



## Rekonstruktion und Tracking in dynamischen Umgebungen

Maskenbasierte Anpassungen mithilfe eines Problem  
spezifisch trainierten Segmentierungsmodells

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

# Inhaltsverzeichnis

<b>Abkürzungsverzeichnis</b>	<b>2</b>
<b>1. Einleitung</b>	<b>3</b>
<b>2. Theoretische Grundlagen</b>	<b>4</b>
2.1. Problemkontext . . . . .	4
2.2. SLAM und Visual SLAM . . . . .	4
2.3. visuelle Odometrie . . . . .	5
2.3.1. Rigid-Body-Motion . . . . .	6
2.3.2. Notation . . . . .	6
2.3.3. Problem Formulierung . . . . .	7
2.3.4. Warpingfunktion . . . . .	7
2.3.5. Fehlerfunktion . . . . .	8
2.4. Voxel-Block-Grid TSDF Tracking . . . . .	9
2.4.1. VoxelBlockGrid . . . . .	10
2.4.2. Integration . . . . .	10
2.4.3. TSDF-Tracking . . . . .	11
<b>3. Masken basiertes Tracking und Szenen Rekonstruktion</b>	<b>12</b>
3.1. maskierte visuelle Odometrie . . . . .	12
3.1.1. Optimierung visueller Odometrie . . . . .	12
3.1.2. Masken Einbindung . . . . .	14
3.2. maskierte TSDF Integration . . . . .	15
3.3. Auswertung . . . . .	17
3.3.1. Testdatensatz . . . . .	17
3.3.2. Metriken . . . . .	18
3.3.3. maskierte Odometrie Auswertung . . . . .	18
3.3.4. maskiertes TSDF-Tracking Auswertung . . . . .	21
3.3.5. Bewertung . . . . .	23
<b>4. Semantische Bild Segmentierung</b>	<b>26</b>
4.1. Datensatz . . . . .	26
4.1.1. Annotation . . . . .	27
4.1.2. Evaluation und Trainingsdatensatz . . . . .	27
4.1.3. Augmentation . . . . .	28
4.2. Model-Architektur . . . . .	28
4.2.1. CNN-Backbone . . . . .	29
4.2.2. Encoder . . . . .	30

4.2.3. Decoder . . . . .	32
4.3. Trainingskonfiguration . . . . .	32
4.3.1. Hyperparameter . . . . .	32
4.3.2. Hyperparameter Optimierer . . . . .	34
4.4. Trainingsergebnisse . . . . .	35
4.4.1. Metriken . . . . .	35
4.4.2. Ergebnisse . . . . .	36
<b>5. Fazit und Ausblick</b>	<b>38</b>
<b>A. Rohdaten</b>	<b>39</b>
<b>Literatur</b>	<b>42</b>

# Abkürzungsverzeichnis

<b>SLAM</b>	Simultaneous Localization and Mapping . . . . .	4
<b>RGB-D</b>	Red-Green-Blue plus Depth . . . . .	4
<b>vSLAM</b>	visual SLAM . . . . .	4
<b>TSDF</b>	Truncated Signed Distance Field . . . . .	3
<b>ICP</b>	Iterative Closest Point . . . . .	4
<b>SfM</b>	Structure from Motion . . . . .	4
<b>RPE</b>	Relative Pose Error . . . . .	18
<b>ATE</b>	Absolute Trajectory Error . . . . .	18
<b>P2P</b>	Point to Plane . . . . .	9
<b>CNN</b>	Convolutional Neural Network . . . . .	28
<b>FCN</b>	Fully Convolutional Network . . . . .	28
<b>ASPP</b>	Atrous Spatial Pyramid Pooling . . . . .	31
<b>GIS</b>	Generic Image Segmentation . . . . .	26
<b>PIS</b>	Promtable Image Segmentation . . . . .	26
<b>IoU</b>	Intersection over Union . . . . .	35

# 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, werden Verfahren benötigt, die sowohl die eigene Bewegung im Raum genau schätzt als auch störende bewegte Objekte erkennt und aus der Rekonstruktion ausschließt.

In diesem Kontext wird ein Fokus auf die Methode der visuellen Odometrie und der Verbindung mit dem Truncated Signed Distance Field (TSDF)-Tracking, die den Grundbaustein für die Rekonstruktion darstellt. Dabei ist Anpassung dieser Verfahren für dynamische Szenen durch einbeziehen von vorsegmentierten Masken, essenziell.

Die automatische Erstellung von den benötigten Masken, aufgrund eines erstellten Segmentierungsdatensatzes, bildet den zweiten Schwerpunkt der Arbeit.

## 2. Theoretische Grundlagen

### 2.1. Problemkontext

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

Die zentrale Herausforderung, die sich aus der Problemstellung ergibt, ist folgendes: Das Messsystem arbeitet unabhängig vom Schieberoboter und verfügt somit über keine direkten Informationen 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 Simultaneous Localization and Mapping (SLAM), bei dem die Sensor 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 Structure from Motion (SfM), bei denen der optische Fluss zwischen aufeinanderfolgenden Bildern analysiert wird. Daneben existieren geometrische Methoden wie der Iterativ 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 [26]

Die genannten Methoden besitzen alle das Problem, dass statische Umgebungen vorausgesetzt werden. Das Feld des dynamischen vSLAM stellt die Erweiterung und Anpassung von Methoden auf dynamische Umgebungen da. Dabei nehmen Methoden des maschinellen Lernen einen größer werdenden Anteil ein.[26]

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 ist die Rekonstruktion die zweite zentrale Aufgabe. Dabei werden die Bilddaten mithilfe der Sensorposition in ein Speichermedium integriert. Geläufige Representation sind Punktwolken (direkt Darstellung als 3D-Punkte), Meshes (Polygon basiert), Surfels oder auch hierarchische Strukturen wie Octrees. Zunehmend werden auch neuronale Darstellungen erforscht [15].

