Universität Rostock

Traditio et Innovatio

Rekonstruktion und Tracking in dynamischen Umgebungen

Semantische Anpassung ausgewählter Rekonstruktions Algorithmen

Name:	Hans-Hauke Haufe
Matrikelnummer:	2222 0976
Abgabedatum:	21.09.2025
Betreuer und Gutachter:	Dr. rer. nat. Tobias Strauß Universität Rostock Mathematisch-Naturwissenschaftliche Fakultät
Gutachter:	Arne Pointeck Fraunhofer-Institut für Großstrukturen in der Produktionstechnik IGP Messtechnik

Bachelorarbeit am Institut für Mathematik
Mathematisch-Naturwissenschaftliche Fakultät

Inhaltsverzeichnis

Abkürzungsverzeichnis

SLAM Simultaneous Localization and Mapping 3

RGB-D Red-Green-Blue plus Depth 4

vSLAM visual SLAM 4

CNN Convolutional Neural Network 28

ASPP Atrous Spatial Pyramid Pooling 30

TSDF Truncated Signed Distance Field 3

ICP Itarativ Closest Point 4

SfM Structrue from Motion 4

RPE Realtive Pose Error 18

ATE Absolute Trajectory Error 18

P2P Point to Plane 9

GIS Genric Image Segmentation 26

PIS Promtable Image Segmentation 26

1. Einleitung

In modernen Milchviehbetrieben spielt die präzise Fütterung eine entscheidende Rolle für Tiergesundheit, Milchleistung und Ressourceneffizienz. Eine Möglichkeit, die Fütterung zu optimieren, sowie Tiergesundheit zu überwachen, ist die regelmäßige und genaue Messung der verbleibenden Futtermenge im Stall. Die manuelle Erfassung ist jedoch zeitaufwendig, fehleranfällig und auch nicht ununterbrochen möglich. Automatisierte Verfahren können hier unterstützen, insbesondere Systeme, die das Futtervolumen räumlich erfassen und auswerten.

Die räumliche Rekonstruktion in einem Stall mit frei bewegenden Kühen ist technisch anspruchsvoll. Kühe bewegen sich unvorhersehbar, verdecken Teile der Futterfläche und verändern die Szene kontinuierlich. Klassische 3D-Mapping-Algorithmen setzen dagegen meist auf statische Umgebungen, wodurch Fehlschätzungen entstehen.

Um unter diesen Bedingungen eine präzise 3D-Rekonstruktion zu ermöglichen, wird ein Verfahren benötigt, das sowohl die eigene Bewegung im Raum genau schätzt als auch störende bewegte Objekte erkennt und aus der Rekonstruktion ausschließt. Hierfür bietet sich der Einsatz von Simultaneous Localization and Mapping Simultaneous Localization and Mapping (SLAM) in Kombination mit Maskensegmentierung und einer angepassten Odometrie und Truncated Signed Distance Field (TSDF)-Integration an.

Ein großer Fokus liegt hierbei auf der Methode der visuellen Odometrie und der Verbindung mit dem TSDF-Tracking, die den Grundbaustein für die Rekonstruktion darstellt. Dabei ist Anpassung dieser Verfahren für dynamische Szenen durch einbeziehen von vorsegmentierten Masken, essenziell.

Der zweite Bestandteil der Arbeit fokussiert sich darauf die Masken zu generieren. Dies passiert auf einem Anwendung spezifischen Datensatz.

2. Theoretische Grundlagen

2.1. Problemkontext

Das autonome Tracking erfolgt über ein Sensorsystem, welchers auf einem schon vorhanden Futterschiebe-Roboter installiert wird. Schieberoboter sind ein gängiger Bestandteil von lokalen Milchviehbetrieben. Sie fahren in regelmäßigen Abstände an der Futterstelle vorbei und schieben verteilte Silage zusammen. Die werden Fahrten genutzt werden um die Futter-Messung parallel durchzuführen.

Ein zentraler Bestandteil des Systems ist eine Red-Green-Blue plus Depth (RGB-D)-Kamera. Diese erfasst pro Aufnahme sowohl Farbinformationen als auch zugehörige Tiefenwerte für jeden Pixel. Die Farbinformationen dienen der Bildsegmentierung und ermöglichen damit die Erkennung der Tiere in der Szene. Die Tiefen-Informationen liefern die geometrische Struktur der Umgebung und bilden die Grundlage für die räumliche Rekonstruktion.

2.2. SLAM und Visual SLAM

Das zentrale Herausforderung, die sich aus der Problemstellung ergibt, ist folgendes: Das Messsystem arbeitet unabhängig vom Schieberoboter, dass heißt es verfügt über keine direkte Information über dessen aktuelle Position oder Orientierung im Stall. Damit wird jede Erfassung zu einer isolierten Messung ohne globalen Bezugspunkt. Um die erfassten Daten dennoch zu einer konsistenten und vollständigen 3D-Karte zusammenzuführen, muss die Bewegung des Sensors präzise geschätzt und in ein gemeinsames Koordinatensystem überführt werden.

Diese Aufgabe fällt in das Gebiet des Simultaneous Localization and Mapping SLAM, bei dem die eigene Position gleichzeitig mit einer Karte der Umgebung bestimmt wird. SLAM ist ein hoch dynamisches und sich schnell entwickelndes Forschungs-Thema. Dabei spielt die Kombination von Sensordaten mit mehrfachen komplexen Algorithmen eine zentrale Rolle.

Ein Teilgebiet, das sich primär mit der Nutzung von Bilddaten zur gleichzeitigen Lokalisierung und Kartenerstellung befasst, ist das visual SLAM (vSLAM). Die Schätzung der Eigenbewegung aus Bildfolgen kann dabei auf unterschiedlichen Ansätzen basieren. Ein Ansatz sind Verfahren der Structrue from Motion (SfM), bei denen der optische Fluss zwischen aufeinanderfolgenden Bildern analysiert wird. Daneben existieren geometrische Methoden wie der Itarativ Closest Point (ICP)-Algorithmus, bei dem geometrische

Messpunkte zwischen aufeinanderfolgenden Aufnahmen verglichen und zur Bewegungsbestimmung herangezogen werden. In dieser Arbeit wird jedoch der Schwerpunkt auf visuelle Odometrie gelegt, bei der Bilddaten kontinuierlich abgeglichen werden [23]

Die genanten Methoden besitzen alle das Problem, dass statische Umgebungen vorausgesetzt werden. Das Feld des dynamischen vSLAM stellt die Erweiterung und Anpassung von Methoden da auf dynamische Umgebungen. Dabei nehmen Methoden des maschinellen Lernen einen größer werdenden Anteil ein.[23]

Eine Anpassung der visuellen Odometrie Algorithmus's für dynamische Szenen, durch die Hilfe neuronaler Netze ist ein zentraler Punkt der Arbeit.

Neben dem tracking der Sensor-Position spielt die Rekonstruktion eine wichtige Rolle. Dabei werden die Bilddaten mithilfe der Sensorpostion in ein Speichermedium integriert. Die Representation kann verschiedene Formen annehmen kann annehmen, wie Punktwolken (direkt Darstellung asl 3D-Punkte), Meshes (Polygon basiert), Surfels oder hirachische Strukturen wie Octrees. Zunehmend werden auch neuronale Darstellungen erforscht [13].

Die Erstellung solcher Modelle wird durch dynamische Objekte deutlich erschwert. Daher ist eine angepasste Rekonstruktion ein weiterer Bestandteil dieser Arbeit. Die Kombination der angepassten Integrationsmethode mit der modifizierten Odometrie führt zum TSDF-Tracking, welches in dieser Arbeit besonders eingehend untersucht wird.

2.3. visuelle Odometrie

Die Problemstellung, die visuellen Odometrie versucht zu lösen, ist: Zu einem Bildpaar (I_s, I_t) die Relativbewegung des Sensors, bestehend aus Translation und Rotation, zwischen den beiden Aufnahmezeitpunkten zu schätzen. Aus dem sukzessiven Bestimmen solcher Teilbewegungen kann die gesamte Bewegung des Sensors (Bewegungs-Trajektorie) annäherungsweise bestimmt werden. Dabei gibt es verschiedene Ansätze.

Zum einem gibt es Methoden die Feature-Extraction nutzen, um aus Bilder besondere Bildmerkmale zu extrahieren, in Form von hoch dimensionalen Representation von Features, und diese dann iterative abzugleichen. Feature Methoden werden häufig genutzt um zeitlich weiter entfernte Bilder zu vergleichen.[16, 14]

Ein weitere Klasse von Methoden ist die der dichte Odometrie. Dabei werden alle Pixel in die Optimierung miteinbezogen. Ihr Anwendungsgebiet ist das Abgleichen von Bildern mit hoher Frequenz, jedoch mit kleinerer zeitlichen Differenz. Der Schwerpunkt liegt hier nur auf der dichte visuelle Odometrie.

Bemerkung: *Im folgenden wird visuelle Odometrie synonym mi der dichten visuellen Odometrie verwendet*

2.3.1. Rigid-Body-Motion

Rigid-Body-Motion ist das Grundlegende Konzept, dass die unverzerrten Bewegung von festen zusammenhängende Objekten im Raum beschreibt. Mathematisch kann sie beschrieben werden durch $SE(3)$. [15]

Definition 1. *$SE(3)$ ist gegeben durch*

$$\left\{F : \mathbb{R}^3 \to \mathbb{R}^3, x \mapsto Ax + b \;\middle|\; A \in SO(3), b \in \mathbb{R}^n\right\} \tag{2.1}$$

Die Komposition von Abbildungen definiert eine Verknüpfung auf der Menge. $SO(3)$ stellt dabei die 3-dimensionale orthogonale Gruppe da(vgl. [15]).

Bemerkung 2.1. *Es kann verifiziert werden, dass $SE(3)$ durch eine Matrixgruppe beschrieben werden kann.*

$$SE(3) \simeq \left\{ \begin{pmatrix} \Omega & v \\ 0 & 1 \end{pmatrix} \;\middle|\; \Omega \in SO(3),\, v \in \mathbb{R}^3 \right\}$$

Dies erlaubt eine numerische Beschreibung von Rigid-Body-Motion über lineare Abbildungen [15]

Die Gruppe stellt Bewegungen als Kombination zwischen Rotation und Translation da. Das Auszeichnende Merkmal dabei ist, das Abstände und Orientierung von Punkten durch die Abbildung erhalten bleiben.[15]

Im folgender wird oft über eine starre Umgebungen oder starre Szene geredet. Damit ist nicht gemeint das die Szene unbewegt ist, sondern das Veränderungen nur über Rigid-Body-Motion erfolgen.

2.3.2. Notation

RGB-D-Bilder können durch Abbildungen $I_i : \Omega \to [0,1]^3 \times \mathbb{R}_+,\ \Omega \subset \mathbb{Z}^2$ beschrieben werden. Im Kontext der Bewegungsbestimmung werden Bilder in Paaren betrachtet (I_s, I_t). I_s wird als Ursprungsbild und I_t als Zielbild bezeichnet. Pixel Koordinaten, kurz Pixel, werden durch Elemente aus Ω dargestellt.

$I_i^d : \Omega \to \mathbb{R}_+$ stellt die Funktion da, die nur den Tiefen-Anteil jedes Pixels berücksichtigt und $I_i^g : \Omega \to [0,1]$ die zu jedem Pixel einen Graustufenwert zuordnet, berechnet aus den Farbwerten. Für eine Erleichterung der Notation hilft es eine Funktion zu definieren, die einen Pixel plus Tiefen-Wert als 3 dimensionalen Vector darstellt. $d_i : \Omega \to \mathbb{R}^3, \quad d_i(u,v) \mapsto (u, v, I_i^d(u,v))^T$

2.3.3. Problem Formulierung

Motivation: *Die Ausgangslage ist: Die Kamera hat zu zwei Zeitpunkten Aufnahmen einer Szene gemacht. Diese Aufnahmen sind durch eine beobachtete (gemessenen) Teilmenge von 3-dimensionalen Punkten entstanden. Die Punkte werden jeweils im Koordinatensystemen des Sensors zu dem zugehörigen Zeitpunkt beschrieben.*

Da die Kamera ein starrer Körper ist, verändert sich ihre Position im Raum durch eine Rigid-Body-Motion. Unter der Annahme einer statischen Szene lassen sich die in zwei aufeinanderfolgenden Aufnahmen beobachteten Punkte durch genau eine starre Transformation in SE(3) miteinander in Beziehung setzen.

Damit kann das Problem der Kamerabewegung auf das Bestimmen dieser Transformation zwischen den entsprechenden Punktmengen zurückgeführt werden.

Die die Motivation stellt das Problem geometrisch da. Die konkrete Formulierung des Problems erfolgt aber über die Pixel in den Bilder. Diese Darstellung stimmt mit den durch den Sensor gelieferte Daten übereinstimmt.

