Universität Bremen

Vergleich von neuronalen Netzen zur Lidar basierten Pylonerkennung in der Formula Student

Comparison of neural networks for lidar-based pylon detection in Formula Student

Bachelorarbeit

im Rahmen des Studiengangs
Informatik
der Universität Bremen

vorgelegt von
Torge Wessel
Matrikelnummer: 6111046

ausgegeben und betreut von
Prof. Dr.-Ing. Udo Frese
Prof. Dr.-Ing. Maren Petersen

Bremen, den 27. Juli 2026

Universität Bremen

Nachname **Wessel** Matrikelnr. **6111046**

Vorname **Torge**

Hinweise zu den offiziellen Erklärungen

1. Die folgende Seite mit den offiziellen Erklärungen

   A) Eigenständigkeitserklärung

   B) Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

   C) Einverständniserklärung über die Bereitstellung und Nutzung der Bachelorarbeit / Masterarbeit in elektronischer Form zur Überprüfung durch eine Plagiatssoftware

   ist entweder direkt in jedes Exemplar der Bachelor- oder Masterarbeit fest mit einzubinden oder unverändert im Wortlaut in jedes Exemplar der Bachelor- oder Masterarbeit zu übernehmen.

   Bitte achten Sie darauf, jede Erklärung in allen drei Exemplaren der Arbeit zu unterschreiben.

2. In der digitalen Fassung kann auf die Unterschrift verzichtet werden. Die Angaben und Entscheidungen müssen jedoch enthalten sein.

Zu B)

Die Einwilligung kann jederzeit durch Erklärung gegenüber der Universität Bremen, mit Wirkung für die Zukunft, widerrufen werden.

Zu C)

Das Einverständnis der dauerhaften Speicherung des Textes ist freiwillig.

Die Einwilligung kann jederzeit durch Erklärung gegenüber der Universität Bremen, mit Wirkung für die Zukunft, widerrufen werden.

Weitere Informationen zur Überprüfung von schriftlichen Arbeiten durch die Plagiatsoftware sind im Nutzungs- und Datenschutzkonzept enthalten. Diese finden Sie auf der Internetseite der Universität Bremen.

Universität Bremen

Nachname **Wessel** Matrikelnr. **6111046**

Vorname **Torge**

A) Eigenständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Alle Teile meiner Arbeit, die wortwörtlich oder dem Sinn nach anderen Werken entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht. Gleiches gilt auch für Zeichnungen, Skizzen, bildliche Darstellungen sowie für Quellen aus dem Internet, dazu zählen auch KI-basierte Anwendungen oder Werkzeuge. Die Arbeit wurde in gleicher oder ähnlicher Form noch nicht als Prüfungsleistung eingereicht. Die elektronische Fassung der Arbeit stimmt mit der gedruckten Version überein. Mir ist bewusst, dass wahrheitswidrige Angaben als Täuschung behandelt werden.

☒ Ich habe KI-basierte Anwendungen und/oder Werkzeuge genutzt und diese im Anhang "Nutzung KI basierte Anwendungen" dokumentiert.

B) Erklärung zur Veröffentlichung von Bachelor- oder Masterarbeiten

Die Abschlussarbeit wird zwei Jahre nach Studienabschluss dem Archiv der Universität Bremen zur dauerhaften Archivierung angeboten. Archiviert werden:

1) Masterarbeiten mit lokalem oder regionalem Bezug sowie pro Studienfach und Studienjahr 10 % aller Abschlussarbeiten
2) Bachelorarbeiten des jeweils ersten und letzten Bachelorabschlusses pro Studienfach und Jahr.

☒ Ich bin damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin damit einverstanden, dass meine Abschlussarbeit nach 30 Jahren (gem. §7 Abs. 2 BremArchivG) im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

☐ Ich bin nicht damit einverstanden, dass meine Abschlussarbeit im Universitätsarchiv für wissenschaftliche Zwecke von Dritten eingesehen werden darf.

C) Einverständniserklärung zur Überprüfung der elektronischen Fassung der Bachelorarbeit / Masterarbeit durch Plagiatssoftware

Eingereichte Arbeiten können nach § 18 des Allgemeinen Teil der Bachelor- bzw. der Master-prüfungsordnungen der Universität Bremen mit qualifizierter Software auf Plagiatsvorwürfe untersucht werden.
Zum Zweck der Überprüfung auf Plagiate erfolgt das Hochladen auf den Server der von der Universität Bremen aktuell genutzten Plagiatssoftware.

☑ Ich bin damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum oben genannten Zweck dauerhaft auf dem externen Server der aktuell von der Universität Bremen genutzten Plagiatssoftware, in einer institutionseigenen Bibliothek (Zugriff nur durch die Universität Bremen), gespeichert wird.

☐ Ich bin nicht damit einverstanden, dass die von mir vorgelegte und verfasste Arbeit zum o.g. Zweck dauerhaft auf dem externen Server der aktuell von der Universität Bremen genutzten Plagiatssoftware, in einer institutionseigenen Bibliothek (Zugriff nur durch die Universität Bremen), gespeichert wird.

Mit meiner Unterschrift versichere ich, dass ich die obenstehenden Erklärungen gelesen und verstanden habe und bestätige die Richtigkeit der gemachten Angaben.

27.07.2026 Torge Wessel

Kurzfassung Die Formula Student ist ein Wettbewerb, in dem studentische Teams weltweit ihre selbst entwickelten Rennwagen gegeneinander antreten lassen. Bremergy, das Team der Universität Bremen und der Hochschule Bremen, möchte in der fahrerlosen Disziplin antreten und benötigt dafür eine zuverlässige Erkennung der Pylone, die die Fahrbahn markieren. Diese Arbeit vergleicht mehrere neuronale Netzarchitekturen daraufhin, wie zuverlässig sie die Pylone mit Hilfe eines LiDAR erkennen und lokalisieren. Die Forschungsfrage dieser Bachelorarbeit lautet: *„Welche Kombination aus Netzarchitektur und LiDAR-Eingabedarstellung erzielt für die Formula-Student-Pylonerkennung die zuverlässigste Detektionsleistung bei echtzeitfähiger Inferenzlatenz auf dem J4012?“*. Der Vergleich zeigt, dass das BEV-Pillar-Modell mit einem WF1-Score von 0,888 und einer TensorRT-Latenz von 8,3 ms die beste Erkennungsleistung bei niedrigster Inferenzzeit erreicht und damit für den Produktiveinsatz auf dem J4012 empfohlen wird.

Abstract Formula Student is a competition in which student teams from around the world compete against each other with their self-developed race cars. Bremergy, the team of the University of Bremen and Hochschule Bremen, wants to compete in the driverless discipline and needs reliable detection of the pylons that mark the track. This thesis compares several neural network architectures with regard to how reliably they detect and localize the pylons using a LiDAR. The research question of this bachelor's thesis is: *"Which combination of neural network architecture and LiDAR input representation achieves the most reliable pylon detection performance with real-time capable inference latency on the J4012 in Formula Student?"*. The comparison shows that the BEV-Pillar model achieves the best detection performance at the lowest inference time, with a WF1 score of 0.888 and a TensorRT latency of 8.3 ms on the J4012, and is therefore recommended for production use.

Inhaltsverzeichnis

1. Einleitung

Spätestens seit ChatGPT [1] ist Künstliche Intelligenz (KI) im Alltag vieler Menschen angekommen. Was vorher vor allem Gegenstand der Forschung war, wird heute in der Industrie eingesetzt, um Produkte zu optimieren oder den Menschen den Alltag zu erleichtern. Ein sichtbares Beispiel ist das US-Unternehmen Waymo, das in mehreren amerikanischen Großstädten einen bezahlten, vollständig autonomen Fahrdienst betreibt [2].

Für die Autoindustrie ist Motorsport seit jeher ein Mittel, um neue Technik unter harten Bedingungen zu erproben [3] und um Nachwuchs für das Ingenieurwesen zu begeistern. Genau dafür gibt es die Formula Student, einen Wettbewerb, in dem studentische Teams ihre selbst entwickelten Rennwagen gegeneinander antreten lassen. Seit 2017 gibt es dort die Kategorie Driverless, in der die Fahrzeuge vollständig autonom fahren müssen [4].

Eines dieser Teams ist Bremergy, der studentische Verein der Universität Bremen und der Hochschule Bremen [5]. Der Verein wurde 2012 gegründet [6] und ist seitdem fester Teilnehmer der Formula Student. Für die Saison 2025 wurde das Ziel gesetzt, eine eigene Software für Driverless zu entwickeln, für die Saison 2026 soll das Auto damit beim Formula Student Event in Spanien fahrerlos fahren.

Damit das gelingt, braucht Bremergy eine zuverlässige Erkennung der Pylone, die die Fahrbahn markieren. Viele Teams lösen diese Aufgabe mit Mono- oder Stereo-Kameras [7], da eine Kamera meist günstiger ist als ein LiDAR (Light Detection and Ranging). Bremergy verfügt jedoch über einen solchen Sensor. Ein LiDAR tastet die Umgebung mit Laserpulsen ab und erzeugt daraus eine dreidimensionale Punktwolke. Bisher ist weder im Team noch in der Formula Student allgemein untersucht worden, welche neuronale Netzarchitektur sich am besten eignet, um Pylone aus LiDAR-Daten zu erkennen.

Diese Lücke greift die vorliegende Arbeit auf. Untersucht werden vier unterschiedliche Netzarchitekturen daraufhin, wie gut sie Pylone erkennen und lokalisieren. Dabei gelten klare Anforderungen: Das System muss mit 10 Hz laufen und möglichst ressourcenschonend arbeiten, da die im Auto verbaute Recheneinheit (Seeed Studio J4012, Abschnitt 1.1) ein eingebettetes Modul ist. Solche Module arbeiten

mit einem Leistungsbudget von wenigen zehn Watt, während Grafikkarten in Arbeitsplatzrechnern ein Vielfaches davon aufnehmen und entsprechend mehr Rechenleistung bereitstellen. Ein Modell, das auf einem Arbeitsplatzrechner noch flüssig läuft, kann auf einem solchen Modul daher deutlich zu langsam sein. Die Anforderung von 10 Hz ergibt sich unmittelbar aus der Scanrate des verwendeten LiDAR, der zehn vollständige Umdrehungen pro Sekunde liefert [8]. Ein Modell, das langsamer arbeitet als der Sensor misst, müsste Scans verwerfen und würde die Umgebung entsprechend verzögert abbilden.
Daraus ergibt sich die Forschungsfrage: *„Welche Kombination aus Netzarchitektur und LiDAR-Eingabedarstellung erzielt für die Formula-Student-Pylonerkennung die zuverlässigste Detektionsleistung bei echtzeitfähiger Inferenzlatenz auf dem J4012?“.* Dies wird in dieser Arbeit systematisch und methodisch evaluiert.

Die meisten Teams setzen auf Kameras, während der Einstieg über einen LiDAR bislang kaum dokumentiert ist. Genau darin liegt die wissenschaftliche Relevanz dieser Arbeit. Sie soll diesen Einstieg erleichtern, indem sie eine Datengrundlage für alle Teams schafft, die Pylonerkennung mit einem LiDAR und neuronalen Netzen in Betracht ziehen. Abbildung 1.1 zeigt exemplarisch, wie eines der in dieser Arbeit untersuchten Modelle (BEV-Pillar, wobei BEV für Bird's Eye View steht, also die Vogelperspektive auf die Szene) Pylone auf einem realen LiDAR-Frame erkennt und lokalisiert.

Damit die Ergebnisse dieser Arbeit richtig eingeordnet werden können, sei an dieser Stelle der Untersuchungsgegenstand präzisiert. Gegenstand ist ein **Vergleich** der Frage, welche Kombination aus Netzarchitektur und LiDAR-Eingabedarstellung sich für die Pylonerkennung am besten eignet. Es entstehen dabei durchaus vier lauffähige Pylonerkenner, von denen einer produktiv auf dem Fahrzeug eingesetzt wird. Was nicht entwickelt wird, ist eine neuartige Netzarchitektur. Alle vier Basisarchitekturen stammen aus der Literatur. Der eigene Beitrag liegt in ihrer Anpassung an die Formula-Student-Domäne, ihrem Training auf einem gemeinsamen Datensatz und ihrer systematischen Evaluation auf identischer Zielhardware. Das Ergebnis ist entsprechend kein neues Erkennungsverfahren, sondern eine belastbare Entscheidungsgrundlage dafür, welches der bestehenden Verfahren für diesen Anwendungsfall zu wählen ist.

1.1. Fahrzeug, Sensoren und Pipeline

Bremergy baut für jede Saison ein neues Fahrzeug, das den Namen „Bremo“ mit der jeweiligen Jahreszahl trägt. Das in dieser Arbeit referenzierte Fahrzeug ist der Bremo25 [5] mit einem Gewicht von 222 kg (Abbildung 1.2). Bremergy gehört zu den wenigen Teams der Formula Student, die ihren Rennwagen seit der Vereinsgründung

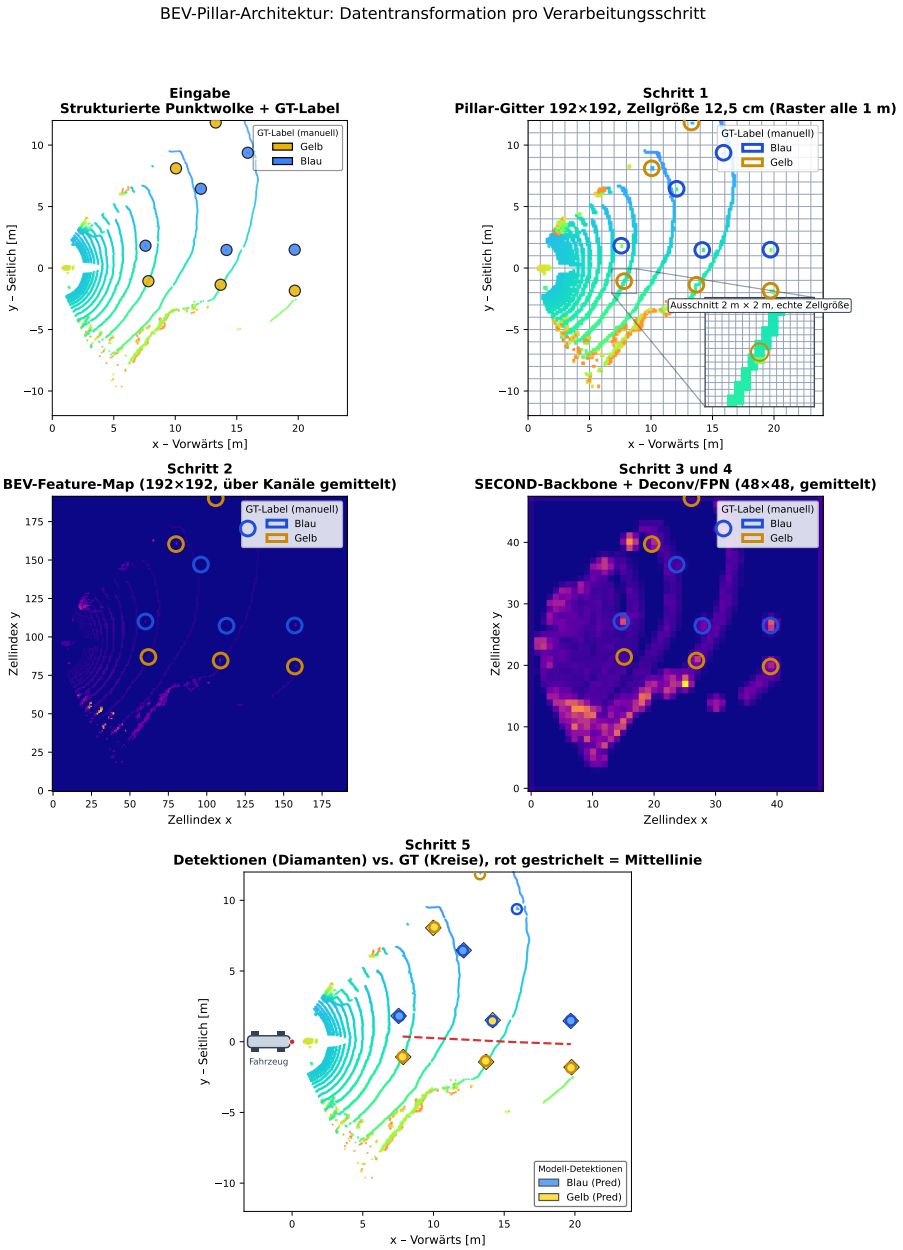

Abbildung 1.1.: Beispiel einer Pylonendetektion mit dem BEV-Pillar-Modell auf einem realen OS1-32-LiDAR-Frame (13.950 Punkte, Testday 2026), von der strukturierten Eingabe-Punktwolke über das Pillar-Gitter bis zu den finalen Detektionen (Diamanten) gegen GT-Labels (Kreise).

rein elektrisch betreiben. Damit erreicht der Wagen eine Höchstgeschwindigkeit von 126 km/h und beschleunigt in 2,6 Sekunden von 0 auf 100 km/h. Der Bremo26 ist zum Zeitpunkt des Verfassens dieser Arbeit fertiggestellt, jedoch liegen noch keine veröffentlichten technischen Daten vor.

Abbildung 1.2.: Der Bremo25 von Bremergy Racing. Die Aufnahme stammt aus dem internen Bremergy-SharePoint.

Für die Umgebungserkennung stehen zwei Sensoren zur Verfügung, eine Intel RealSense D435 als Kamera und ein Ouster OS1-32-U3 als LiDAR [8]. Der OS1-32-U3 tastet die Umgebung mit 32 vertikal versetzten Laserstrahlen ab und rotiert dabei horizontal mit 2048 Messpunkten pro Umdrehung, woraus sich bei einem vollen 360°-Umlauf maximal $2048 \times 32 = 65\,536$ Punkte pro Scan ergeben. Softwareseitig ist der Sensor in dieser Arbeit auf ein 120°-Vorwärtsfenster begrenzt, wodurch pro Scan typischerweise nur rund 22 000 dieser theoretisch möglichen Punkte belegt sind. Anders als bei einem reinen Kamerasystem nutzt diese Arbeit ausschließlich den LiDAR für die Pylonerkennung. Ein rein kamerabasiertes System wurde bereits von Yannis Fynn Meyer umgesetzt [9]. Die Modellausführung erfolgt auf einem Jetson Orin NX 16 GB (Seeed Studio J4012 [10]), im Folgenden J4012 genannt.

Die Software für das autonome Fahren ist in mehrere Funktionsblöcke gegliedert, die nacheinander durchlaufen werden. Wie diese Blöcke im Detail umgesetzt sind, hängt vom jeweiligen Team und Fahrzeug ab. Bei Bremergy ist die Pipeline so aufgebaut, wie in Abbildung 1.3 dargestellt.

Sensor Input: Hierzu zählen die Sensorwerte verschiedener Teile des Fahrzeugs. LiDAR und Kamera nehmen die Umgebung wahr, jedoch zählen auch Rad-Encoder dazu, die zurückgeben, wie schnell sich die Reifen drehen.

Erkennung: Hier werden die Umgebungssensordaten von Kamera und LiDAR verarbeitet. In diesem Block sitzt die in dieser Arbeit untersuchte neuronale Pylonerkennung auf Basis der LiDAR-Punktwolke. Ergebnis dieses Blocks ist die Position der erfassten Pylone.

Lokalisation & Kartierung: Die hier ausgegebenen Informationen liefern die aktuelle Position des Fahrzeugs. Dafür kommt SLAM (Simultaneous Localization and Mapping) zum Einsatz, ein Verfahren, das die eigene Position und eine Karte der Strecke gleichzeitig bestimmt. Dabei wird zusätzlich die Pylonerkennung korrigiert.

Streckenplaner: Ermittelt die Mittellinie und damit den zu fahrenden Pfad durch die Strecke.

Ansteuerung: Hier wird der zu fahrende Pfad in konkrete Steuersignale für Motoren und Lenkungsmotor übersetzt. Anschließend wird ein Wunsch für Richtung und Geschwindigkeit an die Steuerelektronik der Embedded-Software-Abteilung gesendet.

Abbildung 1.3.: Die Driverless Pipeline von Bremergy als sequentieller Ablauf der autonomen Fahrsoftware. Der Erkennung-Block (hervorgehoben) ist Gegenstand dieser Arbeit.

1.2. Formula Student Driverless Cup

Der Formula Student Driverless Cup ist eine eigenständige Kategorie, die es seit 2017 gibt [4]. In dieser Kategorie können Teams ihre Fahrzeuge fahrerlos gegeneinander antreten lassen. Dabei setzen die Teams auf unterschiedlichste Mittel, um die Fahrbahn zu erkennen. Die Fahrbahn sowie die Anforderungen an die Software sind je nach Disziplin verschieden, insgesamt gibt es vier Disziplinen [11].

1. **Acceleration:** Zählt allein die Längsbeschleunigung. Das Auto muss eine 75 m lange Gerade so schnell wie möglich zurücklegen.
2. **Skidpad:** Prüft das Fahrverhalten in Kurven. Gefahren wird eine liegende Acht, die mehrfach durchfahren wird, sodass Auto und Software gleichermaßen gefordert sind.

3. **Autocross:** Eine einzelne Runde auf einem vorher unbekannten, geschlossenen Kurs.

4. **Trackdrive:** Die anspruchsvollste Disziplin, weil die Strecke vorher nicht bekannt ist. Der geschlossene Kurs wird zehnmal durchfahren, eine Runde ist laut Regelwerk 200 bis 500 m lang, insgesamt also 2 bis 5 Kilometer. Gewertet wird die Gesamtzeit über die zehn Runden, für jeden umgefahrenen Pylon kommen 2 Strafsekunden hinzu.

Da das Regelwerk bei Autocross und Trackdrive nur grobe Vorgaben zum Streckenaufbau macht, muss die Software bei beiden Disziplinen mit einem unbekannten Streckenverlauf zurechtkommen.

In allen vier Disziplinen ist die Fehlertoleranz gering. Wird ein Pylon falsch oder gar nicht erkannt, kann das Auto von der Strecke abkommen und im schlimmsten Fall disqualifiziert werden. Eine genaue Pylonerkennung ist damit die Grundlage für jede weitere Berechnung in der Software.

Allen Disziplinen ist gemeinsam, dass die Fahrbahn durch farbige Pylone begrenzt wird und mindestens 3 m breit ist [11]. Der linke Rand wird durch blaue, der rechte durch gelbe Pylone markiert. Start und Ziel sind zusätzlich durch große und kleine orange Pylone gekennzeichnet. Damit unterscheidet die Formula Student vier Pylonarten, die in Abbildung 1.4 dargestellt sind. Diese Arbeit beschränkt sich auf die Erkennung von blauen und gelben Pylonen, da die verwendeten Datensätze (siehe Abschnitt 2.2) ausschließlich diese beiden Klassen labeln. Orange Pylone werden nicht separat klassifiziert.