Die Erstellung solcher Modelle wird durch dynamische Objekte deutlich erschwert. Daher ist eine angepasste Rekonstruktion ein weiterer Fokus 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 visuelle Odometrie löst, ist folgende: 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 wird die gesamte Bewegung des Sensors (Bewegungs-Trajektorie) annäherungsweise bestimmt werden. Dabei gibt es verschiedene Ansätze.

Zum einen 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.[19, 17]

Ein weitere Klasse von Methoden ist die der dichte Odometrie. Sie optimiert direkt auf den Pixel ohne zusätzliche Darstellungen. Ihr Anwendungsgebiet ist das Abgleichen von Bildern mit hoher Frequenz, jedoch mit kleinerer zeitlichen Differenz. Der Schwerpunkt liegt hier nur auf der dichten visuellen 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 zusammenhängenden Objekten im Raum beschreibt. Mathematisch kann sie beschrieben werden durch  $SE(3)$ . [18]

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

$$\left\{ F : \mathbb{R}^3 \rightarrow \mathbb{R}^3, x \mapsto Ax + b \mid A \in SO(3), b \in \mathbb{R}^n \right\} \quad (2.1)$$

$SO(3)$  stellt dabei die 3-dimensionale orthogonale Gruppe da. Durch Komposition von Abbildungen kann eine Verknüpfung auf der Menge definiert werden.  $SE(3)$  verbunden mit dieser Verknüpfung bildet ein Gruppe (vgl. [18]).

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

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

Dies erlaubt eine numerische Beschreibung von Rigid-Body-Motion über lineare Abbildungen (vgl. [18]).

Rigid-Body-Motion ist Kombination aus Rotation und Translation. Das Auszeichnende Merkmal dabei ist, dass Abstände und Orientierung von Punkten durch die Abbildung erhalten bleiben (vgl. [18]).

Im folgender wird häufig der Begriff einer starre Umgebungen/Szene gebraucht. Der Begriff bezeichnet eine Umgebung/Szene dessen Bewegung zeitlich durch Rigid-Body-Motion beschrieben.

### 2.3.2. Notation

RGB-D-Bilder können durch Abbildungen  $I_i : \Omega \rightarrow [0, 1]^3 \times \mathbb{R}_+$ ,  $\Omega \subset \mathbb{Z}^2$  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  $\Omega$  dargestellt.

$I_i^d : \Omega \rightarrow \mathbb{R}_+$  stellt die Funktion da, die nur den Tiefen-Anteil jedes Pixels berücksichtigt.  $I_i^g : \Omega \rightarrow [0, 1]$  Pixel bildet einen Farbpixel auf einen Graustufenwert ab. Zur Vereinfachung der Notation wird eine Funktion definiert, die einen Pixel zusammen mit seinem Tiefenwert als dreidimensionalen Vektor darstellt:  $d_i : \Omega \rightarrow \mathbb{R}^3$ ,  $d_i(u, v) \mapsto (u, v, I_i^d(u, v))^T$

### 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 Koordinatensystem 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 Punktmenzen zurückgeführt werden.

Die Motivation stellt das Problem auf eine geometrisch Art da. Die konkrete Formulierung des Problems erfolgt aber über die Pixel in den Bildern. Diese Darstellung stimmt mit den durch den Sensor gelieferten Daten übereinstimmt.

Das bestimmen 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  $r : \Omega \times SE(3) \rightarrow \mathbb{R}$  eine Fehlerfunktionen ist, die implizit von  $I_s, I_t$  abhängt.  
[24][20]

### 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. Die Projektionsebene wird durch die Menge  $\{(x, y, 1), x, y \in \mathbb{R}\}$  gegeben. 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 Intrinsische Matrix angewandt.

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

Die Matrix  $K$  beruht auf festen Kamerakonstanten den 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 [7]

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

**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

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

Trotz der Bezeichnung ist  $P^{-1}$  nicht die Inverse von  $P$ .[7].

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

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

Für eine Vereinfachung der Notation wird  $Tx = Ax + b$ ,  $A \in SO(3), b \in \mathbb{R}^3$  beschrieben, anstatt über die Matrix Form in Bemerkung 2.1.

**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.

**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  $r_{photo} : \Omega \times SE(3) \rightarrow \mathbb{R}$  ist definiert als

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

[20, 24]

### 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  $r_{depth} : \Omega \times SE(3) \rightarrow \mathbb{R}$  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). Dazu wird zu 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.[20, 29].

**Definition 5.** Sei  $n : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ ,  $x \mapsto n(x) := n_x$  die Abbildung die einem Punkt eine Normale zuteilt. 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  $r_{p2p} : \Omega \times SE(3) \rightarrow \mathbb{R}$  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 visuellen 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 Voxelpuffers 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 Raymarching (Raycasting) synthetische Bilder zu der Szene zu erzeugen, was sich für das Tracking als nützlich erweist.[8]

Ein weiteres zentrales Element des Voxel-Block-Grids ist eine Trennung von lokaler und globaler Geometrie. Dafür werden Voxel in sogenannte Blöcke gruppiert. Jeder Block stellt eine lokale Ansammlung von Voxel-Gittern mit höherer räumlicher Auflösung dar. 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. [8]*

### 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. [8]

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

$$d = d(u, v) - r \quad (2.3)$$

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. Werden Abstandswerte mehrfach in Voxel integriert. Anstatt den Wert mit dem neu integrierten zu überschreiben wird ein kleinste Quadrate Problem konstruiert und gelöst. Der Ansatz biete die Möglichkeit die Integration zu gewichten. [8]*

### 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. Glattere Daten führen zu einer Verbesserung der visuellen Odometrie.