Das bestimmten der Rigid-Body-Motion erfolgt über ein Optimierungsproblems, welches durch die gemessenen Daten gegeben ist.

Definition 2. *Sei (I_s, I_t) ein Paar von RGB-D-Bildern. Das Odometrie Optimierungsproblems ergibt sich durch*

$$\min_{T \in SE(3)} \sum_{x \in \Omega} r(x, T)$$

wobei $r : \Omega \times SE(3) \to \mathbb{R}$ eine Fehlerfunktionen ist, die implizit von I_s, I_t abhängt. [21][17]

2.3.4. Warpingfunktion

Ein Frage die sich dabei stellt ist: Wie wird der Übergang von 2-dimensionalen Pixel (Bildkoordinaten) mit Tiefen-Channel zu 3-dimensionalen Punkten (Kamerakoordinaten) und zurück modelliert.

Dies erfolgt mit den Pinhole-Kameramodell. Es stellt ein vereinfachtes Modell einer Bildaufnahme da. Dabei ist die Kamera im Koordinaten Ursprung positioniert. Dabei zeigt z-Achse zeigt nach vorne. Projektionsebene liegt in der Ebene $\{(x, y, 1), x, y \in \mathbb{R}\}$. Der erste Schritt der Aufnahme eines synthetischen Bildes ist, es die Punkte in die Ebene zu transformieren. Der Schritt heißt Perspektivische Teilung. Daraufhin wird die Kamera Intrinsischen Matrix angewandt

$$K = \begin{pmatrix} f_x & 0 & -c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix},$$

wobei Brennweiten f_x, f_y und die Optischen Zentren c_x, c_y. Sie stellt den Übergang von Punkten in der Ebene zu Pixelkoordinaten da. Somit ist die gesamte Abbildung von Kamerakoordinaten in Bildkoordinaten gegeben durch [6]

$$P : \mathbb{R}^3 \to \mathbb{R}^2, \quad \begin{pmatrix} x \\ y \\ z \end{pmatrix} \mapsto \Pi K \frac{1}{z} \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \quad \Pi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Bemerkung: *In dem Process der Perspektivische Teilung gehen die Tiefendaten verloren. Deshalb sind RGB-D Daten nötig um den inversen Schritt zu gehen.*

Der Übergang von Pixel mit Tiefenchannel zu Kamerakoordinaten ergibt sich durch das Anwenden der Inverse der Intrinsischen Matrix. Hier bezeichnet als

$$P^{-1} : \mathbb{R}^3 \to \mathbb{R}^3, \quad x \mapsto K^{-1}(x)$$

Trotz der Bezeichnung ist P^{-1} nicht die Inverse von P.[6].

Mithilfe dieser beiden Abbildungen kann die sogenannte Warping Funktion ω definiert werden, die einen Bildkoordinaten Wechsel durch T darstellt.

$$\omega : \Omega \times \mathbb{R} \to \mathbb{R}^2, \qquad \omega(x, T) := P(T(P^{-1}(x)))$$

Für eine Vereinfachung der Notation kann $Tx = Ax + b,\ A \in SO(3), b \in \mathbb{R}^3$ angesehen werden. T kann auch in homogenen Koordinaten dargestellt werden, doch dann müssen noch hin und zurück konvertiert werden.

Bemerkung 2.2. *Betrachtet man Warping-Funktion fällt auf, dass im Allgemeinen $\omega(x, T) \notin \Omega$. Wenn die einzelnen Komponenten keine ganzen Zahlen sind werden sie gerundet oder es wird interpoliert. Auch werden in der Warping-Funktion normalerweise die Punkte gegen das Viewing-Volume der Ziel-Kamera geclipped, was die Punkte aus der Rechnung entfernt werden.*

Bemerkung 2.3. *Seien I_s und I_t zwei Aufnahmen derselben Szene zu Zeitpunkten s und t, und $T \in SE(3)$ die Transformation der Kamera von s nach t. Ein Pixel $x \in I_s$ mit bekannter Tiefe repräsentiert implizit einen 3D-Punkt p. Die Abbildung $\omega(T^{-1}, x)$ liefert den Pixel in I_t, der denselben 3D-Punkt p darstellt.*
Also stellt $\omega(T^{-1}, x)$ die Transformation (Warping) von Bild I_s zum Bild I_t da.

2.3.5. Fehlerfunktion

Durch die Warping-Funktion wird es ermöglicht das geometrische Problem, über das Abgleichen von Bildern zu formulieren. Man warpt eines der beiden Bilder und berechnet die Übereinstimmung. Es werden nun einige der wichtigsten und meist verwendeten Fehlerfunktionen vorgestellt.

Photometric Intensity Fehler

Der Photometric Intensity Error r_{photo} stellt den Unterschied zwischen den Helligkeitswerten der einzelnen Pixel da. Dafür werden die RGB Werte in Graustufenwerte umgewandelt.

Definition 3. *Die Photometric Intensity Verlustfunktion* $r_{photo} : \Omega \times SE(3) \to \mathbb{R}$ *ist definiert als*

$$r_{photo}(x, T) = (I_t^g(\omega(d_s(x), T)) - I_s^g(x))^2$$

[17, 21]

Hybrid Fehler

Es kann analog zu dem Intensität's Fehler ein Tiefen Fehler definiert werden. Er wird oft in Hybrid Methoden mit dem Photometrischen Fehler verwendet.

Definition 4. *Der Tiefen Fehler* $r_{depth} : \Omega \times SE(3) \to \mathbb{R}$ *ist definiert als*

$$r_{depth}(x, T) = (I_t^d(\omega(d_s(x), T)) - I_s^d(x))^2$$

Sei $\delta in(0, 1)$ *Dann ist der Hybrid-Fehler definiert durch*

$$r_{hybrid}(x, T) = \delta r_{photo}(x, T) + (1 - \delta) r_{depth}(x, T)$$

PointToPlane Fehler

Ein oft genutzter geometrischer fehler ist der Point to Plane (P2P). Dieser besitzt eine bessere Konvergenzgeschwindigkeit zu anderen Tiefen bassierten Fehlern [20].

Dazu wird jedem Punkt in den Kamerakoordinaten von I_t eine Normale bestimmt, die den Tangentialraum der gemessenen Geometrie in jenem Punkt beschreibt. Mithilfe von den Normalen kann ein Abstand zu den Tangentialräumen (affine Hyperebenen) bestimmt werden.[17, 26].

Definition 5. *Sei* $n : \mathbb{R}^3 \to \mathbb{R}^3, \quad x \mapsto n(x) := n_x$ *die Abbildung die einem Punkt eine Normale zuteil. Weiter ist* $x^* = P^{-1}(d_t(\omega(x, T)))$*, der transformierte Pixel im Zielbild, der in den Raum zurück projiziert wurde. Dann ist der Geometrische PointToPlane Fehler* $r_{p2p} : \Omega \times SE(3) \to \mathbb{R}$ *definiert als*

$$r_{p2p}(x, T) = ((TP^{-1}d_s(x) - x^*)^T n_{x^*})^2$$

Dabei werden die Normale aus dem Zielbild berechnet.

2.4. Voxel-Block-Grid TSDF Tracking

Eine direkt Anwendung der visuelle Odometrie ist das TSDF-Tracking. Es stellt eine Fusion von Rekonstruktions und Tracking Algorithmen da. Dabei wird ein internes Modell konstruiert und die Bilder werden auf diesen Modell getrackt. Dies soll den aufbauenden Fehler durch aufeinanderfolgenden visuelle Odometrie entgegenwirken.

2.4.1. VoxelBlockGrid

Die Representation des internen Modells erfolgt über ein Voxel Block Gitter (Voxel-Block-Grid). Voxel können als eine Diskretisierung von räumlichen Koordinaten angesehen werden. Eine direkte Darstellung einer Szene allein über Voxel ist in der Regel nicht sinnvoll, da die Geometrie stark an die Auflösung des Voxelgitters gebunden wäre. Bei niedriger Auflösung würde die Oberfläche treppen- bzw. kantig erscheinen.

Um dieses Problem zu umgehen, speichern die Voxel TSDF Werte.

- *Distance-Field*: Der Wert gibt den Abstand des Voxels zur nächstgelegenen gemessenen Oberfläche an.
- *Signed:* Das Vorzeichen gibt an, ob sich der Voxel vor oder hinter der Oberfläche befindet.
- *Truncated:* Abstände, die größer als ein definierter Schwellwert sind, werden abgeschnitten und nicht gespeichert.

Ein TSDF ist eine glatte Darstellung einer Oberfläche. Des weiteren ist es möglich effizient durch Raymartsching (Raycasting) synthetische Bilder zu der Szene zu erzeugen, was sich für das Tracking als nützlich erweist.[7]

Ein weiteres zentrales Element des Voxel-Block-Grids ist eine Trennung von lokalen und globaler Geometrie. Dafür wird die Szene ist ein grobes Gitter, dem Block-Gitter aufgeteilt. Jeder Block stellt eine lokale Ansammlung von Voxel-Gittern mit höherer räumlicher Auflösung da. Dies erlaubt effektives und effizientes lokales operieren auf dem Gitter.

Bemerkung 2.4. *Das grobe Block-Gitter, sowie das feine Voxel-Gitter werden in der Implementation durch Hashmaps organisiert. Dies kann speicher und cache freundlich in Parallel auf dem GPU oder in Vektorisierten CPU Code umgesetzt werden. Das heißt paralleler und speicher-lokaler Zugriff auf die Voxel. [7]*

2.4.2. Integration

Das Einfügen von RGB-D Bildern in das Modell erfolgt über das sogenannte Integrations Verfahren. Dazu ist die globale Kamera Position relativ zu dem Gitter nötig. Das Vorgehen ist dabei die Voxel $x \in \mathbb{R}^3$ in die Kamerakoordinaten des Bildes zu Transformieren, mithilfe der Kameraposition T und der Intrinsischen Matrix. Dann werden die Voxel-Koordinaten in die entsprechenden Bildkoordinaten umgewandelt. Aus dem Tiefenwert d des Pixel wird dann der Abstandswert berechnet. [7]

$$(u, v, r)^T = K(T^{-1}x) \tag{2.2}$$

$$d = d(u, v) - r \tag{2.3}$$

Durch die Nutzung des Kamera Sichtfeldes, findet eine sogenannte Blockaktivierung statt. Dabei werden nur die Voxel der Blöcke, die im Kamera Sichtfeld liegen in die Berechnung einbezogen. Dadurch wird lokal und effizient auf dem Gitter operiert.

Bemerkung 2.5. *In der Anwendung wird eine Sequenz von RGB-D-Bildern integriert. Dabei werden Voxel mehrfach getroffenen. Das Vorgehen dabei ist ein Updaten des Abstandswertes. Dabei wird häufig eine Gewichtung der gemessenen Abstandswerte vorgenommen. Daraus wird ein kleinste Quadrate Problem konstruiert, was den Abstand als Minimum einer gewichtet Summe der einzelnen Abstände beschreibt. Das Problem kann iterativ mit jedem neuem Abstandswert gelöst werden. [7]*

2.4.3. TSDF-Tracking

Ein VoxelBlockGrid verbunden mit der Integration von RGB-D-Bildern kann für eine Verbesserung des Tracking der Kamera genutzt werden. Dabei wird mithilfe von Raycasting und der letzten bestimmten Kameraposition ein synthetisches Bild erzeugt (siehe Abb. 3.5a). Diese Bilder haben den Vorteil, dass sie auf Grundlage einer glatten und konsistenten Darstellung der Geometrie entstanden sind. Real gemessene Sensor Daten besitzen, aufgrund von Imperfektionen der Sensorik rauschen. Das wirkt sich negativ auf das Lösen des Optimierungsproblems aus.

Auch kann das Driften der Positionen verbessert werden. Das Modell gibt einen Referenzpunkt an dem sich orientiert werden kann.[26]

3. Masken basiertes Tracking und Szenen Rekonstruktion

Die visuelle Odometrie und auch das Erstellen eines Voxel-Block-Grids der Szenen ist nur möglich, wenn die gemessene Szene statisch ist. In dem Kontext der Rekonstruktion der Futterstelle (siehe 2.1) ist dies jedoch nicht gegeben. Das heißt zusätzlich befinden sich dynamische Objekte, mit eigenen Bewegungs-Trajektorien, in der Szene. Somit ist die beobachtete Geometrie nicht mehr star.

