 vision-msgs Paket basierend auf der ROS2 Dokumentation [43]

- *Objekterkennungsmodelle*, die Objekte innerhalb der eingegeben Daten erkennt und in der Regel mit Hilfe von Boundingboxen diese markieren
- *Segmentierungsmodelle*, die jedem Datenpunkt eine Klasse zuordnen und somit beispielsweise durch darauffolgende Einfärbung erlaubt die verschiedenen Objekte genauer zu unterscheiden

Diese Modellarten umfassen nicht alle Möglichkeiten, da zahlreiche weitere deep learning Ansätze existieren, jedoch bilden diese drei Arten von Modellen einen Großteil der gängingen Anwendungsfälle ab, sodass ausschließlich diese im weiteren Verlauf des Projekts genauer betrachtet werden.
ROS2 bietet vorgefertigte Datentypen, die die Kommunikation mit den Ergebnissen solcher Modelle erleichtern soll. Die Abbildung 3.2 stellt diese graphish dar und zeigt welche Typen für welche Art Modellergebnis genutzt werden können.
Für die Objekterkennung in Bildern wird in der Regel das *Detection2DArray* genutzt, welches einerseits das Eingabebild und andererseits die erkannten Boundingboxen, sowie die Bewertung dieser enthält. Das Selbe gilt für das *Detection3DArray*, wobei dieses anstelle eines Bildes eine Punktewolke als Eingabewert voraussetzt und die Boundingboxen dreidimensional mit Hilfe eines Vektors abbildet.
Wird hingegen lediglich eine Klassifizierung des eingegeben Bildes oder der eingegebenen Punktewolke benötigt, lassen sich diese mit Hilfe der *Classifica-*

tion2D und *Classification3D* Klassen abbilden, in denen zusätzlich zur Originaleingabe die jeweils zugeordnete Klasse, sowie die Bewertung mit welcher Sicherheit sie dieser zuzuordnen ist. Zu beachten ist hierbei, dass diese Typen in der neueren ROS Humble Version durch ein gemeinsames *Classification* Objekt ersetzt wurden. Aus Kompatibilitätsgrunden wären beide Arten im weiteren Verlauf betrachtet, um Endgeräte, die nicht auf den neuesten Softwarestand sind weiter nutzbar zu halten.
Für Segmentierungsmodelle gibt es keinen dedizierten Datentypen, jedoch lassen sich diese Ergebnisse direkt in die Eingabedatentypen *Image* und *PointCloud2* projezieren, da die Klassifizierung auf Pixel- bezhiehungsweise Punktebene stattfindet.
Im bereits existierenden Prototypen des ROS2 Wrappers wird ausschließlich der Anwendungsfall der Objekterkennung in zweidimensionalen RGB-Bildern betrachtet. Im Rahmen dieses Projektes sollen daher die drei genannten Modellarten nutzbar sein und mit Hilfe der ROS2 Datentypen dynamisch an den Fall angepasst und ausgegeben werden.

3.1.2 Vor- und Nachprozessierung von Daten

Damit Bild- oder Punktewolkendaten in ein deep learning Modell gegeben werden können, müssen diese in den meisten Fällen vorprozessiert und somit an das jeweilige Modell angepasst werden. Die Vorprozessierung kann dabei einfache Operationen wie das Verkleinern/Vergrößern von Bildern oder das Ausschneiden bestimmter Bildpunkte, sowie komplexere Operationen wie das Herausfiltern von Störsignalen in LiDAR Daten umfassen. Alle möglichen Vorprozessierungsmethoden für Bild- und Punktewolkendaten in einem Projekt zusammenzufassen ist aufgrund der unterschiedlichen Anforderungen einzelner deep learning Modelle kaum möglich, da nicht alle Modelle mit gleichen Eingaben operieren und je nach Modell einige Schritte einzigartig für die jeweilige Modellart sein können. Demnach gilt es die gängigsten Arten der Vorprozessierung für Bild- und Punktewolkendaten einzubinden und eine möglichst erweiterbare Plattform bereitzustellen, die die Einbindung individueller Vorprozessierungsschritte ermöglicht.
Dafür gilt es auf der einen Seite die möglichst gängigsten Schritte der Vor- und Nachprozessierung herauszuarbeiten, um eine Vielzahl von Anwendungsfällen ohne große Implementierungsaufwand abbilden zu können und auf der anderen Seite zu ermöglichen, diese flexibel an den Anwendungsfall angepasst einzubinden. Es existieren bereits Softwarestacks, die sich mit dem Anwendungsfall der Prozessierung von rohen Sensordaten befassen wie die ROS image_pipeline [44]. Diese gehört zum ROS Softwarestack und stellt einzelne ROS-Nodes zur Verfügung, welche beispielsweise das Kalibrieren von Monokular- und Stereokameras, das Herausfiltern von Kameraverzerrung und andere Vorprozessierungsmethdoen, sowie das Nutzen von Tiefenkameras inklusive des Umwandels in Punktewolken ermöglicht.

Abbildung 3.3: Beispielhafte Prozessierung eines Bildes mit der ROS2 image_pipeline [45]

Abbildung 3.4: Bildprozessierung mit der Nvidia Isaac ROS-pipeline [46]

Für die einzelnen Prozessschritte wie das Ausschneiden bestimmter Bildpunkte oder das Entzerren von Kamerabildern bietet die image_pipeline einzelne Nodes an, die je nach Gebrauchsfall eingebunden werden können. Somit lassen sich beispielsweise verzerrte Kamera Bilder entzerren und mit einem Bayer-Filter bearbeiten wie im Beispiel der Abbildung 3.3 gezeigt wird.
Ein ähnlicher Ansatz wird in Nvidias Isaac ROS Paketen genutzt, welche für NVIDIA Hardware optimierte ROS Bausteine enthält, die beispielsweise das Vorprozessieren von Bilddaten ermöglicht. Hier werden ebenfalls modulare Bausteine zur Verfügung gestellt, die Schritte wie das Entzerren, Größenänderungen oder Farbraumkonvertierungen beinhalten, wie in der Abbildung 3.4 gezeigt wird.
Beide Pakete bieten einen modularen Ansatz die einzelnen Vorprozessierungsschritte für Bilddaten auszuführen und unterstützen zusätzlich zu normalen RGB-Kameradaten auch das prozessieren von Tiefenkameradaten. Jedoch sind die integrierten Anwendungsfälle nur bedingt für die meisten Arten von Klassifizierungs- und Objekterkennungsmodellen nutzbar, da in einigen Fällen je nach Bilddaten noch zusätzliche Schritte durchgeführt werden müssen, um die Bilder an das Modell anzupassen. Zusätzlich bringt die Nutzung von separaten Vorprozessierungsnodes zusätzlichen Orchestrierungsaufwand und weitere Fehlerquellen mit sich, da stets der Status der Nodes überwacht werden muss und bei Fehlern oder Ausfällen der Nodes auf diese reagiert werden

muss. Die Pakete eignen sich insbesondere für einfache Anwendungsfälle mit gewöhnlichen Kameradaten, bieten jedoch wenig Flexibilität in der Anpassung und beinhalten keine Bausteine, die sich mit der Nutzung von beispielsweise LiDAR Sensoren auseinander setzen. Dennoch zeigen die beiden Beispiele, wie mit einem modularen Ansatz Sensordaten den Anwendungsfällen entsprechend vorprozessiert werden können und welche Operationen gängig sind.
Zusätzlich zur Vorprozessierung, müssen die Ausgabedaten aus den deep learning Modell häufig ebenfalls nachprozessiert werden. Auch hier hängen die benötigten Schritte stark von der Art des Modells ab. Beispielsweise werden bei Segmentierungsmodellen typischerweise die Bilder oder Punktewolken abhängig von der zugeordneten Klassenzugehörigkeit eingefärbt, um diese sichtbar zu machen. Alternativ gilt es beispielsweise bei Objekterkennungsmodellen die Sicherheiten der Klassenzugehörigkeit auszuwerten und nur Boundingboxen mit einer Mindestsicherheit anzuzeigen oder bei Klassifizierungsmodellen die jeweilige Zugehörigkeit als Label an das Bild anzuhängen.
Darüberhinaus gibt es viele weitere Beispiele, wie die Ergebnisse aus deep learning Modellen von Sensordaten ausgewertet werden, was es erschwert dafür einheitliche Lösungen zu erstellen, die möglichst viele Fälle abdecken. Für die Ausgabe der Ergebnisse können die oben genannten ROS2 vision_msgs-Klassen genutzt werden, jedoch müssen die Ergebnisse auf die Struktur dieser zugeschnitten werden, um eine einheitliche Ausgabe zu gewährlisten.
Zusammengefasst gilt es demnach die Möglichkeit zu bieten, Vor- und Nachprozessierungsschritte möglichst modular und flexibel in den zu erstellenden ROS2 Node einzubinden. Somit soll ähnlich wie bei der ROS2 image_pipeline die Möglichkeit gegeben sein, die Schritte als modulare Einheiten aneinander ketten zu können. Jedoch soll aufgrund des Orchestrierungsaufwands die Ausführung der Schritte nicht in separaten Nodes stattfinden, sondern im zu erstellenden Wrapper gesammelt werden.