Auch kann das Driften der Positionen verbessert werden. Das Modell gibt einen Referenzpunkt, an dem sich orientiert werden kann.[29]

# 3. Masken basiertes Tracking und Szenen Rekonstruktion

Die visuelle Odometrie und auch das Erstellen eines Voxel-Block-Grids der Szenen ist grundlegend und Voraussetzung gekoppelt, dass die gemessene Szene statisch ist. In dem Kontext der Rekonstruktion der Futterstelle (siehe 2.1) ist dies nicht gegeben. Die Kühe stellen dynamische Objekte dar, mit unabhängigen Bewegungs-Trajektorien.

**Bemerkung 3.1.** *Odometrie ist fundamental eine Optimierung über Rigid-Body-Motion. Im geometrische Sinne wird eine Transformation der Geometrie durch eine Rigid-Body-Motion approximiert. Ist die Transformation nicht starr, ist das Modell Fehlerhaft.*

*IBei der Modellerstellung kann es durch fehlerhafte Trajektorien zu einem verzerrten Warping des Modells kommen. Selbst bei einer exakten Trajektorie führen unabhängig bewegte Objekte zu verfälschter Geometrie und zu zeitlichen Schlieren.*

**Idee:** Durch Aufteilung der gemessenen Punkte in starre Teilmengen kann gezielt die Voraussetzung der starren Umgebung wiederhergestellt. Dafür ist eine Anpassung der Algorithmen nötig, die Einzuschränkung erlaubt.

Im Folgenden werden die starren Untermengen über Menge von Pixeln im Ursprungsbild und Zielbild beschrieben. Sie repräsentieren starre Geometrie. Die Pixelmengen sind im folgenden als fester Eingabe Parameter in den Algorithmen gegeben. Die Mengen werden mit  $M_s, M_t \subset \Omega$  bezeichnet.

## 3.1. maskierte visuelle Odometrie

Die Anpassung der visuellen Odometrie benötigt ein grundlegendes Verständnis des zu grunde liegenden Optimierungsproblem's und dem Aufbau des Lösungs-Algorithmus's.

### 3.1.1. Optimierung visueller Odometrie

Bei der Betrachtung des in 2 gegebenen Problems ist erkenntlich, dass das Optimierungsdomain eine Gruppe ist. Gängige elementare Optimierungsverfahren im  $\mathbb{R}^n$  basieren auf der Berechnung von Ableitungen (Ableitungen höherer Ordnung), für das Finden von Abstiegsrichtungen der Fehlerfunktion. Anhand von Abstiegsrichtungen werden lokal Schritte getätigt um die Funktion iterativ zu minimieren. In dem Fall einer Gruppe ist jedoch weder klar, was mit einer Ableitung, Abstiegsrichtung und einem Schritt gemeint ist.

**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  $\Omega$ :

$J, r \leftarrow \text{CalcJacobian}(x, T)$

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][29, 20])

$SE(3)$  als Gruppe und Menge besitzt bestimmte Glattheits-Eigenschaften. 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 sie 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 [14, Kap.9][1, Kap.8.4][18, Kap.2][18, Apd.A]

Mithilfe der Theorie können gängige Optimierungsverfahren übertragen werden. Die benutzte Problemformulierung erlaubt es allgemeine kleinste Quadrate Methoden auf es anzuwenden. Konkret wird eine erweiterte Version des Gauss-Newton Verfahrens angewendet (vgl. [1, Kap.8.4])

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 . Als Schrittfunktion dient die Matrixexponentialfunktion (vgl. [18, Kap.2]).

**Bemerkung 3.2.** In den Buch [18, Kap.2] ist eine konkrete physisch motivierte Herleitung der Form der Abstiegsrichtungen zu finden. Dabei wird die Matrixexponentialfunktion durch das lineare Differenzialgleichungs System mit genannten Abstiegsmatrizen motivierte.

Der Algorithmus zum Lösen der Odometrie ist beschrieben in 1. Er nutzt die Linearität der Jacobimatrix akkumulativ aus Teilberechnungen der Jacobimatrizen der Pixel zu berechnen. Die Aufteilung der Rechnung bietet den Zugang das Problem auf bestimmte

Mengen von Pixeln einzuschränken.

Die konkreten Jacobimatrizen für die Fehler sind gegeben durch folgenden Formeln (vgl. [20, S.146, Fom.28-30]).

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

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

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

$\nabla I_s^d, \nabla I_s^g$  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  $\nabla I_s^g$  und  $\nabla I_s^d$  werden über sogenannte Sobel-Filter berechnet. Sie stellen Convolution zu den Sobel-Kernen da und approximieren Bild-Gradienten.  $J_P$  kann analytische durch die Kamera Intrinsische Werte berechnet werden. Für den Ausdruck  $J_T$  existiert analytische Representation die hier jedoch nicht angegeben wird (siehe [10]).

**Bemerkung 3.3.** Durch die Berechnung von Bild-Gradienten und Normalen wird implizit die Annahme der Differenzierbarkeit von aufgenommener Oberfläche und des Aufnahme-Prozess's getroffen. Das ist in generellen realen Aufnahmen nicht gegeben, was zu einer schlechten Konvergenz des Optimierungs-Algorithmus führen kann.