Bemerkung 3.1. *Odometrie ist fundamental eine Optimierung über Rigid-Body-Motion. Das heißt im geometrische Sinne wird versucht die Transformation der Geometrie durch eine starre Bewegung zu approximieren. Wenn die Transformation nicht starr ist, ist das Modell Fehlerhaft. Auch die Integration in ein Grid wird dadurch beeinflusst. Dies führt zu fehlerhafter und artefalthaltiger Geometrie in dem Modell, was auch das TSDF-Tracking verschlechtert.*

Idee: *Wenn eine Untermenge der Punkte sich starr verhält, kann versucht werden nur über diese Menge an Punkten im Odometrie Optimierungsproblems zu optimieren. Auch kann versucht werden nur die Menge in das Szenen-Modell integriert werden.*

Im Folgenden werden die starren Untermengen über Menge von Pixeln im Ursprungs- und Zielbild beschrieben. Sie stellen die korrespondierenden Pixel zu der starren Menge da. Idealerweise sind sie durch die Projection P der Punkte der Menge gegeben. Die Pixelmengen sind im folgenden als Eingabe in den Algorithmus gegeben und werden mit $M_s, M_t \subset \Omega$ bezeichnet.

3.1. maskierte visuelle Odometrie

Damit die visuelle Odometrie auf die Einbindung von Masken angepasst werden kann, ist es nötig das zugrunde legende Optimierungsproblem zu betrachten und wie es gelöst wird.

3.1.1. Optimierung visueller Odometrie

Die Besonderheit des Problems 2 ist die Fehlerfunktion. Sie besitzt als Definitionsbereich eine Gruppe besitzt. Gängige elementare Optimierungsverfahren im $\mathbb{R}^n$ basieren auf der Berechnung von Ableitungen (Ableitungen höherer Ordnung), um Abstiegsrichtungen der Fehlerfunktion zu finden. Anhand der Abstiegsrichtung wird lokal ein Schritt getätigt. Somit wird iterativ versuch eine lokale Lösung zu finden. In dem Fall

Input: RGBD Bilder I_s, I_t, Anfangs Transform T_0
Output: Angenäherte Transform T von I_j zu I_i
$T \leftarrow T_0$;
repeat
  Parallel for each pixel x in Ω: $J, r \leftarrow CalcJacobian(x, T)$
  Parallel reduction to accumulate J,r:

$$H = \sum J^\top J, \quad b = \sum J^\top r$$

  Löse Gleichung:

$$\delta = -H^{-1} b$$

  Update Transform:

$$T \leftarrow \exp(\delta) \cdot T$$

until *Konvergenzkriterium erfüllt*;
return T;

Algorithmus 1: Gauss-Newton Verfahren für visuelle Odometrie (vgl.[1, Kap.8.4][26, 17])

einer Gruppe ist jedoch weder trivial, was mit einer Ableitung, Abstiegsrichtung und Schritt gemeint ist.

Es stellt sich heraus, dass $SE(3)$ bestimmte Glattheits-Eigenschaften besitzt. Sie kann als glatte Untermannigfaltigkeit des $\mathbb{R}^{16}$ angesehen werden. Dies ermöglicht Methoden der Differenzial Geometrie nutzen um Konzepte aus dem $\mathbb{R}^n$ zu übertragen auf $SE(3)$ zu übertragen. Auf diese Theorie wird jedoch in dieser Arbeit nicht genauer eingegangen. Eine mathematisch genaue Betrachtung eines Teils der Theorie ist zu finden in [12, Kap.9][1, Kap.8.4][15, Kap.2][15, Apd.A]

Das Problem stellt ein verallgemeinertes nichtlineares kleinste Quadrate Problem da. In der Anwendung wird es entweder Gauss-Newton oder Levenberg-Marquard Iterationen angewendet um Ableitungen höherer Ordnungen zu vermeiden. Eine genauer Beschreibung dieser Algorithmen und der Optimierungstheorie hinter ähnlichen Problemen ist in[1, Kap.8.4] beschrieben.

Im weiteren wird weiterhin der Begriff der Jacobimatrix genutzt, obwohl er nur das Analog zu dem mathematischen Objekt darstellt. Schrittrichtungen sind selber Matrizen, die eine Infinitesimale Rotation und Translation darstellen (siehe [15, Kap.2]). Als Schrittfunktion dient die Matrixexponentialfunktion.

Bemerkung 3.2. *In den Buch [15, Kap.2] ist eine konkrete physisch motivierte Herleitung der Form der Abstiegsrichtungen zu finden. Dabei wird die Matrixexponentialfunktion durch das lineare Differenzialgleichungs Systeme mit genanten Abstiegsmatrizen motivierte.*

Der Algorithmus zum Lösen der Odometrie ist beschrieben in 1. Dabei wird die Linea-

rität der Jacobimatrix stark ausgenutzt um sie für jeden Pixel separat zu berechnen. Die Matrizen-Anteile werden in einer Matrix parallel gesammelt und das GaussNewton Gleichungssystemen wird gelöst. Die Jacobimatrizen für die konkreten Fehler sind gegeben durch folgenden Formeln (vgl. [17, S.146, Fom.28-30]).

$$J_{r_{p2p}}(x,T) = n_{x^*}^T J_T(P^{-1}d_s(x)) + n_{x^*}^T J_T$$

$$\begin{aligned} J_{r_{photo}}(x,T) &= 2 * \left(\sqrt{r_{photo}(x,T)}\right) \nabla I_s^g(\omega(x,T)) J_P(TP^{-1}d_s(x)) J_T(P^{-1}d_s(x)) \\ &= 2 * \left(\sqrt{r_{photo}}\right) \nabla I_s^g J_P J_T \end{aligned}$$

$$\begin{aligned} J_{r_{depth}}(x,T) &= 2 * \left(\sqrt{r_{depth}(x,T)}\right) \nabla I_s^d(\omega(x,T)) J_P(TP^{-1}d_s(x)) J_T(P^{-1}d_s(x)) \\ &= 2 * \left(\sqrt{r_{depth}}\right) \nabla I_s^d J_P J_T \end{aligned}$$

$\nabla I_s^d, \nabla I_s^g$ sind die Gradienten der Bildfunktionen, J_P die Jacobimatrix für die Projektionsmatrix P und J_T die Jacobimatrix in $SE(3)$ in dem Punkt T.

Die Ausdrücke ∇I_s^g und ∇I_s^d werden über sogenannte Sobel-Filter berechnet. Sie stellen Convolution zu den Sobel-Kernen da und approximieren den Gradienten. J_P kann analytische durch die Kamera Intrinsische Werte berechnet werden. Für den Ausdruck J_T existiert auch eine analytische Representation die hier jedoch nicht angegeben wird.

Bemerkung 3.3. *Für die Optimierung muss die Annahme getroffenen, dass die Kamera Punkte auf einer differenzierbare Oberfläche aufnimmt. Nur durch diese Annahme ist es sinnvoll über normalen in Punkten zu sprechen, da sonst nicht der Tangentialraum definiert werden kann. ∇I_s^g und ∇I_s^d sind auch an diese Annahme gebunden. In der Anwendung werden die Elemente stumpf approximiert und der Algorithmus wird angewendet.*

Bemerkung 3.4. *Um die Konvergenz des Algorithmus zu verbessern, wird eine Multiskalen-Implementierung eingesetzt. Dabei wird eine sogenannte Bildpyramide der Eingabebilder über nn Stufen erzeugt. Die Auflösungen der Stufen unterscheiden sich jeweils um den Faktor 2. Die Verarbeitung beginnt auf der niedrigsten Auflösungsebene, da hier grobe Bewegungen effizient erfasst werden können. Die in dieser Stufe geschätzten Parameter dienen als Initialisierung für die nächsthöhere Auflösungsebene. Auf diese Weise werden die Ergebnisse mit jedem Schritt in der Pyramide sukzessive verfeinert. [6]*

3.1.2. Masken Einbindung

Eine zentrale Frage ist, wie die Masken in den Optimierungsprozess integriert werden. Die Ursprungsmaske M_s kann unmittelbar berücksichtigt werden: Für Pixel $x \notin M_s$ werden weder Ableitungen noch Residuen berechnet, sodass diese störenden Punkte nicht in den Rekonstruktionsraum projiziert werden. Damit gehen sie nicht in das akkumulierte Gleichungssystem (siehe 1) ein.

Die Zielmaske M_t bleibt zwar unverändert, dennoch muss in jeder Iteration geprüft werden, ob die durch die Warping-Funktion verschobenen Pixel innerhalb der durch M_t definierten gültigen Bereiche liegen. Durch die Verschiebung kann es vorkommen, dass ursprünglich gültige Bildbereiche auf ungültige Regionen abgebildet werden. Nur wenn die Transformation T exakt der tatsächlichen Bewegung entspricht, werden valide Pixel konsequent auf valide Pixel abgebildet. Während der Optimierung ist dies jedoch nicht gewährleistet.

Die Lösung des Problems ist es nur valide Pixel aus I_s, die durch T auf valide Pixel in I_t abgebildet werden in die Optimierung einzubeziehen. Die maskierte Version des Algorithmus's 1 ergibt sich aus der Anpassung der Funktion 'CalcJacobian'.

Input: Pixel x, Transform T, Masken M_s, M_t
Output: Ableitung J
if $x \in M_s$ **then**
  $J \leftarrow 0$;
  $r \leftarrow 0$;
end
else
  if $\omega(d_s(x), T) \in M_t$ **then**
    $J \leftarrow 0$;
    $r \leftarrow 0$;
  end
  else
    $r \leftarrow r(x)$;
    $J \leftarrow D_r$
  end
end
return J, r;

Algorithmus 2: CalcJacobianMaskout

Bemerkung 3.5. *Bei Berechnung von $n_x, \nabla I_t^g, \nabla I_t^d$ muss vorsichtig behandelt werden. Wenn die Randpunkte der starren Geometrie betrachtet werden, haben lokale Punkte immer Einfluss auf die Berechnung von $n_x, \nabla I_t^g, \nabla I_t^d$. Wenn der Einfluss von Störregionen komplett aus der Optimierung entfernt werden soll, müssen die Ableitung von Randpunkten mit Vorsicht berechnet oder gleich 0 gesetz werden.*

3.2. Maskout TSDF Integration

Das in Abschnitt 2.4.2 angegeben Integration-Vorgehen kann einfach durch die Einbindung von Masken angepasst werden. Wenn ein Transformierter Voxel in das Bild projiziert wird, kann im Integration-Verfahren der Distanzwert entsprechen gewichtet

werden. Das erfolgt wie in der Bemerkung 2.5 erläutert.

Input: Aktive Blockmenge A, Tiefenbild I_t, Kamerapose T, Trunkationsdistanz μ, Maske $M_t \subset \Omega$
foreach *Block* $b \in A$ **do**
  foreach *Voxel* $v \in b$ **do**
    $p_w \leftarrow$ Weltposition des Voxelzentrums;
    $p_c \leftarrow T^{-1} \cdot p_w$;
    if p_c *außerhalb des Clipping volume* **then**
      continue
    end
    $(u, v) \leftarrow$ Projektion von p_c ins Bild;
    if (u, v) *inerhalb von* M_t **then**
      $w_{meas} \leftarrow 0$
    end
    $sdf \leftarrow \text{clamp}(z_{meas} - p_c.z, -\mu, \mu)/\mu$;
    if $|sdf| > 1$ **then**
      continue
    end
    $tsdf_{neu} \leftarrow \frac{tsdf_{alt} \cdot w_{alt} + sdf \cdot w_{meas}}{w_{alt} + w_{meas}}$; `// Lösen des weighting Problems 2.5`
    $w_{neu} \leftarrow \min(w_{alt} + w_{meas}, w_{max})$; `// Gewichtungswert update`
    update_voxel(v, tsdf$_{neu}$, w_{neu});
  end
end

Algorithmus 3: maskierte TSDF-Integration in aktivierte Voxel-Blöcke

Bemerkung 3.6. *Da in der Anwendung die Masken durch ein neuronales Netz generiert werden, ist keine Pixel genaue Segmentierung zu erwarten. Um das Problem zu lösen können die binären Masken durch Unsicherheits-Bilder ersetzt werden. Der im Voxel gespeicherte Gewichtungswert kann als Vertrauen in die Geometrie interpretiert werden und wenn er unter einem bestimmten Schwellenwert ist im Modell ignoriert werden.*

Der Masken basierte TSDF-Tracking Algorithmus ergibt sich daraus wie folgt.

Input: Tiefenbild I_t, Maske I_s, Kamera-Intinsics K, Vorpose T_{t-1}, TSDF-Volumen V
Output: Aktualisierte Pose T_t, aktualisiertes Volumen V
$\hat{I_{t-1}} \leftarrow \text{Raycast}(V, K, T_{t-1})$;
$T_t \leftarrow MaskoutOdometire(\hat{I_{t-1}}, I_t, M_t, K)$;
$MaskoutIntegrate(V, I_t, M_t, T_t)$;
return T_t, V;

Algorithmus 4: maskiertes TSDF-Tracking

Bemerkung 3.7. *In der Optimierung werden $\nabla I_s^d, \nabla I_s^g, n_x$ im Zielbild bestimmt. Aus diesem Grund werden die synthetisch erzeugten Bilder als Zielbild im Algorithmus 1 benutzt. Es ist auch keine Maske Zielbild benötigt, da es aus dem bereinigten internen Modell entstanden ist.*

3.3. Auswertung

Die grundlegenden Algorithmen aus dem Kapitel 2 sind bereits in einer Modulare Form in der Opensource Bibliothek Open3D implementiert. Das Softwaresystem stellt optimierte und parallelisierte Algorithmen und Datenstrukturen, für das Arbeiten mit 3D Daten in vielseitiger Form bereit.