3.1.3 Konzept

Der Prototyp des ROS2 Wrappers beinhaltet die oben genannten Konzepte nur bedingt. Die Implementierung ist ausschließlich auf den Anwendungsfall der Erkennung von Mais- und Weizenpflanzen angepasst. Demnach sind die Vor- und Nachprozessierungsschritte, sowie die genutzten Ein- und Ausgabedatentypen statisch an die Anwendung angepasst und ließen sich nur mit zusätzlicher Implementierung an andere Modelle anpassen.
Ein erweiteter ROS2 Node sollte demnach folgende Kriterien erfüllen:

- **Flexibilität:** Der ROS2-Wrapper soll es ermöglichen flexibel zwischen verschiedenen Modellen und Eingabetypen zu wechseln, ohne große Änderungen an der Implementierung des Wrappers vornehmen zu müssen
- **Erweiterbarkeit:** Da die Einbindung jedes möglichen Modells und den verschiedenen Vor- und Nachprozessierungsschritten in einem einzigen

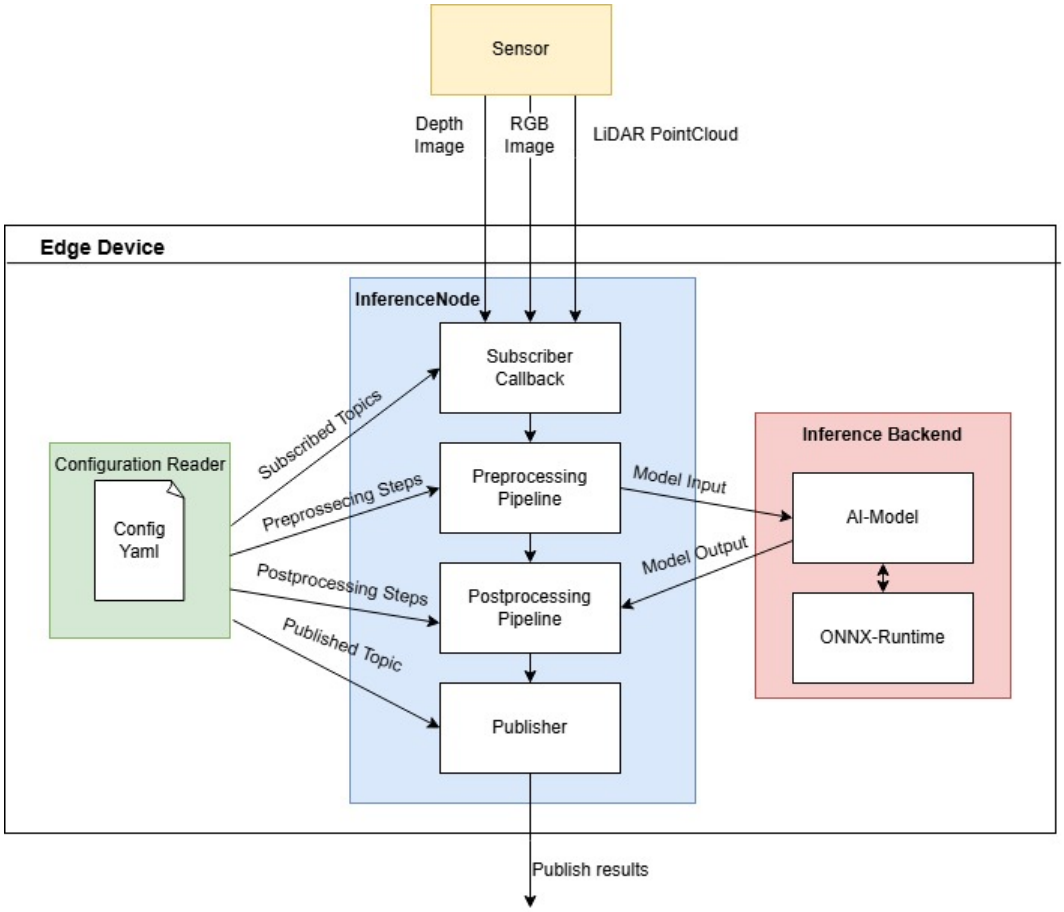

Abbildung 3.5: Konzept für den Aufbau des ROS2 Nodes

Baustein kaum möglich ist, soll der Wrapper die Möglichkeit bieten, neue Modellarten und Prozessierungsschritte in die Software einbinden zu können, ohne den Kern der Anwendung anpassen zu müssen

- **Performanz:** Der Wrapper soll trotz hoher Flexibilität möglichst performant operieren, da bei den genutzten Sensordaten hoher Datenverkehr auftreten kann

Basierend auf den genannten Kriterien wurde das Konzept des vorhandenen ROS2 Wrappers weiterentwickelt, sodass das Abbilden von Vor- und Nachprozessieren innerhalb des Nodes ermöglicht wird. In Abbildung 3.5 wird der grobe Aufbau exemplarisch dargestellt.
Die Funktionalität des zentralen Hardware Nodes wird um einen Pre- und Postprocessing Baustein erweitern. Die einzelnen Schritte werden durch die hinterliegende Konfiguration vorgegeben und sollen in einer modularen Art und Weise implementiert werden, die es ermöglicht, dass die Schritte beliebig verbunden werden können. Dafür wird ein neues Konzept entwickelt, wie die hinterlegte Konfigurationsdatei erweitert werden kann, damit flexibel zwischen

den Ein- und Ausgabetypen, sowie den Vor- und Nachprozessierungsschritten gewechselt werden kann. Die Idee wird in folgender YAML Datei festgehalten, die als erstes Konzept erarbeitet wurde:

```yaml
model_path: Dummy.onnx

sub_topics:
  - topic: rgb-camera
    subscribed_type: Image
    preprocessing_steps:
    - name: Step 1
      params:
        - name: Param 1
          value: 1
        - name: Param 2
          value: 2
  - topic: depth-camera
    subscribed_type: Image
    preprocessing_steps:
    - name: Step 1
      params:
        - name: Param 1
          value: 1

    - name: Step 2
      params:
        - name: Param 1
          value: 1

postprocessing_steps:
  - name: Step 1
    params:
      - name: Bounding Box
        value: X

publish_type: Detection2DArray
pub_topic: /BoundingBoxes
```

Die Grundidee des Konzepts ist es, flexibel Datentypen und Prozessschritte einzubinden. Dabei soll es ermöglicht werden beliebig viele Sensortypen zu integrieren, da einige Modelle beispielsweise Punktwolken und RGB Kameradaten oder RGB- und Tiefenkameradaten entgegennehmen. Abhänging vom konfigurierten Subscriber-Typen soll ein ROS2 Subscriber erstellt werden, der die jeweiligen Daten entgegennimmt und bei mehreren konfgurierten Sensoren diese vor der Eingabe in das Modell synchronisiert. Für jedes der konfigurierten Topics können beliebig viele Vorprozessierungsschritte angehangen werden. Diese sollen einem standardisierten Format unterliegen, welches vorgibt, dass in jeden Schritt die Daten aus dem vorherigen Schritt übergeben werden, und das jeder Schritt die weiterprozessierten Daten wiederum ausgibt. So ließen sich beispielsweise gängige Bildoperationen aneinander Ketten wie das Vergrößern, gefolgt vom Normalisieren und Ausschneiden, um diese anschließend in das

Modell übergeben zu können. Darüberhinaus können Typanpassung und Datenumformungen in den Schritten übernommen werden. Für die Inferenz der Modelle wird wie im erstellten Prototyp ONNX [42] genutzt, da das Format die Möglichkeit bietet Modelle einfach auszutauschen und zu im- und exportieren, ohne dass der gesamte Code in die Software integriert werden muss.
Dieser Ansatz bietet sehr viel Flexibilität, birgt jedoch die Gefahr, dass die Konfigurationsreihenfolge fehlerhaft sein kann. Das Risiko der Fehlkonfiguration, soll durch eindeutige Namensgebung, sowie ausführliche Dokumentation möglichst gering gehalten werden, um somit dem konfigurierenden Entwickler möglichst viel Arbeit abzunehmen. Die prozessierten Sensordaten können anschließend in das KI-Modell gegeben werden, woraufhin die Ausgaben in die konfigurierten Nachprozessierungsschritte gegeben werden. Hier gilt das gleiche Konzept, wie bei der Vorprozessierung, sodass die Erbebnisse jedes Schrittes in den nächsten Schritt übertragen werden können. Zusätzlich soll in der Konfiguration angegeben werden, welcher Ausgabetyp erwartet wird, sodass ein gegebener Publisher erstellt werden kann und die Ergebnisse an den Typ angepasst werden können. Dieser Ablauf wird zusätlich in Abbildung 3.6 graphisch dargestellt.