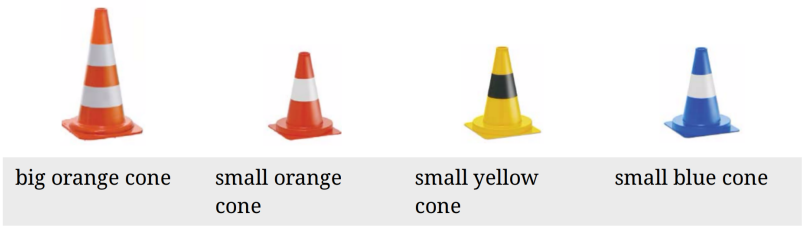

Abbildung 1.4.: Die vier Pylonarten einer Formula Student Driverless Strecke, also Orange (groß), Orange (klein), Gelb und Blau [12]. Deutlich erkennbar sind die horizontalen Streifen, die bei gelben Pylonen schwarz und bei blauen weiß ausgeführt sind. Dieser Unterschied ist die Grundlage der Farbklassifikation über die Reflektivität (Abschnitt 2.1).

1.3. Aufbau der Arbeit

Neben der Einleitung und der Zusammenfassung am Ende gliedert sich diese Arbeit in die folgenden vier Kapitel. Häufig verwendete Abkürzungen sind zusätzlich im Abkürzungsverzeichnis am Ende der Arbeit aufgelistet.

Kapitel 2 vermittelt die für alle vier Modelle gemeinsamen fachlichen Grundlagen, also den Stand der Technik in der LiDAR-basierten 3D-Objekterkennung, den verwendeten Datensatz sowie die Metriken, mit denen alle Modelle einheitlich bewertet werden, und den gemeinsamen Trainingsaufbau.

Kapitel 3 beschreibt die Vorgehensweise dieser Arbeit, also die Fragestellung, den eigenen Anteil an Datenerhebung, Anpassung, Training und Auswertung sowie den wiederkehrenden Aufbau der folgenden Modellarchitekturen.

Kapitel 4 behandelt die vier untersuchten Architekturen. Nach den gemeinsamen Komponenten aus SECOND-Backbone, Deconv/FPN und CenterHead (Abschnitt 4.1) beschreibt Abschnitt 4.2 den Datenfluss der BEV-Pillar-Architektur von der strukturierten Punktwolke über den Pillar-Encoder bis zur anchorfreien Heatmap-Detektion, Abschnitt 4.3 den Unterschied zum Pillar-Modell durch den dichten 3D-Faltungs-Encoder als Ersatz für Sparse-Convolution, Abschnitt 4.4 die sphärische Projektion und den daran angepassten Backbone sowie die Eigenschaften der Range-Image-Projektion bei kleinen Objekten, und Abschnitt 4.5 die direkte Verarbeitung der strukturierten Punktwolke mit IA-SSD [13]. Jeder Abschnitt bewertet das jeweils trainierte Modell auf dem gemeinsamen Testdatensatz.

Kapitel 5 vergleicht alle vier Architekturen anhand der Ergebnisse des gemeinsamen Testdatensatzes. Es diskutiert Erkennungsleistung, Farbgenauigkeit, Latenz und Einsatzfähigkeit und gibt eine Empfehlung für den Produktivbetrieb.

Anhang A ergänzt die Modellkapitel um großformatige Abbildungen. Für jede der vier Architekturen finden sich dort ein Diagramm des vollständigen Verarbeitungsablaufs sowie eine Übersicht, die an einem realen LiDAR-Frame zeigt, wie sich die Daten in jedem einzelnen Verarbeitungsschritt verändern. Wer nachvollziehen möchte, was ein bestimmter Schritt bildlich bewirkt, findet die entsprechende Darstellung dort.

1.4. Beitrag der Arbeit

Die zentralen Beiträge dieser Arbeit lassen sich wie folgt zusammenfassen.

- Erster systematischer Vergleich von vier grundlegend verschiedenen LiDAR-Eingabemethoden für die Pylonerkennung im Formula-Student-Kontext auf produktionstauglicher Hardware.
- Implementation, Training und Evaluation von vier Architekturen (BEV-Pillar, BEV-Voxel, Range-Image, Raw-Point) auf einem gemeinsamen 49-Frame-Testdatensatz.
- Geometrisch begründeter Nachweis einer strukturellen Erkennungsgrenze für Range-Image-Detektoren auf dem OS1-32-U3. Ab 9,4 m passt ein Pylon nicht mehr in eine Bildzelle. Zehn Modellversionen und ein Kontrollexperiment bestätigen diesen Befund empirisch.
- Identifikation der physikalischen Ursache des Fernbereich-Einbruchs beim Raw-Point-Modell. In durchgeführten Experimenten zeigte sich, dass Pylone mit nur einem LiDAR-Treffer von der Ball-Query-Gruppierung [14] nicht zuverlässig verarbeitet werden können. Bei zwei oder mehr Treffern liegt der Recall auf BEV-Niveau.
- Entwicklung eines Metrik-Pakets, das auf Abstandsmatching statt IoU basiert und Recall, Precision und Farbgenauigkeit getrennt nach Entfernungsbereichen ausweist.
- Adaption des Voxel-Encoders auf reguläre 3D-Faltungen, um TensorRT-Exportierbarkeit auf dem J4012 herzustellen.

2. Grundlagen der LiDAR-basierten Objekterkennung

2.1. Stand der LiDAR-Technik

Die 3D-Objekterkennung in LiDAR-Punktwolken ist ein aktives Forschungsfeld, das durch neuronale Netze seit 2018 erheblich an Leistung gewonnen hat [15]. Dabei haben sich vier grundlegend verschiedene Eingabemethoden etabliert.

BEV-Pillar: PointPillars [16] teilt den Raum in vertikale Säulen auf, kollabiert die Höhendimension zu einem Merkmalsvektor und verarbeitet das resultierende 2D-Vogelperspektiv-Bild mit einem Standard-CNN. Der Ansatz ist besonders recheneffizient, verliert durch das Kollabieren der Höhendimension jedoch vertikale Information.

BEV-Voxel: VoxelNet [17] teilt die strukturierte Punktwolke in ein dreidimensionales Gitter und verarbeitet es mit 3D-Faltungen. Die Höhendimension bleibt erhalten, was einen höheren Rechenaufwand bedeutet, dafür aber mehr räumliche Information liefert.

Range-Image: RangeDet [18] projiziert die strukturierte Punktwolke aus Sensorsicht in ein sphärisches Panoramabild, bei dem jede Zeile einem Laserstrahl entspricht. Klassische 2D-Bildverarbeitungsarchitekturen können direkt auf dieser kompakten Darstellung arbeiten, bei kleinen, entfernten Objekten stößt die Winkelauflösung dieser Darstellung jedoch an eine geometrische Grenze.

Raw-Point: Methoden wie PointNet++ [14] und IA-SSD [13] verarbeiten die Punkte ohne Rasterschritt direkt, indem sie für jeden Punkt die Merkmale seiner räumlichen Nachbarn zusammenfassen und diesen Vorgang über mehrere Stufen mit wachsendem Suchradius wiederholen. Dies vermeidet Quantisierungsverluste, also den Genauigkeitsverlust, der entsteht, wenn kontinuierliche Punktkoordinaten einem Gitter mit fester Zellgröße zugeordnet werden. Der Ansatz ist jedoch schwieriger auf dedizierte Inferenz-Hardware zu exportieren.

Drei dieser vier Verfahren geben ihre Detektionen nicht direkt als Liste von Objekten aus, sondern über eine Heatmap [19]. Eine Heatmap ist ein Bild in der Auflösung des Detektionskopfes, in dem jede Zelle einen Wert zwischen 0 und 1 trägt, nämlich

die Sicherheit des Netzes, dass an dieser Stelle der Mittelpunkt eines Objekts liegt. Für jede Objektklasse entsteht eine eigene Heatmap, und die Detektionen ergeben sich anschließend aus deren lokalen Maxima. Raw-Point arbeitet ohne Raster und damit auch ohne Heatmap.

Im Formula-Student-Kontext hat AMZ Racing [20] gezeigt, wie ein vollständiges autonomes Rennsystem mit LiDAR-Erkennung aufgebaut werden kann.

Klassische Farbkameras unterscheiden Farben über das sichtbare Lichtspektrum. LiDAR-Sensoren hingegen arbeiten im Infrarotbereich und messen die Reflektivität von Oberflächen, nicht deren Farbe. Für die Pylonerkennung ergibt sich daraus eine indirekte Methode zur Farbunterscheidung, die dieselbe strukturierte Punktwolke nutzt wie die eigentliche Positionserkennung. Die Erkennung der Farbe erfolgt somit nicht über die eigentliche Farbe, da Blau und Gelb im Infrarotbereich kaum unterscheidbar sind.

Formula-Student-Pylone sind mit horizontalen Pylonstreifen versehen [12]. Blaue Pylone haben weiße Streifen, gelbe Pylone schwarze, wie in Abbildung 1.4 zu sehen ist. Weiße Oberflächen reflektieren Infrarotlicht weitgehend, schwarze Oberflächen absorbieren es größtenteils. Treffen mehrere der 32 Laserstrahlen einen Pylon aus unterschiedlichen Höhen, entsteht ein vertikales Muster aus abwechselnd hohen und niedrigen Reflektivitätswerten [8]. Dieses Muster ist für blaue und gelbe Pylone spiegelverkehrt und dient den neuronalen Netzen als Grundlage für die Farbklassifikation.

Voraussetzung ist, dass mindestens zwei Strahlen den Pylon aus verschiedenen Höhen treffen und somit beide Streifenarten sichtbar machen. Im Fernbereich, wo oft nur ein einziger Strahl auftrifft, ist eine zuverlässige Farbklassifikation entsprechend schwieriger.

2.2. Datensatz und Metriken

Damit ein neuronales Netz trainiert werden kann, werden Daten benötigt, aus denen das Netz lernen kann. Alle vier Modelle dieser Arbeit werden überwacht trainiert, das Netz bekommt also zu jeder Aufnahme die zugehörige richtige Lösung vorgelegt und lernt aus der Abweichung seiner Vorhersage. Zu Beginn der Bachelorarbeit verfügte das Team Bremergy über keine eigenen Daten, daher wurde ein externer Datensatz beschafft. Dieser wurde vom Formula Student Team der Chalmers University of Technology (Chalmers Formula Student) erstellt [21]. Es handelt sich dabei um 63 Frames, die mit Labeln versehen wurden, in diesem Datensatz kommen `blue_cone` und `yellow_cone` zum Einsatz, weder große noch kleine orange Pylone sind gelabelt. Diese manuell vergebenen Referenzangaben werden als Ground-Truth-Label

(GT-Label) bezeichnet, ein so gekennzeichneter Pylon wird im Folgenden als GT-Pylon bezeichnet. Anhand von ihnen lässt sich im Training vergleichen, inwieweit die Vorhersage des Modells von der tatsächlichen Position und Farbe des Pylons abweicht.

Im Verlauf der Arbeit wurden eigene Fahraufnahmen mit dem LiDAR von Bremergy erstellt und manuell gelabelt. Der Gesamtdatensatz setzt sich damit aus dem Chalmers-Datensatz [21] und den eigenen Aufnahmen zusammen. Insgesamt stehen damit 128 real aufgenommene und gelabelte Frames zur Verfügung. Davon entfallen 79 Frames mit 2491 Pylonen auf den Trainingsdatensatz und 49 Frames mit 1018 Pylonen auf den davon getrennten Testdatensatz. Gemessen an der Zahl der gelabelten Pylone entspricht das einer Aufteilung von 71 zu 29 Prozent. Beide Teile enthalten sowohl Aufnahmen aus dem Chalmers-Datensatz als auch eigene Aufnahmen mit dem Sensor des Fahrzeugs, sodass die Auswertung auch die Übertragbarkeit auf den im Fahrzeug verbauten Sensor abdeckt. Auf dem Testdatensatz werden alle in dieser Arbeit berichteten Ergebnisse der vier Modelle gemessen. Während des Trainings kommt zusätzlich ein Validierungsdatensatz zum Einsatz, der dem Testdatensatz entnommen ist. Auf ihm wird nach jeder Epoche der Fortschritt gemessen, um den besten Modellstand auszuwählen und das Training zu beenden, sobald es sich nicht mehr verbessert. Er liefert außerdem die Grundlage für Entscheidungen über einzelne Einstellungen der Modelle, etwa den Schwellwert, ab dem eine Detektion ausgegeben wird. In das Lernen der Gewichte fließt er dagegen nicht ein, das Netz bekommt diese Aufnahmen im Training nie zu sehen. Diese Aufteilung ist dem Umfang der verfügbaren Daten geschuldet. Eine dritte, vollständig eigenständige Teilmenge würde entweder dem Training oder dem Testdatensatz einen erheblichen Teil der gelabelten Pylone entziehen und damit die Aussagekraft der berichteten Ergebnisse verringern.

Der Chalmers-Datensatz liegt in einem Label-Format vor, das mit dem in dieser Arbeit eingesetzten Trainingsframework mmdetection3d von OpenMMLab [22] kompatibel ist. Das Framework wurde auch deshalb gewählt, weil es Referenzarchitekturen bekannter Datensätze wie KITTI [23] und nuScenes [24] mitliefert und auf PyTorch [25] aufbaut, was den Export dreier der vier trainierten Modelle nach ONNX und TensorRT ermöglicht.

Um zu bewerten, was ein Modell richtig und falsch erkannt hat, wird jede Detektion mit den GT-Labeln abgeglichen. Daraus ergeben sich die gängigen Kennzahlen True Positive (TP), False Positive (FP) und False Negative (FN). Als Matching-Kriterium gilt ein zweidimensionaler BEV-Abstand von maximal 0,5 m zwischen erkanntem und tatsächlichem Pylon. Eine IoU-basierte Bewertung scheidet aus, da ein Pylon mit einem Basisdurchmesser von 22,8 cm nur wenige Gitterzellen groß ist und die IoU bei so kleinen Boxen schon bei wenigen Zentimetern Versatz nahe null liegt. Der Schwellwert von 0,5 m ist so gewählt, dass er deutlich unter dem

geringsten Abstand zweier Pylone auf der Strecke liegt. Gegenüberliegende Pylone sind nach Regelwerk mindestens 3 m voneinander entfernt, da die Fahrbahn diese Mindestbreite hat [11], und im Testdatensatz beträgt der kleinste gemessene Abstand zweier Pylone 1,16 m und damit mehr als das Doppelte des Schwellwerts. Eine Detektion kann so nie versehentlich dem falschen Pylon zugeordnet werden. Gleichzeitig ist er groß genug, dass eine für die Streckenplanung noch brauchbare Detektion nicht als Fehler gewertet wird.

Recall und **Precision** sind Grundlage des WF1-Scores. Der Recall gibt an, wie viele der vorhandenen Pylone das Modell gefunden hat (0 bis 1, wobei 1 bedeutet, dass alle Pylone gefunden wurden). Die Precision gibt an, wie viele der gefundenen Pylone auch wirklich vorhanden waren (0 bis 1, wobei 1 bedeutet, dass keine Fehldetektionen auftreten). Formal ergeben sie sich als Recall $= TP/(TP + FN)$ und Precision $= TP/(TP + FP)$. Beide Kennzahlen sowie der daraus gebildete F1-Score als ihr harmonisches Mittel sind Standardmaße der Objekterkennung [23].

WF1 (Weighted F1 Score, 0 bis 1) ist eine distanzgewichtete Variante des F1-Scores. Der F1-Score ist das harmonische Mittel aus Precision und Recall.

Der Erkennungsbereich reicht 24 m vor das Fahrzeug (Abschnitt 2.3). Alle Metriken dieser Arbeit werden auf diesem Bereich berechnet. Für den WF1-Score wird er in mehrere Entfernungsbereiche unterteilt, also in aufeinanderfolgende Intervalle des Abstands zum Sensor, nämlich 0–8 m, 8–16 m und 16–24 m. Jede Detektion und jeder GT-Pylon wird anhand seines radialen Abstands genau einem dieser Entfernungsbereiche b zugeordnet. Für jeden Bereich wird anschließend ein eigener F1-Score $F_{1,b}$ bestimmt, in den ausschließlich die Detektionen und GT-Pylone dieses Bereichs eingehen. Die Bereiche werden abschließend zu einem gewichteten Mittel zusammengefasst.

$$\mathrm{WF1} = \frac{\sum_b w_b \cdot F_{1,b}}{\sum_b w_b}$$

Dabei bezeichnet w_b das Gewicht des Entfernungsbereichs b. Die Gewichtung priorisiert dabei den Nahbereich gegenüber dem Fernbereich. Enthält ein Entfernungsbereich weder GT-Pylone noch Detektionen, ist $F_{1,b}$ nicht definiert und der Bereich bleibt bei der Mittelung unberücksichtigt, also auch im Nenner. Die in der Literatur für 3D-Objekterkennung übliche mittlere Average Precision (mAP) [23, 24] wurde hier bewusst nicht verwendet. Er mittelt die Precision über alle möglichen Score-Schwellwerte hinweg und bewertet damit vor allem die Rangfähigkeit eines Modells. Im produktiven Einsatz wird jedoch pro Modell nur ein einziger, fest konfigurierter Schwellwert verwendet (siehe jeweiliger Anpassungen-Abschnitt in Kapitel 4), weshalb die tatsächliche Leistung an genau diesem Betriebspunkt die für diese Arbeit relevantere Größe ist.

wRecall und **wPrec** (jeweils 0 bis 1) sind nach demselben Schema gebildet wie der WF1-Score. Statt eines F1-Scores wird je Entfernungsbereich der Recall beziehungsweise die Precision bestimmt und anschließend mit denselben Gewichten w_b gemittelt, wobei unbesetzte Bereiche genauso unberücksichtigt bleiben. Beide zeigen, ob ein Modell eher Pylone übersieht oder eher zu viele Detektionen ausgibt, während der WF1-Score beides zu einer Zahl zusammenfasst. Der WF1-Score wird dabei aus den F1-Werten der einzelnen Entfernungsbereiche gebildet und nicht aus wRecall und wPrec.

R@Distanz (0 bis 1) gibt den Recall pro Entfernungsbereich an und zeigt, in welchem Bereich die Erkennungsleistung eines Modells nachlässt. Es ist derselbe Recall wie oben, jedoch nur über die Pylone eines Entfernungsbereichs b berechnet.

$$\mathrm{R@}b = \frac{TP_b}{TP_b + FN_b}$$

TP_b ist die Zahl der korrekt erkannten GT-Pylone dieses Bereichs, FN_b die Zahl der dort übersehenen. Verwendet werden dieselben Entfernungsbereiche wie beim WF1-Score. Sie teilen den Erkennungsbereich in drei gleich große Abschnitte von je 8 m.

ColorAcc (Color Accuracy, 0 bis 1) ist der Anteil der korrekt erkannten Positionen, bei denen auch die Farbe stimmt. Der Positionsabgleich erfolgt dabei farbunabhängig, bewertet wird anschließend nur, ob die vorhergesagte Farbklasse mit der des GT-Pylons übereinstimmt.

$$\mathrm{ColorAcc} = \frac{TP_{\text{Farbe korrekt}}}{TP}$$

Sie wird separat ausgewiesen, weil im Gesamtsystem die Kamera die Farbe zuverlässiger bestimmt als der LiDAR.

MAE (Mean Absolute Error, in Metern, 0 = perfekte Lokalisierung) ist der mittlere euklidische Abstand in der BEV-Ebene zwischen erkannter Position $\hat{p}_i$ und tatsächlicher Pylonposition p_i. Berechnet wird er nur über die Menge der korrekten Detektionen (TP). Gegenüber einem quadratischen Fehlermaß wie dem RMSE wurde der MAE gewählt, da er einzelne große Ausreißer nicht überproportional gewichtet und damit die typische Lokalisierungsgenauigkeit robuster abbildet.

torch-ms / TRT-ms ist die Ende-zu-Ende-Latenz für einen Frame in Millisekunden. torch-ms steht für die vollständige mmdetection3d-Pipeline in PyTorch, TRT-ms für die exportierte TensorRT-Engine inklusive Vor- und Nachverarbeitung. Gemessen wurde auf dem J4012 im FP32-Modus mit deaktiviertem TF32.

Trainingszeit (in Stunden) ist die Dauer von Trainingsbeginn bis zu dem durch Early Stopping ausgelösten Ende oder bis zum Erreichen des Epochenlimits, gemessen auf der für das Training verwendeten Desktop-GPU (NVIDIA RTX 5080).

Sie zeigt, wie aufwendig ein Modell in der Entwicklung ist, unabhängig von seiner späteren Inferenzlatenz auf dem J4012.

Für die Bewertung der vier Modelle in dieser Arbeit sind Erkennungsleistung (WF1, R@Distanz, MAE) und Latenz (torch-ms/TRT-ms) die entscheidenden Metriken, da eine zuverlässige, präzise und rechtzeitige Pylonerkennung die Grundvoraussetzung für die Streckenplanung ist. ColorAcc wird dagegen nachrangig gewichtet, da im Gesamtsystem die Kamera die Farbklassifikation zuverlässiger bestimmt als der LiDAR und ein Modell mit niedrigerer Farbgenauigkeit, aber hoher Positionsgenauigkeit für den produktiven Einsatz damit nicht automatisch schlechter geeignet ist.

2.3. Gemeinsamer Trainingsaufbau

Die Modelle haben viele Gemeinsamkeiten, was das Training angeht. Begonnen wurde damit, die Distanz des Erkennungshorizonts festzulegen, ein LiDAR-System ist theoretisch in der Lage über viele hundert Meter Objekte zu erkennen, jedoch tritt das im betrachteten Szenario nicht ein. In Abstimmung mit dem Team Bremergy wurde festgelegt, dass der Erkennungsbereich 24 m vor dem Fahrzeug und jeweils 12 m seitlich beträgt. Da der Sensor so konfiguriert ist, dass er nur den 120°-Vorwärtsbereich aufnimmt statt den vollen 360°-Umlauf, enthält ein Scan maximal rund 22 000 Punkte statt der theoretischen 65 536. Nach Filterung auf den rechteckigen Erkennungsbereich verbleiben typischerweise einige tausend Punkte für die Netzverarbeitung, dieser Wert gilt für alle vier Modelle. 24 m hat jedoch auch noch den technischen Grund, dass $24\,\text{m} / 0{,}125\,\text{m} = 192$ Gitterzellen ergibt, was ohne Rest durch 16 teilbar ist, dies ist eine Voraussetzung der SECONDFPN-Deconvolution-Schichten. Darüber hinaus wurde als Optimizer AdamW [26] verwendet, da er in den Referenzimplementierungen aller vier verglichenen Basisarchitekturen zum Einsatz kommt. Ein einheitlicher Optimizer stellt sicher, dass gemessene Leistungsunterschiede auf die Architektur zurückgehen und nicht auf unterschiedliche Optimierungsverfahren. Jedes Modell wurde mit FP32 trainiert, also mit 32-Bit-Gleitkommazahlen. Die Gründe gegen das sparsamere FP16 sind in Abschnitt 4.2 beschrieben. Des Weiteren hat jedes Modell augmentierte LiDAR-Scans erhalten, folgende Augmentationsfunktionen wurden verwendet.