Das Tensor-Api bietet ein Backend für das Interfacing mit Maschine-Learning Bibliotheken wie PyTorch und TensorFlow bereit. Dies erleichtert das Pipelining und Erstellen von SLAM-Systemen verbunden mit Methoden des maschinellen.[26]

Die Implementation der angepassten Algorithmen ist direkt im Tensor-Api von Open3D integriert, was eine einfache Einbindung eines Segmentierungsmodells ermöglicht. Alle Algorithmen liegen in einer parallelisierter Version als vektorisierten CPU code und GPU CUDA-Kernels bereit.

3.3.1. Testdatensatz

Im Kontext der Rekonstruktion von der Futterstelle von Kühe (siehe 2.1), stehen keine präsisen wahren Trajektorien für die Evaluation zur Verfügung. Als Test wird deshalb ein RGB-D-Slam Benchmark der Technischen Universität München genutzt [22]

Dieser enthält eine Vielzahl unterschiedlichen, mit einer RGB-D Kamera aufgenommenen Szenen, die gezielt unterschiedliche Schwierigkeiten des Slam versuchen zu simulieren. Der hier relevante Teil des Datensatzes ist der 'Dynamic Objects' Teil. Dort wird eine Tisch-Szene gefilmt, mit Menschen bewegten Menschen. Dabei werden folgende Szenen betrachtet:

- *static xyz:* Die Kamera bewegt sich durch Translations-Bewegung und filmt einen statischen Tisch-Szene
- *static rpy:* Die Kamera bewegt sich durch Rotation-Bewegung und filmt einen statischen Tisch-Szene
- *walking static:* Die Kamera bewegt sich nicht und filmt Tisch-Szene mit bewegenden Menschen
- *walking xyz:* Die Kamera bewegt sich durch Translations-Bewegung und filmt Tisch-Szene mit bewegenden Menschen
- *walking rpy:* Die Kamera bewegt sich durch Rotations-Bewegung und filmt Tisch-Szene mit bewegenden Menschen

Der Datensatz stellt keine Masken für die Menschen in den Bildern zur Verfügung. Aus diesem Grund wurden automatisch Masken für die Menschen erzeugt, mithilfe des Segmentierungsmodell Detectron2. [25] Dabei liegen die Masken in unbearbeiteten From vor.

3.3.2. Metriken

Fur das Evaluieren von SLAM Algorithmen werden mehrere Metriken genuzt. Dabei wird eine Referenztrajektorie (Ground Truth) $\{P_t^{gt} | t \in I\}$ mit einer geschätzten Trajektorie $\{P_t^{est} | t \in I\}$ verglichen. $I := \{0, \ldots, n\}$ die Menge der Zeitindizes der aufeinanderfolgenden Messungen (z. B. Bildframes).

Für den Vergleich der Position lassen sich im Wesentlichen zwei Fehlerarten messen: Translationsfehler und Rotationsfehler. Der Unterschied zwischen zwei Posen lässt sich durch eine Fehlertransformation darstellen, aus der sowohl der Translations- als auch der Rotationsanteil berechnet werden kann.

ATE

Der Absolute Trajectory Error (ATE) misst die Genauigkeit der gesamten Trajektorie. Zu jedem Zeitpunkt i wird die Fehlermatrix $P_i^{est}(P_i^{gt})^{-1}$ betrachtet. Aus dieser Transformation werden separat die Rotations- und Translationsanteile extrahiert (vlg. [22]):

$$ATE_{rot}(\{P_t^{gt}\}, \{P_t^{est}\}) = \frac{1}{|I|} \sum_{i=1}^{n} rot(P_i^{est}(P_i^{gt})^{-1})$$
$$ATE_{trans}(\{P_t^{gt}\}, \{P_t^{est}\}) = \frac{1}{|I|} \sum_{i=1}^{n} trans(P_i^{est}(P_i^{gt})^{-1})$$

Bemerkung 3.8. *Für den ATE werden in der Regel die Trajektorie zueinander ausgerichtet. Dies soll eine bessere Vergleichbarkeit der Trajektorien liefern unabhängig von der Startposition.[22]*

RPE

Der Realtive Pose Error (RPE) ist ein Maß für den Drift eines SLAM-Algorithmus. Er beschreibt den Fehler, der in den relativen Bewegungen zwischen aufeinanderfolgenden Zeitpunkten entsteht. Hierzu werden die relativen Transformationen der Ground-Truth- und der Schätztrajektorie verglichen (vlg. [22]):

$$RPE_{rot}(\{P_t^{gt}\}, \{P_t^{est}\}) = \frac{1}{|I|} \sum_{i=1}^{n} rot((P_{i-1}^{gt})^{-1} P_i^{gt} \left((P_{i-1}^{est})^{-1} P_i^{est}\right)^{-1})$$
$$RPE_{trans}(\{P_t^{gt}\}, \{P_t^{est}\}) = \frac{1}{|I|} \sum_{i=1}^{n} trans((P_{i-1}^{gt})^{-1} P_i^{gt} \left((P_{i-1}^{est})^{-1} P_i^{est}\right)^{-1})$$

3.3.3. maskierte Odometrie Auswertung

Die Testkonfiguration basiert auf dem reinen Tracking mit dem Algorithmus der visuellen Odometrie-Algorithmus, mit einer gewählten Fehlerfunktion und Maskierung's Variante. Die im Kapitel 2.3.5 vorgestellen Fehlerfunktionen werden im folgenden Verglichen. Der Algorithmus wird hier in einer Multiskalen-Architektur eingesetzt (vlg.

Bem. 3.4). Dafür werden drei Bild-Pyramidenstufen verwendet, mit jeweils zehn Iterationen pro Stufe. Dabei wurden mehrere Tracking-Durchläufe simuliert, jedoch liegt dabei die Run-to-Run Varianz unter 0.1% für RPE und ATE. Deshalb können die Werte als deterministisch angesehen werden.

Das genutzte Gerät bei den Versuchen ist die Grafikkarte. Dies hat den Grund, dass die Laufzeit erheblich schneller im Vergleich zu dem CPU ist. Die Grafikkarte und auch Allgemein ein CUDA taugliches ist die Zielhardware für die genanten Algorithmen.

Laufzeit

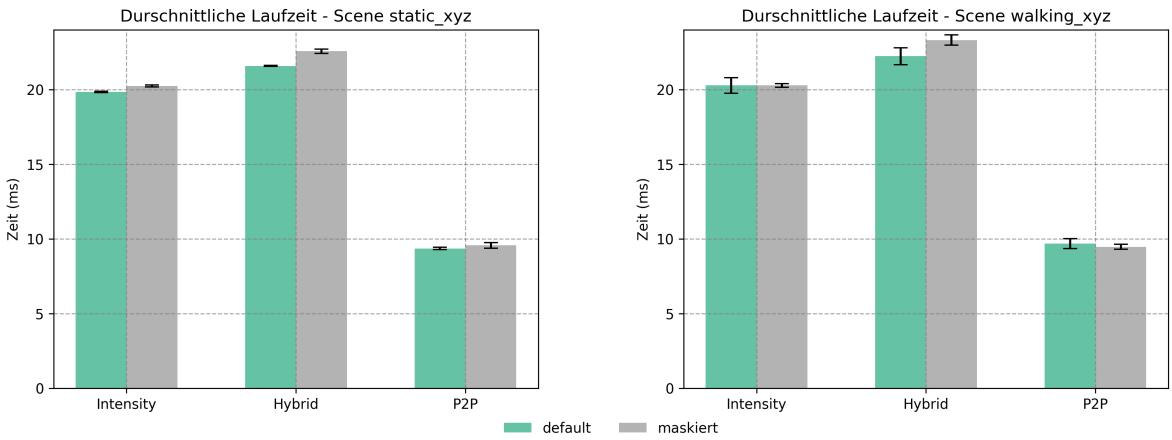


Abbildung 3.1.: durchschnittliche Laufzeit visueller Odometrie. Vergleich von statischer und dynamischer Umgebung

Aus der Grafik 3.1 lässt sich schließen, dass jede Variante des Algorithmus es möglich ist schneller, als die für das dichte Tracking übliche Bildrate von 30 FPS, zu laufen. Dabei ist die Laufzeit der Intensität's und Hybrid Fehlerfunktion deutlich höher als die des P2P. Dies kann durch eine erhöhte Speicher und Speicherbandbreiten erklärt werden. Auch ist der Rechenaufwand zum Bestimmen der Gradientenbildern über die Sobel-Filter pro Pyramiden-Stufe, ein Faktor.

Der Unterschied zwischen der maskierten Variante und der Open3D Implementation ist dagegen kleiner. Das kann mit dem kleineren Speicheraufwand der Masken erklärt werden. Diese werden als 1-Byte pro Pixel Buffer abgespeichert.

In statischen Szenen ist die Laufzeit der maskierten Variante geringfügig größer. Jedoch führt die Verringerung der Jacobimatrix Berechnungen bei dem P2P in dynamischen Szenen zu einem Laufzeit Ausgleich. Die deutliche Speicher Beanspruchung führt für den Intensity und Hybrid-Fehler zu keinen großen Veränderungen.

Bemerkung 3.9. *In der Laufzeit ist ein nichtlinearer Anstieg im Vergleich zu Speicherverbrauch zu erkennen zwischen P2P und den anderen Fehlern. Dies kann mit verbesserten Caching versucht werden zu erklären, da weniger Daten aus verschiedene Datenbuffer pro Kernel benötigt werden.*

Drift Analyse

Visuelle Odometrie bildet einen zentralen Baustein zur Bewegungsbestimmung aus Bilddaten, ist jedoch nicht direkt mit einem SLAM-System vergleichbar. Dies zeigt sich auch in Tabelle A.2: Eine durchschnittliche Abweichung von rund einem halben Meter ist für eine präzise Rekonstruktion ungeeignet. Für die Bewertung eines solchen Algorithmus ist daher der RPE angebracht.

In den Daten A.1 ist ein klarer Trend erkennbar: Beim Translationsfehler weist das P2P-Verfahren einen geringeren Drift auf, unabhängig davon, ob der Bewegung eine Translation oder Rotation zugrunde liegt.

Für den Rotationsdrift zeigt sich ein ähnliches Muster, allerdings weniger ausgeprägt, sodass dies nicht zwingend als Beleg für eine höhere rotatorische Robustheit des P2P gewertet werden kann. Dagegen sprechen insbesondere Szenen mit starker Rotation und ohne dynamische Objekte („static rpy“), in denen das P2P-Verfahren deutlich schlechter abschneidet.

Auf den statischen Szenen verhalten sich die maskierten Methode gleich zu den un-

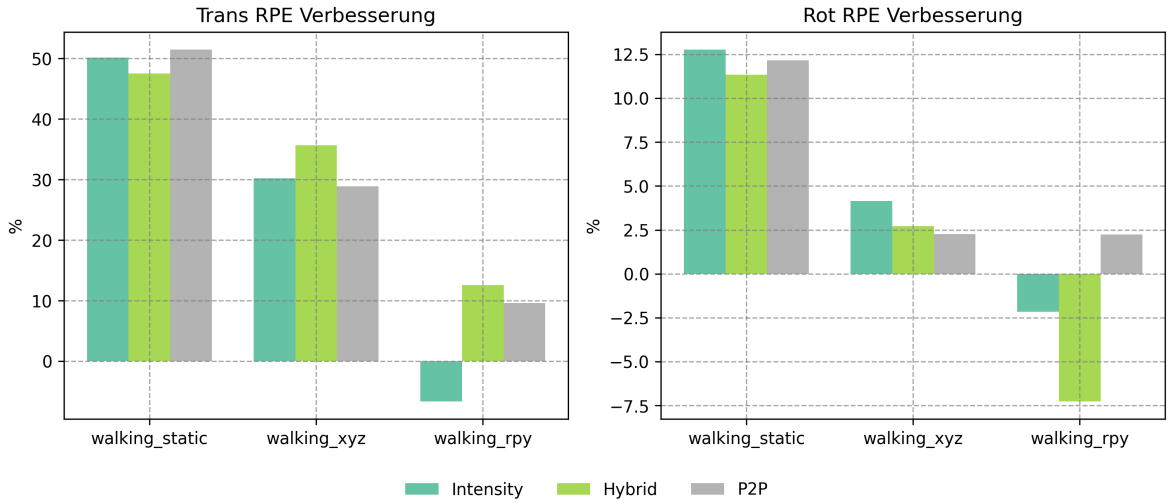


Abbildung 3.2.: Prozentuale Verbesserung des RPE durch maskierten Odometrie

maskierten. Dies liegt daran, dass die Iteration ohne heraus maskierte Pixel zu den unmaskierten degenerieren.