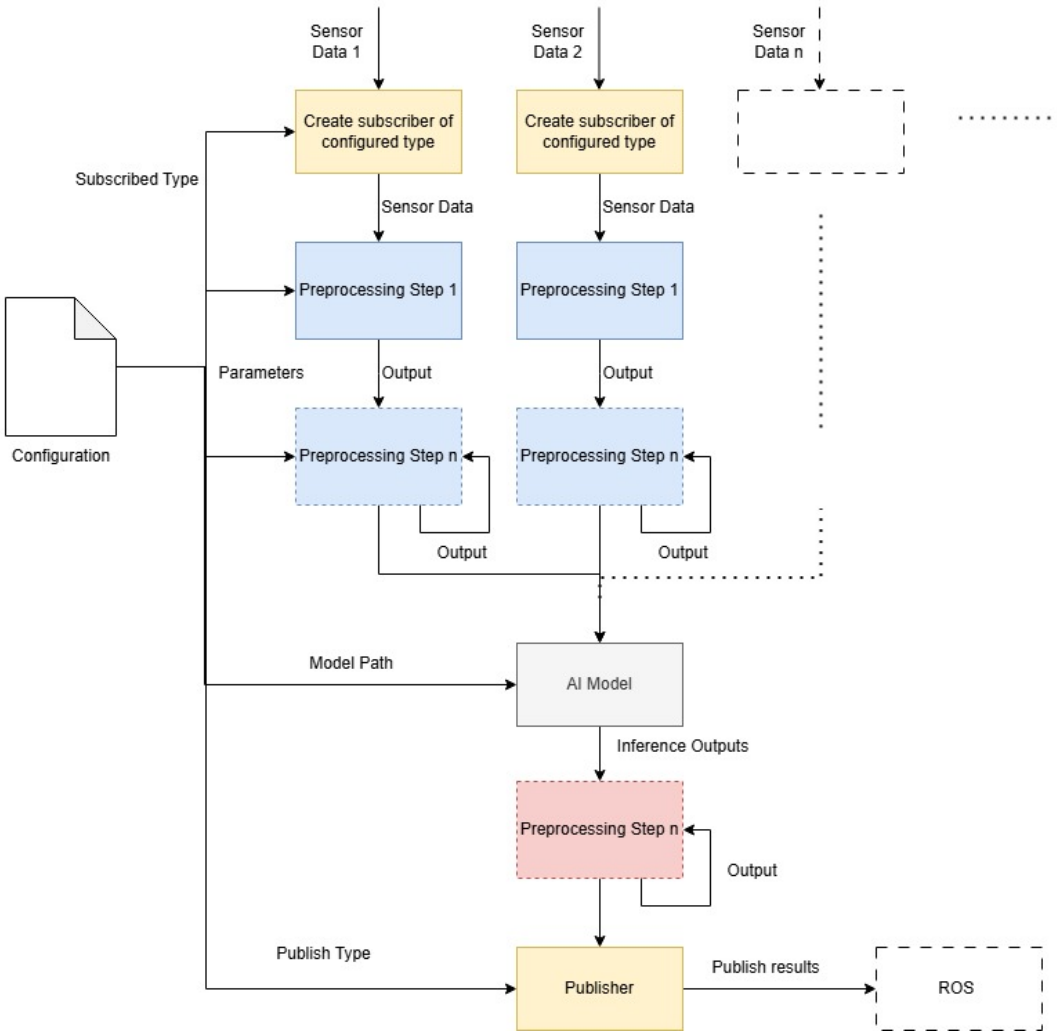

Abbildung 3.6: Konzept für den Ablauf der Prozessschritte

3.2 Entwicklung des ROS Wrappers

Um das entwickelte Konzept auf möglichste unterschiedliche Modelle anpassen zu können wurden zahlreiche Modellarten und Anwendungsbeispiele untersucht. Dafür wurden Modellsammlungen wie der ONNX-Modelzoo betrachtet. Dieses Projekt sammelt unterschiedliche ONNX-Modelle [47] mit verschiedenen Anwendungsfällen wie Klassifizierung, Objekterkennung, Bildsegmentierung und viele mehr. Zusätzlich wurden gängige Modelle untersucht, die neben RGB-Kameradaten auch Tiefenkamera- oder LiDAR-Sensordaten entegegennehmen. Aus diesen Beispielen lassen sich Vor- und Nachprozessierungsschritte ableiten, die vermehrt auftreten und somit in einer standardisierten Form in den ROS2 Node integriert werden können.
Die herausgearbeiteten Prozessschritte werden dabei möglichst modular gewählt, sodass diese für unterschiedliche Projekte wiederverwendet werden können. Dazu wird pro ausführbaren Schritt eine Methode erstellt, deren Eingabe eindeutig durch die gegebenen Parameter definiert wird. Jeder dieser Schritte folgt dem Schema als ersten Eingabeparameter den Wert *data* zu erwarten. Dieser wird dynamisch bei der Ausführung der Vorprozessierungsschritte mit dem Wert des vorangegangenen Schrittes gefüllt. Folgender Code-Ausschnitt zeigt wie die Vorprozessierung im Detail abläuft:

```
1  def preprocess_pipeline(self, orig_input, topic_num, configs):
2      #Set data to original input for first step
3      kwargs = {"data": orig_input}
4      for step in
       ↪  configs["sub_topics"][topic_num]["preprocessing_steps"]:
5      #Execute all configured preprocessing steps for sub_topic x
6          if step["params"] != None:
7              for param in step["params"]:
8                  #Add configured parameters from yaml to keyword
                   ↪  arguments
9                  kwargs[param["name"]] = param["value"]
10
11         #Call configured method with arguments
12         input = globals()[step["name"]](**kwargs)
13
14         #Add output of method to input for next step
15         kwargs = {"data": input}
16
17     return input
```

Dieser Ablauf gewährt die Möglichkeit beliebig viele Schritte aneinander zu ketten und die Ausgaben jedes Schrittes weiter zu geben. Dadurch können die Sensordaten in beliebiger Reihenfolge mit den für das Modell benötigten Schritten bearbeitet werden, sodass am Ende die vorprozessierten Daten herauskommen, die direkt in das ONNX Modell gegeben werden können. Diese

Logik wird ebenfalls für die Nachprozessierung genutzt. Zu beachten ist hierbei, dass die Ausgabe der Modelle in der Regel keinem Standard folgt und je nach Aufbau des neuronalen Netzes unterschiedlich ausfällt. Die Ausgaben können von Boundingbox-Größen über Klassenzugehörigkeiten bis hin zu Klassenzuordnungen pro Pixel/Punkt reichen, weshalb es schwer fällt diese in einen standardisierten Datentypen zu überführen. Daher werden die Nachprozessierungsschritte auf die selbe Art wie die Vorprozessierung ausgeführt, wodurch erlaubt wird benutzerdefinierte Schritte hinzuzufügen, falls diese benötigt werden, ohne den Kern der Anwendung anpassen zu müssen.
Zusätzlich werden für die Vor- und Nachprozessierung Methoden zur Datentypumwandlung bereitgestellt. Dazu gehören beispielsweise die Umwandlung eines Bildes in ein Python Array, die Umwandlung von Tiefenbildern zu Punktewolken oder die Umwandlung von Modellausgabewerten in ROS2 Nachrichten wie *Detection2DArray* oder *Classification*. Der vollständige Ablauf der Inferenz von Bilddaten, sieht dabei beispielsweise aus wie in Abbildung 3.7.
Der ROS Node erstellt bei der Initialisierung die Subscriber und Publisher, die in der Konfigurationsdatei definiert werden. Darauf folgt in der Regel als erstes eine Typumwandlung des ROS *Image*-Objekts in einen für das Modell passenden Datentypen wie Array. Anschließend folgen beliebig viele Vorprozessierungsschritte wie zum Beispiel das Ändern der Bildgröße und das Transponieren, um die Daten an die Eingabeparameter des Modells anzupassen. Wurden alle Schritte ausgeführt, werden die Daten in das Modell gegeben und über die ONNXRuntime ausgeführt. Anschließend werden beliebig viele Nachprozessierungsschritte ausgeführt, wie das Zurückumwandeln der Bildgröße, gefolgt von der Umwandlung in den Ausgabetypen. Die dynamischen Vor- und Nachprozessierungsschritte werden aus den Methoden in den jeweiligen Dateien importiert und über den Namen in der Konfigurationsdatei ausgeführt.
Um zu gewährleisten, dass die Methodennamen einzigartig sind und nicht von anderen Paketen genutzt werden, folgen die Schritte einer klaren Namensstruktur. Beispielsweise werden Bildoperation unterteilt in die Vor- und Nachprozessierungsoperationen und starten bei der Namensgebung immer mit dem prefix *image_*. Um einen neuen Schritt hinzuzufügen muss demnach lediglich eine Methode in der jeweiligen Datei hinzugefügt werden, die dem Namensschema folgt, die Ausgabe eines vorausgehenden Schrittes entgegennimmt und die prozessierten Daten wieder ausgibt. In dieses Konzept lassen sich beliebig viele Modelle integrieren, was im Folgenden Kapitel genauer beschrieben wird.
Da die Ausführung des Nodes in Zukunft auf einem Edge-Gerät wie einem Nvidia Jetson Orin stattfindet, gilt es zusätzlich ein Konzept zu entwickeln, wie das Ausrollen des Nodes geschehen soll. Wie in Kapitel 2 beschrieben, bietet die Plattform Agri-Gaia die Möglichkeit Edge-Geräte hinzuzufügen und das Ausrollen von Software und KI-Modellen zu übernehmen. Dieses Konzept wurde bereits im vorhandenen Prototypen umgesetzt und geschieht anhand eines erstellten Docker-Containers, der den ROS2 Node aufruft und kontinuierlich ausführt. Andere Ansätze zur Ausführung von Modellinferenz in Contai-