RandomFlip3D [16]: Es wird auf X und Y Ebene so gespiegelt, dass aus einer echten aufgenommenen Linkskurve eine augmentierte Rechtskurve entsteht. Dies hilft Kurven in beide Richtungen gleich gut zu erkennen.

Random Z-Rotation [27]: Dreht die gesamte Szene um die Z-Achse des Sensors. So wird erreicht, dass eine Erkennung von Pylonen möglich ist, auch wenn der Sensor nicht exakt in Fahrtrichtung ausgerichtet ist. Sie ermöglicht dem Modell die Pylone zu erkennen, auch wenn die Winkel zu den Pylonen sich verändert haben.

RingDropout: Dies ist eines der wichtigsten Mittel, um eine robuste LiDAR-Pylonerkennung zu gewährleisten. Bei dieser Augmentierung werden einzelne der 32 Strahlenringe vollständig aus dem Scan entfernt, sodass das Modell lernt, auch mit fehlenden Zeilen umzugehen. Das Fahrzeug wird sich beim Bremsen und bei Kurvenfahrten in eine Richtung neigen, dies bezeichnet man als Rollen. Wenn dies passiert, steht der LiDAR nicht mehr parallel zum Boden und es kann aufgrund der geänderten Neigung des LiDARs dazu kommen, dass Teile der strukturierten Punktwolke fehlen. Dies stellt eine Herausforderung dar, denn selbst bei unvollständigem Frame soll das Modell weiterhin Pylone erkennen, sofern diese in der strukturierten Punktwolke vorhanden sind.

3. Methodisches Vorgehen

Die in Kapitel 1 formulierte Forschungsfrage lässt sich nur beantworten, wenn alle vier Architekturen unter identischen Bedingungen bewertet werden, also auf demselben Testdatensatz, mit denselben Metriken und auf derselben Zielhardware. Daraus ergibt sich der eigene Anteil an dieser Arbeit. Zu Beginn lag kein ausreichender eigener Datensatz vor, weshalb der vorhandene externe Datensatz durch eigene, manuell gelabelte LiDAR-Aufnahmen erweitert wurde. Jede Basisarchitektur wurde anschließend an die Formula-Student-Domäne angepasst, etwa an Größe und Form der Pylone, den begrenzten Erkennungsbereich und die Zielhardware J4012. Das Training der vier Modelle und die Behebung der dabei aufgetretenen Instabilitäten wurden eigenständig durchgeführt. Für den abschließenden Vergleich entstand ein gemeinsames Evaluationsverfahren, das alle vier Modelle auf demselben Testdatensatz und mit denselben Metriken bewertet.

Für den Vergleich wurden vier Architekturfamilien ausgewählt, die den Design-Raum der LiDAR-Objekterkennung hinsichtlich Eingabedarstellung und Verarbeitung der Höheninformation weitgehend abdecken.

BEV-Pillar ist als CenterPoint [19] in der Pillar-Konfiguration umgesetzt, die den Pillar-Encoder aus PointPillars [16] mit einem anchorfreien Head und dem SECOND-Backbone [27] kombiniert und dabei die Höheninformation kollabiert.

BEV-Voxel basiert auf demselben CenterPoint-Head, erhält die Höheninformation jedoch in einem dreidimensionalen Voxelgitter [17].

Range-Image orientiert sich an RangeDet [18, 28] und projiziert die strukturierte Punktwolke aus Sensorsicht in ein sphärisches Panoramabild.

Raw-Point ist als IA-SSD [13] umgesetzt, das auf den Set-Abstraction-Schichten von PointNet++ [14] aufbaut. Es verzichtet vollständig auf ein Zwischenformat und verarbeitet die strukturierte Punktwolke direkt, so wie sie der Sensor liefert.

Um die vier Architekturen trotz ihrer strukturellen Unterschiede vergleichbar zu behandeln, folgt jeder der Modell-Abschnitte Abschnitt 4.2 bis Abschnitt 4.5 in Kapitel 4 derselben Gliederung aus Architektur, Anpassungen, Training und Ergebnissen. Vorangestellt sind die gemeinsamen Netzkomponenten (Abschnitt 4.1), die Pillar

und Voxel gemeinsam nutzen, damit sie nicht mehrfach erklärt werden müssen. Der Architektur-Abschnitt beschreibt den Datenfluss der jeweiligen Basisarchitektur und begründet ihre Auswahl, der Anpassungs-Abschnitt begründet die Änderungen, die für die Übertragung auf die Formula-Student-Pylonerkennung und den J4012 notwendig waren. Der Trainings-Abschnitt geht nur auf Abweichungen vom gemeinsamen Trainingsaufbau aus Abschnitt 2.3 ein, um Wiederholungen zu vermeiden. Der Ergebnis-Abschnitt interpretiert jeweils, ob sich die aus der Literatur erwarteten Eigenschaften der Architektur in den eigenen Daten bestätigen. Wo eine Architektur auf Konzepten eines vorherigen Abschnitts aufbaut, etwa der Heatmap-Detektion oder dem SECOND-Backbone, wird auf die entsprechende Stelle verwiesen, anstatt sie erneut zu erklären, damit durchgängig erkennbar bleibt, was aus der Literatur übernommen und was eigene Anpassung oder Interpretation ist. Von jeder Architektur entstanden im Verlauf der Arbeit mehrere Versionen, die jeweils ein konkretes Problem der Vorgängerversion beheben sollten, etwa eine unpassende Auflösung des Detektionskopfes oder eine instabile Loss-Funktion. Beschrieben und verglichen wird jeweils nur die beste Version einer Architektur, sodass jede Architektur mit ihrem besten erreichten Stand in den Vergleich eingeht. Die vier Einzelergebnisse werden abschließend in Kapitel 5 gegenübergestellt.

4. Anpassung, Training und Evaluation der Netzarchitekturen

4.1. Gemeinsame Netzkomponenten

BEV-Pillar (Abschnitt 4.2) und BEV-Voxel (Abschnitt 4.3) unterscheiden sich hauptsächlich darin, wie die Punktwolke zu einer zweidimensionalen BEV-Feature-Map codiert wird. Backbone, Neck, Head und Dekodierung sind bei beiden identisch und werden hier einmal beschrieben, statt sie in jedem Abschnitt zu wiederholen. Diese Codierung bildet Schritt 1 der Verarbeitung und wird bei Pillar in Abschnitt 4.2 und bei Voxel in Abschnitt 4.3 beschrieben, die hier folgenden Schritte 2 bis 5 sind für beide dieselben. Die Nummerierung entspricht den Marken in den Ablaufdiagrammen im Anhang. Range-Image (Abschnitt 4.4) verwendet dieselben Grundprinzipien in einer an seine Bildgeometrie angepassten Form, die dort separat beschrieben wird. Das Range-Image ist mit 32 Zeilen und 2048 Spalten stark ungleichmäßig geformt, weshalb seine Faltungen in vertikaler und horizontaler Richtung unterschiedlich stark zusammenfassen. Ein solches richtungsabhängiges Verhalten wird als anisotrop bezeichnet, im Unterschied zu den isotropen Faltungen bei Pillar und Voxel, die beide Bildrichtungen gleich behandeln. Raw-Point (Abschnitt 4.5) folgt einem eigenständigen Paradigma ohne gemeinsame Komponenten.

Gemeinsame Eingabe: Pillar, Voxel und Range-Image erhalten als Input dieselbe strukturierte Punktwolke mit $2048 \times 32 \times 4$ Werten (x, y, z, reflectivity), was dem vollen 360°-Abtastumfang des Sensors bei 32 Beams entspricht. Da der Sensor nur den 120°-Vorwärtsbereich aufnimmt, sind typischerweise rund 22 000 der theoretisch möglichen 65 536 Punkte belegt. Raw-Point erhält dieselben Punkte als flache, unstrukturierte Liste (Abschnitt 4.5).

2. SECOND-Backbone [27]: Der Backbone verarbeitet die vom jeweiligen Encoder erzeugte BEV-Feature-Map mithilfe von drei Faltungsstufen, jede Stufe halbiert die räumliche Auflösung und erhöht die Kanaltiefe. Durch die Reduzierung der räumlichen Auflösung lassen sich feine sowie grobe Features aus der BEV-Feature-Map abstrahieren, jede Stufe schickt ihre Ausgabe zusätzlich an die benachbarte Stufe im Deconv/FPN.

3. Deconv/FPN (Neck) [29]: Die drei Backbone-Ausgaben werden auf eine einheitliche Auflösung von 48×48 gebracht und konkateniert, sodass feine und grobe Features kombiniert und einheitlich an den Head übergeben werden.

4. CenterHead [19]: Pro Farbklasse, blau oder gelb, werden Heatmaps mit der Auflösung 48×48 erstellt. Ein hoher Ausschlag von Features wird als Pylonposition markiert, zusätzlich gibt der Head vier weitere Ausgabewerte pro Zelle aus, nämlich x/y-Offset, Höhe, Dimension und Ausrichtung.

x/y-Offset beinhaltet die Koordinaten des Pylon-Featureausschlags relativ zum Zentrum der Gitternetzzelle.

Höhe ist die geschätzte z-Position des Boxzentrums. Sie wird für die Pylonerkennung nicht ausgewertet, da Pylone im betrachteten Szenario immer auf dem Boden stehen und ihre Höhe damit vorab bekannt ist.

Dimension gibt die dreidimensionale Bounding Box an, die Dimensionen sind hier Breite, Länge und Höhe der vorhergesagten Bounding Box.

Ausrichtung gibt die Rotation des Objektes um die Hochachse an (Gierwinkel, engl. *yaw*), kodiert als sin(yaw) und cos(yaw). Jedoch wird dies nicht weiter betrachtet, da für die Pylonerkennung die Rotation der Pylone keine relevante Information für die Streckenplanung enthält.

5. Dekodierung [19] und Circle-NMS [22]: Lokale Maxima in der Heatmap werden markiert, sobald der Heatmapwert einen modellspezifischen Schwellwert überschreitet (siehe jeweiliger Anpassungen-Abschnitt). Im Anschluss wird mithilfe des x/y-Offsets die Koordinate im Gitternetz verfeinert und zurück in Meter umgerechnet. Circle-NMS sortiert die Detektionen nach Score, die besten werden behalten. Pro bestem Wert werden alle anderen Detektionen in einem Umkreis von rund 1,4 m entfernt, da die Annahme ist, dass diese denselben Pylon beschreiben und der höchste Score bereits die beste Detektion darstellt.

4.2. BEV-Pillar

4.2.1. Architektur

CenterPoint [19] ist ursprünglich für autoähnliche Objekte im Bereich von mehreren zehn Metern entwickelt worden. Pylone sind mit 22,8 cm Basisdurchmesser deutlich kleiner, stehen direkt auf dem Boden, und der Erkennungsbereich ist auf 24 m begrenzt. Aus diesem Größen- und Distanzunterschied ergeben sich die im folgenden Abschnitt beschriebenen Anpassungen, die auch das Voxel-Modell (Abschnitt 4.3) übernimmt.

Den Überblick über den gesamten Verarbeitungsablauf zeigt Abbildung A.1 im Anhang.

Ausgehend von der in Abschnitt 4.1 beschriebenen gemeinsamen Eingabe wird der Bereich 24 m vor dem Auto und jeweils 12 m zu beiden Seiten vom Auto in ein 192×192 Gitternetz aufgeteilt. Alle Punkte, die in eine Gitternetzzelle fallen, werden in einen Pillar zusammengefasst. Die Höhe (z) wird hier ignoriert und durch die Zusammenfassung in eine Säule stehen die echten Höhenwerte für Backbone, Deconv nicht zur Verfügung. Im Anschluss wird ein punktweises Multi-Layer-Perceptron (MLP) mit anschließendem Max-Pooling verwendet [30], bezeichnet als PillarFeatureNet, es entsteht ein 64-dimensionaler Vektor pro Pillar. Danach werden die Vektoren mit PointPillarsScatter zurück ins Gitter projiziert und es entsteht die BEV-Feature-Map.

Backbone, Neck und Head entsprechen anschließend exakt den in Abschnitt 4.1 beschriebenen gemeinsamen Komponenten. Abbildung 4.1 zeigt die daraus resultierenden finalen Detektionen.

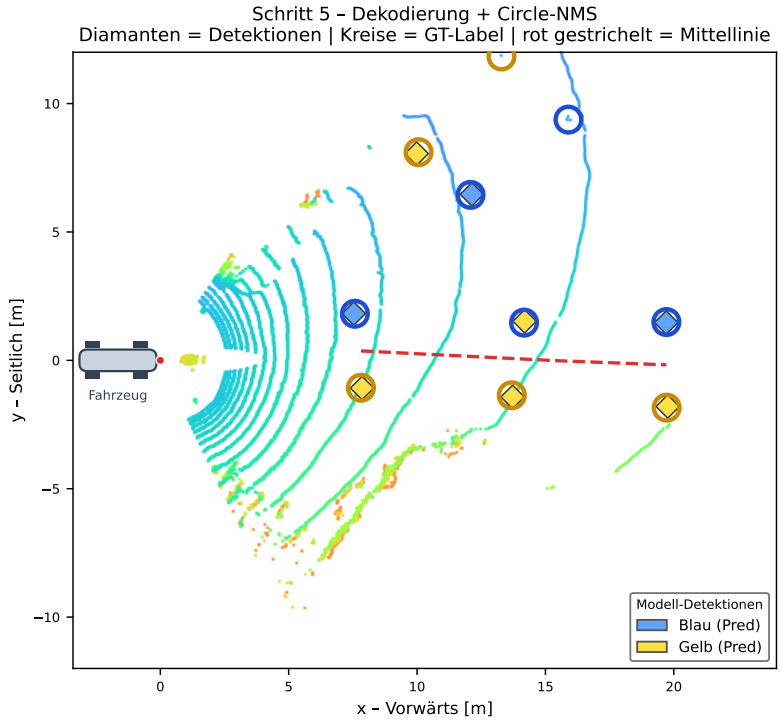

Abbildung 4.1.: Finale Pylondetektionen nach der Dekodierung (Diamanten, farbig nach Klasse) auf der BEV-Karte gegen GT-Labels (Kreise).

Die Datentransformation an einem konkreten realen Beispielframe über alle Schritte hinweg ist in Abbildung A.2 im Anhang zusammengefasst.

4.2.2. Anpassungen für die Pylonerkennung

Heatmapauflösung: Der Skalierungsfaktor zwischen Gitternetz und Heatmap wurde auf 4 gesetzt, damit hat die Heatmap eine Auflösung von 48×48 Zellen und jede Zelle entspricht 0,5 m. Mit dem ursprünglichen Faktor 8 ergab sich eine 24×24 Heatmap, jede Zelle entsprach dort 1 m, was zu einem mittleren Lokalisierungsfehler von 0,5 bis 1,0 m führte. Durch den Faktor 4 in Kombination mit Gauß-Regression konnte dieser auf 6,1 cm reduziert werden. Die Gauß-Regression platziert um jeden GT-Pylon eine zweidimensionale Gauß-Glocke in der Heatmap, sodass das Netz nicht nur lernt, ob ein Pylon vorhanden ist, sondern auch, mit welcher Präzision er innerhalb einer Zelle liegt.
Klassengewichtung: Im Training wird der Verlust für gelbe Pylone doppelt so stark gewichtet wie für blaue. Auswertungen des Trainings zeigten wiederholt eine systematisch schwächere Erkennung gelber Pylone gegenüber blauen, obwohl beide Klassen im Datensatz ausgeglichen vertreten sind. Durch die stärkere Gewichtung von Gelb im Loss wird dieser Bias korrigiert.
Distanzgewichtung: Der Verlust für weit entfernte Boxen wurde mit dem Faktor 0,7 gewichtet, da die Regression für große Abstände schwieriger ist. Eine Analyse von 336 eigenen Fahraufnahmen zeigte mit dem ursprünglichen Wert von 0,4 einen Recall von nur 0,03 bis 0,14 im Bereich von 16 bis 20 m und nahezu 0 jenseits von 20 m, obwohl real über die Hälfte der Pylone weiter als 16 m entfernt liegt. Nach einem Retraining mit dem angepassten Wert von 0,7 sowie begleitenden Anpassungen an der Trainingsdaten-Gewichtung stieg der Recall im Bereich 16 bis 24 m auf 0,665 (siehe Tabelle 4.1).
Erkennungsschwellwert: Ein Heatmapwert muss den Schwellwert von 0,4 überschreiten, damit eine Detektion ausgegeben wird. Der Wert liegt höher als beim Voxel-Modell (0,35, siehe Abschnitt 4.3), weil das Pillar-Modell auf dem Validierungsdatensatz zu mehr unsicheren, potenziell falschen Detektionen neigte. Der höhere Schwellwert filtert diese unsicheren Detektionen heraus und schlägt sich damit im Endergebnis nieder. Die Precision des Pillar-Modells liegt mit 0,928 über der des Voxel-Modells mit 0,867 (Tabelle 4.1 und Tabelle 4.2).
NMS-Radius: Circle-NMS entfernt alle Detektionen im Umkreis von rund 1,4 m um den jeweils besten Treffer. Der Radius ist bewusst deutlich größer als der erreichte mittlere Lokalisierungsfehler von 6,1 cm gewählt, damit auch ungenau platzierte Mehrfachdetektionen desselben Pylons zuverlässig zusammengeführt werden. Circle-NMS arbeitet dabei getrennt je Farbklasse, ein blauer Pylon kann eine gelbe Detektion also nie unterdrücken.

4.2.3. Training und Ergebnisse

Das Training basiert auf dem gemeinsamen Setup aus Abschnitt 2.3. Optimizer und Lernraten-Schedule folgen CenterPoint [19]. Zusätzlich wurden EMA und Warmup aus der öffentlichen CenterPoint-Referenzimplementierung übernommen. Das exponentiell gleitende Mittel (EMA) führt einen zweiten, über die Epochen geglätteten Satz von Gewichten mit und dämpft damit die Schwankungen einzelner Epochen, die bei einem kleinen Trainingsdatensatz besonders stark ausfallen. Das Warmup erhöht die Lernrate über die ersten Epochen langsam auf ihren Zielwert, statt sofort mit voller Lernrate zu starten, was zu Beginn des Trainings zu großen, destabilisierenden Gewichtsänderungen führen kann. Konfiguriert sind maximal 300 Epochen, in der Praxis stoppt das Training jedoch regelmäßig um Epoche 80, da Early Stopping das Training beendet, sobald sich der WF1-Score auf dem Validierungsdatensatz über 30 Epochen nicht mehr signifikant verbessert hat. Die Geduld von 30 Epochen liegt deutlich über der Länge der beobachteten Plateaus, in denen sich der WF1-Score vorübergehend nicht verbessert, bevor er erneut ansteigt.

In frühen Trainingsläufen blieb der Recall für blaue Pylone über sämtliche Epochen bei 0, während gelbe Pylone normal erkannt wurden. Ursache war ein invertiertes Label-Mapping zwischen den verwendeten Datenquellen. In einem Teil der Label-Dateien war Gelb als Klasse 0 und Blau als Klasse 1 codiert, im anderen genau umgekehrt. Nach dem Angleichen der Label-Konvention stieg der Blau-Recall auf das erwartete Niveau.

Im Verlauf des Trainings stellte sich außerdem heraus, dass FP16-Inferenz auf dem J4012 zu NaN- und Inf-Werten in etwa 10 % der Detektionen führte. FP16 steht für 16-Bit-Gleitkommazahlen und benötigt weniger Speicher und Rechenzeit als FP32 mit 32 Bit, verliert dafür aber an numerischer Stabilität. Da dieses Problem ausschließlich auf der Zielhardware auftrat und nicht im Training selbst, wurde das Modell auf FP32 umgestellt.

Ob die vollständige Umstellung auf FP32 der einzig gangbare Weg war, lässt sich nicht abschließend beantworten. Theoretisch wäre auch eine gemischte Präzision denkbar gewesen, bei der nur die numerisch instabilen Schichten, etwa die Sigmoid- und Exponentialfunktionen der Heatmap-Berechnung, gezielt auf FP32 gezwungen werden, während der Rest des Netzes in FP16 verbleibt. Das hätte jedoch vorausgesetzt, die genaue Operation zu isolieren, die den Overflow verursacht, was einen deutlich höheren Debugging-Aufwand bedeutet hätte als die pauschale Umstellung. Da die TensorRT-Engine bereits mit vollständigem FP32 nur 8,3 ms benötigt und damit weit unter dem 100 ms-Budget liegt, hätte der potenzielle Geschwindigkeitsgewinn von FP16 hier ohnehin keinen praktischen Nutzen gehabt. FP32 bietet gegenüber FP16 deutlich höhere numerische Stabilität und ist einfacher vorherzusagen, kostet dafür aber die doppelte Speichermenge pro Wert und verzichtet auf die für FP16

optimierten Tensor-Core-Pfade der Jetson-Hardware. Bei bereits ausreichender Latenzreserve überwog hier der Stabilitätsvorteil klar gegenüber diesen Nachteilen.

Das Training lief insgesamt etwa 4 Stunden.

Die Ergebnisse zum Training des BEV-Pillar-Modells sind in Tabelle 4.1 dargestellt. Mit einem WF1-Score von 0,888 erkennt das Modell Pylone gut. Auffällig ist die hohe Precision von 0,928 bei einem Recall von 0,853, das Modell gibt also lieber keine Detektion aus als eine falsche. Das ist für die Streckenplanung die bessere Wahl, da ein einzelner übersehener Pylon durch die SLAM-gestützte Kartierung aus Folgeframes noch ergänzt werden kann (siehe Kapitel 1), während eine falsch positionierte Detektion die Streckenkarte direkt verfälscht und nicht ohne Weiteres nachträglich korrigiert werden kann. Der mittlere Lokalisierungsfehler von 6,1 cm zeigt, dass erkannte Pylone sehr genau lokalisiert werden.
Der Recall steigt von 0 bis 8 m mit 0,837 auf 0,869 im Entfernungsbereich von 8 bis 16 m an, im Fernbereich von 16 bis 24 m fällt er auf 0,665, was an der geringer werdenden Punktdichte liegt.
Da beim Pillar-Encoding die Höhe kollabiert wird, kann das Modell die Pylonstreifen nicht räumlich auflösen, was die Farbgenauigkeit von 0,721 erklärt.
Die Latenz liegt bei 32 ms im PyTorch-Betrieb und 8,3 ms mit TensorRT auf dem J4012, damit liegt das Modell weit unter dem 100 ms Budget für den 10-Hz-Betrieb.
Diese Ergebnisse bestätigen die aus der Literatur erwartete Charakteristik von PointPillars [16], also die geringste Latenz aller vier Modelle bei gleichzeitig höchster Genauigkeit (WF1, MAE) stützt die Einordnung als recheneffizienteste, aber höheninformationslose Architektur (siehe Kapitel 2), was sich auch in der niedrigen Farbgenauigkeit von 0,721 widerspiegelt.