In den dynamischen Szenen zeigt sich eine deutliche Verbesserung des RPE (siehe

Abb. 3.2). Eine Ausnahme bilden die Rotationsszenen: Hier weist ausschließlich das P2P-Verfahren eine Verbesserung auf.

Aus den dynamischen Szenen mit unbewegter Kamera lässt sich ableiten, dass das Maskieren zu einer stabileren Variante des Algorithmus führt. Unter dieser Annahme könnte das Verhalten in den Rotationsszenen als ein generelles Problem der visuellen Odometrie bei starker Rotation interpretiert werden. Unterstützt wird diese Annahme durch den in diesen Szenen erhöhten translationalen Drift.

3.3.4. TSDF-Tracking Auswertung

Der Testaufbau für das TSDF-Tracking ist ähnlich zu dem der visuellen Odometrie. Dabei wird für die in dem Algorithmus 4 genutzte visuelle Odometrie sie selbe Konfiguration genutzt. Das Raycasting Verfahren ist dabei das Standart-Verfahren aus [7]. Die Vergleichs-Version des TSDF-Tracking in Open3D nutzt zudem auch ein Gewichtungswert pro Voxel um die Stabilität zu erhöhen. Auch in diesem Aufbau wird das Tracking mehrfach auf der gleichen Szene simuliert.

Laufzeit

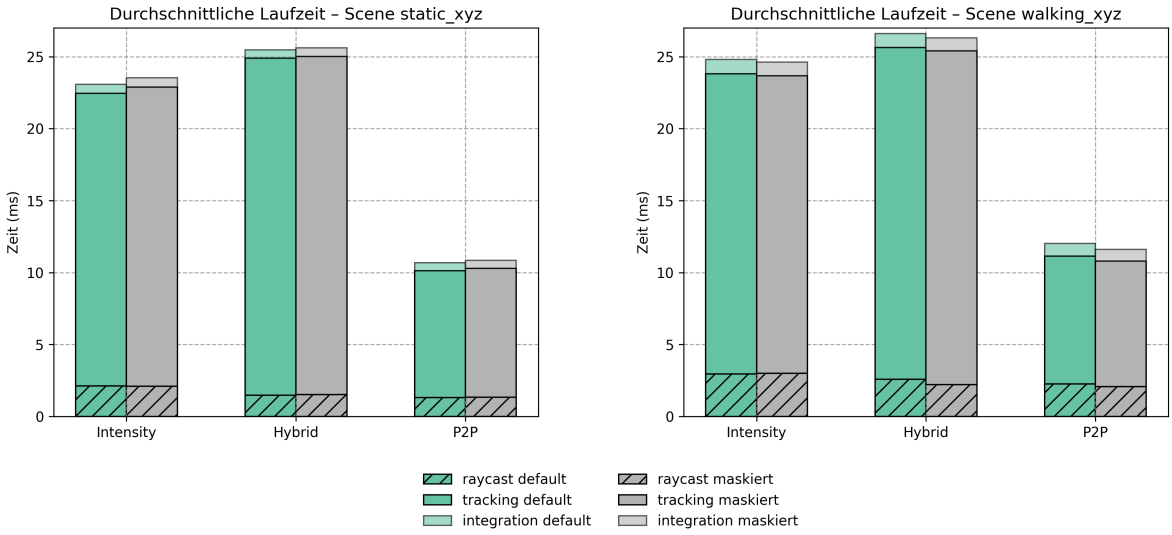


Abbildung 3.3.: Durchschnittliche Laufzeit des TSDF-Trackings

Ein Großteil der Laufzeit pro Iteration wird von dem Tracking in Anspruch genommen (vgl. Abb. 3.3). Dementsprechend ist die das Verhältnis der Laufzeiten fur die verschiedenen Fehlerfunktionen vergleichbar zu Abb. 3.1.

Ein weiteres bobachtes Phänomen ist: Die durchschnittliche Raycasting und Integration Zeit unterscheidet sich zwischen statischer und dynamischer Szene. Ein Grund für das Verhalten kann durch die erhöhte Anzahl von aktivierten Blocks und Voxel gefunden

werden. In den dynamischen Szenen entsteht mehr fehlerhafte Geometrie, die einen negativen Einfluss auf die Rechenzeit des Raycasting under Integration haben.

Auch jeder TSDF-Algorithmus Variante ist das Echtzeit-Tracking mit einer Bildrate von 30 FPS möglich (auf dem GPU).

Tracking-Verlust

Wenn die Daten in den Tabllen A.1 und A.3 betrachtet werden, fällt auf: Es bestehen starke Schwankungen in ATE und RPE selbst auf den statischen Szenen. Besonders in den ATE-Werten A.4 sind großen Schwankungen auf der gleichen Szene zu beobachten.

Dies ist auf ein generelles Problem des Trackings mit einem TSDF-VoxelBlockGrid zurückzuführen, dem Tracking-Verlust. Der Voxel Weighting-Mechanismus, der für die Konsistenz des Modells zuständig ist, kann zu einem Verlust der Referenz von Kamera zu Modell führen. Durch schnelle Bewegungen kann es passieren, dass Kamerasichtfeld auf noch invalide Geometrie gerichtet ist. Dadurch entstehen unvollständige synthetische Bilder. Das Tracking auf Bildern mit nur wenig validen Pixel kann zu starken Drift, sowie singulären Gleichungssystemen in der visuellen Odometrie führen.

Das Verlieren der Referenz für dazu, dass die Kameraposition beliebig driftet bis es wieder ein Modell erstellt wird auf dem getrackt werden kann. Das Ergebnis der Rekonstruktion ist dabei Menge von getrackten Fragmenten in unterschiedlichen Orientierung und Abständen.

Ein Kriterium um einen Tracking-Loss direkt aus den Daten abzulesen, ist eine hohe Varianz in dem ATE. Durch das nahezu zufällige Driften entstehen stark unterschiedliche Trajektorien.

Fehlerfunktionen

Wenn die RPE und ATE Werte auf den statischen Sznen verglichen werden, ist deutlich zu erkennen, das der P2P-Fehler sich am stabilsten verhält. Dies ist gut an der Varianz des ATE zu erkennen. Der Intensität's-Fehler ist stark anfällig fur einen Tracking-Verlust (auf jeder Szene). Dies ist auch zu erwarten. Die synthetisch erstellten Bilder simulieren nicht mit akkurate Licht- und Schatten und somit ist ein Vergleich von diesen nicht sinnvoll.[7]

Ähnliche Probleme sind auch in der Hybrid-Methode zu erkennen. Durch das Einbeziehen der Tiefendaten wird das Tracking verbessert, jedoch tritt auch in der Basis Szenen „static rpy“ein Tracking-Loss auf. Beide Fehlerfunktionen sind nicht für das TSDF-Tracking geeignet.

Die Fehlerfunktion, die von dem Model profitiert ist die P2P. Aus der Abb. 3.4 ist eine deutlicher Verbesserung des Drifts zu erkennen. Im folgenden wird für das Tracking nur

noch der P2P betrachtet.

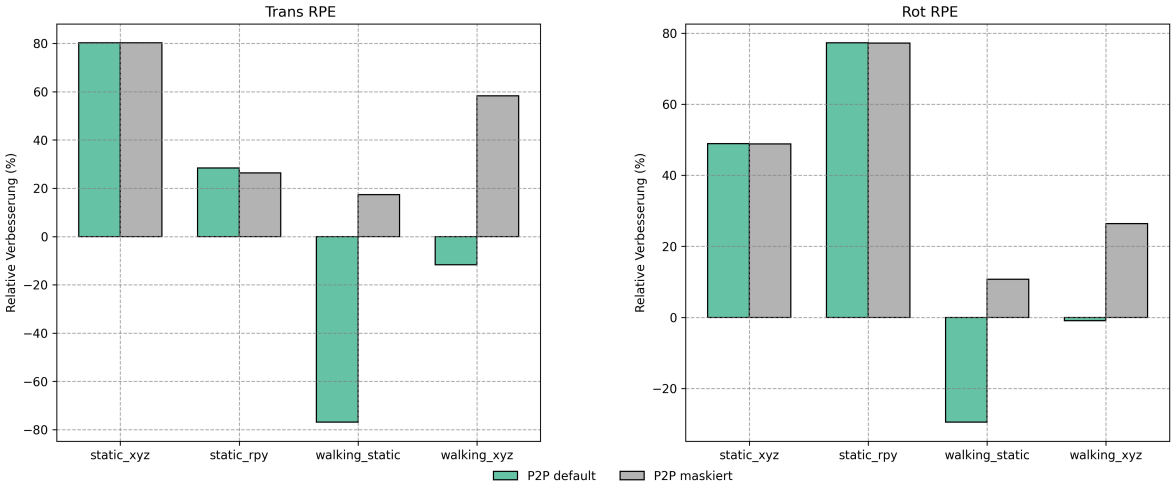


Abbildung 3.4.: Relative Verbesserung des Drifts der P2P-Fehlerfunktion

Dynamische Szenen

In dynamischen Szenen kommt ein weiterer Faktor dazu, der zu einem Tracking-Verlust führen kann. Die Qualität des Trackings hängt stark von der der geraycasteten-Bilder ab. Wenn das „Standard“TSDF-Tracking auf den dynamischen Szenen angewendet wird führt dies zu einer Verschlechterung des interne Modell an dem sich das System orientiert (vgl. Abb. 3.5).

Aus den Varianzwerten des ATE auf den dynamischen Szenen ist erkenntlich, dass das „Standard“TSDF-Tracking auf jeder das Tracking verliert. Am besten ist dies auf der Szene „walking static“zu erkennen. Trotz keiner Kamera Bewegung entsteht ein Tracking-Verlust. Im der Abb. 3.5b ist eine Verschiebung der Szene deutlich zu erkennen.

Die maskierte Variante des P2P Trackings gelingt es selbst auf den dynamischen Szene eine Tracking-Loss zu vermeiden. Dabei steht die dynamische Roationsbewegungen-Szene hervor. Die Verbindung aus rotation und bewegter Szene erschwert das Tracken immens. Wie schon in Section 3.3.3 betrachtet hat der visuelle Odometrie Algorithmus Schwierigkeiten auf der Rotations-Szene.

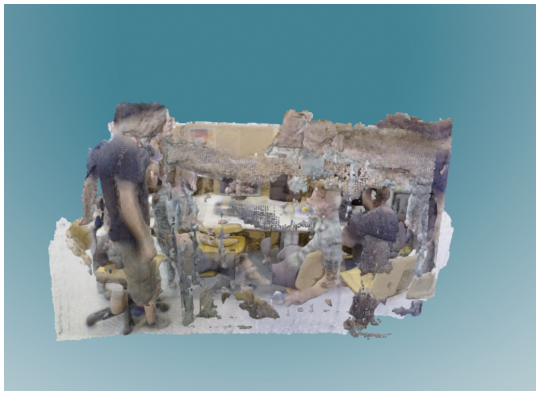
3.3.5. Bewertung

Die beschrieben Experimente zeigen deutlich das der P2P den Intensität's und Hybrid Fehler überlegen ist. Das ist leicht in der visuelle Odometrie zu erkennen. Inder Laufzeit ist die Fehlerfunktion auch deutlich überlegen. Das Muster setzt sich auch aud das TSDF-Tracking fort. Die Trend ist jedoch erwartbar, da die synthetischen Bilder das Tracking per Graustufen behindert.

(a) Raycast-Frame maskierte Integration

(b) Raycast-Frame Open3d standart Integration

(c) internes Modell erstellt durch maskierte Integration



(d) internes Modell erstellt durch Open3d standart Integration

Abbildung 3.5.: Integrations Algorithmen Vergleich

Die maskierten Varianten der Methoden stellen eine deutliche Verbesserung in den einzelnen Algorithmen da. Besonders ist dieser Trend bei dem TSDF-Tracking zu erkennen. Es ermöglicht erst eine Fortsetzung des Verfahrens.

Die Ergebnisse zeigen jedoch auch, dass weder die visuelle maskierte Odometrie noch das TSDF-Tracking ein zuverlässiges und robustes Tracking gewährleisten. Wie in Tabelle A.2 ersichtlich, ist die reine Odometrie stark fehleranfällig. Das maskierte TSDF-Tracking mit der P2P-Fehlerfunktion bietet zwar eine deutliche Verbesserung, leidet jedoch unter hoher Instabilität: Bei langsamer Bewegung und geringer Rotation arbeitet es zuverlässig, während stark dynamische Bewegungen zu einem vollständigen Trackingausfall führen.