**Bemerkung 3.4.** Um die Konvergenz des Algorithmus's 1 zu verbessern, wird eine Multiskalen-Implementierung eingesetzt. Dabei wird eine sogenannte Bildpyramide der Eingabebilder über  $n$  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 sukzessiv verfeinert. [7]

### 3.1.2. Masken Einbindung

Die Einbindung der Bildmasken findet direkt in der parallelen Berechnung der Teil-Jacobimatrizen statt. Dabei kann die Ursprungsmaske  $M_s$  unmittelbar integriert werden: Für Pixel  $x \notin M_s$  werden weder Ableitungen noch Residuen berechnet, sodass invalide Punkte keinen Einfluss auf das Gauss-Newton-Gleichungssystem haben (siehe 1).

Die Zielmaske  $M_t$  bleibt 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 können gültige Bildbereiche auf ungültige 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 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 durch eine Anpassung der Funktion „CalcJacobian“ in dem Algorithmus.

**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:** maskierte Version der CalcJacobian Funktion aus dem Algorithmus 1

**Bemerkung 3.5.** Die Berechnung von  $n_x, \nabla I_t^g, \nabla I_t^d$  muss vorsichtig behandelt werden. Wenn Randpunkte von starren Geometrie betrachtet werden, haben lokale Messdaten Einfluss auf die Berechnung von  $n_x, \nabla I_t^g, \nabla I_t^d$ . Wenn der Einfluss von Störregionen vollständig in der Optimierung entfernt werden soll, müssen die Ableitung von Randpunkten mit gesondert berechnet werden.

## 3.2. maskierte TSDF Integration

Das in Abschnitt 2.4.2 angegeben Integration-Vorgehen kann direkt durch die Einbindung von Masken angepasst werden. Wenn ein Voxel auf einen invaliden Bildbereich transformiert wird, kann im Integration-Verfahren der Distanzwert entsprechen gewich-

tet werden. Das erfolgt wie in der Bemerkung 2.5 erläutert.

**Input:** Aktive Blockmenge  $A$ , Tiefenbild  $I_t$ , Welt Kameraposition  $T$ , Trunkationsdistanz  $\mu$ , Maske  $M_t \subset \Omega$

```

foreach Block  $b \in A$  do
    foreach Voxel  $v \in b$  do
         $p_w \leftarrow$  Weltposition des Voxels;
         $p_c \leftarrow KT^{-1} \cdot p_w;$            // Transformation in Kamerakoordinaten
        if  $p_c$  außerhalb des Sichtfeldes then
            | continue
        end
         $(u, v) \leftarrow$  Projektion von  $p_c$  in Bildkoordinaten;
        if  $(u, v)$  innerhalb von  $M_t$  then
            |  $w_{meas} \leftarrow 0;$  // setzt das Gewicht des Abstandswertes gleich 0
        end
         $sdf \leftarrow \text{clamp}(z_{meas} - p_c.z, -\mu, \mu)/\mu;$ 
        if  $|sdf| > 1$  then
            | continue;                                // Truncation
        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 pixelgenaue Segmentierung zu erwarten. Um das Problem zu lösen, können die binären Masken durch Unsicherheit's-Bilder ersetzt werden. Der im Voxel gespeicherte Gewichtungswert kann als Vertrauen in die Geometrie interpretiert werden.

Der Masken basierte TSDF-Tracking Algorithmus ergibt sich aus maskierte visueller Odometrie und maskierte Integration.

**Input:** Tiefenbild  $I_t$ , Maske  $I_s$ , Kamera-Intinsics  $K$ , Vorpose  $T_{t-1}$ , TSDF-Volumen  $V$

**Output:** Aktualisierte Pose  $T_t$ , aktualisiertes Volumen  $V$

```

 $\hat{I}_{t-1} \leftarrow \text{Raycast}(V, K, T_{t-1});$ 
 $T_t \leftarrow \text{MaskoutOdometry}(\hat{I}_{t-1}, I_t, M_t, K);$ 
 $\text{MaskoutIntegrate}(V, I_t, M_t, T_t);$ 
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. Die synthetisch erzeugten Bilder werden als Zielbilder gewählt, da sie das in Bemerkung 3.3 verbessern.

### 3.3. Auswertung

Die grundlegenden Algorithmen aus dem Kapitel 2 sind in einer Modularen 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 (siehe [29]).

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 Lernens.[29]

Die Implementation der angepassten Algorithmen ist direkt im Tensor-API von Open3D integriert. Alle Algorithmen liegen in einer parallelisierten Version als multithreaded CPU code und GPU CUDA-Kernels bereit.

#### 3.3.1. Testdatensatz

Im Kontext der Rekonstruktion von der Futterstelle von Kuh (siehe 2.1), stehen keine präzisen wahren Trajektorien für die Evaluation zur Verfügung. Als Test wird deshalb ein RGB-D-Slam Benchmark der Technischen Universität München genutzt [25]

Dieser enthält eine Vielzahl unterschiedlichen, mit einer RGB-D Kamera aufgenommenen Szenen, die gezielt unterschiedliche Schwierigkeiten des Slam simulieren. Der hier relevante Teil des Datensatzes ist der ‘Dynamic Objects’ Teildatensatz. In diesem wird eine Tisch-Szene, mit Menschen bewegten Menschen aufgenommen. Folgende Szenen werden für die Evaluation genutzt:

- *static xyz*: Die Kamera bewegt sich durch Translations-Bewegung und filmt eine statische Tisch-Szene
- *static rpy*: Die Kamera bewegt sich durch Rotation-Bewegung und filmt eine statische 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 Segmentierungsmodells Detectron2 [28]. Dabei liegen die Masken in unbearbeiteten Form vor.