Tabelle 4.1.: Evaluationsergebnisse des BEV-Pillar-Modells (J4012, FP32). Metrikdefinitionen siehe Abschnitt 2.2.

WF1	wRecall	wPrec	ColorAcc	MAE	R@0–8 m	R@8–16 m	R@16–24 m	torch / TRT	Trainingszeit
0,888	0,853	0,928	0,721	6,1 cm	0,837	0,869	0,665	32 ms / 8,3 ms	4 h

4.3. BEV-Voxel

4.3.1. Architektur

Den Überblick über den gesamten Verarbeitungsablauf zeigt Abbildung A.3 im Anhang.

Ausgehend von der in Abschnitt 4.1 beschriebenen gemeinsamen Eingabe wird wie beim Pillar-Modell der Detektionsbereich in ein 192×192 Gitternetz aufgeteilt, jedoch diesmal zusätzlich in der Höhe in 12 Schichten zu je 0,125 Metern, das ergibt ein dreidimensionales Gitter mit $192 \times 192 \times 12$ Zellen. Die Schichtdicke entspricht bewusst der Kantenlänge der BEV-Zellen, sodass die Voxel würfelförmig sind und in allen drei Achsen dieselbe räumliche Auflösung besitzen. Zwölf Schichten decken einen Höhenbereich von 1,5 m ab, der den Boden und die 32,5 cm hohen blauen und gelben Pylone vollständig einschließt und darüber liegende Strukturen ausschließt. Da die Punktwolke zuvor auf die Bodenebene normiert wird (Abschnitt 4.5), verschiebt sich dieser Bereich bei Steigungen und Neigungen des Untergrunds mit. Alle Punkte, die in eine Zelle fallen, werden zu einem Mittelwert zusammengefasst. Im Gegensatz zum Pillar-Modell wird hier kein gelerntes Netz pro Zelle angewendet.

Dieses dreidimensionale Gitter wird über mehrere 3D-Faltungen [17, 27] auf eine zweidimensionale BEV-Feature-Map reduziert, dabei fließt die Höheninformation in die Features ein. Ursprünglich sollte hier das Standard-Framework spconv verwendet werden. In durchgeführten Exportversuchen ließ sich dieses auf dem J4012 jedoch nicht als TensorRT-Engine exportieren, daher kommt ein Encoder auf Basis regulärer dichter 3D-Faltungen zum Einsatz (DenseEncoder). Eine Sparse-Convolution rechnet nur auf den tatsächlich belegten Zellen des Gitters und überspringt die leeren, die bei einer Punktwolke den weitaus größten Teil ausmachen. Eine dichte Faltung rechnet dagegen auf allen Zellen, auch auf den leeren. Sie kostet dadurch deutlich mehr Rechenzeit, kommt aber mit Standardoperationen aus, die sich ohne zusätzliche Bibliothek als TensorRT-Engine exportieren lassen. Backbone, Neck und Head entsprechen anschließend exakt den in Abschnitt 4.1 beschriebenen gemeinsamen Komponenten. Abbildung 4.2 zeigt die daraus resultierenden finalen Detektionen.

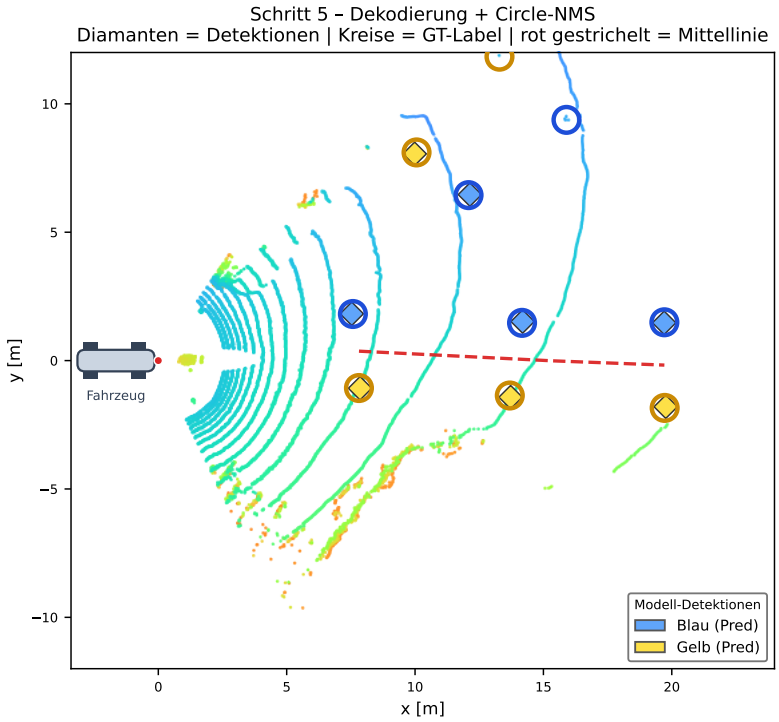

Abbildung 4.2.: Finale Detektionen des Voxel-Modells (Diamanten) gegen GT-Labels (Kreise) auf der BEV-Karte.

Die Datentransformation an einem konkreten realen Beispielframe über alle Schritte hinweg ist in Abbildung A.4 im Anhang zusammengefasst.

4.3.2. Anpassungen für die Pylonerkennung

Backbone, Neck und Head sind mit dem Pillar-Modell identisch, entsprechend übernimmt dieses Modell auch dessen Anpassungen aus Abschnitt 4.2 für Heatmapauflösung, Klassengewichtung und Distanzgewichtung. Eigenständige Anpassungen betreffen ausschließlich den Encoder und den davon abhängigen Erkennungsschwellwert.

Encoder: In durchgeführten Exportversuchen ließen sich Sparse-Convolution-Bibliotheken auf dem J4012 nicht als TRT-Engine exportieren. In der Community wird dafür häufig empfohlen, auf eine ältere Version der Systemsoftware zurückzugehen, unter der sich spconv zuverlässiger exportieren lässt. Diese Option schied hier aus, da das separate Kameramodell zur nachgelagerten Farbkorrektur der Pylone die neueste Version voraussetzt und ein Downgrade diese andere Komponente der Pipeline unbrauchbar gemacht hätte. Stattdessen arbeitet das Modell mit einem

Encoder aus regulären dichten Faltungen, der sich auf der aktuellen Systemsoftware als einzelne TRT-Engine exportieren lässt. Eine erste Auswertung zeigte, dass die gefundenen Pylone bereits sehr genau lokalisiert waren, praktisch alle korrekten Detektionen lagen weniger als 0,25 m vom tatsächlichen Zentrum entfernt. Die Positionsgenauigkeit war somit nicht weiter optimierbar. Der begrenzende Faktor war stattdessen die Precision, das Modell gab also zu viele Falschdetektionen aus. Deshalb wurde nicht der CenterHead verändert, der die Position verfeinert, sondern die Kanalzahl des DenseEncoders von 16 auf 32 Basiskanäle erhöht. Ziel war, die vorhandene Höheninformation stärker für die Unterscheidung zwischen echten Pylonen und Störstrukturen wie Boden oder Rauschen nutzbar zu machen.
Erkennungsschwellwert: Ein Heatmapwert muss den Schwellwert von 0,35 überschreiten, damit eine Detektion ausgegeben wird. Der Wert liegt niedriger als beim Pillar-Modell (0,40, siehe Abschnitt 4.2). Die höhere Encoder-Kapazität und die erhaltene Höheninformation erlauben es dem Voxel-Modell, Falschdetektionen bereits vor der Schwellwert-Filterung sicherer zu verwerfen, sodass ein niedrigerer Schwellwert ausreicht.

4.3.3. Training und Ergebnisse

Das Training folgt dem gleichen Ansatz wie in Abschnitt 4.2. Statt eines festen Lernraten-Schedules wird die Lernrate hier bei ausbleibender Verbesserung des WF1-Scores halbiert, da der 3D-Encoder mehr Schichten hat und der Konvergenzprozess schwieriger vorherzusagen ist. Konfiguriert sind maximal 500 Epochen mit Early Stopping, also mehr als die 300 beim Pillar-Modell, da der tiefere Encoder mehr Epochen bis zur Konvergenz benötigt. Das Training lief insgesamt etwa 5 Stunden, gegenüber 4 Stunden beim Pillar-Modell, was den höheren Rechenaufwand der zusätzlichen Encoder-Kanäle und der dritten Raumdimension widerspiegelt.

Die Ergebnisse sind in Tabelle 4.2 dargestellt. Mit einem WF1-Score von 0,804 erkennt das Modell Pylone gut. Die Farbgenauigkeit beträgt 0,856 und ist damit die höchste aller vier Modelle. Anders als beim Pillar-Modell (Abschnitt 4.2) kollabiert die Höheninformation hier nicht, wodurch das Modell die vertikalen Pylonstreifen räumlich auflösen kann. Dieses Ergebnis bestätigt die aus der Literatur erwartete Charakteristik von VoxelNet [17] (siehe Kapitel 2), also mehr räumliche Information bei höherem Rechenaufwand.
Der Recall im Entfernungsbereich von 8 bis 16 m liegt mit 0,678 unter dem Nahbereich mit 0,825, im Fernbereich von 16 bis 24 m beträgt er 0,671.
Im PyTorch-Betrieb liegt die Latenz bei 297 ms, da der DenseEncoder alle $192 \times 192 \times 12$ Zellen verarbeitet, auch wenn die meisten leer sind. Mit der exportierten TensorRT-Engine beträgt die Latenz 70,6 ms auf dem J4012 und liegt damit unter dem Budget von 100 ms für den 10-Hz-Betrieb. Der im Vergleich zum Pillar-Modell

(8,3 ms) deutlich höhere Wert bestätigt ebenfalls die literaturbekannte Charakteristik. Die erhaltene dritte Raumdimension kostet Rechenzeit.

Tabelle 4.2.: Evaluationsergebnisse des BEV-Voxel-Modells (J4012, FP32). Metrikdefinitionen siehe Abschnitt 2.2.

WF1	wRecall	wPrec	ColorAcc	MAE	R@0–8 m	R@8–16 m	R@16–24 m	torch / TRT	Trainingszeit
0,804	0,752	0,867	0,856	6,4 cm	0,825	0,678	0,671	297 ms / 70,6 ms	5 h

4.4. Range-Image

4.4.1. Architektur

RangeDet [18] und LaserNet [28] sind für vollständige 360°-Fahrszenen mit autoähnlichen Objekten entwickelt worden. In dieser Arbeit ist der Sensor auf einen 120°-Vorwärtsbereich begrenzt und das Zielobjekt ist ein 22,8 cm kleiner Pylon, entsprechend übernimmt das Modell die Grundarchitektur, muss die Projektion und Nachverarbeitung jedoch an diese beiden Unterschiede anpassen. Anders als Pillar und Voxel (Abschnitt 4.1) verarbeitet dieses Modell die Punktwolke nicht aus der Vogelperspektive, sondern aus Sensorsicht, weshalb Backbone, Neck und Head hier in eigener, anisotroper Form beschrieben werden.

Den Überblick über den gesamten Verarbeitungsablauf zeigt Abbildung A.5 im Anhang. Die folgenden Abschnitte erläutern die fünf nummerierten Schritte im Detail.

LiDAR-Scan: Der Input ist identisch zu den vorherigen Modellen (Abschnitt 4.1). Nur die rund 683 Spalten im 120°-Vorwärtsfenster sind mit Messwerten belegt, der restliche Teil des Arrays ist leer. Abbildung 4.3 zeigt das vollständige Array. Von den 2048 Spalten, die einem vollen 360°-Umlauf entsprechen, trägt nur der grün markierte Ausschnitt Messwerte.

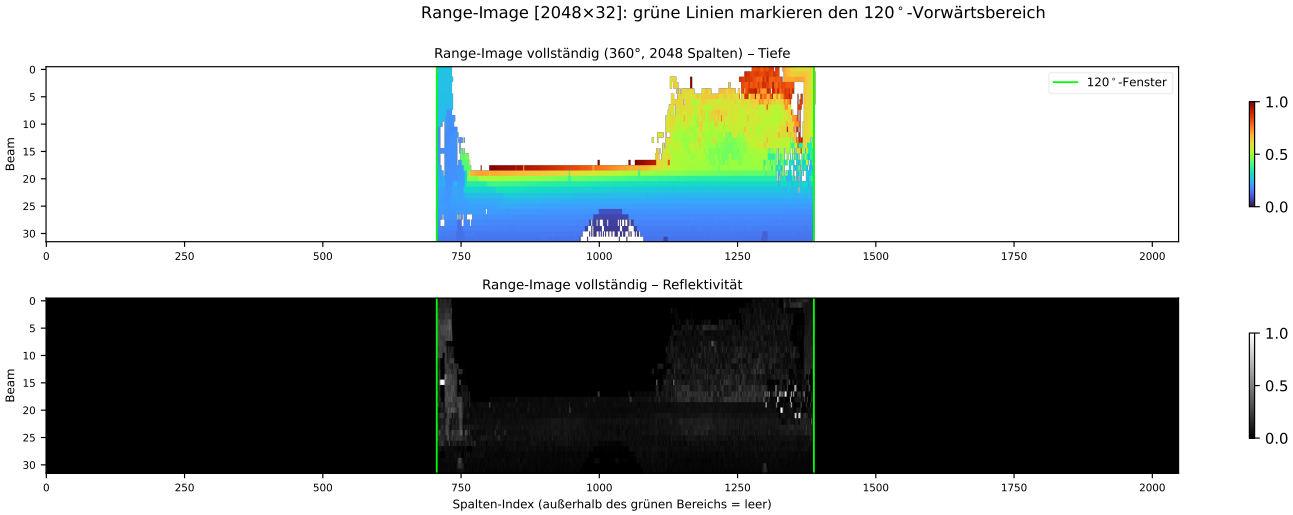

Abbildung 4.3.: Vollständiges Range-Image (2048×32) für Tiefe (oben) und Reflektivität (unten). Die grünen Linien begrenzen das 120°-Vorwärtsfenster. Außerhalb dieses Ausschnitts bleibt das Array leer, es werden also nur rund 683 der 2048 Spalten genutzt.

1. Range-Image-Projektion: Jeder Punkt wird anhand seines Elevationswinkels einer Zeile zugeordnet, der Elevationswinkel ist der vertikale Neigungswinkel des Laserstrahls, beim Ouster OS1-32-U3 liegen diese zwischen $-22{,}5°$ und $+22{,}5°$, was einem vertikalen Sichtfeld von 45° entspricht [8]. Die Spalte ergibt sich aus dem Azimutwinkel, also der horizontalen Drehposition des Sensors im aktuellen Scan. Das ergibt ein 2048×32 großes Bild, das sich grundlegend vom BEV-Bild unterscheidet, da es nicht von oben auf die Szene schaut, sondern diese so abbildet, wie der Sensor sie abtastet. Pro Pixel werden fünf Kanäle befüllt, die Koordinaten x, y und z des Punktes, der Reflektivitätswert sowie die Tiefe. Die Tiefe ist der euklidische Abstand des Punktes vom Sensor und wird unverändert in Metern eingetragen.

2. SECOND-Backbone [27]: Der Backbone verarbeitet das Range-Image mit anisotropen Faltungsstufen, die Spalten werden schrittweise von 2048 auf 512, 256 und 128 komprimiert, die 32 Zeilen bleiben dabei erhalten. Würde man wie bei einem Standard-CNN auch die Zeilen halbieren, hätte ein Pylon ab wenigen Metern Abstand nur noch einen einzigen Pixel in der vertikalen Dimension, was jede verlässliche Merkmalserkennung verhindert.

3. RangeImageFPN [18]: Die drei Backbone-Ausgaben werden auf eine gemeinsame Größe gebracht und konkateniert, das ergibt eine Feature-Map von 256×32 Zellen mit je 384 Kanälen, also $256 \times 32 \times 384$.

4. RangeImageCenterHead [18, 19]: Pro Farbklasse wird eine Heatmap über

das Range-Image berechnet, zusammen mit Regressionsausgaben für Tiefe, Breite und Höhe des Pylons. Die Tiefe wird dabei nicht in Metern vorhergesagt, sondern logarithmisch. Ohne diese Umrechnung würde ein Fehler von einem Meter bei einem 24 m entfernten Pylon im Loss genauso schwer wiegen wie bei einem 3 m entfernten, obwohl er dort relativ zur Entfernung achtmal größer ist. Die logarithmische Zielgröße bewertet stattdessen den relativen Fehler, sodass Pylone aus allen Entfernungen mit vergleichbarem Gewicht ins Training eingehen. Die Umrechnung ist umkehrbar und wird bei der Dekodierung exakt zurückgenommen, es geht dabei keine Information verloren.

5. Dekodierung und NMS: Die erkannten Pixel werden mit ihrer vorhergesagten Tiefe zurück in dreidimensionale Koordinaten im LiDAR-Frame umgerechnet. Anschließend werden Detektionen außerhalb des konfigurierten 120°-Vorwärtsfensters verworfen. Beim NMS wird ein Mindestabstand von 0,9 m verwendet, da bei großen Abständen der Winkelabstand zweier benachbarter Spalten im Range-Image mehreren Dezimetern entspricht, was dazu führt, dass der gleiche Pylon in zwei benachbarten Spalten erkannt wird und ohne ausreichend großen Mindestabstand beide Detektionen erhalten blieben.

Die Datentransformation an einem konkreten realen Beispielframe über alle fünf Schritte hinweg ist in Abbildung A.6 im Anhang zusammengefasst.

4.4.2. Anpassungen für die Pylonerkennung

Beam-Tabelle: Für die Projektion wird eine Tabelle benötigt, die jeden der 32 Laserstrahlen einem Elevationswinkel zuordnet. Die theoretische Tabelle aus dem Ouster-Datenblatt passt nicht zu den echten Daten, da die Punkte bodenrelativ transformiert werden und die Winkel sich dadurch verschieben. Die Tabelle wurde daher mittels k-Means-Clustering neu berechnet, einem Verfahren, das ähnliche Messwerte automatisch zu Gruppen zusammenfasst. Die tatsächlich gemessenen Elevationswinkel aus den Daten wurden zu 32 Gruppen geclustert, deren Mittelwerte die neue, datengetriebene Beam-Tabelle bilden. Damit stieg die Befüllungsrate des Range-Images von 3 % auf 8,7 %.
Erkennungsschwellwert: Ein Heatmapwert muss den Schwellwert von 0,25 überschreiten, damit eine Detektion ausgegeben wird. Der Wert liegt deutlich niedriger als bei Pillar (0,40) und Voxel (0,35, siehe Abschnitt 4.2 und Abschnitt 4.3), da die Heatmap-Scores des Range-Image-Modells systematisch niedriger ausfallen. Wie die geometrische Herleitung im Ergebnisse-Abschnitt zeigt, ist ein Großteil der Pylone jenseits weniger Meter kleiner als eine Gitterspalte der Heatmap, sodass selbst korrekte Detektionen kein scharfes, hoch-konfidentes Aktivierungsmuster erzeugen.

4.4.3. Training und Ergebnisse

Das Training folgt dem gleichen Ansatz wie in Abschnitt 4.2. Early Stopping ist auf eine Geduld von 60 Epochen gesetzt, statt der 30 Epochen bei Pillar (Abschnitt 4.2), da das Training durch die komplexere, mehrteilige Loss-Funktion aus Heatmap- und Tiefenregressions-Termen langsamer konvergiert. Das Training lief insgesamt etwa 3 Stunden und damit kürzer als bei Pillar oder Voxel, obwohl es über mehr Epochen lief. Ausschlaggebend sind die Kosten eines einzelnen Trainingsschritts. Das Range-Image ist mit 2048×32 Werten eine kleinere und dichter belegte Eingabe als ein BEV-Gitter mit 192×192 Zellen, das zum größten Teil leer bleibt, und der Verarbeitung ist kein eigener Encoder-Schritt vorgeschaltet.

Die Ergebnisse sind in Tabelle 4.3 dargestellt. Mit einem WF1-Score von 0,238 und einem Recall von 0,228 erkennt das Modell deutlich weniger Pylone als die BEV-Varianten. Der Grund liegt in einer geometrischen Grenze der Eingaberepräsentation. Zwar tastet der OS1-32-U3 mit $360°/2048 = 0{,}176°$ pro Spalte ab [8], für die Erkennbarkeit eines Pylons ist aber nicht diese native Sensorauflösung entscheidend, sondern die Auflösung der RangeImageFPN-Ausgabe, auf der die Heatmap berechnet wird. Diese beträgt 256 Spalten über die vollen 360° (Abschnitt 4.4), also $360°/256 = 1{,}406°$ pro Spalte.

Ein Pylon hat nach Regelwerk einen Basisdurchmesser von 228 mm [12], für die folgende Rechnung auf 0,23 m aufgerundet, also einen Radius $r = 0{,}115\,\text{m}$. Aus einer Entfernung R erscheint er unter dem vollen Sichtwinkel $\theta(R) = 2 \cdot \arctan(r/R)$. Die Strecke vom Sensor zum Pylonzentrum (Länge R) und der Pylonradius r senkrecht dazu bilden dabei ein rechtwinkliges Dreieck, in dem r die Gegenkathete und R die Ankathete des Winkels zwischen der Sichtlinie zum Zentrum und der Sichtlinie zum Rand des Pylons ist. Der Tangens dieses Winkels ist per Definition das Verhältnis r/R, der Winkel selbst also $\arctan(r/R)$. Dieser Winkel tritt symmetrisch zu beiden Seiten der Mittellinie auf, zum linken wie zum rechten Rand des Pylons, weshalb die Verdopplung den vollen Sichtwinkel über den gesamten Pylondurchmesser ergibt. Setzt man $\theta(R)$ gleich der Spaltenbreite von 1,406° und löst nach R auf, ergibt sich der folgende Grenzabstand.

$$
\begin{aligned}
2 \cdot \arctan(r/R) &= 1{,}406° \\
\arctan(r/R) &= 0{,}703° \\
r/R &= \tan(0{,}703°) = 0{,}01227 \\
R &= r/0{,}01227 = 0{,}115\,\text{m}/0{,}01227 \approx 9{,}4\,\text{m}
\end{aligned}
$$

Bei diesem Abstand füllt der Pylon gerade noch genau eine Gitterspalte der Heatmap. Für alle größeren Abstände ist der Pylon kleiner als ein Pixel. Ist ein Objekt kleiner als ein Pixel, hängt sein Beitrag zur Pixelaktivierung von der genauen Position innerhalb des Pixels ab, die von Scan zu Scan variiert, das Netz sieht damit

kein konsistentes Muster und kann keinen zuverlässigen Detektor lernen. Das zeigt sich im Recall, der im Nahbereich von 0 bis 8 m bei 0,380 liegt und im Entfernungsbereich von 8 bis 16 m auf 0,076 einbricht.
Die Farbgenauigkeit beträgt 0,894 und ist damit die höchste aller vier Modelle. Wenn das Modell einen Pylon erkennt, ist die Farbe fast immer korrekt, da das Range-Image die Reflektivität räumlich pro Strahl auflöst.
Die Latenz liegt bei 30,5 ms im PyTorch-Betrieb und 30,0 ms mit TensorRT auf dem J4012, da die Projektion und Dekodierung als Python-Code außerhalb der TRT-Engine laufen und der Geschwindigkeitsgewinn der Engine damit nicht zum Tragen kommt.
Diese Ergebnisse bestätigen die in Kapitel 2 aus der Literatur beschriebene Charakteristik von RangeDet [18]. Die kompakte, sensornahe Darstellung ermöglicht zwar eine feine, strahlgenaue Farbauflösung, stößt bei kleinen, entfernten Objekten jedoch an genau die geometrische Grenze der Winkelauflösung, die oben hergeleitet wurde und sich im starken Recall-Einbruch ab 8 m niederschlägt.