Somit stellen beide Verfahren für sich genommen kein robustes SLAM-System dar. Sie können jedoch als Komponenten in einem System eingesetzt werden, das mehrere Tracking-Algorithmen kombiniert und deren Ergebnisse fusioniert. Ein solches Zusam-

menspiel unterschiedlicher Verfahren mit jeweils eigenen Stärken und Schwächen ist charakteristisch für leistungsfähige SLAM-Systeme. Ein möglicher Ansatz dabei wäre das Tracking durch die visuell basierenden Fehlerfunkionen zu überwachen um einen möglichen Trackingverlust zu erkennen. Das führ jedoch in das diverse und umfangreiche Feld der vSLAM-Systeme welche hier nicht betrachtet werden.

Für den Einsatz im Stahlumfeld ist das TSDF-Tracking trotz der genannten Einschränkungen geeignet. Der Roboter auf dem das System montiert ist bewegt sich mit einer Transitiven Bewegung. Auch ist die Geschwindigkeit des des Roboters gering. In diesem stellt es eine gute Methode da.

4. Semantische Bild Segmentierung

In dem Gebiet des vSLAM sind Deep Learning Methoden essenziell für das Tracking in dynamischen Umgebungen. Dabei sind Bild-Segmentierungs Netzwerke ein wichtiger Bestandteil.

Bei der Bildsegmentierung wird unterschieden zwischen der Genric Image Segmentation (GIS) und Promtable Image Segmentation (PIS). Das automatische Generieren von Masken dynamischer Objekten fällt ind das Gebiet des GIS (vgl. [27]). Grundlegende Problemstellungen sind dabei:

- **Semantische Segmentierung:** Jeder Pixel in einem Bild wird ein Klassenlabel zugeteilt
- **Instanzen Segmentierung:** Pixel werden in zusammenhängende Regionen gruppiert
- **Panoptic Segmentation:** Verbindung Instanzen und semantischer Segmentierung

Bei der Rekonstruktion der Futterstelle ist es nicht erforderlich zwischen den Kühen zu unterscheiden, deshalb wird hier die Semantischen Segmentierung betrachtet.

4.1. Datensatz

Der Datensatz ist eine Menge an Kamerafahrten, die mit an einem Tag in einem Stall aufgenommen wurden. Dabei wird in jeder Kamerafahrt eine Lauf des Schieberoboters simuliert. Sie unterschieden sich untereinander in der Orientierung des Sensors zu der Futterstelle.

Die Bilder stellen hoch dynamische Szenen da, mit vielen sich im Hintergrund und Vordergrund bewegenden Kühen. Viele Kühe sind dabei Großteil verdeckt oder klein in dem Hintergrund. Auch herrschen teilweise schlechte Lichtverhältnisse, die die Qualität der Bilder mindert. Insgesamt besteht der Datensatz aus 6 Läufen und insgesamt 350 Bildern. Die Auflösung der Bilder in dem Datensatz ist 640x480.

Die in dem Datensatz annotierten Klassen sind:

- **Kühe:** Für das Maskieren in den Tracking und Rekonstruktion-Algorithmen
- **Silage:** Als Grundlage in der Futtermessung

- **Background:** Die restliche Szene

Durch die Unterschiede der Kamera-Positionierung herrscht in bestimmten Kamerafahrten eine starke Klassen-Ungleichheit. Dabei stellt der Hintergrund und die Silage einen Großteil der Pixel da. In anderen Kamerafahrten dominieren die Kühe als Klasse die Bilder.

4.1.1. Annotation

Die Annotation stellt einen aufwendigen Teil in der Erstellung des Datensatzes da. Für die semantische Segmentierung muss eine Klassifikation pro Pixel annotiert werden. Aufgrund des hohen Aufwandes werden schon existierende Segmentierung in den Labeling Prozess mit eingebunden.

Die einfache Nutzung von GIS Segmentierungsmodell führt zu Schwierigkeiten. Aufgrund der Komplexität der gefilmten Szene, sowohl als auch der spezifischen Anforderungen der Klassen, ist viel manuelles Postprocessing nötig.

Für effektiveres Labeln können PIS Segmentierungs-Modells genutzt werden. Konkret wurde das SAM2 [19] verwendet. Das Modell bietet die Möglichkeit Segmentierung aufgrund von Promts zu erreichen. Dafür können Punkte, Bounding-Boxes und Vorsegmentierung als Input in das Modell gegeben werden um Regionen zu spezifizieren. Dies erlaubt ein flexibles und schnelles Segmentieren von selbst schwer semantisch zu erkennenden Regionen.

Selbst mit dem flexiblen Labeling Setup ist noch manuelles Postprocessing nötig, für schwer zu erkennende Regionen. Die Segmentierungen sind nicht pixelgenau. Aufgrund ungünstiger Lichtverhältnisse lassen sich Kühe im Hintergrund teilweise nur schwer oder gar nicht erkennen.

4.1.2. Evaluation und Trainingsdatensatz

Der Datensatz weist einige Herausforderungen auf. Bilder aus demselben Durchlauf sind aufgrund temporaler Überschneidungen stark miteinander korreliert. Daher ist bei der Trennung in Trainings- und Evaluationsdaten besondere Vorsicht geboten. Ein akkurates Vorgehen wäre, Trainings- und Evaluationsdatensatz aus Aufnahme aus verschiedenen Ställen zu verwenden. Das ist jedoch zu diesem Zeitpunkt nicht möglich, da bislang nur an einem Tag in einem einzigen Stall aufgezeichnet wurde. Zukünftig ist geplant, den Datensatz in dieser Hinsicht zu diversifizieren.

Auf dem momentanen Datensatz ist ein sauberes Vorgehen eine direkte Trennung der Kamerafahrten. Das dabei erstehende Problem ist, dass eine Imbalance zwischen der Anzahl der Bilder in den einzelnen Läufen besteht. Das führt dazu, dass eine faire Trennung nicht ohne große Variation der große des Evaluationsdatensatz oder einem Verlust von annotierten Bilder möglich ist.

Die deshalb gewählte Trennung besteht aus einer Unterteilung in den Kamerafahrten. Damit die Trainings und Evaluationsdatensatz nicht zu stark korrelieren werden zeitliche Pufferzonen eingebaut. Diese Lösung ist nicht optimal, da trotzdem starke Zusammenhänge zwischen Splits bestehen.

4.1.3. Crossvalidation

Die Wahl der Splits beinflusst stark zeitliche Position der Evaluation's-Bilder. Um diesen Faktor zu gut wie möglich zu kontrollieren wird ein Teilung mithilfe von 5-Folds Crossvalidation festgelegt. Dabei werden die temporalen Abschnitte zufällig in jeder Fahrt gewählt, ohne das dabei einen eine Menge von Bildern mehrfach als Validationsdatensatz vorkommt.

Der Testdatensatz stellt eine extra annotierte Fahrt da, die nicht in die Crossvalidation eingeht. Jedoch stammt er von dem selben Tag und Stall, was die Genrealisierbarkeit, der auf ihm erzielten Ergebnisse deutlich infrage stellt.

4.1.4. Augmentation

Aufgrund der kleinen Große des Datensatzes werden die Bilder im Training konstant augmentiert. Dafür wird eine zufälliger Spiegelung und eine zufälliger Verkleinerung um 20 % des Bildbereiches auf ein Bild angewendet (vlg. [3, 9]). Auch wird die Helligkeit und der Kontrast zufällig variiert. Dies soll die schwankende Lichtverhältnisse im Stall emulieren.

4.2. Architektur

Eine klassische Herangehensweise semantische Segmentierung zu erreichen ist es, bewehrte Modelle aus der Bildklassifikation zu modifizieren.

Die hier genutzte Architektur ist die DeeplabV3plus. Dabei stellt ein Convolutional Neural Network (CNN) die Grundlage bzw. den sogenannten Backbone da. Die Idee ist, dass das Backbone Netzwerke die sogenannte Feature Extraction (das effiziente erkennen von hochdimensionen Bildstrukturen) effizient durchführt, aufgrund welcher die Pixel klassifizierte werden. [4]

4.2.1. CNN

Für diese Idee muss zuerst die Architektur und Funktionsweise von CNN's verstanden werden. Die grundlegende Operation, die CNN's ihren Namen gibt, ist die Convolution (Faltung). Diese stellt eine Möglichkeit da, lokal Bild Information zu erfassen.

Definition 6. *Sei* $\Omega := \{0, \dots, W-1\} \times \{0, \dots, H-1\} \times \{0, \dots, C_{in}\}$*, dann wird ein Bild (Feature Map), mit Auflösung* $H \times W$ *und* C_{in} *Featurechannels, definiert durch*

$I : \Omega \to \mathbb{R}$.

Weiter sei $\Omega^* := \{0, \dots, k_w\} \times \{0, \dots, k_h\} \times \{0, \dots, C_{in}\} \times \{0, \dots, C_{out}\}$. *Dann ist ein Convolution Kernel (Faltungskerne) mit Kernel Größe* $k_w \times k_h$ *für* C_{in} *Input Feature Channels Feature Channels und* C_{out} *Output Channels gegeben durch* $K : \Omega^* \to \mathbb{R}$.

Eine $k_w \times k_h$ *Bildfaltung (Image-Convolution) zu dem Kernel K, ist definiert als:*

$$(I * K)(x, y, c) = \sum_{i=0}^{k_h-1} \sum_{j=0}^{k_w-1} \sum_{d=0}^{C_{in}-1} I(x - i + a, y - j + b, d) K(i, j, d, c)$$

Dabei sind $a = \lfloor \frac{k_w}{2} \rfloor, b = \lfloor \frac{k_h}{2} \rfloor$. *Eine Convolution mit K erzeugt ein neues Bild* $I * K$ *mit* C_{out} *Feature Channels.[18]*

Bemerkung 4.1. *Wenn sich die Definition einer Bildfaltung angeschaut wird fällt auf, dass die Abbildung nicht für alle* (x, y, c) *definiert ist. Dies passiert dadurch, dass für die* (x, y) *am Rand des Bildes* $(x - i + a, y - j + b) \notin \{0, \dots, W - 1\} \times \{0, \dots, H - 1\}$. *Effektiv verkleinert sich also die Auflösung des Output Bildes* $I * K$. *Dies wird in der Praxis durch Padding, das Erweitern des Definitionsbereiches des Input Bildes durch Default Werte, gelöst. [18]*

Die Faltung mit einem Kern K stellt eine gewichtet Summe über die Feature Channel einer lokalen Umgebung k_w, k_h da. Dies passiert für jeden Output Feature Channel. Damit soll es den Modellen ermöglicht werden unterschiedliche Aspekte des Bildes zu extrahieren.

Ziel ist es die Anzahl der Output Channels eines neuen Bildes zu erhöhen, damit über die Channels ein Klassifikation gemacht werden kann. Dies ist jedoch nicht ohne weiteres möglich da der Speicherplatz linear in den Channel Ansteigt. Deshalb ist die Strategie von CNN's die Auflösung $H \times W$ zu reduzieren und gleichzeitig die Anzahl der Feature Channels zu erhöhen. Hierzu werden sogenannte strided Convolutions genutzt.

Definition 7. *Sei I ein Feature-Map und K ein Faltungskern. Weiter seien* $s_h, s_w \in \mathbb{N}$, *dann ist die strided Bildfaltung zu dem Kernel K definiert durch*

$$(I * k)(i, j, c) = \sum_{i=0}^{k_h-1} \sum_{j=0}^{k_w-1} \sum_{d=0}^{C_{in}-1} I(x * s_h - i + a, y * s_w - j + b, d) K_{i,j,d,c}$$

Die Werte s_h, s_w stellen die Schrittweiten der Mittelpunkte des Kernelfenster da. Mit geschickter Wahl von Padding und Schrittweiten wird die Auflösung der Bilder halbiert und gleichzeitig die Anzahl der feature Channels verdoppelt. Der Faktor um den Auflösung reduziert wird, wird als der Outputstride bezeichnet. Viele Modelle haben eine Outputstride von 32 und größer. Auf diesen Feature-Maps mit kleiner Auflösung und vielen Features-Channels wird dann eine Klassifikation durchgeführt.

4.2.2. Encoder

Der Encoder von Deeplab besteht zum großen Teil aus dem Backbone Model. Dieses liegt jedoch in modifizierter Form vor. Das Problem welches auftritt, wenn das unveränderte Backbone Model genutzt wird ist, dass der Outputstride sehr groß ist. Um also auf die originale Auflösung zurück zukommen muss die Klassifikation dementsprechend hochskaliert werden, was zu einer groben Akkuratheit der Pixel führt. Um das Problem zu umgehen wird versucht den Outputstride so klein wie möglich zuhalten.

Atrous Convolutions

Dies bringt jedoch Probleme mit sich. Convolutions in späteren Schichten haben die die Eigenschaft, dass sie durch die Verringerung der Auflösung größere Bildbereiche wahrnehmen. Um das Verhalten zu emulieren werden sogenannte Atrous Convolutions genutzt.