### 3.3.2. Metriken

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

Für den Vergleich der Position können zwei Fehlerarten betrachtet werden: Translations- und Rotationsfehler. Der Unterschied zwischen zwei Posen lässt sich durch eine Fehlertransformation darstellen, aus der sowohl der Translations- als auch der Rotationsfehler 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 (vlg. [25]):

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

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

**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. [25]

#### RPE

Der Realitive Pose Error (RPE) ist ein Maß für den Drift eines SLAM-Algorithmus's. 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. [25]):

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

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

### 3.3.3. maskierte Odometrie Auswertung

Die Testkonaufbau besteht aus der Trajektorie-Erstellung mit der visuellen Odometrie. Dazu werden Konfigurationen mit verschiedenen Fehlerfunktionen (siehe 2.3.5) und Maskierungen betrachtet. Der Algorithmus wird 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 die Run-to-Run Varianz unter 0.1% für RPE und ATE. Im folgenden wird deshalb die visuelle Odometrie als deterministische betrachtet.

Das genutzte Gerät bei den Versuchen ist die Grafikkarte. Sie stellt die geeignete Hardware für die Verarbeitung von Bildern da und liefert eine Geschwindigkeits-Verbesserung von bis zu 10 %.

## Laufzeit

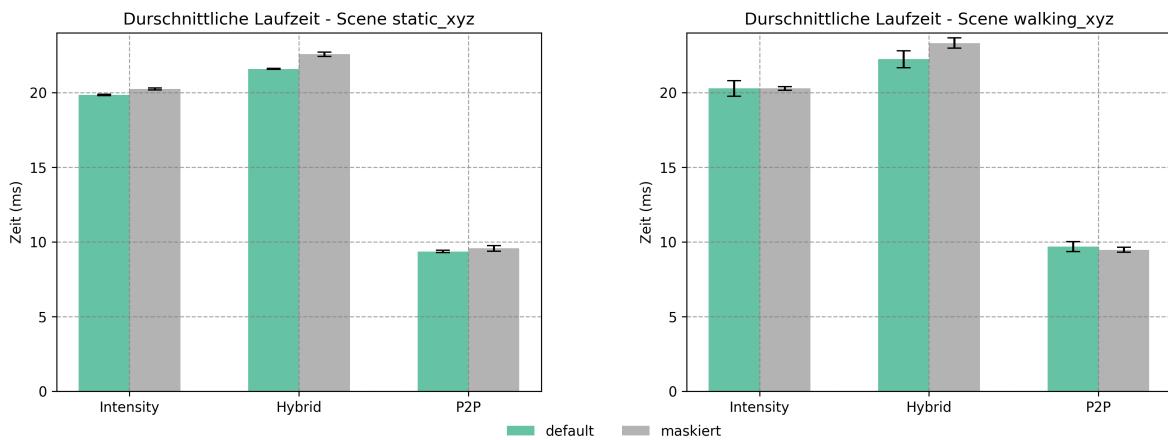


Abbildung 3.1.: durchschnittliche Laufzeit visueller Odometrie. Vergleich von statischer und dynamischer Umgebung

In der Grafik 3.1 ist zu erkennen, dass jede betrachtete Konfiguration in Echtzeit läuft (für dichtes visuelles Tracking ist eine Framerate von 30 Fps üblich). Die Laufzeit der Intensität's und Hybrid Fehlerfunktion deutlich höher als die des P2P. Dies kann durch einen erhöhten Belastung des Speicher erklärt werden. Auch ist der Rechenaufwand zum Bestimmen der Gradienten-Bildern 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 gering. In den statischen Szenen ist ein leichter Trend der Verringerung der Laufzeit in den maskierten Varianten zu erkennen. Ein Entfallen von Jacobimatrix Berechnungen kann ein Grund für das Verhalten liefern.

## Drift Analyse

Visuelle Odometrie bildet einen zentralen Baustein zur Bewegungsbestimmung aus Bilddaten, ist jedoch nicht direkt mit einem SLAM-System vergleichbar. In Tabelle A.2 ist

### 3. Masken basiertes Tracking und Szenen Rekonstruktion

---

eine starke Abweichung der Trajektorie zu erkennen. Für die Bewertung eines solchen Algorithmus ist der RPE angebracht.

In den Daten A.1 ist ein klarer Trend erkennbar: Beim Translationsfehler weist das P2P-Verfahren einen geringeren Drift auf, unabhängig von der grundlegenden Bewegung.

Für den Rotationsdrift zeigt sich ein ähnliches Muster, allerdings weniger ausgeprägt. Die Beobachtung ist nicht als ein Beleg für eine höhere rotatorische Robustheit des P2P zu sehen. Dafür sprechen die Szenen mit starker Rotation und ohne dynamische Objekte („static rpy“), in denen das P2P-Verfahren deutlich schlechter performt.

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



Abbildung 3.2.: Prozentuale Verbesserung des RPE durch maskierten Odometrie

maskierten.

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 darstellt. Unter dieser Annahme kann 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 und Verhalten der Methoden auf der „static rpy“ Szenen.

### 3.3.4. maskiertes 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 die selbe Konfiguration genutzt. Für das Raycasting wird das Standart-Verfahren aus [8, 29] genutzt. Die Vergleichs-Version des TSDF-Tracking in Open3D nutzt zudem auch einen Gewichtungswert pro Voxel um die Stabilität zu erhöhen. Auch in diesem Aufbau wird das Tracking mehrfach auf der gleichen Szene simuliert.