Damit die geometrische Herleitung nicht als alleinige Erklärung stehen bleibt, wurde der Befund zusätzlich abgesichert.

Kontrollexperiment. Der stärkste Beleg dafür, dass diese Grenze in der Eingaberepräsentation liegt und nicht in den Daten, im Training oder in den GT-Labeln, ergibt sich aus dem Vergleich mit den BEV-Modellen. Pillar und Voxel wurden auf demselben Datensatz, mit denselben GT-Boxen und derselben Aufteilung in Trainings- und Validierungsdaten trainiert wie das Range-Image-Modell und erreichen ein Vielfaches von dessen WF1-Score (Kapitel 5). Die einzige geänderte Größe ist die Art, wie dem Netz die Punktwolke präsentiert wird. Ein Problem in den Daten, in den GT-Labeln oder in der Konvergenz müsste sich bei allen drei Modellen gleichermaßen auswirken. Da nur das Range-Image-Modell einbricht, bleibt die Eingaberepräsentation als Erklärung.

Auch die Literatur stützt diesen Befund. RangeDet [18] räumt trotz seines Titels selbst einen deutlichen Rückstand der Range-View-Verfahren gegenüber BEV-Verfahren ein und wird auf Objekten von der Größe eines Fahrzeugs oder Fußgängers evaluiert, die auch auf größere Entfernung noch mehrere Bildzellen und mehrere Laserstrahlen belegen. Die einzige gefundene Arbeit, die Range-View-Detektion auf pylonartigen Objekten erfolgreich zeigt [31], arbeitet auf einem Datensatz mit effektiv 64 statt 32 Laserstrahlen und benennt die Erhöhung der Auflösung als ihren größten Wirkhebel. Die Auflösung wirkt dabei in beide Bildrichtungen, und die obige Rechnung erfasst nur eine davon. Horizontal setzt die Spaltenbreite die hergeleitete Grenze von 9,4 m. Vertikal entscheidet der Abstand der Laserstrahlen darüber, ob ein Pylon überhaupt von mehr als einem Strahl getroffen wird, und erst mehrere Treffer aus verschiedenen Höhen ergeben ein auswertbares Muster im Range-Image (Abschnitt 2.1). Beim OS1-32 liegen die 32 Strahlen $45°/31 = 1{,}45°$ auseinander,

ein 32,5 cm hoher Pylon wird damit im Mittel nur bis etwa 6,4 m von zwei Strahlen erfasst. Bei doppelter Strahlenzahl verschiebt sich diese Schwelle auf etwa 13 m. Die zitierte Arbeit verfügt genau über diese höhere vertikale Auflösung, ihr Ergebnis stützt den Befund dieser Arbeit.

Tabelle 4.3.: Evaluationsergebnisse des Range-Image-Modells (J4012, FP32). Metrikdefinitionen siehe Abschnitt 2.2.

WF1	wRecall	wPrec	ColorAcc	MAE	R@0–8 m	R@8–16 m	R@16–24 m	torch / TRT	Trainingszeit
0,238	0,228	0,249	0,894	22,3 cm	0,380	0,076	0,026	30,5 ms / 30,0 ms	3 h

4.5. Raw-Point

4.5.1. Architektur

IA-SSD [13] ist ursprünglich für Fahrzeuge auf KITTI-Straßenszenen entwickelt worden, die mit rund 4 m Länge weit größer sind als ein 22,8 cm breiter Pylon. Entsprechend mussten vor allem die Suchradien der Set-Abstraction-Stufen und die Nachverarbeitung an die deutlich kleinere und dünner besetzte Zielstruktur angepasst werden. Dieses Modell teilt sich keine Komponenten mit den drei vorherigen Abschnitten (Abschnitt 4.1), da es ohne Raster oder Panoramabild arbeitet.

Den Überblick über den gesamten Verarbeitungsablauf zeigt Abbildung A.7 im Anhang. Die folgenden Abschnitte erläutern die vier nummerierten Schritte im Detail.

LiDAR-Scan: Die Eingabe ist dieselbe strukturierte Punktwolke wie bei den anderen Modellen (Abschnitt 4.1), die für die Verarbeitung jedoch als flache Liste übergeben wird.
1. Set-Abstraction-Stufen [14]: IA-SSD [13] löst das Problem, unstrukturierte Punktmengen ohne Raster zu verarbeiten, auf eine Weise, die sich gut erklären lässt. Die Punkte werden in drei Schritten schrittweise verdichtet, jeder Schritt produziert eine kleinere aber informationsreichere Menge.

In jedem Schritt passiert dasselbe, eine Teilmenge von Saatpunkten wird so ausgewählt, dass je zwei möglichst weit voneinander entfernt sind, kein Bereich bekommt dabei zu viele Saatpunkte und kein Bereich zu wenige. Jeder Saatpunkt zieht dann seine Nachbarn im Umkreis eines festen Radius an, verarbeitet sie durch geteilte Schichten und fasst sie über ein Maximum zusammen. Wie viele Nachbarn dabei gefunden werden spielt keine Rolle, das Ergebnis ist immer ein gleich großer Merkmalsvektor. Die erste Stufe reduziert so auf 4096 Saatpunkte mit Suchradien von 0,1 m, 0,2 m und 0,4 m.

Was die zweite Stufe auszeichnet, ist, dass sie nicht mehr gleichmäßig auswählt. An diesem Punkt hat das Netz bereits gelernt, welche Regionen wahrscheinlich Pylone enthalten und wählt die nächsten 512 Saatpunkte entsprechend bevorzugt dort aus. Die dritte Stufe nimmt größere Radien von 0,5 m, 1,0 m und 1,5 m und gibt 256 Punkte aus, die sie zu gleichen Teilen aus einer geometrischen und einer merkmalsbasierten Auswahl zusammensetzt, also je 128 Punkte. Aus diesen 256 Punkten wählt die nächste Stufe die 128 Saatpunkte aus, mit denen das VoteModule arbeitet.

2. VoteModule [32]: Mit den 128 Saatpunkten liegt man meistens irgendwo neben einem Pylon, selten exakt auf seinem Zentrum. Das VoteModule berechnet deshalb für jeden Saatpunkt einen Versatzvektor, der ihn auf das nächste Pylonzentrum zeigen lässt. Nach dem Verschieben aller 128 Punkte ballen sich die Saatpunkte, die zum selben Pylon gehören, zu einem dichten Cluster, während der Rest sich diffus im Raum verteilt. Genau an diesen Clustern wird dann detektiert.

3. SSD3DHead [13]: Für jeden verschobenen Saatpunkt gibt der Kopf direkt Klasse, Position, Abmessungen und Drehwinkel aus, ohne vordefinierte Referenzboxen. Klassische Detektoren lernen Abweichungen von vordefinierten Boxengrößen, hier lernt das Netz die Ausgaben vollständig aus den lokalen Merkmalen, was bei Pylonen gut funktioniert, weil alle Pylone der Formula Student genormt sind.

4. Dekodierung und NMS: Die Vorhersagen werden mit einem entfernungsabhängigen Konfidenzschwellwert gefiltert, nämlich 0,30 für den Nahbereich bis 10 m, wo die Punktdichte ausreicht und ein strenger Schwellwert Falschdetektionen begrenzt, und 0,06 im Fernbereich über 10 m, wo Scores wegen dünnerer Punktwolke systematisch niedriger ausfallen. Anschließend reduziert ein NMS die verbleibenden Treffer auf finale Detektionen, wie in Abbildung 4.4 zu sehen. Anders als bei den BEV-Modellen arbeitet es hier nicht über einen festen Umkreis, sondern über die dreidimensionale IoU zweier Boxen mit einem Schwellwert von 0,1. Das ist kein Widerspruch zum Verzicht auf die IoU bei der Bewertung in Abschnitt 2.2, denn dort wird eine Detektion gegen ein GT-Label geprüft, hier dagegen werden zwei Vorhersagen desselben Modells miteinander verglichen. Da beide dieselbe feste Pylongröße von 0,23 m Kantenlänge tragen, hängt ihre IoU allein vom Abstand ihrer Zentren ab, ein Schwellwert von 0,1 entspricht damit einem Mindestabstand von rund 19 cm.

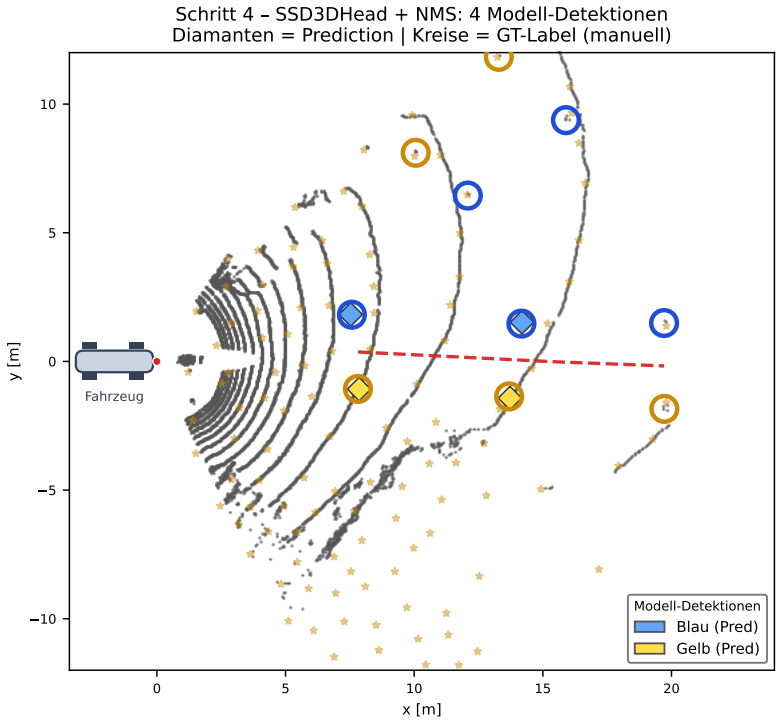

Abbildung 4.4.: Finale Detektionen (Diamanten) gegen GT-Labels (Kreise) beim Raw-Point-Modell in der BEV-Ansicht nach dem NMS.

Die Datentransformation an einem konkreten realen Beispielframe über alle vier Schritte hinweg ist in Abbildung A.8 im Anhang zusammengefasst.

4.5.2. Anpassungen für die Pylonerkennung

SA-Radien [13]: Die Standard-Radien aus dem IA-SSD-Paper [13] sind auf Fahrzeuge ausgelegt, die deutlich größer als Pylone sind, daher wurden die Suchradien der ersten Stufe auf 0,1 m, 0,2 m und 0,4 m reduziert, um die lokale Punktstruktur der Pylone aufzulösen.
Punktzahl je Set-Abstraction-Stufe: Die erste Stufe wählt 4096 Saatpunkte aus. Zunächst waren es 2048, jedoch tastete diese Stufe rein geometrisch ab und verwarf dabei weit entfernte Pylone, die nur aus ein bis zwei Punkten bestehen, bereits bevor die nachfolgende lernbasierte Auswahl sie überhaupt bewerten konnte. Das äußerte sich in einem Recall von lediglich 0,15 im Bereich von 10 bis 20 m. Die Verdopplung auf 4096 sorgt dafür, dass auch punktarme Pylone die geometrische Vorauswahl überstehen und die Vordergrund-Selektion erreichen. Die zweite Stufe gibt 512 Punkte aus, was der Eingabegröße entspricht, die die dritte Stufe erwartet.

Deren Ausgabe wird auf 128 Saatpunkte reduziert, die dem VoteModule als Grundlage dienen. Diese Zahl muss kleiner sein als die Anzahl der von der dritten Stufe gelieferten Punkte, damit die Saatpunkte eine echte Teilmenge bilden.
Detektionen je Saatpunkt: In der Standardkonfiguration gibt der Kopf für jeden Saatpunkt eine eigene Detektion je Farbklasse aus. Jeder Saatpunkt erzeugt damit an derselben Stelle eine blaue und eine gelbe Box, von denen höchstens eine richtig sein kann. Die Hälfte aller Ausgaben ist dadurch zwangsläufig falsch und die erreichbare Precision auf 0,5 begrenzt, unabhängig davon, wie genau das Modell die Positionen trifft. Umgestellt auf eine einzige Detektion je Saatpunkt, für die der Kopf die wahrscheinlichste Klasse wählt, stieg die Precision auf den in Tabelle 4.4 berichteten Wert von 0,800.
Distanzabhängiger Erkennungsschwellwert: Im Nahbereich wird ein Schwellwert von 0,30 verwendet, im Fernbereich 0,06, da das Modell für weiter entfernte Pylone niedrigere Konfidenzwerte vorhersagt und ein einheitlicher Schwellwert den Recall ab 10 m stark einschränkt. Mit einem einheitlichen, für den Nahbereich kalibrierten Schwellwert lag der Recall im Bereich 10 bis 16 m bei nur 0,19. Mit dem distanzabhängigen Schwellwert stieg er auf 0,71 bei weiterhin hoher Precision.
AlignToGroundPlane: Die z-Koordinaten werden auf die Bodenebene normiert, da der Detektionskopf absolute z-Werte für die Zentrumsvorhersage verwendet und sonst durch Schwankungen in der Montagehöhe des Sensors gestört wird. Die Normierung stellt sicher, dass die Höhenwerte in Training und Inferenz dieselbe Bezugsebene haben.

4.5.3. Training und Ergebnisse

Das Training folgt dem gleichen Ansatz wie in Abschnitt 4.2, der Vergleich mit den GT-Labels läuft hier aber anders. Welcher Saatpunkt zu welchem Pylon gehört, entscheidet der Abstand nach dem Voting. Liegt ein Saatpunkt nah genug an einem GT-Zentrum, wird er ihm zugewiesen. Für zugewiesene Saatpunkte berechnet sich dann ein zweiteiliger Loss. Der Vote-Loss misst, wie weit die verschobene Position noch vom GT-Zentrum entfernt ist, der Head-Loss, wie weit die vorhergesagte Klasse, Position, Abmessung und Ausrichtung vom GT-Label entfernt ist. Beide Teile werden gemeinsam backpropagiert, das Netz lernt also gleichzeitig besser zu voten und besser zu detektieren.

Die Lernrate wird nach einem festen Stufenplan verringert, analog zum Trainings-Setup von 3DSSD [32]. In eigenen Versuchen konvergierte der vote-basierte Detektionskopf damit zuverlässiger als mit einem adaptiven Verfahren. Vor IA-SSD wurde zunächst 3DSSD [32] eingesetzt, dessen Furthest-Point-Sampling an der Flaschenhals-Stufe bevorzugt Vordergrundpunkte verwarf. Der Wechsel auf das centerness-geführte

Sampling von IA-SSD steigerte den WF1-Score von 0,291 auf 0,403. Das Training lief insgesamt etwa 7 Stunden.

Die Ergebnisse der Raw-Point-Detektion sind in Tabelle 4.4 dargestellt. Mit einem WF1-Score von 0,788 erkennt das Modell Pylone gut, bleibt aber hinter dem Pillar-Modell zurück. Im Nahbereich von 0 bis 8 m liegt der Recall bei 0,819 und damit auf dem Niveau der BEV-Modelle, im Entfernungsbereich von 8 bis 16 m bei 0,733, im Fernbereich von 16 bis 24 m bricht er auf 0,533 ein. Die Precision von 0,800 zeigt, dass acht von zehn Detektionen einen echten Pylon treffen. Schlechter als die anderen Modelle schneidet das Modell bei der Farbgenauigkeit ab. Ein Wert von 0,715 bedeutet, dass bei etwa drei von vier erkannten Pylonen die Farbe stimmt. Dass der Wert nicht höher liegt, hat seinen Grund darin, dass das Netz die unterschiedliche Reflektivität der Pylonstreifen nur aus einzelnen Punkten im Cluster lesen kann. Der Positionsfehler liegt mit 6,4 cm auf dem Niveau der BEV-Modelle. Damit bestätigt sich die in Kapitel 2 aus der Literatur beschriebene Charakteristik direkter Punktverarbeitung [14, 13], also verlustfreie Eingabe ohne Quantisierungsrauschen, aber empfindlich gegenüber dünn besetzten Regionen, in der strukturierten Punktwolke.

Was den Fernbereich-Recall angeht, konnte eine klare Ursache gefunden werden. Der Einbruch trifft ausschließlich Pylone, die mit einem einzigen Punkt in der Punktwolke vertreten sind. Einen Suchradius mit nur einem Punkt darin kann keine lokale Struktur bilden, ab zwei Punkten pro Pylon liegt der Recall auf dem Niveau der BEV-Modelle.

Die Latenz beträgt 129 ms auf dem J4012 und überschreitet damit das 100-ms-Budget für den 10-Hz-Betrieb. Ein TensorRT-Export, der sie senken könnte, ist nicht möglich. Ball-Query und Farthest-Point-Sampling erzeugen dynamische Ausgabegrößen, die sich nicht in einen zur Compile-Zeit festen Berechnungsgraphen exportieren lassen [33].

Tabelle 4.4.: Evaluationsergebnisse des Raw-Point-Modells (J4012, FP32). Metrikdefinitionen siehe Abschnitt 2.2.

WF1	wRecall	wPrec	ColorAcc	MAE	R@0–8 m	R@8–16 m	R@16–24 m	torch / TRT	Trainingszeit
0,788	0,776	0,800	0,715	6,4 cm	0,819	0,733	0,533	129 ms / n. v.	7 h

5. Vergleichende Bewertung und Diskussion

Der folgende Vergleich stellt die vier Architekturen einander gegenüber. Die Modelle hatten die Aufgabe, Pylone zuverlässig zu lokalisieren und zu detektieren, und zwar unter der Nebenbedingung, dass sie auf dem J4012 in unter 100 ms je Frame laufen. Beide Teile der Forschungsfrage werden deshalb getrennt betrachtet, zuerst die Erkennungsleistung und anschließend die Inferenzlatenz. Die Zahlen in der Tabelle zeigen einen klaren Gewinner, nämlich das Pillar-Modell.

Tabelle 5.1.: Vergleich aller vier Architekturen auf dem gemeinsamen Testdatensatz (49 Frames, J4012, FP32). Latenz als PyTorch- und TRT-Inferenz, n. v. steht für nicht verfügbar, da Raw-Point nicht TRT-exportierbar ist.

Modell	WF1	wRec	wPrec	ColorAcc	MAE	R@0–8	R@8–16	R@16–24	PT / TRT [ms]	Trainingszeit
Pillar	**0,888**	**0,853**	**0,928**	0,721	**6,1 cm**	**0,837**	**0,869**	0,665	**32 / 8,3**	4 h
Voxel	0,804	0,752	0,867	0,856	6,4 cm	0,825	0,678	**0,671**	297 / 70,6	5 h
Raw-Point	0,788	0,776	0,800	0,715	6,4 cm	0,819	0,733	0,533	129 / n. v.	7 h
Range-Image	0,238	0,228	0,249	**0,894**	22,3 cm	0,380	0,076	0,026	30,5 / 30,0	**3 h**

5.1. Erkennungsleistung nach Distanzbereichen

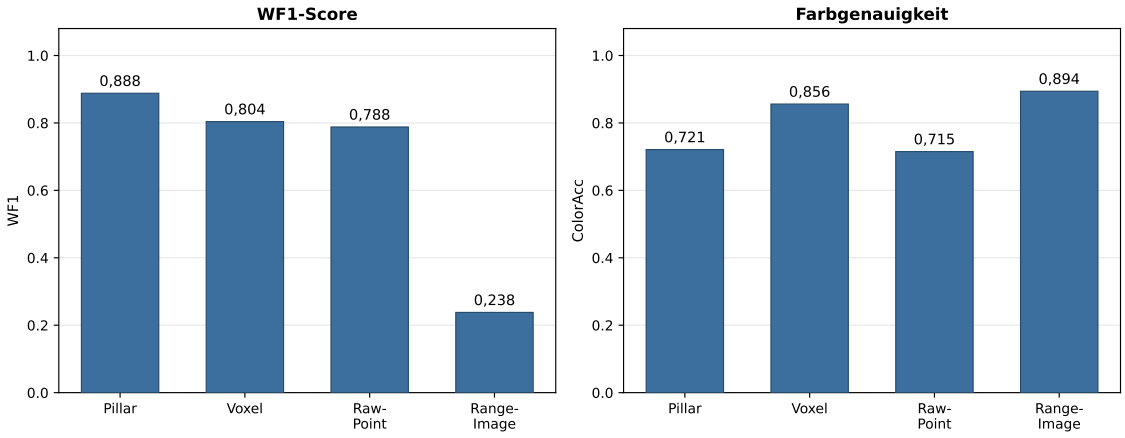

Abbildung 5.1.: WF1-Score (links) und Farbgenauigkeit ColorAcc (rechts) aller vier Architekturen im Vergleich.

Wie in Tabelle 5.1 und Abbildung 5.1 zu sehen, ergibt sich nach WF1 eine Rangliste mit Pillar (0,888), Voxel (0,804), Raw-Point (0,788) und Range-Image (0,238). Zwischen den ersten drei liegen maximal 0,100 Punkte, zwischen Raw-Point und Range-Image dagegen 0,550 Punkte. Die ersten drei Modelle bewegen sich damit in derselben Größenordnung, während sich das Range-Image-Modell um mehr als das Fünffache dieses Abstands nach unten absetzt. Der Abstand innerhalb der ersten drei beschreibt damit einen graduellen Unterschied in der Erkennungsleistung, der Abstand zum Range-Image-Modell dagegen einen grundsätzlichen. Welche der ersten drei Architekturen sich für den Produktiveinsatz eignet, entscheidet sich deshalb nicht am WF1-Score allein, sondern erst zusammen mit Latenz und Exportierbarkeit auf dem J4012.