Definition 8. *Sei I ein Bild und K ein Faltungskern. Weiter seien $r_h, r_w \in \mathbb{N}$, dann ist die Atrous Bildfaltung zu dem Kernel K definiert durch*

$$(I * k)(i, j, c) = \sum_{i=0}^{k_h-1} \sum_{j=0}^{k_w-1} \sum_{d=0}^{C_{in}-1} I(x - i * r_h + a, y - j * r_w + b, d)K(i, j, d, c)$$

Atrous Convolutions simulieren eine Ausbreitung des Faltungsfensters. Ab einem bestimmten Outputstride werden stridet Convolution durch die entsprechenden Atrous Convolution ersetzt [3].

Bemerkung 4.2. *In der Implementation werden vortrainierte Parameter von dem Backbone Model genutzt werden (vgl. [3, 9]). Da die Anzahl der In- und Output-Channels unverändert bleibt, sowie die Dimension der Kernelfenster, stimmen die Dimension für die Parameter überein. Auch bleibt der Sichtbereich der Fenster durch die Ausbreitung gleich, was eine Rechtfertigung für die Sinnhaftigkeit der Parameter des Backbone-Models gibt.*

Wichtig dabei ist anzumerken, dass aufgrund der höheren Auflösung der Speicherplatz der Feature Maps ansteigt, sowie der Rechenaufwand, da die Fenster auf mehr Pixel operieren . [8]

Seperable Convolutions

Astrous Spatial Pyramid Pooling

Eine weitere Strategie um Features auf unterschiedlichen Skalen zu lernen, ist das Atrous Spatial Pyramid Pooling (ASPP). Dabei wird auf die letzte Feature Map mehrere separable Atrous Convolution mit unterschiedlichen Raten und Padding in parallel angewendet. Zudem wird noch eine 1x1 Convolution ($k_w = k_h = 1$) und ein 2x2 Average Pooling Layer angewendet. Das Padding in den Atrous Convolution wird dabei so gewählt, dass die Auflösung der Feature Map erhalten bleibt. Der Output des Pooling

Layers wird dabei auf die Auflösung hoch interpoliert. Der Output wird aneinander gehängt und Mithilfe von einer weiteren 1x1 Convolution werden die Feature Channels auf runter projiziert.[3]

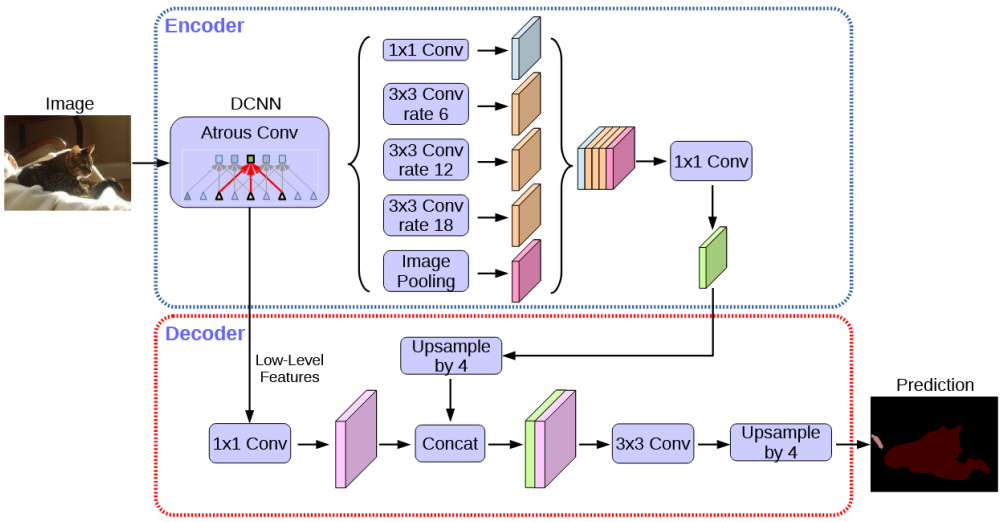


Abbildung 4.1.: DeepLabv3+ Architecture (Original aus [4])

4.2.3. Decoder

Frühere Deeplab Models hatten nur einen primitiven Decoder, der die Ergebnisse des ASPP Modules nur Bilinear auf die originale Auflösung hoch interpoliert. In den Paper [4] wird ein komplexerer Decoder präsentiert, der den Process des Hochskalierens erleichtert.

Die Idee ist eine Feature Map aus dem Backbone Model mit höherer Auflösung und groberen Features zu speichern während der Inference und dann diese bei der Hochskalierung zu nutzen. Ziel dabei ist es schärfere lokale Segmentierung zu erreichen. Dafür wird die Faturemap mit einem Outputstride von 4 genommen, die sogenannten Low-Level-Features [4]. Das ASPP Module bleib wie es ist und wird bei einem Outputstride von 8 oder 16 angewendet 4.1.

Die Ergebnisse werden Bilinear hoch interpoliert, bis auf den Outputstride 4. Die Anzahl der Feature-Channels in den Low-Level-Features wird durch eine 1x1 Convolution reduziert und darauf hin mit den hochskalierten High-Level-Features aneinander gehängt.

Eine darauf folgenden 3x3 Convolution, auf den kombinierten Feature soll eine Verfeinerung der High-Level-Features Auflösung durch die Low-Level-Features erreichen. Erst dann wird die finale Klassifikation der Features durch eine 1x1 Convolution erreicht.

4.3. Trainingskonfiguration

Für das Training des Model wurde eine modulare Implementation von dem Deeplabv3plus Model in Pytorch gewählt [8]. Das Repository erlaubt es vortrainierten Parameter einfach für verschiedene Backbone Modelle zu laden. Dabei stehen die Modelle ResNet, MobilenetV2, Xception, und Hrnet zu Verfügung. Somit ist es möglich folgenden Experimente mit unterschiedlichen Modell einfach fortzusetzen.

Als Backbone-Model für die Experimente wurde das ResNet-50 gewählt. Es handelt sich um ein neuronales Netz, das auf Standard-Convolutions basiert. Im Gegensatz dazu setzen Modelle wie MobileNetV2 oder Xception auf Depthwise-Separable-Convolutions, was zu deutlich weniger Parametern und geringerem Rechenaufwand führt. Die ResNet Architektur wird auch als Backbone in dem Paper [3, 9] genutzt und stellt ein bewährtes Modell da.

ResNet-50 weist daher eine höhere Parameteranzahl und einen größeren Rechenaufwand auf, bietet aber auch ein höheres Potenzial für maximale Genauigkeit. Die Wahl fiel bewusst auf ResNet-50, um das Leistungspotenzial des Ansatzes zu evaluieren. In späteren Trainingsiterationen ist vorgesehen, auch ressourcenschonendere Backbones zu untersuchen, die sich für effiziente Systeme besser eignen.

Das ResNet50 ist eine kleinere Version des ResNet Modells, mit 50 Schichten, was es ermöglicht es auf Verbraucher freundlichen Hardware zu Trainieren. Die vortrainierten Parameter stammen aus dem Torchvision Modul und wurden auf dem ImageNet-1k Datensatz [5] trainiert. Dabei wurden nur die Parameter von einem Datensatz genutzt, da in [9] kein starker Einfluss durch den Datensatz auf festgestellt werden konnte.

4.3.1. Hyperparameter

Die Konfiguration fur Training des Modells bietet eine Vielzahl von zu wählenden Parameter. Diese Hyperparameter werden nicht durch den Traningsprozess optimiert, sondern müssen im Vorhinein gewählt werden.

Loss

Für die Bildsegmentierung, sowie allgemein Klassifikation-Problemen ist der Cross-Entropy-Loss eine gängige Fehlerfunktion (siehe [3]).

Definition 9. *Sei $C \subset \mathbb{N}$ die Menge der Klassen. Weiter $x_i, i \in C$ die Ausgabe des Modells zu der Klasse i. Weiter sei w_i eine Gewichtung der Klasse i. Der Cross-Entropy-Loss für die Ausgabe x des Modells yu dem Label y ist definiert durch*

$$l_{cross}(x, y) = -w_y log\left(\frac{exp(x_y)}{\sum_{c=1}^{C} exp(x_c)}\right) \tag{4.1}$$

Der Gewichtungswert wird für das manuelle Ausgleichen von Klassenungleichgewichten genutzt. In dem Fall des Datensatzes wird nur die Hintergrund Gewichtung angepasst

mit der Range $[0.1, 1]$.

Ein weiterer Loss, der besonders in der Bild-Segmentierung genutzt wird ist der Focal-Loss.

Definition 10. *Sei die Notation gleicht zu der in 4.1. Dann ist der Focal-Loss gegeben durch (vgl.[10]).*

$$l_{FL}(x, y) = -\alpha(1-p)^{\gamma} log(p) \tag{4.2}$$

Der Focal-Loss stellt eine adaptive Gewichtung des Cross-Entropy-Losses da, bei dem Pixel mit hoher vorhergesagter Confidence geringer und Pixel mit niedriger Confidence hoher gewichtet werden. Der Vorteil ist, dass automatisch der einen Fokus auf schwer zu klassifizierende Pixel gelegt wird. Dabei kann dies bei ungenauer Pixel Annotation zu Trainingsschwierigkeiten führen.

Weight-Decay

Der Weight-Decay stellt eine Regularisierung des Modells im Training da, um Overfitting zu vermeiden. Dabei wird ein Regularisierungsterm in die Fehlerfunktion hinzugefügt, der eine Gewichtung der Norm des Modell-Parameter darstellt.

$$l_{L2}(x, y) = l(x, y) + \lambda ||W||^2 \tag{4.3}$$

W stellt den Vektor mit allen trainierebaren Parametern des Modells da. λ ist der Weight-Decay Faktor gewählt wird. Er stellt die Gewichtung zwischen Fehlerfunktion und Größe des Modell Modells da. Dadurch soll die Komplexität eines Modells betraft werden um Overfitting auf dem Trainingsdatensatz zu vermeiden.

Optimierer

Für das Training wurde der AdamW-Optimierer eingesetzt. AdamW ist ein Variantenoptimierer auf Basis von Adam, bei dem das Weight-Decay korrekt vom Gradientenupdate entkoppelt ist [11]. Er stellt einen robusten Optimierer für Tiefe CNN's da. Dabei ist im Gegensatz zu oft dem genutzten Momenten basierte Stochastic-Gradient-Decent ein geringe Wahl von Hyperparameter nötig. Hier wird nur die Lernrate optimiert.

Backbone Training

Aufgrund der Größe des Datensatzes ist es nicht nicht sinnvoll ein Tiefes Convolutionelles Modell zu Trainieren. Um das Problem zu umgehen wird versucht die vortrainierte Parameter effektiv zu nutzen. Dazu wird für das Backbone-Modell die Lernrate um einen Faktor von 0.1 verringert. Diese Strategie wird auch in den Paper [3] genutzt.

Eine andere Herangehensweise ist es die Parameter aus dem Training komplett zu entfernen. Dabei muss beachtet werden nur die Parameter zu entfernen die auch unverändert auf der jeweiligen Feature-Map operieren. Wenn also eine Erhöhung eine strided Convolution durch eine Atrous-Convolution vertauscht wird, operieren die Erweiterten

Convolutions auf dem gleiche Sichtfenstern, aber auch auf einer größeren Feature-Map (Bem. 4.2). Aus diesem Grund sollten sie in das Training einbezogen werden.

Outputstride

Für die Konfiguration des Backbone-Models muss auch der Outputstride festgelegt werden. Dabei stehen 8, 16 praktisch zur Auswahl. Ist der Outputstride zu klein steigt der Rechenaufwand, sowie der Speicherverbrauch dramatisch an [3]. Ist er zu klein kann es zu unpräzisen Versagen führen, da der Up-Sampeling Faktor zu groß ist, selbst mit Hilfe der hinzugefügten Decoders.

4.3.2. Hyperparameter Optimierer

Aufgrund des kleinen Trainingsdatensatzes ist die Konfiguration der Parameter schwer im Vorhinein eingrenzen durch fehlende Referenz Experimente. Die Konfiguration Bereiche sind große gewählt um Preconditioning des Trainings zu vermindern. Für die Bestimmung einer guten Konfiguration wird eine Kombination aus Exploration und Exploitation genutzt.

Für die Hyperparameter-Optimierung wird eine Kombination aus einem Random-Search- und einem TPE-Sampler eingesetzt. Der Explorationsanteil wird durch den Random-Search-Sampler abgedeckt, der zu Beginn der Suche zufällige Kombinationen der Parameter aus ihren jeweiligen Suchräumen auswählt [2]. Dies bildet die Initialphase der Optimierung, in der der Parameterraum breit abgedeckt wird.

Anschließend kommt der TPE-Sampler (Tree-structured Parzen Estimator) zum Einsatz, ein bayesianisches Optimierungsverfahren, das auf Basis der bereits getesteten Konfigurationen gezielt vielversprechende Bereiche weiter untersucht [24]. Die Implementierung beider Algorithmen sowie die Steuerung des gesamten Optimierungsprozesses erfolgt mithilfe des Hyperparameter-Optimierungsframeworks Optuna [2].