#### Laufzeit

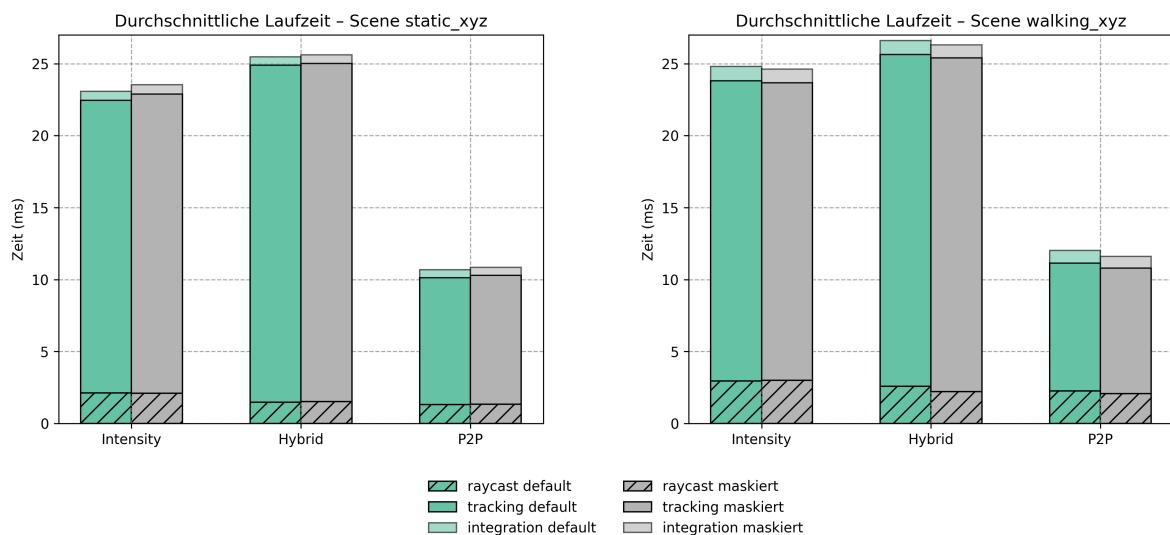


Abbildung 3.3.: Durchschnittliche Laufzeit des TSDF-Trackings

Ein Großteil der Laufzeit pro Iteration stellt das Tracking durch visuelle Odometrie dar (vgl. Abb. 3.3). Dementsprechend ist das Verhältnis der Laufzeiten für die verschiedenen Fehlerfunktionen vergleichbar zu Abb. 3.1.

Ein weiteres beobachtetes 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 aktivierte Blocks und Voxel gefunden werden. In den dynamischen Szenen entsteht mehr fehlerhafte Geometrie, die einen negativen Einfluss auf die Rechenzeit des Raycasting unter Integration haben.

#### Tracking-Verlust

In den Daten der Tabellen A.1 und A.3 ist ein Phänomen zu beobachten: Es bestehen starke Schwankungen in ATE und RPE selbst auf den statischen Szenen. Besonders in den ATE-Werten A.4 sind großen Schwankungen auf gleichen Szene zu beobachten.

Das ist ein Symptom eines generellen Problem des Trackings mit einem TSDF-VoxelBlockGrid, 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 Kamerarasichtfeld auf noch invalide Geometrie gerichtet ist. Dadurch entstehen unvollständige synthetische Bilder. Das Tracking auf Bildern mit einer geringen Anzahl valider 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, wieder eine Referenz zu Geometrie gefunden 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 von Trajektorien.

#### **Fehlerfunktionen**

Anhand der Betrachtung der ATE-Varianz ist erkennbar, dass der Intensität's-Fehler anfällig für Tracking-Verlust ist. Die schlechte Performance ist auch erwartbar. Die synthetisch erstellten Bilder simulieren nicht mit akkurate Licht- und Schatten und somit ist ein Vergleich von diesen nicht sinnvoll.[8]

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. Die RPE- und ATE-Werte der statischen Szenen zeigen, dass der P2P-Fehler im Vergleich zu den anderen Verlustfunktionen eine stabile und präzise Schätzung liefert. Aus der Abb. 3.4 ist eine deutlicher Verbesserung des Drifts zu erkennen. Aufgrund der Beobachtung wird im folgenden wird explicit nur der P2P genauer in dynamischen Szenen betrachtet.

#### **Dynamische Szenen**

Die Betrachtung des „Standard“ TSDF-Tracking auf den dynamischen Szenen zeigt eine Verschlechterung gegenüber der reinen visuellen Odometrie. 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 Qualität des Trackings ist stark an die der geraycasteten-Bilder gekoppelt. Feh-