Das erste Entfernungsdrittel von 0 bis 8 m, der linke Block in Abbildung 5.2, liefert keine Überraschungen. Pillar 0,837, Voxel 0,825, Raw-Point 0,819 liegen gleichauf. Pylone in dieser Entfernung werden mit mehreren Punkten abgetastet und geben allen drei Architekturen genug Punkte zum Arbeiten. Das Range-Image-Modell erreicht hier nur 0,380 und ist damit schon im Nahbereich deutlich schlechter, obwohl die Sub-Pixel-Grenze erst bei 9,4 m liegt (hergeleitet in Abschnitt 4.4). Dafür gibt es zwei Gründe. Erstens endet dieser Entfernungsbereich bei 8 m und damit knapp vor der Grenze, denn ein Pylon auf 8 m belegt nur noch 1,2 Spalten. Der Mittelwert über den Bereich mischt also gut aufgelöste Pylone im unmittelbaren Nahbereich mit bereits grenzwertigen am oberen Rand. Zweitens muss dieses Modell die Entfernung schätzen, während die BEV-Modelle sie unmittelbar aus der Gitterzelle ablesen. Die Position eines Pylons ergibt sich beim Range-Image aus Azimut, Elevation und der vorhergesagten Tiefe, ein Fehler in dieser Tiefe verschiebt die Detektion also radial vom Sensor weg oder auf ihn zu. Übersteigt die Verschiebung den Zuordnungsradius von 0,5 m, gilt der Pylon als nicht gefunden. Wie stark das wiegt, zeigt der Lokalisierungsfehler in Abbildung 5.3 mit 22,3 cm, gemessen an genau den Detektionen, die den Zuordnungsradius noch eingehalten haben.

Ab 8 bis 16 m treten die Unterschiede deutlich hervor, wie in Abbildung 5.2 zu sehen. Pillar hält 0,869, Raw-Point fällt auf 0,733, Voxel auf 0,678 und Range-Image bricht auf 0,076 ein. Das erklärt Abschnitt 4.4. Ab 9,4 m passt ein Pylon rechnerisch nicht mehr in eine einzelne Bildzelle der Projektion. Was das Netz dann sieht, ist kein klares Muster mehr, sondern Rauschen. Interessanter ist die Frage, warum Voxel hier hinter Pillar liegt, obwohl es mehr Information hat. Der Grund liegt im Training. Der DenseEncoder verarbeitet ein zwölfmal größeres Gitter als Pillar und besitzt entsprechend mehr freie Parameter, die aus derselben, unveränderten Datenmenge gelernt werden müssen. Bei gleichem Datensatz und gleichem Abbruchkriterium fällt es dem größeren Modell damit schwerer, seine zusätzliche Kapazität auszuschöpfen, weshalb mit einem größeren Trainingsdatensatz oder gezielt auf den Encoder abgestimmten Trainingsparametern ein besseres Ergebnis nicht auszuschließen ist.

Von 16 bis 24 m, dem rechten Block in Abbildung 5.2, sind Pillar und Voxel mit 0,665 und 0,671 fast gleichauf, ein Unterschied, der vernachlässigt werden kann. Raw-Point fällt auf 0,533, der Grund wird bereits in Abschnitt 4.5 beschrieben. Ab etwa 16 m trifft meistens nur noch ein Strahl einen Pylon. In durchgeführten Experimenten zeigte sich, dass die Ball-Query-Gruppierung des Raw-Point-Modells [13] ab diesem Punkt mindestens zwei Treffer pro Pylon benötigt, um etwas zu erkennen. Ab zwei Treffern pro Pylon erreicht Raw-Point das Recall-Niveau der BEV-Modelle, bei nur einem Treffer bricht der Recall hingegen ein.

Abbildung 5.3 stellt den Lokalisierungsfehler gegenüber. Pillar (6,1 cm), Voxel und Raw-Point (je 6,4 cm) liegen so nah beieinander, dass der Unterschied von 3 mm für die Streckenplanung ohne Bedeutung ist. Er liegt weit unterhalb der Genauigkeit, mit der die Pylone beim Aufbau der Strecke selbst positioniert werden. Range-Image kommt auf 22,3 cm. Die Ursache liegt nicht am Netz selbst, sondern an der begrenzten Auflösung der Eingaberepräsentation, aus der sich unabhängig vom Trainingszustand keine präzise Rückprojektion gewinnen lässt.

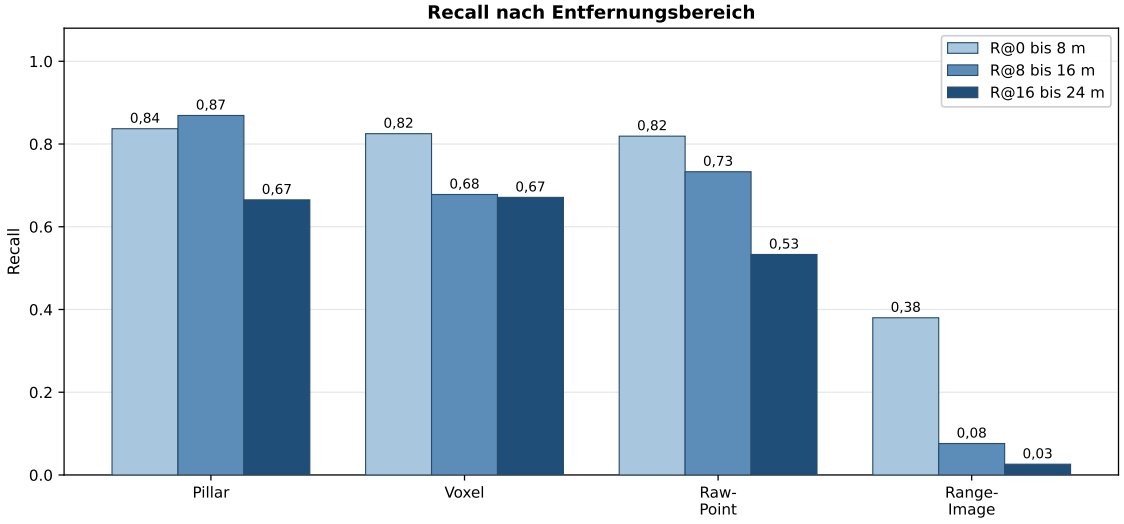

Abbildung 5.2.: Recall je Entfernungsbereich für alle vier Architekturen. Pillar erreicht im kritischen Bereich von 8 bis 16 m den höchsten Recall, beim Range-Image-Modell bricht er ab 8 m ein.

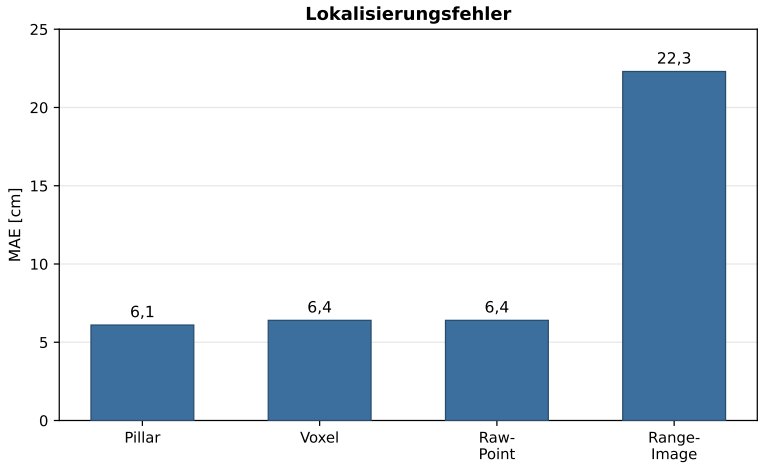

Abbildung 5.3.: Mittlerer Lokalisierungsfehler (MAE) aller vier Architekturen. Pillar, Voxel und Raw-Point liegen innerhalb von 3 mm gleichauf, Range-Image liegt um mehr als das Dreifache darüber.

5.2. Farbgenauigkeit und Höhenauflösung

Bei der Spalte zur Farbgenauigkeit (ColorAcc) aus Tabelle 5.1, die im rechten Diagramm von Abbildung 5.1 zusätzlich grafisch dargestellt ist, zeigt sich dagegen ein völlig anderer Trend. Für Farberkennung wäre Range-Image das beste Modell, mit 0,894 vor Voxel (0,856), Pillar (0,721) und Raw-Point (0,715). ColorAcc wertet nur diejenigen Pylone aus, die auch gefunden wurden, alle nicht gefundenen Pylone werden in dieser Metrik nicht beachtet. Range-Image findet insgesamt sehr wenige Pylone (Recall 0,228). Jedoch sind die wenigen Treffer fast alle nah und gut sichtbar, daher kann der weiße und schwarze Streifen der Pylone gut erkannt werden, was in einer guten ColorAcc resultiert.

Dass Voxel besser als Pillar beim Farberkennen ist, ist dagegen ein echter Unterschied. Mit 12 Höhenschichten sieht der Encoder die vertikalen Pylonstreifen. Blau und Gelb unterscheiden sich im Intensitätssignal je nach Scanzeile, und dies geht verloren, sobald man alle Höhenpositionen in einen einzigen Merkmalsvektor zusammenfügt, genau so wie Pillar dies tut. Der Abstand von 13,5 Prozentpunkten zwischen Voxel und Pillar spiegelt diesen Verlust direkt wider. Für den Einsatz bei Bremergy spielt dies eine untergeordnete Rolle, da eine Fusion der Farbinformation der Kamera mit der vom LiDAR vorhergesagten Position zu überdenken wäre. Dies wird aktuell intern bei Bremergy diskutiert.

5.3. Latenz und Einsatzfähigkeit

Die Latenztests wurden für eine konsistente Vergleichsbasis für alle Modelle einmal in PyTorch [25] durchgeführt, da das Raw-Point-Modell nicht TensorRT-fähig ist. Die PyTorch-Latenz auf dem J4012 liegt bei Range-Image bei 30,5 ms, bei Pillar bei 32 ms, bei Raw-Point bei 129 ms und bei Voxel bei 297 ms. Diese Reihenfolge ist ein klares Ergebnis für Range-Image und Pillar, sofern nur die Laufzeit betrachtet wird.

Da jedoch auch relevant ist, wie sich die Modelle latenztechnisch verändern, wenn sie auf die Zielhardware optimiert exportiert werden, stellt Abbildung 5.4 zusätzlich die TensorRT-Latenz. Pillar zeigt hierfür eindeutige Stärke. Als TRT-Engine-Modell ist es mit 8,3 ms das schnellste Modell und erfüllt das 10-Hz-Ziel ohne Probleme.
Voxel, das in der PyTorch-Latenz mit Abstand letzter war, kommt als TRT-Engine-Modell auf 70,6 ms, auch hier wird das 10-Hz-Ziel eingehalten.
Beim Range-Image-Modell hat das TRT-Engine-Modell nahezu keine Auswirkung, nur 30,0 ms statt 30,5 ms. Die Projektion der strukturierten Punktwolke und die Rückrechnung der Detektionen laufen außerhalb der TRT-Engine und benötigen dabei genauso lange wie die TRT-Engine selbst. Das 10-Hz-Ziel wird dennoch erreicht, nur ändert die TRT-Exportierung hier nicht so viel wie bei Pillar und Voxel. Raw-Point lässt sich gar nicht exportieren, da FPS und Ball-Query Ausgaben variabler Größe erzeugen, mit denen TRT nichts anfangen kann [33], die 129 ms bleiben bestehen.

Neben Latenz und Trainingszeit unterscheiden sich die Modelle auch in ihrer Größe. Die Anzahl der trainierbaren Parameter beträgt 1,40 Millionen beim Pillar-Modell, 1,70 Millionen beim Voxel-Modell, 2,52 Millionen beim Raw-Point-Modell und 4,57 Millionen beim Range-Image-Modell. Damit ist das kleinste Modell zugleich das genaueste und schnellste, während das mit Abstand größte die schwächste Erkennungsleistung zeigt. Das stützt die Einordnung aus Abschnitt 4.4. Der Rückstand des Range-Image-Modells liegt nicht an zu geringer Modellkapazität, über die es im Gegenteil reichlich verfügt, sondern an der Eingaberepräsentation, die die benötigte Information gar nicht erst abbildet.

Neben der Inferenzlatenz zeigt Tabelle 5.1 auch die Trainingszeit als Maß für den Entwicklungsaufwand. Pillar liegt mit rund 4 h im unteren Bereich und ist damit nicht nur im Betrieb, sondern auch in der Entwicklung günstig. Voxel liegt mit rund 5 h darüber, was zum zwölfmal größeren Gitter seines Encoders passt. Raw-Point benötigt mit rund 7 h am längsten und bleibt als einziges Modell ohne TensorRT-Pfad auch im Betrieb auf die langsamere PyTorch-Ausführung angewiesen. Range-Image trainiert mit rund 3 h am schnellsten, allerdings nicht wegen eines sparsameren Aufbaus, sondern weil ein einzelner Trainingsschritt auf dem 2048×32 großen Bild deutlich weniger Rechenzeit kostet als auf einem BEV-Gitter (Abschnitt 4.4). Die

angegebenen Zeiten gelten für die Trainingsläufe der hier verglichenen Modellstände. Im Verlauf der Entwicklung lagen einzelne Vorgängerversionen mit über zehn Stunden deutlich darüber, bevor Konfiguration und Abbruchkriterium auf die Pylonerkennung zugeschnitten waren.

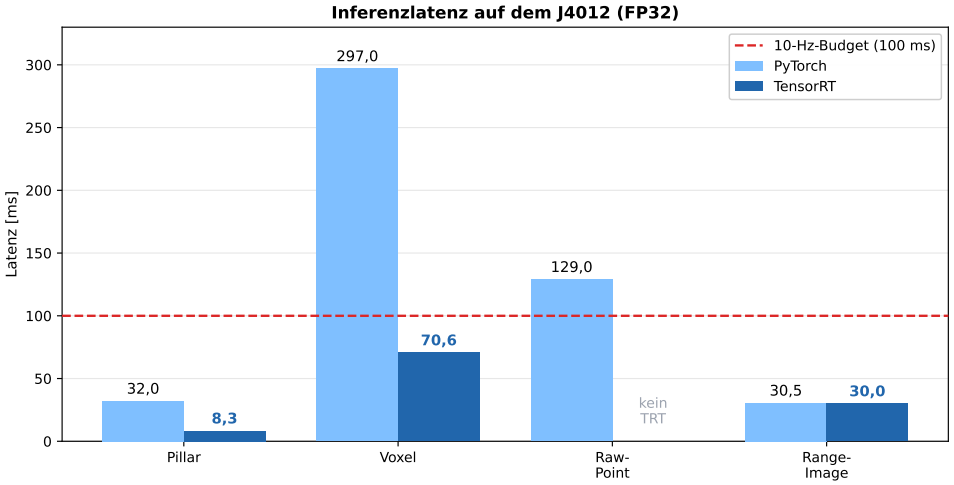

Abbildung 5.4.: Inferenzlatenz auf dem J4012 im PyTorch- und TensorRT-Betrieb. Die gestrichelte Linie markiert das 100-ms-Budget für 10-Hz-Betrieb. Raw-Point ist nicht TRT-exportierbar.

5.4. Einordnung der Ergebnisse in den Anwendungsfall

Das Überraschendste an diesen Ergebnissen ist nicht, welches Modell die beste Performance gezeigt hat, sondern der Grund dafür. Pillar [16, 19] ist die einfachste Architektur und hat trotzdem die beste Erkennung und die kürzeste Latenz. Die Erklärung liegt im Anwendungsfall. Pylone sind klein und vertikal ohne besonderes Profil. Was zählt, ist ihre Position auf dem Boden und ihr Intensitätssignal, nicht ihre Höhenverteilung. Wer die Höhe weglässt, verliert bei Pylonen fast nichts und bekommt dafür ein kleineres Modell, das schnell auf einem J4012 läuft.

Voxel [17] macht es anders. Es behält die Höhendimension, verarbeitet zwölfmal mehr Gitterzellen, und erkauft sich damit eine bessere Farbunterscheidung, aber schlechtere Gesamterkennung und viel höhere Latenz ohne TRT. Die Fülle an Informationen bringt hier mehr Nachteile als Vorteile. Das ist kein allgemeines Ergebnis, sondern eines, das direkt am Modelltyp hängt.

Das Raw-Point-Modell [13] zeigt etwas anderes. Auf 0 bis 16 m, wo der Sensor genug Punkte liefert, ist es so gut wie die BEV-Modelle. Was es bremst, ist nicht die Architektur, sondern der Sensor. Ein Pylon auf 20 m schlägt mit einem einzigen Punkt ein, und ein Punkt allein reicht für Ball-Query nicht. Das ist kein Designfehler von IA-SSD, das ist eine physikalische Begrenzung durch den Sensor. Wo BEV-Modelle bei wenigen Punkten bis hin zu einem einzelnen Punkt noch Features erkennen können, erhält Ball-Query nicht genügend Informationen, um Vorhersagen zu treffen.

Beim Range-Image-Modell [18, 28] liegt das Problem tiefer. CNN auf geordneten 2D-Panoramabildern funktioniert für Autos und Fußgänger gut, weil diese Objekte auf ausreichender Distanz noch mehrere Pixel belegen. Ein Pylon jenseits von 9,4 m belegt weniger als eine Bildzelle, wie in Abschnitt 4.4 hergeleitet. Das Netz bekommt dort kein konsistentes Signal mehr, sondern ein sporadisches. Zehn Modellversionen haben daran nichts geändert, weil es ebenfalls nicht am Modell liegt, sondern an der Winkelauflösung, mit der ein Pylon im Range-Image überhaupt abgebildet wird.

Dass die Höhenkollabierung von Pillar bei kleinen, bodennahen Objekten mit bekannter Ausdehnung kaum Genauigkeit kostet, deckt sich mit den Ergebnissen von PointPillars [16], das für Fahrzeuge und Fußgänger auf KITTI trotz Höhenkollabierung eine konkurrenzfähige Genauigkeit gegenüber voxelbasierten Verfahren erreicht. Ein direkter Vergleich mit Ergebnissen anderer Formula-Student-Teams ist nicht möglich, da bislang, wie in Kapitel 1 dargelegt, kein vergleichbarer, veröffentlichter Benchmark für die LiDAR-basierte Pylonerkennung existiert.

5.5. Grenzen des Vergleichs

Die in dieser Arbeit erzielten Ergebnisse gelten für den konkret verwendeten Ouster OS1-32-U3, einen mechanisch rotierenden LiDAR mit 32 fest positionierten Lasern, die bei jeder Umdrehung dieselben, regelmäßig verteilten Elevationswinkel abtasten [8]. Diese feste Beam-Struktur ist die Voraussetzung für die Range-Image-Projektion (Abschnitt 4.4). Ein Range-Image setzt voraus, dass sich jedem Punkt eindeutig eine feste Zeile zuordnen lässt. Solid-State-LiDAR-Sensoren mit nicht-repetitivem Scanmuster besitzen keine solche feste Beam-Zeile, wodurch sich das Range-Image-Modell auf einem solchen Sensor nicht ohne grundlegende Anpassung der Projektion einsetzen ließe. Solid-State-Sensoren mit einem festen, kameraähnlichen 2D-Raster, etwa Flash-LiDAR, könnten dagegen weiterhin eine Range-Image-artige Projektion erlauben.

Pillar, Voxel und Raw-Point sind von dieser Einschränkung nicht betroffen, da sie die Punktwolke unabhängig von der Scan-Geometrie des Sensors als Menge von (x, y, z, Reflektivität)-Werten verarbeiten und keine feste Beam-Zuordnung voraussetzen.

Sie sollten sich daher grundsätzlich auch auf Punktwolken anderer LiDAR-Bauarten, einschließlich Solid-State-Sensoren, anwenden lassen. Unterschiede in Punktdichte, Sichtfeld und Rauschcharakteristik zwischen Sensoren wurden in dieser Arbeit jedoch nicht untersucht. Ein direkter Transfer der hier trainierten Modelle auf einen anderen Sensortyp ohne erneutes Training ist entsprechend nicht zu erwarten.

Eine weitere Grenze betrifft das Zeitbudget. Im Rahmen dieser Arbeit wurde für alle vier Modelle dieselbe Entwicklungszeit aufgewendet, unabhängig davon, wie viel Optimierungspotenzial in der jeweiligen Architektur tatsächlich steckt. Die berichteten Ergebnisse spiegeln damit den erreichten Stand innerhalb eines gleich verteilten Zeitbudgets wider, nicht das theoretische Optimum jeder Architektur. Insbesondere für Voxel, dessen deutlich größeres Gitter mehr Trainingsepochen bis zur Konvergenz benötigen würde (Abschnitt 4.3), ist ein signifikant besseres Ergebnis bei zusätzlichem Optimierungsaufwand nicht auszuschließen.

5.6. Empfehlung für den Produktiveinsatz

Damit lässt sich die eingangs gestellte Forschungsfrage, *„Welche Kombination aus Netzarchitektur und LiDAR-Eingabedarstellung erzielt für die Formula-Student-Pylonerkennung die zuverlässigste Detektionsleistung bei echtzeitfähiger Inferenzlatenz auf dem J4012?“*, klar beantworten. Es ist die Kombination aus CenterPoint-Head und BEV-Pillar-Eingabedarstellung, die unter den vier untersuchten Architekturen sowohl die höchste Detektionsleistung als auch die niedrigste, echtzeitfähige Inferenzlatenz auf dem J4012 erreicht.

Für den Produktiveinsatz bei Bremergy ist das Pillar-Modell die klare Empfehlung, mit WF1 0,888, MAE 6,1 cm und TRT-Latenz 8,3 ms. Es bleibt genug Puffer, damit das System auch unter Volllast noch zuverlässig innerhalb von 10 Hz gute Vorhersagen mit präziser Positionsbestimmung liefert.

Wer auf LiDAR-basierte Farberkennung angewiesen ist, findet im Voxel-Modell eine sinnvolle Alternative, mit einer Farbgenauigkeit von 0,856 statt 0,721 und einer TRT-Latenz von 70,6 ms, die noch im Budget liegt. Wie in Abschnitt 5.5 beschrieben, könnte Voxel mit zusätzlichem Optimierungsaufwand signifikant verbessert werden, sodass es Pillar perspektivisch ablösen könnte. Selbst mit den aktuellen Werten liegt mit Voxel aber bereits ein gutes sekundäres Modell vor, das schon jetzt genutzt werden kann.

Das Ergebnis des Raw-Point-Modells ist wissenschaftlich das interessanteste dieser Arbeit. Eine Architektur, die Punkte direkt verarbeitet, ohne sie vorher zu rastern, kommt auf 0 bis 16 m an BEV heran, trotz eines Sensors, der dafür nicht hochauflösend genug ist. Für den aktuellen Betrieb auf dem J4012 scheidet es wegen fehlender

TRT-Exportierbarkeit aus. Als Ansatz für zukünftige Sensorkonfigurationen bleibt es interessant.

Das Range-Image-Modell hat mit dem OS1-32-U3 keine tragfähige Grundlage. Ein höher auflösender Sensor oder gestapelte Mehrframe-Scans könnten das ändern. Innerhalb der Rahmenbedingungen dieser Arbeit ist das Range-Image-Modell am wenigsten geeignet, nicht wegen mangelnder Optimierung, sondern wegen einer geometrischen Grenze, die man mithilfe von Optimierung nicht überwinden kann.

6. Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde untersucht, welche Architektur und welche Methode der Eingabeverarbeitung für LiDAR-Punktwolken am besten für die Erkennung und Lokalisierung von Pylonen im Formula-Student-Kontext geeignet ist. Vier grundlegend verschiedene Ansätze wurden implementiert, trainiert und auf identischer Hardware verglichen. Das Ergebnis zeigt einen klaren Unterschied zwischen den vier Architekturen.