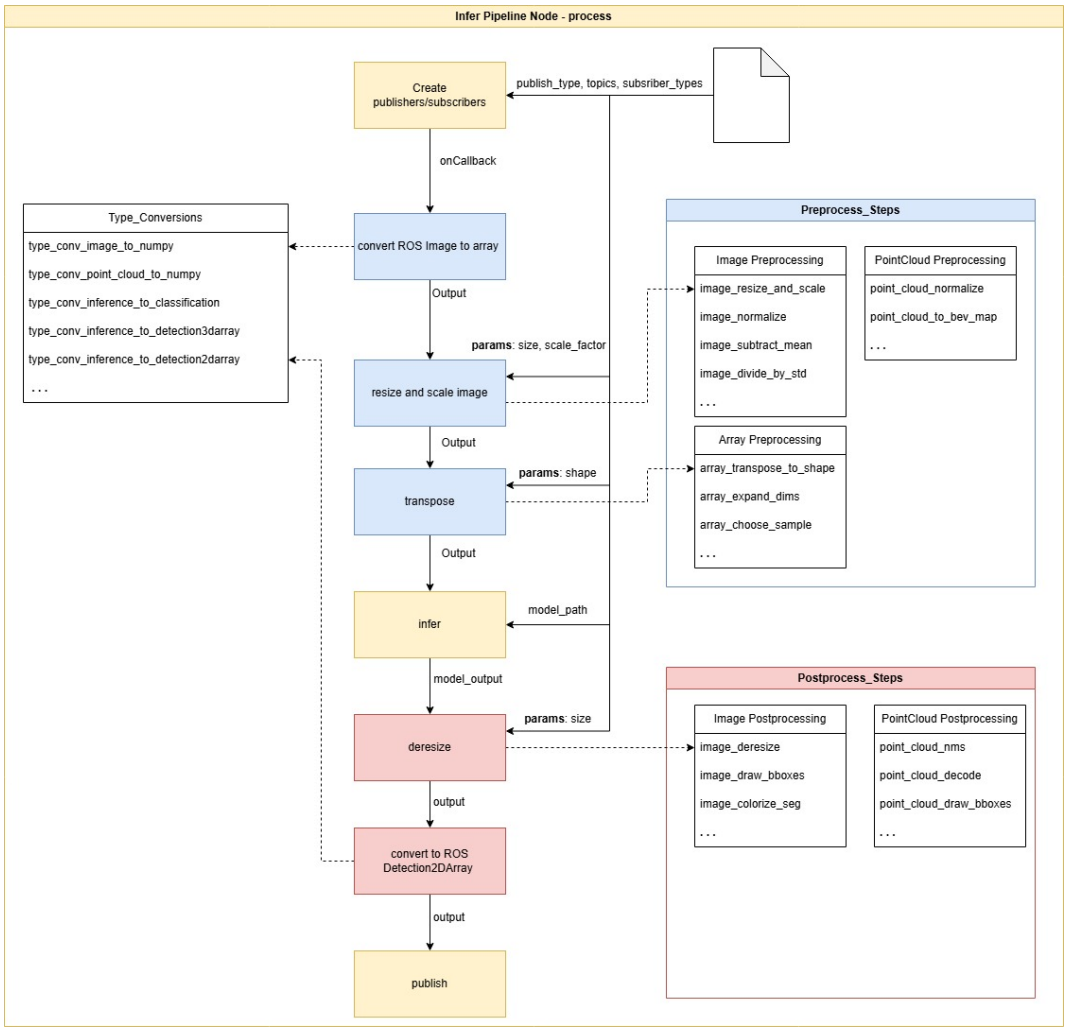

Abbildung 3.7: Beispielablauf der Inferenz im ROS Node

nern wie KServe [48] bieten jedeoch interessante Alternativen. KServe kann als Schnittstelle für die Modellinferenz genutzt werden und erlaubt es beispielsweise Modelle als Teil eines Kubernetes-Cluster in eigenen Containern seperat vom ausführenden Code laufen zu lassen, sodass Plattformen wie Torch und ONNX abstrahiert werden. Dabei werden die Inferenzanfragen als standardisierte Aufrufe an den jeweiligen Container gestaltet, wodurch flexibel zahlreiche Modelle gleichzeitig im Cluster laufen können und ohne Ausfallzeit zwischen diesen gewechselt werden kann. Allerdings stellt sich bei einigen Tests mit der KServe Plattform heraus, dass dieser Ansatz einen großen Mehraufwand mit sich bringt und der genutzte Kubernetes Cluster selbst im kleinsten Format mehrere Gigabyte Speicher beansprucht. Dies würde die Kapazitäten von gängigen Edge-Geräten schnell übersteigen und ist für den benötigten Anwendungsfall nur bedingt nutzbar. Daher wird das Konzept des Protoypen beibehalten, wobei der Ausrollprozess über Agri-Gaia im Rahmen des Projektes nicht erneut getestet wird, da der Berechtigungs- und Einbindungsprozess zu langwierig für den Zeitrahmen ist. Stattdessen wird die Erstellung eines Docker Containers, sowie dessen Nutzung auf einem Nvidia Jetson Orin getestet, um zu validieren, dass der Container ohne Probleme auf dem Edge-Gerät läuft und somit im Nachhinein auch voraussichtlich problemlos in die Agri-Gaia Plattform integriert werden kann.

3.3 Ausführung und Tests

Um die Funktionalität des Nodes zu testen wurden einige Modelle aus verschiedenen Anwendungsszenarien herangezogen, sowie einige Hilfskomponenten erstellt, die das Testen erleichtern. Für die Tests mit Bilddaten steht eine Intel Realsense D435i zur Verfügung, die sowohl RGB- als auch Tiefenbilder liefert und diese über ROS an das konfigurierte Topic senden kann. Um mit LiDAR Punktewolken zu arbeiten, steht allerdings kein physischer Sensor zur Verfügung, der über eine ROS Integeration verfügt. Dazu wurde über die Simulationplattform Gazebo eine Umwelt, sowie ein Roboter erstellt, welcher mit einem LiDAR Sensor versehen ist. Abbildung 3.8 zeigt die Gazebo Umgebung, in der der integrierte Sensor die simulierten Daten an ROS gibt und worüber sich Hindernisse wie Häuser, Fahrzeuge und Ähnliches simulieren lassen. Darüber hinaus wurde ein einfacher ROS Publisher entwickelt, der beliebige Bild- und Punktewolkendaten aus Dateien ausliest und diese an das konfigurierte Topic sendet.
Auf dieser Grundlage können unterschiedliche Modelle integriert und auf die Funktionalität getestet werden. Dabei werden insbesondere die drei verschiedenen Eingabetypen RGB-Bilder, Tiefenbilder und LiDAR-Punktewolken berücksichtigt und dessen Integration in den erstellten ROS-Wrapper getestet. Für die Nutzung von RGB Bilddaten werden Modelle für die Segmentierung, Klassifizierung und die Objekterkennung herangezogen. Die Objekterkennung wird an-

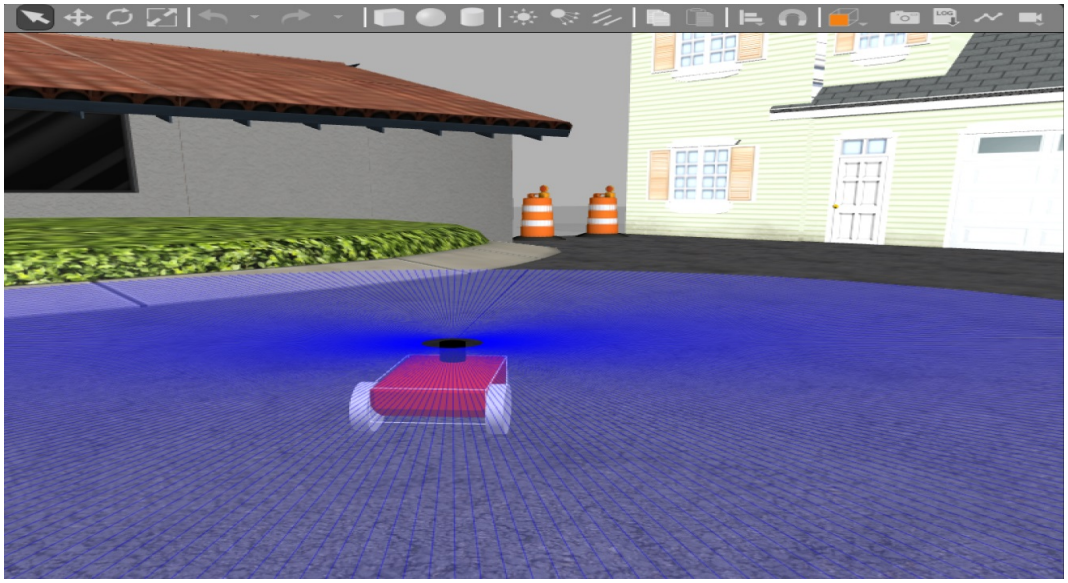

Abbildung 3.8: LiDAR Integration in Gazebo

hand des Faster-RCNN-Models [49] getestet, welches mit dem COCO-Datensatz [50] auf die Erkennung gängier Objekte wie Menschen, Tiere oder Autos, trainiert wurde. Die benötigte Vorprozessierung umfasst dabei das Skalieren des Bildes, sowie das Auffüllen und lässt sich somit ohne große Änderungen als Konfigurationsdatei integrieren. Auf ähnliche Weise wurde das Bisenet-Model [51] integriert, welches mit dem KITTI-Datensatz auf die Segmentierung von Straßenverkehrsobjekten trainiert wurde. Dazu wurde ein bestehender ROS2 KITTI-Datensatz Publisher herangezogen [52], welcher dynamisch sowohl Bild- als auch LiDAR-Punktewolkendaten in das ROS gibt und somit die Frequenz einer Videokamera simuliert. Dadurch lässt sich die Funktionalität des Wrappers sowohl für die Nutzung statischer Bilder als auch dynamischer Videos validieren. Abbildungen 3.9a und 3.9b zeigen die Ergebnisse der Inferenz und dessen Darstellung in RVIZ, wodurch die Funktionalität des Wrappers gewährleistet ist.