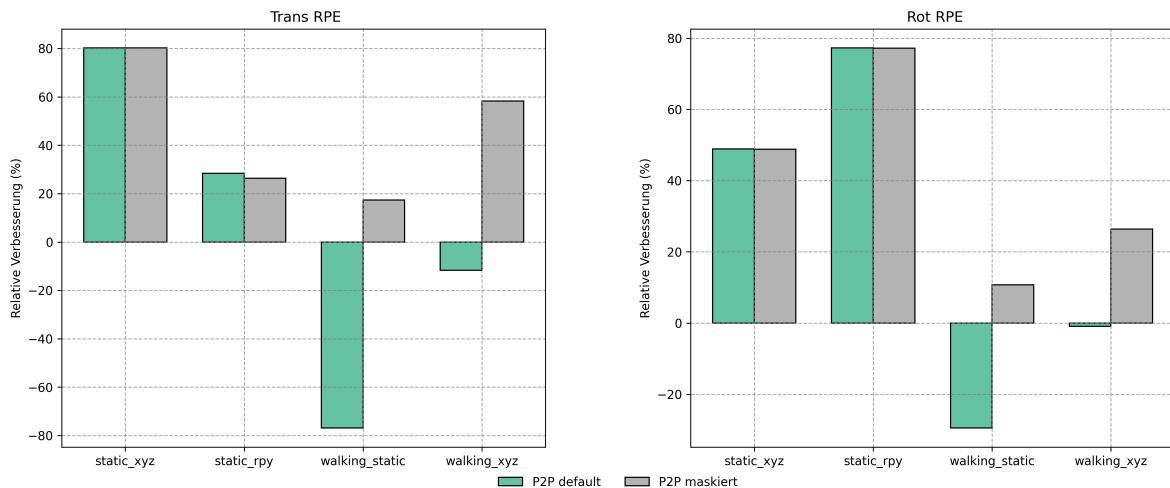


Abbildung 3.4.: Relative Verbesserung des Drifts der P2P-Fehlerfunktion

lerhafte Geometrie die durch bewegende Objekte in das interne Modell eingeführt wird verschlechtert die Referenzpunkte (vgl. Abb. 3.5). Der positive Effekt der synthetischen Bilder ist somit nicht mehr vorhanden und es wird zusätzlich das Tracking behindert.

Die maskierte Variante des P2P Trackings ist es als einzigen Konfiguration des TSDF-Tracking möglich auf den dynamischen Szene eine Tracking-Loss. 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 besitzt der visuelle Odometrie Algorithmus Schwierigkeiten auf der Rotations-Szene.

### 3.3.5. Bewertung

Die beschriebenen Experimente zeigen deutlich das der P2P den Intensität's und Hybrid Fehler überlegen ist. Ähnliche Beobachtung wurden auch in [23] getätigt. In der Laufzeit ist die P2P Fehlerfunktion auch deutlich überlegen. Das Muster setzt sich auch auf das TSDF-Tracking fort. Dieser Trend ist jedoch erwartbar, da die synthetischen Bilder das Tracking per Intensität's Werten behindert.

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

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 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.

### 3. Masken basiertes Tracking und Szenen Rekonstruktion



(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

Somit stellen beide Verfahren kein robustes vSLAM-System dar. Sie können jedoch als Komponenten in einem System eingesetzt werden, das mehrere Tracking-Algorithmen kombiniert und deren Ergebnisse fusioniert. Ein solches Zusammenspiel unterschiedlicher Verfahren mit jeweils eigenen Stärken und Schwächen ist charakteristisch für leistungsfähige SLAM-Systeme.

Ein möglicher Ansatz wäre das Tracking durch die visuell basierenden Fehlerfunktionen zu überwachen, um einen möglichen Trackingverlust zu erkennen. Das führt jedoch in das diverse und umfangreiche Feld der vSLAM-Systeme, welches nicht genauer 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 Roboters gering. Somit treten die Hauptkomponenten für eine Tracking-Verlust nicht auf. Die Tiere stellen

dabei ein teilweise kontrollierbaren Faktor dar, welchem mit maskierten Tracking und Integration entgegenwirkt werden kann.

# 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. Speziell bei der Rekonstruktion der Futterstelle sind Masken, als Eingabe in die maskierten Algorithmen nötig (siehe 3).

Bei der Bildsegmentierung wird unterschieden zwischen der Genric Image Segmentation (GIS) und Promtable Image Segmentation (PIS). Während PIS speziell für prompt-fähige Modelle entwickelt ist, die ihre Segmentierung auf textuellen oder exemplarischen Eingaben basieren, beschreibt GIS das automatische Klassensegmentieren ohne zusätzliche Benutzereingaben (vgl. [30]). Das automatische Generieren von Kuh-Masken fällt in das Gebiet des GIS. Grundlegende Problemstellungen des GIS sind (vgl. [30, 5]):

- **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

## 4.1. Datensatz

Der Datensatz besteht aus einer Menge an Kamerafahrten, die in dem selben Stall aufgenommen wurden. Dabei wird in jeder Kamerafahrt ein Fahrt des Schieberoboters simuliert. Sie unterscheiden sich untereinander in der Orientierung des Sensors zu der Futterstelle und in dem Sichtfeld.

Die Aufnahmen stellen hoch dynamische Szenen da, mit vielen sich im Hintergrund und Vordergrund bewegenden Kühen. Viele Kühe sind dabei verdeckt oder klein in dem Hintergrund sichtbar. Die Lichtverhältnisse variieren, wodurch die Qualität der Bilder gemindert wird. Insgesamt besteht der Datensatz aus 6 Kamera-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 Unterschiede in der Kamera-Positionierung besteht 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 dar. Für die semantische Segmentierung muss eine Klassifikation pro Pixel annotiert werden. Um den Prozess zu erleichtern ist die Nutzung von existierenden Segmentierungs-Modellen im Labeling Prozess essenziell.

Die unmittelbare Nutzung von GIS Segmentierungsmodellen ist nicht möglich. Aufgrund der Komplexität der gefilmten Szene, sowohl als auch der spezifischen Anforderungen der Klassen, ist viel manuelles Postprocessing für die Masken nötig.

Für effektiveres und interaktives Labeln können PIS Segmentierungs-Modelle genutzt werden. Konkret wurde das SAM2 [22] verwendet. Das Modell bietet die Möglichkeit Segmentierung aufgrund von visuellen Prompts zu erreichen. Dafür können Punkte, Bounding-Boxes und Vorsegmentierung als Input in das Modell gegeben werden. Somit ist ein flexibles und schnelles Segmentieren von selbst schwer semantisch zu erkennenden Regionen möglich.

Trotz dem flexiblen Labeling Setup ist manuelles Postprocessing nötig.

Die Segmentierungen sind nicht pixelgenau. Aufgrund ungünstiger Lichtverhältnisse lassen sich Kühe im Hintergrund schwer oder gar nicht erkennen. Teile der Bilder stellen nicht erkennbare Regionen dar in denen eine Annotation nicht möglich ist.