6.1. Zentrale Ergebnisse

Das Pillar-Modell, das die strukturierte Punktwolke in eine flache BEV-Projektion verwandelt, ist das schnellste und genaueste, mit 8,3 ms als TRT-Engine und einem Lokalisierungsfehler von 6,1 cm. Das Voxel-Modell, das die dritte Dimension im Gitter erhält, hat von allen vier die beste Farbgenauigkeit (0,856), ist aber ohne TRT-Export deutlich zu langsam für 10 Hz. Mit der TRT-Engine kommt es auf 70,6 ms, was weiteres Optimierungspotenzial nahelegt. Das Raw-Point-Modell, das die Punkte ohne Zwischenschritt direkt verarbeitet, hält auf 0 bis 16 Metern BEV-Niveau, ein TRT-Export ist aber nicht möglich, also kein 10-Hz-Betrieb. Das Range-Image-Modell, das den Scan als geordnetes Panoramabild verarbeitet, landet bei WF1 0,238 und bricht ab 8 Metern nahezu vollständig ein.

Warum das so ist, hat jeweils einen klaren Grund. Pillar profitiert davon, dass Pylone vertikal keine besondere Struktur aufweisen. Das Weglassen der Höhe bringt daher kaum Informationsverlust mit sich. Voxel zeigt, dass mehr Information nicht automatisch eine bessere Erkennungsperformanz bedeutet. Die Höhendimension hilft bei der Farbe, kostet aber Trainingszeit und Latenz. Raw-Point trifft auf eine physikalische Schranke. Die Ball-Query-Gruppierung [13] benötigt in durchgeführten Experimenten mindestens zwei Treffer pro Pylon, und ab 16 Metern kommen diese nicht mehr verlässlich an. Range-Image trifft auf eine geometrische Grenze. Ab 9,4 Metern passt ein Pylon nicht mehr in eine Bildzelle. Beide Grenzen von Raw-Point und Range-Image sind im Sensor verankert, nicht im Modell.

Das Team Bremergy hat sich daher für das Pillar-Modell entschieden und ist bereits erfolgreich auf einer nach VSV-Regeln [11] aufgebauten Strecke im Betriebshof

der Universität Bremen mit dem Pillar-Modell gefahren. Ein Video davon ist auf YouTube verfügbar (`https://youtu.be/lCw0xjQr-eQ`).

6.2. Ausblick und offene Fragen

Ein naheliegender nächster Schritt wäre eine Änderung der Sensor-Montierung nach vorne. Zum Zeitpunkt des Verfassens dieser Arbeit ist der LiDAR-Sensor parallel zum Boden ausgerichtet, dadurch treffen 16 von 32 Strahlen den Boden und 16 von 32 werden über dem Horizont verloren. Eine Neigung würde mehr Strahlen in den Vorwärtsbereich lenken und die Punktdichte im Fernbereich erhöhen, da dann alle 32 Strahlen in Richtung Boden geneigt sind. Der Single-Point-Bottleneck des Raw-Point-Modells würde sich damit physikalisch verringern. Ein Neutraining mit Daten aus dieser neuen Geometrie wäre für Pillar und Voxel sinnvoll.

Farbgenauigkeit wurde in dieser Arbeit nur über das LiDAR-Signal bewertet. Eine Fusion von LiDAR-Position und Kamerafarbe könnte die aktuellen Werte deutlich übersteigen und die Verwechslungen zwischen Gelb und Blau beim Pillar-Modell praktisch eliminieren.

Für das Range-Image-Modell gibt es einen theoretischen Ausweg aus der geometrischen Grenze, nämlich mehrere aufeinanderfolgende Scans zu stapeln. Durch die Eigenbewegung des Fahrzeugs taucht ein Pylon in jedem Frame an einer anderen Azimutposition auf, gestapelt belegt er mehr Bildzellen. Das wäre aufwendiger, aber der einzige Ansatz, der die Grenze mit dem vorhandenen Sensor verschiebt. Ob sich dieser Aufwand lohnt, bleibt jedoch zu prüfen. Bei hoher Fahrzeuggeschwindigkeit kann die zeitliche Lücke zwischen den gestapelten Scans so groß werden, dass das Fahrzeug den erfassten Streckenabschnitt bereits passiert hat, bevor die Vorhersage vorliegt. Eine Vorhersage auf Basis bereits vergangener Daten wäre dann nicht mehr sinnvoll.

Des Weiteren kann untersucht werden, ob sich nicht abseits vom mmdetection3d-Framework noch Modellarten finden lassen, die genau für das Formula-Student-Szenario passen und ressourcenschonender sind oder sogar besser Pylone lokalisieren und detektieren können.

A. Ergänzende Abbildungen

Dieses Kapitel enthält die Abbildungen zu den einzelnen Verarbeitungsschritten der Modell-Architekturen. Im Fließtext der jeweiligen Kapitel wird darauf verwiesen, im Haupttext verbleiben nur die Eingabe- und die finale Ausgabe-Abbildung jedes Modells.

A.1. BEV-Pillar-Detektion

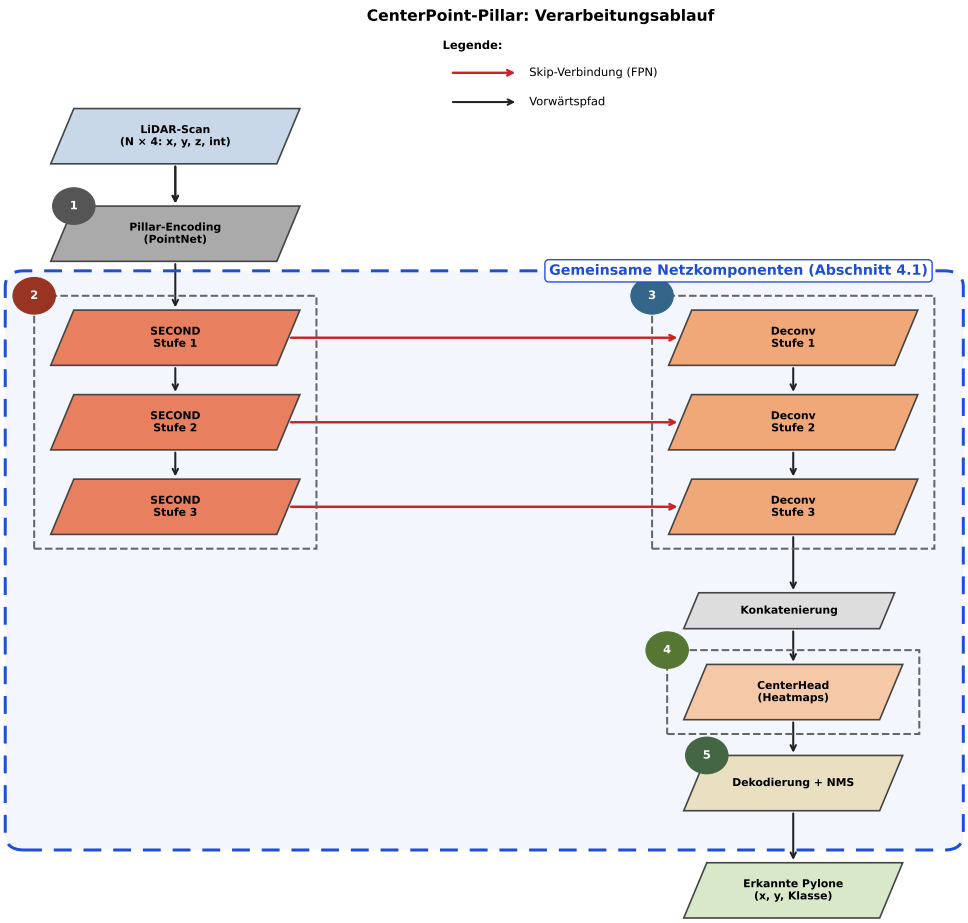

Abbildung A.1.: Verarbeitungsablauf der BEV-Pillar-Architektur von der Eingabe des LiDAR-Scans bis zur Ausgabe der erkannten Pylone. Beschriftete Stufen entsprechen den nummerierten Schritten im Text.

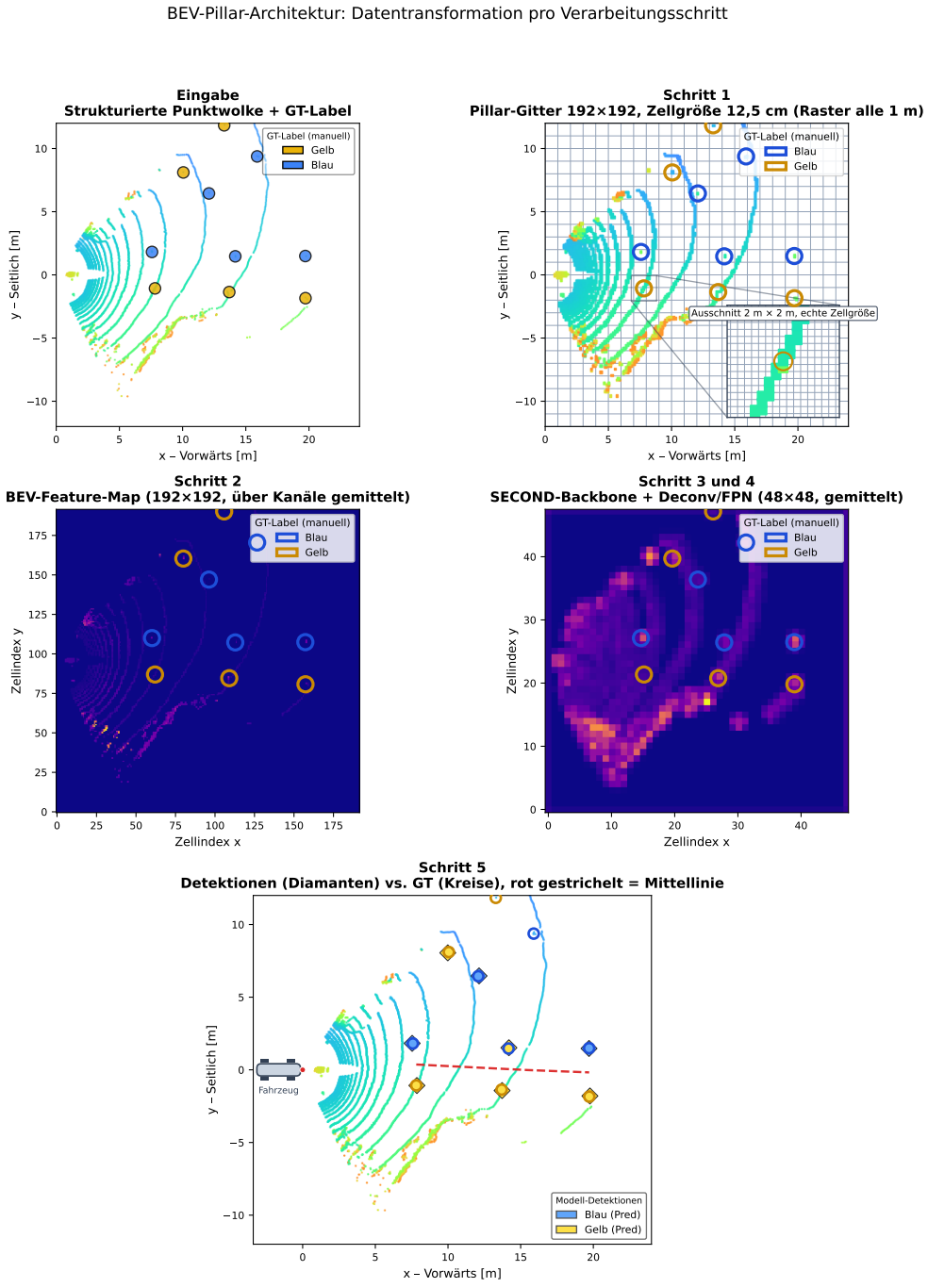

Abbildung A.2.: Datentransformation an einem realen LiDAR-Frame (VSV-Testday, 13.950 Punkte): von der strukturierten Eingabe-Punktwolke über das Pillar-Gitter (192 × 192, Zellgröße 12,5 cm), die BEV-Feature-Map und den SECOND-Backbone bis zu den finalen Detektionen (Diamanten) gegen GT-Labels (Kreise).

A.2. BEV-Voxel-Detektion

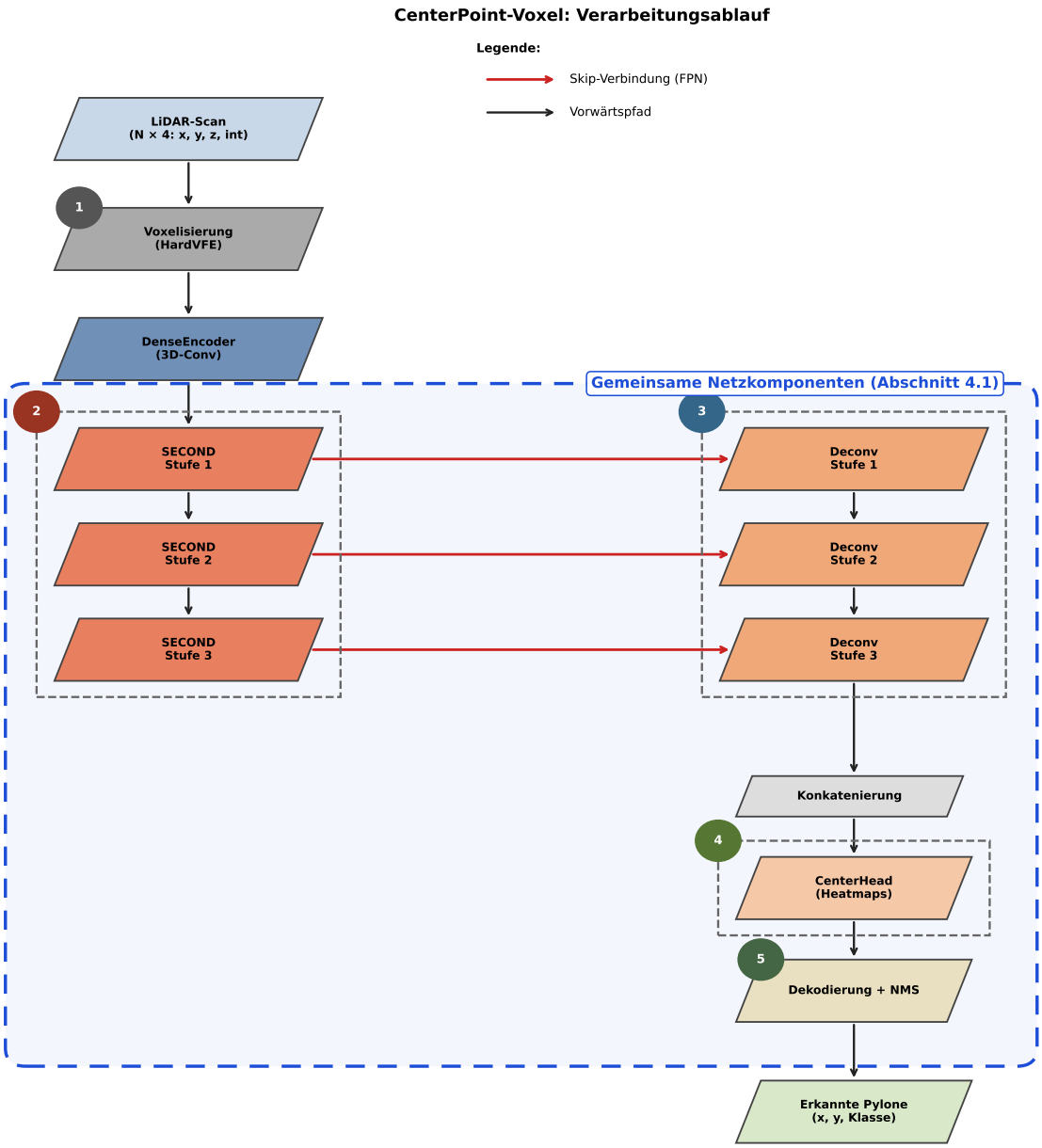

Abbildung A.3.: Verarbeitungsablauf der BEV-Voxel-Architektur von der Eingabe des LiDAR-Scans bis zur Ausgabe der erkannten Pylone. Beschriftete Stufen entsprechen den nummerierten Schritten im Text.

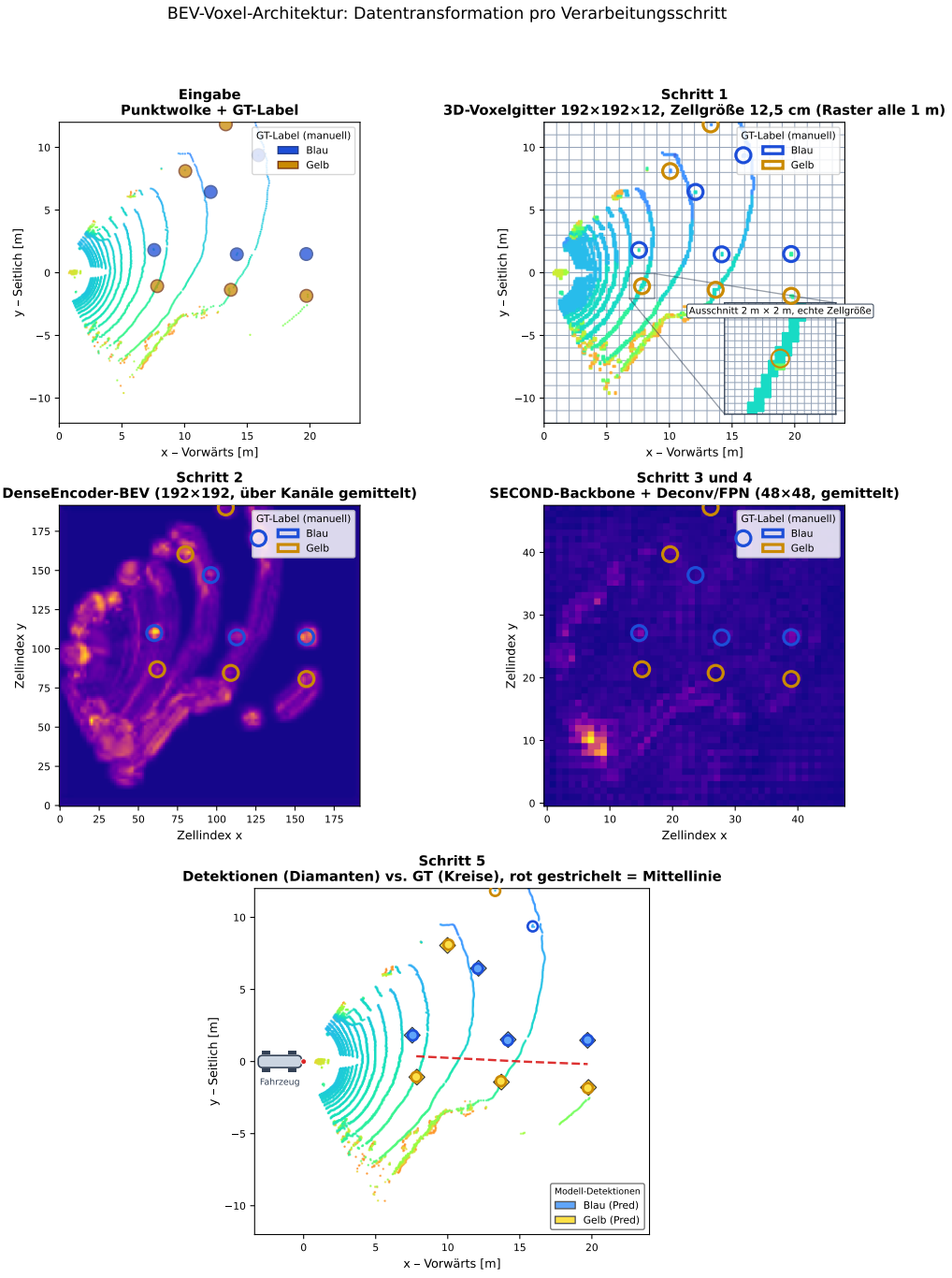

Abbildung A.4.: Datentransformation an einem realen LiDAR-Frame: strukturierte Eingabe-Punktwolke, 3D-Voxelgitter ($192 \times 192 \times 12$, Zellgröße 12,5 cm), DenseEncoder-Feature-Map, SECOND-Backbone-Ausgabe und finale Detektionen (Diamanten) gegen GT-Labels (Kreise). Auf die Darstellung der Höhenaufteilung wurde aufgrund der Darstellbarkeit verzichtet.

A.3. Range-Image-Detektion

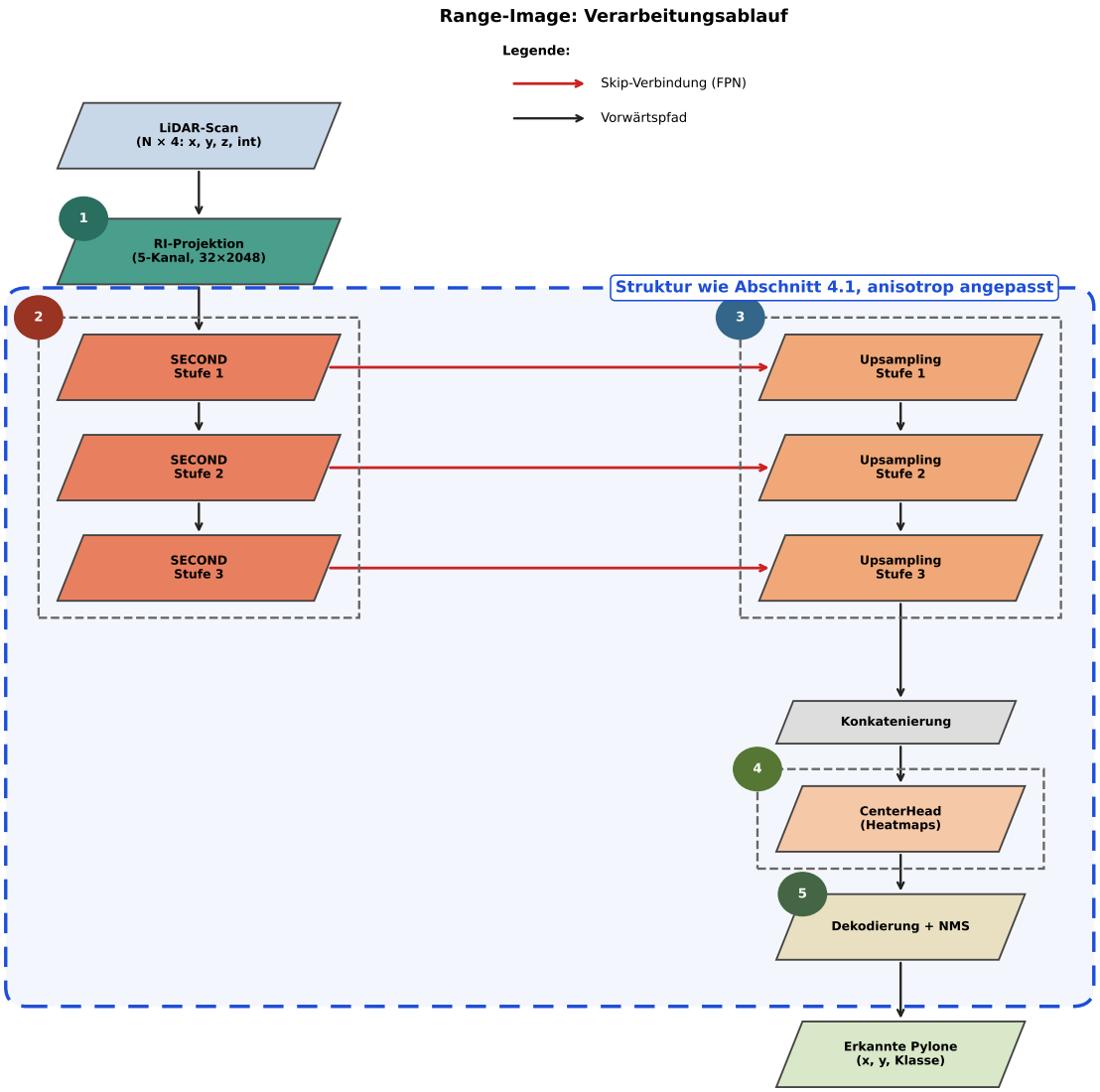

Abbildung A.5.: Verarbeitungsablauf der Range-Image-Architektur von der Eingabe des LiDAR-Scans bis zur Ausgabe der erkannten Pylone. Beschriftete Stufen entsprechen den nummerierten Schritten im Text.