Nach diesem Schema wurden anschließend weitere Modelle integriert, wobei Prozessschritte, die noch nicht im Wrapper integriert waren hinzugefügt wurden und ähnliche Schritte an die standardisierten angepasst wurden.

Während sich zahlreiche Beispiele für die Nutzung von Kameradaten in unterschiedlichen Anwendungsfällen finden lassen, stehen nur bedingt Modelle zur Verfügung, die LiDAR Punktewolken nutzen und mit ausreichend Daten trainiert sind. Der Großteil der genutzten Modelle kommen nahezu ausschließlich aus Anwendungsfällen im Straßenverkehr, bei denen die Modelle darauf trainiert werden Objekte im Straßenverkehr zu erkennen, was insbesondere für das autonome Fahren genutzt werden kann. Diese Modelle dienen dennoch als geeignete Grundlage, um die Integration von Lidar Sensoren zu Testen und anhand von Modellen wie Rangenet [53], welches zur Segmentierung von Punkte-

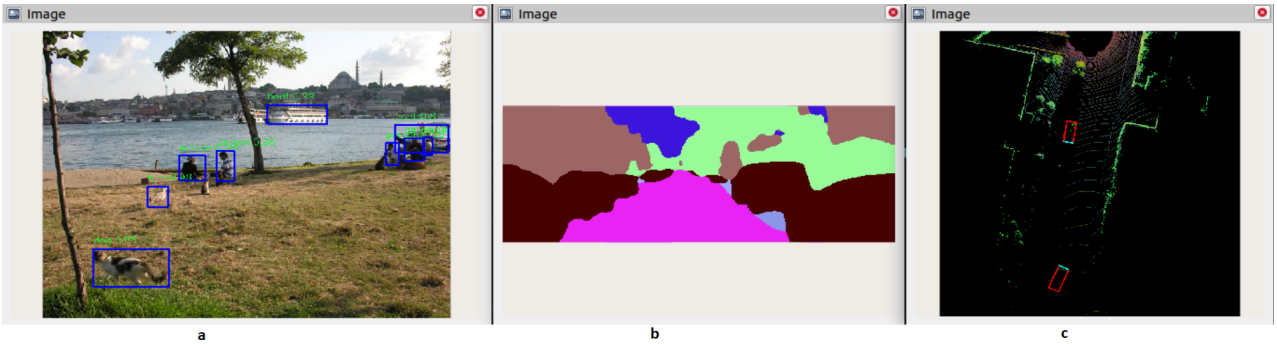

Abbildung 3.9: Inferenztests mit Segmentierung und Objekterkennung

wolken dient oder SFA3D [54], welches Punktewolken in eine Vogelperspektive umwandelt (siehe Abbildung 3.9c), die Funktionalität zu validieren.
Die Handhabung mehrerer Eingabetopics wird anhand des Esanet-Modells validiert [55], welches sowohl RGB- als auch Tiefenbilddaten als Eingabe für das Modell erwartet. ROS2 bietet die Möglichkeit über das *message_filters* Paket mehrere Subscriptions zu synchronisieren, solange diese den gleichen Zeitstempel besitzen. Sind mehrere Subscriptions definiert, werden diese demnach über die *message_filters* synchronisiert und für jedes der topics die jeweilige Vorprozessierung durchlaufen, bevor die Ergebnisse zusammengeführt werden.
Für jedes dieser gesammelten Modelle ist eine Konfigurationsdatei erstellt worden, über die sich einfach zwischen den ausgeführten Modellen wechseln lässt. Dieser Wechsel wird über ROS Parameter gestaltet. Diese lassen sich an die ROS Middleware senden und von allen Nodes auswerten, sodass neben dem Wrapper auch andere Nodes Zugriff auf die aktuelle Konfiguration des Nodes haben und somit auf das aktuell konfigurierte Modell reagieren können.

4 Evaluation

Um die Funktionalität des erstellten Wrappers zu evaluieren werden die in 3.1.3 genannten Kriterien **Flexibilität**, **Erweiterbarkeit** und **Performanz** herangezogen. Die **Flexibilität** wird durch die Nutzung unterschiedlicher Modellarten und Ein-/Ausgabetypen gewährleistet, sowie durch die Unabhängigkeit von der ausführenden Hardware. Wie in Kapitel 3 beschrieben, wurden zahlreiche Tests mit Hilfe von externen, sowie eigenen ROS Publishern durchgeführt, um die Funktionalität zu testen, da insbesondere Sensorik wie ein LiDAR nicht zu Verfügung stehen.
Diese Tests wurden in der Regel auf lokalen Rechnern und virtuellen Maschinen durchgeführt und nicht auf dem Edge-Gerät. Um die Funktionalität auf dem Nvidia Jetson Orin nachzuweisen wurde der Wrapper als Docker Container auf dem Gerät ausgerollt. Nvidia stellt dabei Images zur Verfügung, die Nvidias TensorRT Bibliothek nutzen [56], welche zur Optimierung von Nvidia GPUs dient und insbesondere für das Trainieren und die Inferenz von KI-Modellen geeignet ist. Über die erstelle Dockerfile, sowie angelegte Skripte zum Bauen und Ausführen des Containers kann dieser auf dem Jetson Orin Gerät ausgeführt werden und ohne weitere Konfiguration mit der Inferenz von eingehenden Sensordaten beginnen.
Für die Nutzung von Kameradaten steht eine Intel Realsense D435i Kamera zur Verfügung, die an das Edge-Gerät angeschlossen werden kann. Über RVIZ lassen sich die gesendeten Kamerabilder inklusive der Tiefeninformation anzeigen, sowie die ausgegebenen Inferenzbilder, die der ROS-Wrapper ausgibt. Zum Testen wurde der Wrapper über die Konfigurationsdatei mit dem RCNN-Netz für die Objekterkennung, sowie dem EsaNet-Netz für die Segmentierung von RGB- und Tiefenbilddaten ausgeführt. Für die Objekterkennung mit dem RCNN-Netz wurde die RealSense Kamera auf eine angrenzende Straße gerichtet und die Ergebnisse im RVIZ betrachtet, welche in Abbildung 4.1 dargestellt werden.

Das Ergebnis zeigt die Funktionalität des Wrappers auch in Echtzeit mit angeschlossener Kamera. Neben den RGB-Bildern lässt sich die Funktionalität der Synchronisation mehrerer Topics anhand des EsaNet Modells zeigen, welches sowohl die Tiefenkameradaten als auch die RGB-Daten entgegen nimmt. Abbildung 4.2 zeigt die Ausgabe der Segmentierung, wobei sowohl das RGB- als auch das Tiefenbildtopic der Realsense Kamera abonniert werden. Diese beiden Anwendungsfälle zeigen die Flexibilität der Anwendung hinsichtlich der Möglichkeit Modelle austauschen zu können, sowie mehrere Eingabetopics auf

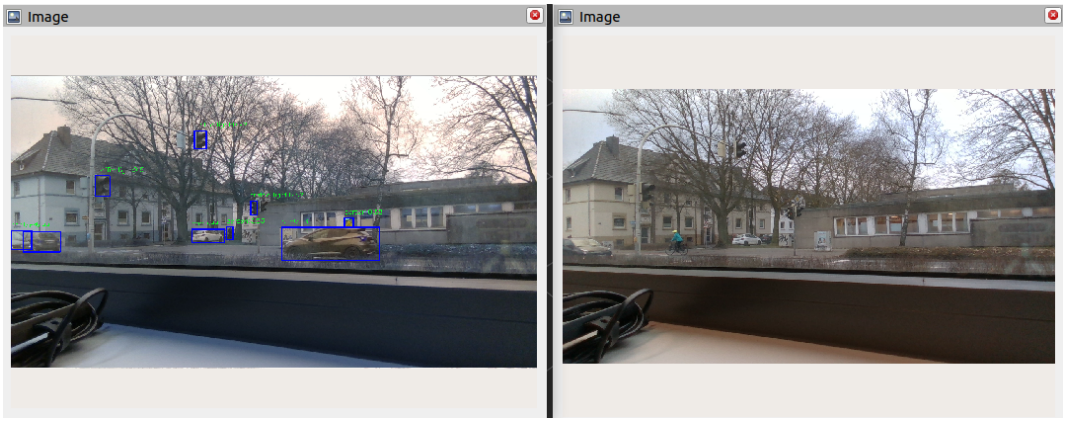

Abbildung 4.1: Objekterkennung mit dem ROS Wrapper durch RealSense D435i Aufnahmen