4.4. Traningsergebnisse

4.4.1. Metriken

IOU

A. Rohdaten

Tabelle A.1.: RPE visuelle Odoemtrie Multiscale 3 Pyramiden Level per 10 Iterationen

Methode	**static xyz**		**static rpy**		**walking static**		**walking xyz**		**walking rpy**	
	Trans. [m]	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**
Intensität	0.0222	0.0206	0.0199	0.0700	0.0156	0.0090	0.0302	0.0214	0.0308	0.0390
Intensität maskiert	0.0223	0.0206	0.0209	0.0706	0.0078	0.0079	0.0211	0.0205	0.0328	0.0399
Hybrid	0.0232	0.0217	0.0227	0.0761	0.0126	0.0086	0.0294	0.0200	0.0281	0.0329
Hybrid maskiert	0.0232	0.0217	0.0230	0.0769	0.0066	0.0076	0.0189	0.0194	0.0245	0.0352
P2P	0.0225	0.0213	0.0105	0.0883	0.0102	0.0073	0.0265	0.0175	0.0230	0.0367
P2P maskiert	0.0225	0.0213	0.0105	0.0883	0.0050	0.0064	0.0188	0.0171	0.0208	0.0359

Tabelle A.2.: ATE visuelle Odoemtrie 3 Pyramiden Level per 10 Iterationen

Methode	**static xyz**		**static rpy**		**walking static**		**walking xyz**		**walking rpy**	
	Trans. [m]	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**
Intensität	0.4480	0.4976	0.4733	1.3482	0.6179	0.2808	1.4618	0.6426	1.6596	1.2408
Intensität maskiert	0.4425	0.4686	0.7497	1.3497	0.3292	0.2123	0.6265	0.3568	1.4056	1.1543
Hybrid	0.5359	0.4932	1.6487	2.0307	0.4870	0.2723	1.5046	0.7136	1.7352	1.1641
Hybrid maskiert	0.5273	0.4811	1.6585	2.0124	0.2554	0.2015	0.5800	0.3681	1.1007	1.0712
P2P	0.5836	0.3370	0.3117	1.8171	0.4192	0.4001	2.7121	0.7409	1.3975	1.3223
P2P maskiert	0.5821	0.3400	0.3360	1.8257	0.3190	0.3884	1.2506	0.4905	0.7510	1.1356

Tabelle A.3.: RPE TSDF-Tracking mit 3 Pyramiden Level per 10 Iterationen und gewichteter Integration

Methode	static xyz		static rpy		walking static		walking xyz		walking rpy	
	Trans. [m]	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**
Intensität default	$0.0212 \pm 7.96\%$	$0.0195 \pm 6.34\%$	$0.0184 \pm 7.47\%$	$0.0474 \pm 0.83\%$	$0.0124 \pm 3.66\%$	$0.0065 \pm 1.41\%$	$0.0879 \pm 25.90\%$	$0.0323 \pm 21.51\%$	$- \pm -\%$	$- \pm -\%$
Intensität maskiert	$0.0209 \pm 3.64\%$	$0.0194 \pm 2.68\%$	$0.0180 \pm 3.27\%$	$0.0471 \pm 0.68\%$	$0.0165 \pm 19.65\%$	$0.0075 \pm 10.14\%$	$0.0940 \pm 37.86\%$	$0.0343 \pm 34.52\%$	$- \pm -\%$	$- \pm -\%$
Hybrid default	$0.0115 \pm 1.12\%$	$0.0144 \pm 0.36\%$	$0.0318 \pm 6.39\%$	$0.0470 \pm 2.81\%$	$0.0101 \pm 12.80\%$	$0.0061 \pm 5.66\%$	$0.0287 \pm 9.87\%$	$0.0158 \pm 3.96\%$	$- \pm -\%$	$- \pm -\%$
Hybrid maskiert	$0.0114 \pm 0.58\%$	$0.0144 \pm 0.24\%$	$0.0329 \pm 7.02\%$	$0.0486 \pm 4.72\%$	$0.0075 \pm 1.26\%$	$0.0056 \pm 0.68\%$	$0.0280 \pm 2.91\%$	$0.0141 \pm 1.17\%$	$- \pm -\%$	$- \pm -\%$
P2P default	$0.0044 \pm 0.04\%$	$0.0109 \pm 0.02\%$	$0.0075 \pm 0.07\%$	$0.0201 \pm 0.02\%$	$0.0181 \pm 7.50\%$	$0.0094 \pm 6.75\%$	$0.0296 \pm 9.46\%$	$0.0177 \pm 7.61\%$	$- \pm -\%$	$- \pm -\%$
P2P maskiert	$0.0045 \pm 0.05\%$	$0.0109 \pm 0.02\%$	$0.0078 \pm 0.08\%$	$0.0201 \pm 0.03\%$	$0.0041 \pm 0.22\%$	$0.0057 \pm 0.03\%$	$0.0079 \pm 0.52\%$	$0.0126 \pm 0.04\%$	$0.0611 \pm 51.08\%$	$0.0399 \pm 62.34\%$

Tabelle A.4.: ATE TSDF-Tracking mit 3 Pyramiden Level per 10 Iterationen und gewichteter Integration

Methode	static xyz		static rpy		walking static		walking xyz		walking rpy	
	Trans. [m]	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**	**Trans. [m]**	**Rot. [rad]**
Intensität default	$1.0399 \pm 18.94\%$	$0.6811 \pm 21.08\%$	$0.6808 \pm 16.23\%$	$1.0558 \pm 12.88\%$	$0.0683 \pm 3.91\%$	$0.0215 \pm 3.68\%$	$2.7599 \pm 33.63\%$	$1.0619 \pm 52.83\%$	$- \pm -\%$	$- \pm -\%$
Intensität maskiert	$1.2910 \pm 32.83\%$	$0.9918 \pm 26.01\%$	$0.7712 \pm 9.00\%$	$1.0139 \pm 14.92\%$	$0.0425 \pm 25.70\%$	$0.0173 \pm 13.49\%$	$2.8556 \pm 31.88\%$	$1.1483 \pm 56.38\%$	$- \pm -\%$	$- \pm -\%$
Hybrid default	$0.0578 \pm 2.17\%$	$0.1179 \pm 1.29\%$	$1.0401 \pm 38.40\%$	$0.8918 \pm 20.38\%$	$0.1712 \pm 87.18\%$	$0.0543 \pm 83.24\%$	$2.0014 \pm 24.41\%$	$0.9059 \pm 14.65\%$	$- \pm -\%$	$- \pm -\%$
Hybrid maskiert	$0.0544 \pm 1.36\%$	$0.1178 \pm 0.72\%$	$1.0982 \pm 43.08\%$	$0.9033 \pm 19.95\%$	$0.0310 \pm 1.87\%$	$0.0150 \pm 0.77\%$	$0.2893 \pm 3.71\%$	$0.0655 \pm 17.25\%$	$- \pm -\%$	$- \pm -\%$
P2P default	$0.0330 \pm 0.03\%$	$0.0336 \pm 0.02\%$	$0.0766 \pm 0.10\%$	$0.1021 \pm 0.02\%$	$0.8969 \pm 19.69\%$	$0.2534 \pm 35.27\%$	$1.2095 \pm 17.29\%$	$0.8985 \pm 10.30\%$	$- \pm -\%$	$- \pm -\%$
P2P maskiert	$0.0329 \pm 0.02\%$	$0.0336 \pm 0.02\%$	$0.0759 \pm 0.07\%$	$0.1022 \pm 0.02\%$	$0.0466 \pm 0.03\%$	$0.0136 \pm 0.04\%$	$0.1053 \pm 0.05\%$	$0.0320 \pm 0.05\%$	$1.1831 \pm 56.88\%$	$0.8706 \pm 75.58\%$

Literatur

[1] P.-A. Absil, Robert Mahony und Rodolphe Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ und Oxford, UK: Princeton University Press, 2008.

[2] Takuya Akiba u. a. *Optuna: A Next-generation Hyperparameter Optimization Framework*. 2019. arXiv: 1907.10902 [cs.LG].

[3] Liang-Chieh Chen u. a. *Rethinking Atrous Convolution for Semantic Image Segmentation*. 2017. arXiv: 1706.05587 [cs.CV].

[4] Liang-Chieh Chen u. a. *Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation*. 2018. arXiv: 1802.02611 [cs.CV].

[5] Jia Deng u. a. „Imagenet: A large-scale hierarchical image database“. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, S. 248–255.

[6] Slimane Djema u. a. *Dense Visual Odometry Using Genetic Algorithm*. 2023. arXiv: 2311.06149 [cs.RO].

[7] Wei Dong u. a. *ASH: A Modern Framework for Parallel Spatial Hashing in 3D Perception*. 2023. arXiv: 2110.00511 [cs.CV].

[8] Gongfan Fang. *DeepLabv3Plus-Pytorch*. https://github.com/VainF/DeepLabV3Plus-Pytorch. Accessed: 2025-06-02. 2022.

[9] Dennis Haitz u. a. „Semantic segmentation with small training datasets: A case study for corrosion detection on the surface of industrial objects“. Englisch. In: *Forum Bildverarbeitung 2022. Ed.: T. Längle; M. Heizmann*. KIT Scientific Publishing, 2022, S. 73–85.

[10] Tsung-Yi Lin u. a. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[11] Ilya Loshchilov und Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].

[12] Wolfgang Lück. *Algebraische Topologie: Homologie und Mannigfaltigkeiten*. 1. Aufl. vieweg studium. Aufbaukurs Mathematik (VSAM). Wiesbaden, Deutschland: Vieweg+Teubner / Springer, 2005. Kap. 9, S. 158–175.

[13] Ruben Mascaro und Margarita Chli. „Scene Representations for Robotic Spatial Perception“. In: *Annual Review of Control, Robotics, and Autonomous Systems* 8.Volume 8, 2025 (2025), S. 351–377. DOI: https://doi.org/10.1146/annurev-control-040423-030709.

[14] Raul Mur-Artal, J. M. M. Montiel und Juan D. Tardos. „ORB-SLAM: A Versatile and Accurate Monocular SLAM System“. In: *IEEE Transactions on Robotics* 31.5 (Okt. 2015), S. 1147–1163. DOI: 10.1109/tro.2015.2463671.

[15] Richard M. Murray, Zexiang Li und S. Shankar Sastry. *A Mathematical Introduction to Robotic Manipulation*. Boca Raton, FL: CRC Press, 1994. Kap. 2, S. 19–61.

[16] OpenCV Team. *Feature Matching*. Accessed: 2025-07-08. OpenCV. 2024.

[17] Jaesik Park, Qian-Yi Zhou und Vladlen Koltun. „Colored Point Cloud Registration Revisited“. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Okt. 2017, S. 143–152. DOI: 10.1109/ICCV.2017.25.

[18] PyTorch Developers. *torch.nn.Conv2d - Pytorch Documentation*. Techn. Ber. Zugriff: 1-05-2025. Pytorch, 2025.

[19] Nikhila Ravi u. a. *SAM 2: Segment Anything in Images and Videos*. 2024. arXiv: 2408.00714 [cs.CV].

[20] Szymon Rusinkiewicz und Marc Levoy. „Efficient Variants of the ICP Algorithm“. In: *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*. IEEE, 2001, S. 145–152. DOI: 10.1109/IM.2001.924423.

[21] Frank Steinbrücker, Jürgen Sturm und Daniel Cremers. „Real-Time Visual Odometry from Dense RGB-D Images“. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. IEEE. Barcelona, Spain, 2011.

[22] J. Sturm u. a. „A Benchmark for the Evaluation of RGB-D SLAM Systems“. In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. Okt. 2012.

[23] Yanan Wang u. a. „A Survey of Visual SLAM in Dynamic Environment: The Evolution From Geometric to Semantic Approaches“. In: *IEEE Transactions on Instrumentation and Measurement* 73 (2024), S. 1–21. DOI: 10.1109/TIM.2024.3420374.

[24] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. 2023. arXiv: 2304.11127 [cs.LG].

[25] Yuxin Wu u. a. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[26] Qian-Yi Zhou, Jaesik Park und Vladlen Koltun. „Open3D: A Modern Library for 3D Data Processing“. In: *arXiv:1801.09847* (2018).

[27] Tianfei Zhou u. a. *Image Segmentation in Foundation Model Era: A Survey*. 2024. arXiv: 2408.12957 [cs.CV].

Erklärung

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig angefertigt und ohne fremde Hilfe verfasst habe. Dazu habe ich keine außer den von mir angegebenen Hilfsmitteln und Quellen verwendet und die den benutzten Werken inhaltlich und wörtlich entnommenen Stellen habe ich als solche kenntlich gemacht. Ich versichere, dass die eingereichte elektronische Fassung mit den gedruckten Exemplaren übereinstimmt.

Rostock, den 21.09.2025

Hans-Hauke Haufe