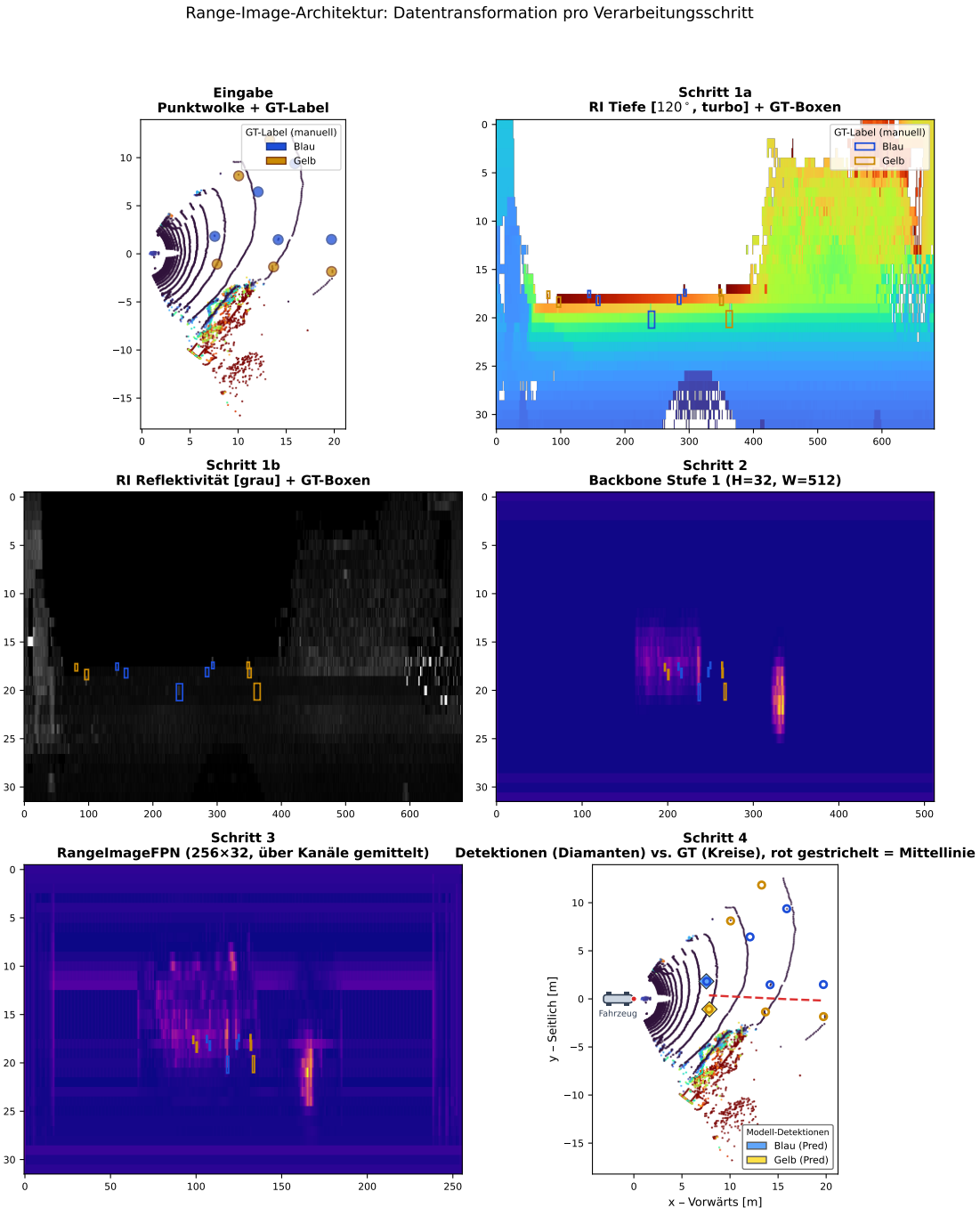

Abbildung A.6.: Datentransformation an einem realen LiDAR-Frame: vollständiges Range-Image (2048 × 32, grüne Linien markieren das 120°-Vorwärtsfenster), anisotroper SECOND-Backbone, RangeImageFPN-Ausgabe und finale Detektionen.

A.4. Raw-Point-Detektion

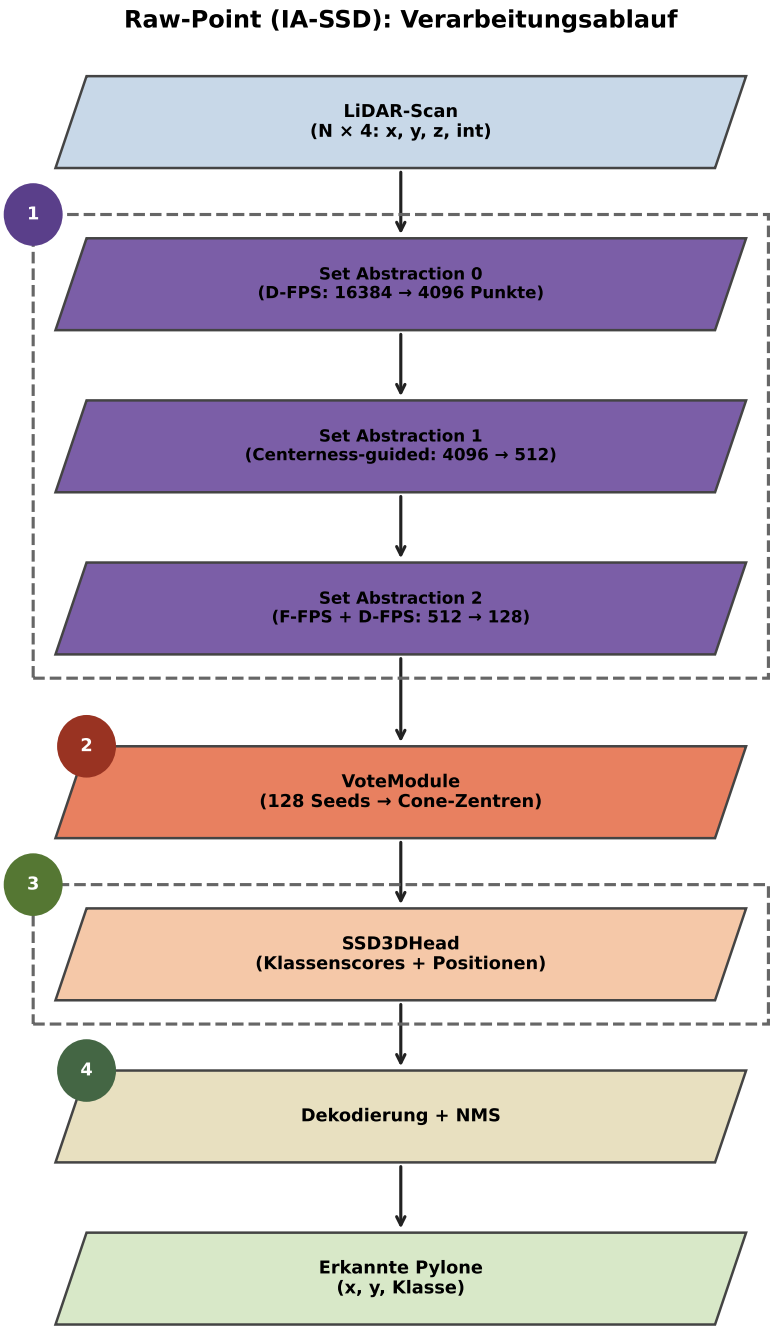

Abbildung A.7.: Verarbeitungsablauf der Raw-Point-Architektur (IA-SSD) von der Eingabe des LiDAR-Scans bis zur Ausgabe der erkannten Pylone. Beschriftete Stufen entsprechen den nummerierten Schritten im Text.

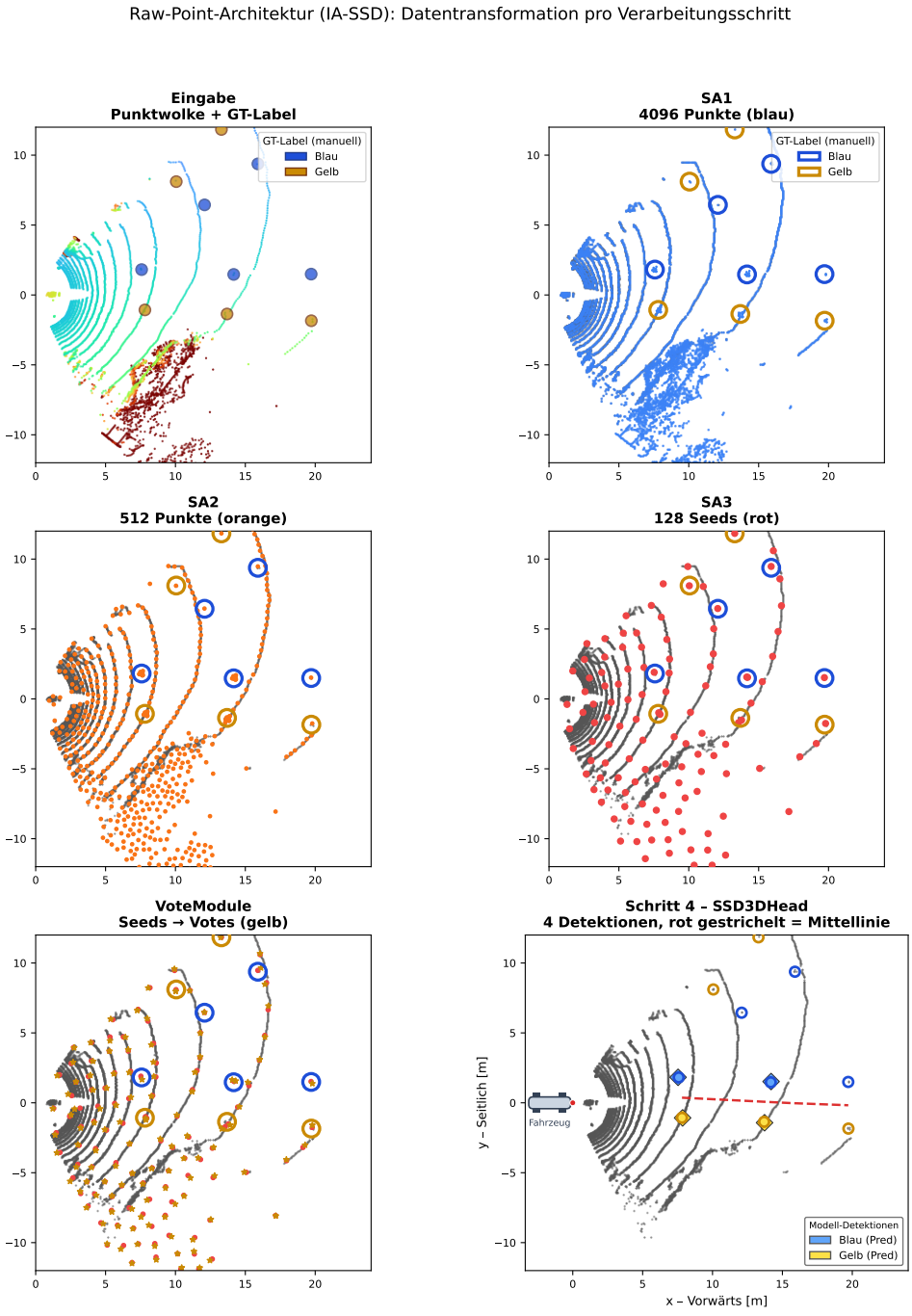

Abbildung A.8.: Datentransformation an einem realen LiDAR-Frame: Set-Abstraction-Stufen SA1 (4096 Punkte, D-FPS), SA2 (512 Punkte, centerness-guided) und SA3 (128 Saatpunkte), VoteModule-Verschiebung und finale Detektionen (Diamanten) gegen GT-Labels (Kreise).

Erklärung zur Nutzung von KI-gestützten Werkzeugen

Gemäß den Transparenzanforderungen der Universität Bremen wird die Nutzung von KI-gestützten Werkzeugen in dieser Arbeit anhand der Vorlage der Studierwerkstatt (Alternative 1, tabellarische Dokumentation) wie folgt dokumentiert:

KI-basiertes Hilfsmittel	**Einsatzform**	**Betroffene Teile der Arbeit**	**Beschreibung der Eingabe (Prompt)**	**Bemerkungen**
Claude (Anthropic)	Literaturrecherche und Modellauswahl	Kapitel 2, Abschnitt Stand der Technik	Sinngemäß: Welche neuronalen Netzarchitekturen eignen sich für die 3D-Objekterkennung in LiDAR-Punktwolken, mit Fokus auf kleine Objekte (Pylone) und Eignung für Edge-Hardware (Jetson Orin NX)?	Die KI hat Hinweise auf relevante Architekturen geliefert (PointPillars, CenterPoint, IA-SSD, RangeDet). Die finale Auswahl und Bewertung der Eignung für die Pylonerkennung erfolgte eigenständig.
Claude (Anthropic)	Auswertung von Trainingsprotokollen	Kein direkter Textabschnitt (Arbeitsprozess: Trainingsdurchläufe aller vier Modelle)	Sinngemäß: Fasse die wesentlichen Auffälligkeiten in dieser Trainingslog-Datei zusammen (Loss-Verlauf, Metrik-Entwicklung, Auffälligkeiten).	Die KI hat Auffälligkeiten in den Logs zusammengefasst. Die Ursachenanalyse und alle daraus folgenden Anpassungen wurden vom Autor eigenständig erarbeitet und durch weitere Experimente validiert.

Hinweis: Die Prompt-Beschreibungen geben den Inhalt sinngemäß wieder, da keine wortwörtlichen Prompt-Protokolle der frühen Recherchephase vorliegen.

Alle fachlichen Entscheidungen, die Implementierung, die Durchführung der Experimente, die Analyse der Ergebnisse sowie die wissenschaftliche Auseinandersetzung mit dem Thema wurden vollständig eigenständig erarbeitet.

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
AdamW	Adam-Optimierer mit Weight Decay
BEV	Bird's Eye View (Vogelperspektive)
CNN	Convolutional Neural Network (Faltendes Neuronales Netz)
EMA	Exponential Moving Average (Exponentieller gleitender Mittelwert)
F1	F1-Score (harmonisches Mittel aus Precision und Recall)
FN	False Negative (Falsch Negativ)
FP	False Positive (Falsch Positiv)
FP16	Fließkommazahl mit 16-Bit-Genauigkeit
FP32	Fließkommazahl mit 32-Bit-Genauigkeit
FPN	Feature Pyramid Network
FPS	Farthest Point Sampling (Weiteste-Punkt-Auswahl)
FS	Formula Student
GT	Ground Truth (Referenzannotation)
Hz	Hertz
IA-SSD	Instance-Aware Single-Stage Object Detector
Inf	Infinity (Unendlich, numerischer Überlauf)
IoU	Intersection over Union (Schnittmenge über Vereinigung)
J4012	Seeed Studio reComputer J4012, Trägerplatine mit Jetson Orin NX 16 GB
KI	Künstliche Intelligenz
LiDAR	Light Detection and Ranging
MAE	Mean Absolute Error (Mittlerer absoluter Fehler)
ms	Millisekunde
NaN	Not a Number (ungültiger Zahlenwert)
NMS	Non-Maximum Suppression
NX	NVIDIA Jetson Orin NX (Modellbezeichnung)
ONNX	Open Neural Network Exchange
OS1-32-U3	LiDAR-Sensor Produktname
SA	Set Abstraction
SECOND	Sparsely Embedded Convolutional Detection
SLAM	Simultaneous Localization and Mapping
TF32	TensorFloat-32
TP	True Positive (Richtig Positiv)
TRT	TensorRT (NVIDIA-Inferenz-Engine)
VSV	Vehicle Status Video (Formula Student Regelwerk)
WF1	Weighted F1 Score (Gewichteter F1-Score)

Literaturverzeichnis

[1] OpenAI, “Introducing ChatGPT,” https://openai.com/blog/chatgpt, 2022, abgerufen am 19. Juli 2026.

[2] Waymo, “Delivering more for our riders in a year of incredible growth,” https://waymo.com/blog/2025/12/2025-year-in-review/, 2025, abgerufen am 21. Juli 2026.

[3] J.-P. Skeete, “The obscure link between motorsport and energy efficient, low-carbon innovation: Evidence from the UK and European Union,” *Journal of Cleaner Production*, vol. 214, pp. 674–684, 2019.

[4] Formula Student Germany, “History of Formula Student Germany,” https://www.formulastudent.de/fsg/about/history, 2024, abgerufen am 21. Juli 2026.

[5] Bremergy Racing, “Bremo25 – Bremergy Racing: Technische Daten des Fahrzeugs,” https://www.bremergy.de/garage/251, 2025, abgerufen am 20. Juli 2026.

[6] Formula Student Germany, “Bremergy e.V. – Team Details,” https://www.formulastudent.de/teams/fse/details/tid/423, 2024, abgerufen am 22. Juli 2026.

[7] N. Vödisch, D. Dodel, and M. Schötz, “FSOCO: The Formula Student Objects in Context Dataset,” *SAE International Journal of Connected and Automated Vehicles*, vol. 5, no. 1, pp. 23–32, 2022.

[8] Ouster, Inc., “Ouster OS1 Sensor Datasheet, Rev. 06 v2.5,” https://data.ouster.io/downloads/datasheets/datasheet-rev06-v2p5-os1.pdf, 2024, abgerufen am 19. Juli 2026.

[9] Y. F. Meyer, “Entwicklung eines neuronalen Netzes zur Pylonenerkennung und Fahrbahnberechnung für autonome Fahrzeugsysteme in der Formula Student,” Bachelorarbeit, Universität Bremen, 2025, betreut von Prof. Dr.-Ing. Udo Frese und Prof. Dr.-Ing. Maren Petersen. [Online]. Available: https://www.informatik.uni-bremen.de/agebv2/downloads/published/meyer_thesis_25.pdf

[10] Seeed Studio, “reComputer J4012 – Edge AI Computer with Jetson Orin NX 16GB,” https://www.seeedstudio.com/reComputer-J4012-p-5586.html, 2023, abgerufen am 19. Juli 2026.

[11] Formula Student Germany, "Formula Student Rules 2026, Version 1.1," Formula Student Germany, Tech. Rep., 2025. [Online]. Available: https://www.formulastudent.de/fileadmin/user_upload/all/2026/rules/FS-Rules_2026_v1.1.pdf

[12] ——, "Formula Student Driverless Specification 2026, Version 1.1," Formula Student Germany, Tech. Rep., 2025. [Online]. Available: https://www.formulastudent.de/fileadmin/user_upload/all/2026/rules/FS_Driverless_Specification_2026_v1.1.pdf

[13] Y. Zhang, Q. Hu, G. Xu, Y. Ma, J. Wan, and Y. Guo, "Not All Points Are Equal: Learning Highly Efficient Point-based Detectors for 3D LiDAR Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 18 931–18 940.

[14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017.

[15] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu, and M. Bennamoun, "Deep Learning for 3D Point Clouds: A Survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 12, pp. 4338–4364, 2021.

[16] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "PointPillars: Fast Encoders for Object Detection from Point Clouds," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 697–12 705.

[17] Y. Zhou and O. Tuzel, "VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4490–4499.

[18] L. Fan, X. Xiong, F. Wang, N. Wang, and Z. Zhang, "RangeDet: In Defense of Range View for LiDAR-based 3D Object Detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021, pp. 2898–2907.

[19] T. Yin, X. Zhou, and P. Krähenbühl, "Center-based 3D Object Detection and Tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 11 784–11 793.

[20] J. Kabzan, M. I. Valls, V. J. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan *et al.*, "AMZ Driverless: The Full Autonomous Racing System," *Journal of Field Robotics*, vol. 37, no. 7, pp. 1267–1294, 2020.

[21] Chalmers Formula Student, "coneScenes: LiDAR Point Cloud Dataset for Formula Student Cone Detection," https://conescenes.chalmersformulastudent.se/, 2024, abgerufen am 19. Juli 2026.

[22] MMDetection3D Contributors, "MMDetection3D: OpenMMLab next-generation platform for general 3D object detection," https://github.com/open-mmlab/mmdetection3d, 2020.

[23] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354–3361.

[24] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuScenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 618–11 628.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019.

[26] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization," in *International Conference on Learning Representations (ICLR)*, 2019.

[27] Y. Yan, Y. Mao, and B. Li, "SECOND: Sparsely Embedded Convolutional Detection," *Sensors*, vol. 18, no. 10, p. 3337, 2018.

[28] G. P. Meyer, A. Laddha, E. Kee, C. Vallespi-Gonzalez, and C. K. Wellington, "LaserNet: An Efficient Probabilistic 3D Object Detector for Autonomous Driving," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 12 677–12 686.

[29] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 936–944.

[30] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 77–85.

[31] B. Wilson, N. A. Mitchell, J. K. Pontes, and J. Hays, "What matters in range view 3d object detection," *arXiv preprint arXiv:2407.16789*, 2024.

[32] Z. Yang, Y. Sun, S. Liu, and J. Jia, "3DSSD: Point-based 3D Single Stage Object Detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 037–11 045.

[33] NVIDIA, “Working with Dynamic Shapes — NVIDIA TensorRT Documentation,” https://docs.nvidia.com/deeplearning/tensorrt/latest/inference-library/work-with-dynamic-shapes.html, 2024, abgerufen am 21. Juli 2026.