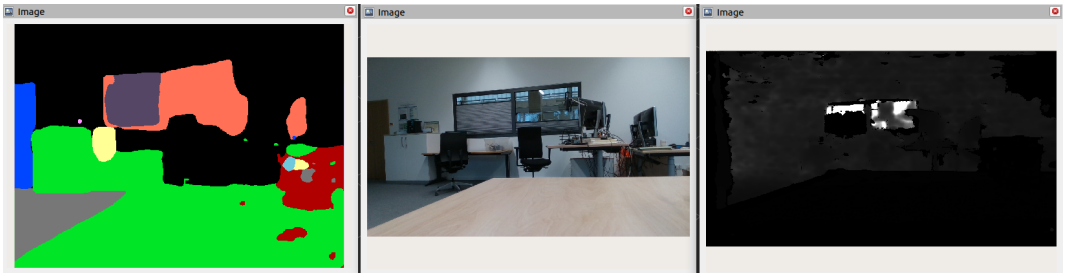

Abbildung 4.2: Segmentierung von RGB- und Tiefenbildern mit dem ROS Wrapper durch RealSense D435i Aufnahmen

einmal zu erlauben.
Die **Erweiterbarkeit** wird durch die Einbindung der Vor- und Nachprozessierungspipeline gewährleistet. Neue Schritte können einfach integriert werden, was in der Code-Dokumentation genauer beschrieben wird. Grundsätzlich müssen neue Schritte als Methode in den vorgesehenen Prozesschrittdateien hinzugefügt werden und der Konvention folgen Daten aus dem vorherigen Schritt per Parameter entgegenzunehmen und das Ergebnis auszugeben. Die Definition der Parameter geschieht über die YAML-Konfigurationsdateien und kann so beliebig an den Anwendungsfall angepasst werden. Die Performanz wird anhand von Laufzeittests ausgewertet, die mit den unterschiedlichen Modellen durchgeführt werden, wobei die durchsschnittliche Laufzeit der Vor- und Nachprozessierung, sowie der Inferenz untersucht wird. Dazu wurden Zeitstempel in das Logging hinzugefügt, die die Dauer der einzelnen Schritte, sowie die Gesamtdauer der Prozessierung dokumentieren. Die Auswertung der Schritte wird in Abbildung 4.3 dargestellt.
Die hier aufgezeichneten Daten wurden unter der Ausführung auf einer lokalen VM ohne Hardwarebeschleunigung durchgeführt, um den maximalen Einfluss der Schritte auf die Rechenzeit sehen zu können. Dabei fällt auf, dass in der Regel die Inferenz die höchste Prozesszeit beansprucht, wobei diese stark von

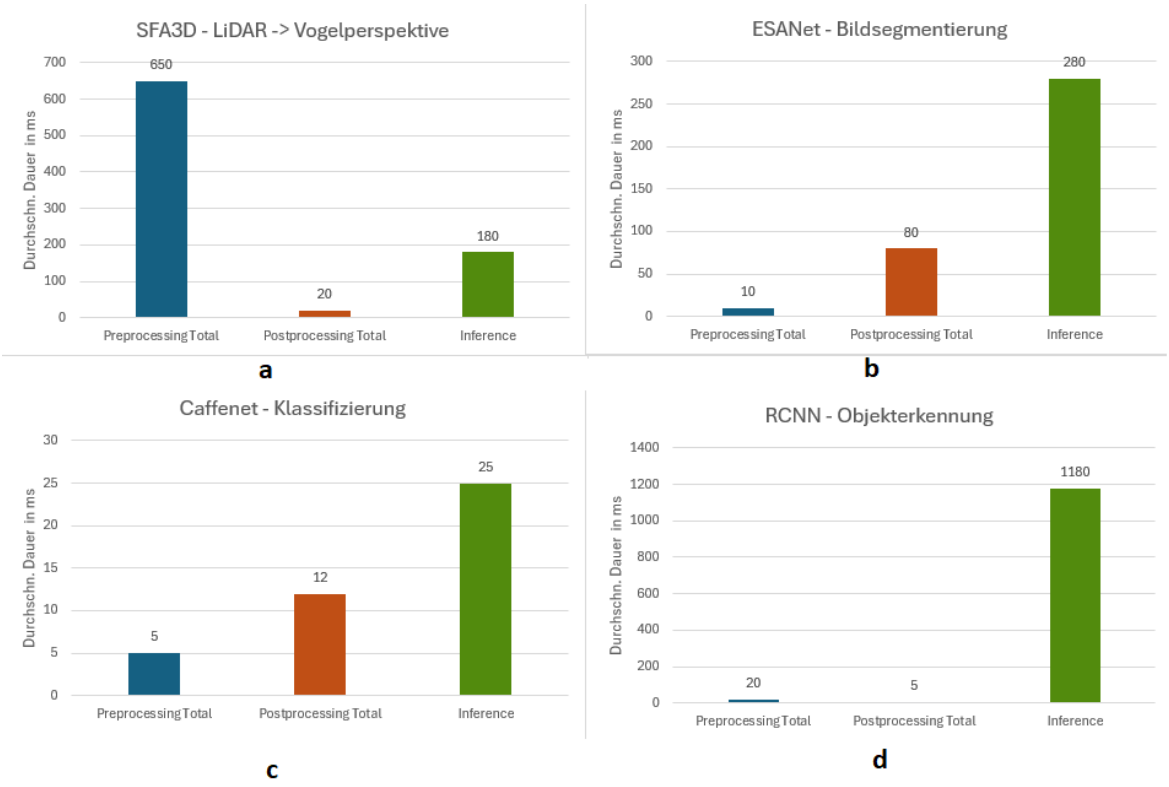

Abbildung 4.3: Ausführungsdauer der einzelnen Modelltypen

der Modellart abhängt und von wenigen Millisekunden bis hin zu einer Sekunde pro Ausführung dauern kann. Diese hängt jedoch ausschließlich vom Modell und der ONNXRuntime ab und lässt sich nur durch optimierte Modelle und Hardware verbessern. Die Vor- und Nachrpozessierung fallen in der Regel deutlich geringer aus und können in den Fällen der Bildprozessierung von der Dauer her vernachlässigt werden. Lediglich bei der Nutzung der LiDAR Punktewolken ist eine deutliche Verzögerung in der Vorprozessierung zu erkennen (siehe Abbildung 4.3a).
Bei genauerer Betrachtung der Dauer jedes einzelnen Schrittes fällt auf, dass diese nahezu ausschließlich auf die Typumwandlung aus der ROS PointCloud2 in ein Array Format stammt. Die Verzögerung ruht dabei auf der Tatsache, dass der genutzte Punktewolken Datensatz aus dem KITTI-Datensatz stammt und eine große Datenmenge mit sich bringt. Die Umwandlung selbst findet durch die in ROS2 integrierte Funktionalität statt, sodass davon auszugehen ist, dass diese auf den Anwendungsfall bereits optimiert ist. Da das Verhalten aufgrund von fehlender Hardware nicht mit einem realen LiDAR Sensor getestet werden kann muss dies in der zukünftigen Nutzung genauer betrachtet werden, um festzustellen, ob ein Optimierungsbedarf herrscht. Grundsätzlich ist davon auszugehen, dass diese Umwandlung mit optimierter Hardware schneller durchgeführt werden kann, sodass diese bei einer realen Anwendung keinen starken Einfluss auf die Ausführungsdauer hat.

Alles in allem ist die Ausführungsdauer der Vor- und Nachprozessierung dennoch zufriedenstellend und wirkt sich nicht merklich auf die Gesamtdauer des Wrappers aus. Das Logging ermöglicht die Dauer jedes einzelnen Schrittes zu überprüfen, sodass bei Bedarf der jeweilige Prozessschritt optimiert werden kann. Der erstellte Wrapper erfüllt somit die gesetzten Kriterien und bietet eine flexible Möglichkeit die Inferenz von KI-Modellen in ROS zu integrieren.

5 Fazit und Ausblick

Im Rahmen dieser Arbeit wurde die Entwicklung eines ROS2-Wrappers zur Ausführung von KI-Modellinferenz durchgeführt, welcher flexibel verschiedene Modelltypen, sowie unterschiedliche Eingabetypen wie RGB-Bilder, Tiefenbilder und Punktewolken nutzen kann. Hierbei wurde ein bestehender ROS Wrapper Prototyp, der ausschließlich auf die Objektdetektion von Bildern verschiedener Pflanzenarten ausgelegt ist, erweitert und an die neuen Anforderungen angepasst.
Die hierfür erstellte Anwendung ist möglichst flexibel gestaltet und erlaubt das Konfigurieren des Modells, sowie der Vor- und Nachprozessierungsschritte über eine standardisierte YAML Datei, wodurch ohne Änderungen am Code die Modelle, sowie die damit verbundene Datenaufbereitung ausgetauscht werden können. Das Konzept umfasst eine Standardisierung der Prozessschritte, sodass diese als eine Pipeline aneinander gekettet werden können, wobei jeder Schritt modulare Operationen durchführt und dabei die Ergebnisse des vorangehenden Schrittes entegennimmt und die Ergebnisse an den nächsten Schritt übergibt. Darüberhinaus umfasst der Wrapper zahlreiche Typumwandlungen von den in ROS genutzen Nachrichtentypen zu Datentypen, die bei der Modellinferenz genutzt werden, sowie die Umwandlung in gängige Ausgabetypen zur Ausgabe von Klassifizierungen oder Boundingboxen. Die KI-Modelle werden im ONNX Format in den Wrapper integriert, sodass diese einfach ausgetauscht werden können, ohne den Code des Modells in den Wrapper einbauen zu müssen.
Zur Validierung wurden zahlreiche vortrainierte ONNX Modelle aus Modellsammlungen und anderen Projekten herangezogen und die Nutzung des Wrappers mit den unterschiedlichen Eingabetypen getestet. Dabei wurden Anwendungsfälle wie die Bild- und Punktewolkensegmentierung, Objekterkennung in Bildern und Punktewolken, sowie die Klassifzierung von RGB- und Tiefenkamerabildern getestet und über angepasste Konfigurationsdateien in den Wrapper integriert.
Das Ausrollen des Modells wurde mit Hilfe eines Docker Containers auf dem zur Verfügung gestellt Nvidia Jetson Orin durchgeführt, wodurch der Wrapper Hardware unabhängig auf beliebigen Edge-Geräten ausgeführt werden kann. Die Bildsegmentierungs- und Klassifizierungsmodelle wurden auf dem Orin in Verbindung mit einer angeschlossenen RealSense D435i Kamera getestet und die Funktionalität des Wrappers, Kameradaten in Echtzeit entgegenzunehmen, validiert. Da kein LiDAR Sensor zur Verfügung stand, um die Handhabung von Punktewolken mit realen Sensoren zu testen wurde sowohl eine Gazebo

Simulationsumgebung aufgesetzt als auch ein eigener Publisher entwickelt, der Demodaten an den Wrapper sendet.
In Zukunft gilt es dennoch diese Funktionalität insbesondere im Hinblick auf die Performanz mit Hilfe eines realen LiDAR Sensors zu validieren, um die Handhabung von den damit verbundenen großen Datenmengen zu testen. Darüberhinaus soll in einem weiteren Schritt die Integration des Wrappers in die Agri-Gaia Plattform stattfinden, damit über diese flexibel der Wrapper und das dazugehörige Modell auf ein Edge-Gerät ausgerollt werden können. Dieses Konzept besteht bereits in Verbindung mit dem zur Verfügung stehenden Prototypen, ist aber aufgrund von auslaufenden Berechtigungen und Einbindung der Edge-Geräte in die Plattform aktuell nicht möglich.
Alles in Allem erfüllt der erstellte ROS Wrapper die Anforderungen beliebige Modelle und Sensordaten entgegenzunehmen und die Inferenzdaten in das ROS auszugeben zur vollsten Zufriedenheit und bietet in Zukunft eine leicht erweiterbare und flexible Lösung KI-Inferenz auf Edge-Geräten zur Nutzung mit Agrarrobotern durchzuführen.

Literaturverzeichnis

[1] FAO: *The State of Food and Agriculture*. https://openknowledge.fao.org/items/2f47c5f5-0950-4b4f-8d14-ba796e618a49. Version: 2017. – Zugriff: 14.12.2024

[2] Eruopäisches Parlament: *Präzisionslandwirtschaftund die Zukunft derLandwirtschaft in Europa*. https://www.europarl.europa.eu/RegData/etudes/STUD/2016/581892/EPRS_STU(2016)581892_DE.pdf. Version: 2024. – Zugriff: 24.10.2024

[3] Duckett, Tom ; Pearson, Simon ; Blackmore, Simon ; Grieve, Bruce: *Agricultural Robotics: The Future of Robotic Agriculture*

[4] ROS: *What is ROS?* https://www.ros.org/. Version: 2024. – Zugriff: 24.10.2024

[5] Xie, Dongbo ; Chen, Liang ; Liu, Lichao ; Chen, Liqing ; Wang, Hai: Actuators and Sensors for Application in Agricultural Robots: A Review. In: *Machines* 10 (2022), Nr. 10. https://www.mdpi.com/2075-1702/10/10/913. – ISSN 2075–1702

[6] Gu, Yili ; Li, Zhiqiang ; Zhang, Zhen ; Li, Jun ; Chen, Liqing: Path Tracking Control of Field Information-Collecting Robot Based on Improved Convolutional Neural Network Algorithm. In: *Sensors* 20 (2020), Nr. 3. https://www.mdpi.com/1424-8220/20/3/797. – ISSN 1424–8220

[7] Cubero, Sergio ; Marco-Noales, Ester ; Aleixos, Nuria ; Barbé, Silvia ; Blasco, Jose: RobHortic: A Field Robot to Detect Pests and Diseases in Horticultural Crops by Proximal Sensing. In: *Agriculture* 10 (2020), Nr. 7. http://dx.doi.org/10.3390/agriculture10070276. – DOI 10.3390/agriculture10070276. – ISSN 2077–0472

[8] Siddiquee, Kazy Noor-E-Alam ; Islam, Md ; Rezaul, Karim ; Grout, Vic: Detection, Quantification and Classification of Ripened Tomatoes using a Comparative Analysis of Image Processing and Machine Learning Methods: A case study. In: *IET Image Processing* 14 (2020), 08. http://dx.doi.org/10.1049/iet-ipr.2019.0738. – DOI 10.1049/iet–ipr.2019.0738

[9] Quan, Longzhe ; Li, Hengda ; Li, Hailong ; Jiang, Wei ; Lou, Zhaoxia ; Chen, Liqing: Two-Stream Dense Feature Fusion Network Based on

RGB-D Data for the Real-Time Prediction of Weed Aboveground Fresh Weight in a Field Environment. In: *Remote Sensing* 13 (2021), Nr. 12. http://dx.doi.org/10.3390/rs13122288. – DOI 10.3390/rs13122288. – ISSN 2072–4292

[10] Mirbod, Omeed ; Choi, Daeun ; Thomas, Roderick ; He, Long: Overcurrent-driven LEDs for consistent image colour and brightness in agricultural machine vision applications. In: *Computers and Electronics in Agriculture* 187 (2021), 106266. http://dx.doi.org/https://doi.org/10.1016/j.compag.2021.106266. – DOI https://doi.org/10.1016/j.compag.2021.106266. – ISSN 0168–1699

[11] Underwood, James P. ; Jagbrant, Gustav ; Nieto, Juan I. ; Sukkarieh, Salah: Lidar-Based Tree Recognition and Platform Localization in Orchards. In: *Journal of Field Robotics* 32 (2015), Nr. 8, 1056-1074. http://dx.doi.org/https://doi.org/10.1002/rob.21607. – DOI https://doi.org/10.1002/rob.21607

[12] Weiss, Ulrich ; Biber, Peter: Plant detection and mapping for agricultural robots using a 3D LIDAR sensor. In: *Robotics and Autonomous Systems* 59 (2011), Nr. 5, 265-273. http://dx.doi.org/https://doi.org/10.1016/j.robot.2011.02.011. – DOI https://doi.org/10.1016/j.robot.2011.02.011. – ISSN 0921–8890. – Special Issue ECMR 2009

[13] Jian Yang, Rajeev K. Amit Sharma S. Amit Sharma: IoT-Based Framework for Smart Agriculture. In: *International Journal of Agricultural and Environmental Information Systems (IJAEIS)* 12 (2021), S. 1–14

[14] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey: ImageNet Classification with Deep Convolutional Neural Networks. In: *Neural Information Processing Systems* 25 (2012), 01. http://dx.doi.org/10.1145/3065386. – DOI 10.1145/3065386

[15] Simonyan, K ; Zisserman, A: Very deep convolutional networks for large-scale image recognition, Computational and Biological Learning Society, 2015, S. 1–14

[16] Szegedy, Christian ; Liu, Wei ; Jia, Yangqing ; Sermanet, Pierre ; Reed, Scott ; Anguelov, Dragomir ; Erhan, Dumitru ; Vanhoucke, Vincent ; Rabinovich, Andrew: Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, S. 1–9

[17] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, S. 770–778

[18] Zhang, Xihai ; Qiao, Yue ; Meng, Fanfeng ; Fan, Chengguo ; Zhang, Mingming: Identification of Maize Leaf Diseases Using Improved Deep Convolutional Neural Networks. In: *IEEE Access* 6 (2018), S. 30370–30377. http://dx.doi.org/10.1109/ACCESS.2018.2844405. – DOI 10.1109/ACCESS.2018.2844405

[19] Too, Edna C. ; Yujian, Li ; Njuki, Sam ; Yingchun, Liu: A comparative study of fine-tuning deep learning models for plant disease identification. In: *Computers and Electronics in Agriculture* 161 (2019), 272-279. http://dx.doi.org/https://doi.org/10.1016/j.compag.2018.03.032. – DOI https://doi.org/10.1016/j.compag.2018.03.032. – ISSN 0168–1699. – BigData and DSS in Agriculture

[20] Rico, Francisco M.: *A Concise Introduction to Robot Programming with ROS2*. https://api.taylorfrancis.com/content/books/mono/download?identifierName=doi&identifierValue=10.1201/9781003289623&type=googlepdf. Version: 2023

[21] Yaacoub, Elias ; Alouini, Mohamed-Slim: A Key 6G Challenge and Opportunity—Connecting the Base of the Pyramid: A Survey on Rural Connectivity. In: *Proceedings of the IEEE* 108 (2020), Nr. 4, S. 533–582. http://dx.doi.org/10.1109/JPROC.2020.2976703. – DOI 10.1109/JPROC.2020.2976703

[22] Singh, Raghubir ; Gill, Sukhpal S.: Edge AI: A survey. In: *Internet of Things and Cyber-Physical Systems* 3 (2023), 71-92. http://dx.doi.org/https://doi.org/10.1016/j.iotcps.2023.02.004. – DOI https://doi.org/10.1016/j.iotcps.2023.02.004. – ISSN 2667–3452

[23] Liu, Deyin ; Chen, Xu ; Zhou, Zhi ; Ling, Qing: HierTrain: Fast Hierarchical Edge AI Learning With Hybrid Parallelism in Mobile-Edge-Cloud Computing. In: *IEEE Open Journal of the Communications Society* 1 (2020), S. 634–645. http://dx.doi.org/10.1109/OJCOMS.2020.2994737. – DOI 10.1109/OJCOMS.2020.2994737

[24] In: Jangaraj, Avanija ; Rajyalakshmi, Ch ; Madhavi, Reddy ; Narendra, B ; Kumar Rao, Dr.B.Narendra: *Enabling Smart Farming Through Edge Artificial Intelligence (AI)*. 2024. – ISBN 979–8369320693, S. 69–82

[25] Zhang, Xihai ; Cao, Zhanyuan ; Dong, Wenbin: Overview of Edge Computing in the Agricultural Internet of Things: Key Technologies, Applications, Challenges. In: *IEEE Access* 8 (2020), S. 141748–141761. http://dx.doi.org/10.1109/ACCESS.2020.3013005. – DOI 10.1109/ACCESS.2020.3013005

[26] Majeed, Yaqoob ; Ojo, Mike O. ; Zahid, Azlan: Standalone edge AI-based solution for Tomato diseases detection. In: *Smart Agricultural Technology* 9 (2024), 100547. http://dx.doi.org/https://doi.org/10.1016/j.atech.2024.100547. – DOI https://doi.org/10.1016/j.atech.2024.100547. – ISSN 2772–3755

[27] Bhavan, S ; ., Mohana: YOLOv5 Crop Detection Deep Learning Model using Artificial Intelligence (AI) and Edge Computing, 2022, S. 21–24

[28] Tiozzo Fasiolo, Diego ; Scalera, Lorenzo ; Maset, Eleonora ; Gasparetto, Alessandro: Towards autonomous mapping in agriculture: A review of supportive technologies for ground robotics. In: *Robotics and Autonomous Systems* 169 (2023), 104514. http://dx.doi.org/https://doi.org/10.1016/j.robot.2023.104514. – DOI https://doi.org/10.1016/j.robot.2023.104514. – ISSN 0921–8890

[29] Hymel, Shawn ; Banbury, Colby ; Situnayake, Daniel ; Elium, Alex ; Ward, Carl ; Kelcey, Mat ; Baaijens, Mathijs ; Majchrzycki, Mateusz ; Plunkett, Jenny ; Tischler, David ; Grande, Alessandro ; Moreau, Louis ; Maslov, Dmitry ; Beavis, Artie ; Jongboom, Jan ; Reddi, Vijay J.: *Edge Impulse: An MLOps Platform for Tiny Machine Learning*. https://arxiv.org/abs/2212.03332. Version: 2023

[30] David, Robert ; Duke, Jared ; Jain, Advait ; Reddi, Vijay J. ; Jeffries, Nat ; Li, Jian ; Kreeger, Nick ; Nappier, Ian ; Natraj, Meghna ; Regev, Shlomi ; Rhodes, Rocky ; Wang, Tiezhen ; Warden, Pete: *TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems*. https://arxiv.org/abs/2010.08678. Version: 2021

[31] Edge Impulse : *ROS2 + Edge Impulse*. https://docs.edgeimpulse.com/experts/software-integration-demos/ros2-part1-pubsub-node. Version: 2024. – Zugriff: 02.11.2024

[32] Gookyi, Dennis Agyemanh N. ; Wulnye, Fortunatus A. ; Wilson, Michael ; Danquah, Paul ; Danso, Samuel A. ; Gariba, Awudu A.: Enabling Intelligence on the Edge: Leveraging Edge Impulse to Deploy Multiple Deep Learning Models on Edge Devices for Tomato Leaf Disease Detection. In: *AgriEngineering* 6 (2024), Nr. 4, 3563-3585. http://dx.doi.org/10.3390/agriengineering6040203. – DOI 10.3390/agriengineering6040203. – ISSN 2624–7402

[33] Barusso, Carolina ; Varella, Walter ; Piantoni, Jane ; Dantas, Rogerio: CLASSIFICATION OF SOYBEAN LEAVES USING THE EDGE IMPULSE PLATFORM, 2023

[34] Passalis, N. ; Pedrazzi, S. ; Babuska, R. ; Burgard, W. ; Dias, D. ; Ferro, F. ; Gabbouj, M. ; Green, O. ; Iosifidis, A. ; Kayacan, E.

; Kober, J. ; Michel, O. ; Nikolaidis, N. ; Nousi, P. ; Pieters, R. ; Tzelepi, M. ; Valada, A. ; Tefas, A.: *OpenDR: An Open Toolkit for Enabling High Performance, Low Footprint Deep Learning for Robotics*. https://arxiv.org/abs/2203.00403. Version: 2022

[35] Sirohi, Kshitij ; Mohan, Rohit ; Büscher, Daniel ; Burgard, Wolfram ; Valada, Abhinav: EfficientLPS: Efficient LiDAR Panoptic Segmentation. In: *IEEE Transactions on Robotics* 38 (2022), Nr. 3, S. 1894–1914. http://dx.doi.org/10.1109/TRO.2021.3122069. – DOI 10.1109/TRO.2021.3122069

[36] Kirillov, Alexander ; He, Kaiming ; Girshick, Ross ; Rother, Carsten ; Dollár, Piotr: Panoptic Segmentation. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, S. 9396–9405

[37] Nuzzi, Cristina ; Pasinetti, Simone ; Pagani, Roberto ; Coffetti, Gabriele ; Sansoni, Giovanna: HANDS: an RGB-D dataset of static hand-gestures for human-robot interaction. In: *Data in Brief* 35 (2021), 106791. http://dx.doi.org/https://doi.org/10.1016/j.dib.2021.106791. – DOI https://doi.org/10.1016/j.dib.2021.106791. – ISSN 2352–3409

[38] OpenDR: *rgbd hand gesture learner module*. https://github.com/opendr-eu/opendr/blob/master/docs/reference/rgbd-hand-gesture-learner.md. Version: 2024. – Zugriff: 06.11.2024

[39] AgriGaia: *AgriGaia - Das Ökosystem*. https://www.agri-gaia.de/agri-gaia/das-oekosystem/. Version: 2024. – Zugriff: 06.11.2024

[40] H. Tapken, H. Graf A. Schliebitz L. Hesse M. Fruhner J. T. T. Wamhof W. T. Wamhof: Accelerating the AI lifecycle with the AgriGaia-Platform. In: *81. Internationale Tagung Landtechnik - VDI Berichte* 2444 (2024), S. 63–68

[41] Matthias Igelbrink: *Konzept des Proof-Of-Concept ROS2 Wrappers im Agro-Technicum*. 2024

[42] ONNX: *Open Neural Network Exchange*. https://onnx.ai/. Version: 2024. – Zugriff: 16.12.2024

[43] ROS: *ROS2 vision_msgs Msg/Srv Documentation*. https://docs.ros.org/en/lunar/api/vision_msgs/html/index-msg.html. Version: 2024. – Zugriff: 14.12.2024

[44] ROS: *ROS2 image_pipeline overview*. https://docs.ros.org/en/rolling/p/image_pipeline/. Version: 2024. – Zugriff: 14.12.2024

[45] ROS: *ROS2 image_proc overview*. https://docs.ros.org/en/rolling/p/image_proc/doc/index.html. Version: 2024. – Zugriff: 14.12.2024

[46] NVIDIA: *Isaac ROS Image Pipeline*. https://nvidia-isaac-ros.github.io/repositories_and_packages/isaac_ros_image_pipeline/index.html. Version: 2024. – Zugriff: 15.12.2024

[47] ONNX: *ONNX Model Zoo*. https://github.com/onnx/models. Version: 2024. – Zugriff: 15.12.2024

[48] KSERVE: *KServe - Home*. https://kserve.github.io/website/latest/. Version: 2024. – Zugriff: 16.12.2024

[49] ZOO, ONNX M.: *Faster RCNN*. https://github.com/onnx/models/tree/main/validated/vision/object_detection_segmentation/faster-rcnn. Version: 2024. – Zugriff: 15.12.2024

[50] COCODATASET: *Common objects in context*. https://cocodataset.org/#home. Version: 2024. – Zugriff: 15.12.2024

[51] ROSI, Gabriele ; CUTTANO, Claudia ; CAVAGNERO, Niccolò ; AVERTA, Giuseppe ; CERMELLI, Fabio: *The revenge of BiSeNet: Efficient Multi-Task Image Segmentation*. https://arxiv.org/abs/2404.09570. Version: 2024

[52] UMTCLSKN: *ROS2 Kitti Publishers*. https://github.com/umtclskn/ros2_kitti_publishers. Version: 2024. – Zugriff: 16.12.2024

[53] MILIOTO, Andres ; VIZZO, Ignacio ; BEHLEY, Jens ; STACHNISS, Cyrill: RangeNet ++: Fast and Accurate LiDAR Semantic Segmentation. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, S. 4213–4220

[54] DUNG, Nguyen M.: *Super-Fast-Accurate-3D-Object-Detection-PyTorch*. https://github.com/maudzung/Super-Fast-Accurate-3D-Object-Detection, 2020

[55] SEICHTER, Daniel ; KÖHLER, Mona ; LEWANDOWSKI, Benjamin ; WENGEFELD, Tim ; GROSS, Horst-Michael: Efficient RGB-D Semantic Segmentation for Indoor Scene Analysis. In: *arXiv preprint arXiv:2011.06961* (2020)

[56] NVIDIA: *NVIDIA L4T TensorRT*. https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-tensorrt. Version: 2024. – Zugriff: 16.12.2024

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche einzeln kenntlich gemacht. Es wurden keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

.....

Osnabrück, 22. Dezember 2024, Simon Kösters