### 4.1.2. Evaluation und Trainingsdatensatz

Der Datensatz weist Herausforderungen auf. Bilder aus demselben Durchlauf sind aufgrund temporaler Überschneidungen stark miteinander korreliert. Daher ist bei der Trennung in Trainings- und Validationsdaten besondere Vorsicht geboten. Ein akkurate Vorgehen wäre, für Trainings- und Validationsdatensatz Aufnahme aus verschiedenen Ställen zu verwenden. Das ist jedoch zu diesem Zeitpunkt nicht möglich. Zukünftig ist geplant, den Datensatz in dieser Hinsicht zu diversifizieren.

Eine perfekte Trennung der Daten ohne Korrelation zwischen Splits ist nicht möglich. Der Hintergrund, sowie die Futterstelle bleiben unverändert zwischen Aufnahmen. Das bestmögliche Vorgehen auf dem Datensatz, in seinen derzeitigen Zustand, ist eine Trennung anhand der einzelnen Kamerafahrten.

#### *4. Semantische Bild Segmentierung*

---

Aufgrund einer Imbalance zwischen der Anzahl der Bilder in den einzelnen Läufen ist dieses Vorgehen bringt das Vorgehen Probleme mit sich. Eine faire Trennung ist nicht ohne große Variation der Größe von Validationsdatensatze oder einem großen Verlust von annotierten Bilder möglich.

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

Die Wahl der Splits beeinflusst die zeitliche Position der Validations-Bilder. Um diesen Faktor zu kontrollieren wird eine Teilung mithilfe von 5-Folds Crossvalidation festgelegt. Dabei werden die temporalen Abschnitte zufällig in jeder Fahrt gewählt, ohne dass dabei einen eine Menge von Bildern mehrfach als Validationsdatensatz vorkommt.

Dieses Vorgehen ist nicht akzeptabel für einen Testdatensatz. Daher stellt dieser eine eigenen Kamerafahrt da, mit zeitlichen Abstand zwischen zu den anderen. Ein großes Problem ist: Die Test-Aufnahme stammt von dem selben Tag und Stall. Es bestehen somit starke Korrelation, was die Genrealisierbarkeit der erzielten Ergebnisse deutlich infrage stellt.

#### **4.1.3. Augmentation**

Aufgrund der kleinen Größe 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, 11]). Auch wird die Helligkeit und der Kontrast zufällig variiert.

### **4.2. Model-Architektur**

Für die semantischen Segmentierung von Bildaufnahmen existieren zahlreiche Ansätze. Eine weit verbreitete Herangehensweise basiert auf überwachtem Lernen, wobei insbesondere Architekturen auf Grundlage von Convolutional Neural Network (CNN)'s häufig eingesetzt werden. Darüber hinaus wurden verschiedene Erweiterungen entwickelt, die beispielsweise Attention-Mechanismen, Conditional Random Fields oder Recurrent Neural Networks einbeziehen. In jüngerer Zeit haben zudem Transformer-Modelle zunehmend an Bedeutung gewonnen (vgl. [5, 30, 16]).

Als Architektur wird DeepLabV3+ gewählt, eine aktuelle Weiterentwicklung der DeepLab-Familie. Diese Modelle basieren auf dem Konzept der Atrous Convolutions und integrieren moderne Erweiterungen wie Encoder-Decoder-Strukturen mit separierbaren Faltungen [4, 3]. DeepLabV3+ stellt eine weit verbreitete Architektur dar, die sowohl in der Forschung als auch in zahlreichen industriellen Anwendungen erfolgreich eingesetzt wird (vgl. [5, 16]). Die Grundlage von Deeplab bildet ein abgepasstes Fully Oonvolutional

Network (FCN).

### 4.2.1. CNN-Backbone

Um die Deeplab Architektur zu verstehen muss zuerst die Funktionsweise von FCN's verstanden werden. FCN basieren auf der Convolution (Faltung) als zentrale Operation.

**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  $I : \Omega \rightarrow \mathbb{R}$ .

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  $K : \Omega^* \rightarrow \mathbb{R}$ .

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. [21]

**Bemerkung 4.1.** Betrachtet man die Definition der Bildfaltung, so zeigt sich, dass diese nicht für alle Pixelkoordinaten  $(x, y, c)$  definiert ist. An den Bildrändern ragt das Faltungsfenster über den eigentlichen Bildbereich hinaus. In der Praxis wird dieses Problem durch Padding gelöst, bei dem der Definitionsbereich des Eingabebildes durch Standardwerte erweitert wird. Dadurch bleibt die Auflösung nach der Faltung erhalten und verkleinert sich nicht schrittweise [21].

Die Faltung mit einem Kern  $K$  stellt eine gewichtet Summe über die Input-Feature Channels eines lokalen Fenster's der Größe  $k_w \times k_h$  dar. Durch den Kern  $K$  werden  $C_{out}$  viele Gewichtungen separat durchgeführt.

Einzelne Convolution-Schichten in einem FCN bestehen jeweils aus einem zu lernenden Kern  $K$ . Das allgemeine Prinzip besteht darin, die Anzahl der Feature-Channels sukzessive zu erhöhen, indem Faltungen mit zunehmender Ausgabedimension  $C_{out}$  angewendet werden. Eine kontinuierliche Erhöhung der Kanalanzahl ist jedoch nicht praktikabel, da der Speicherbedarf bei hochauflösten Feature-Maps stark ansteigt. Daher wird die Kanalanzahl typischerweise nur bei gleichzeitiger Verringerung der räumlichen Auflösung der Feature-Maps erhöht. Eine Methode für das Verkleiner'n der Auflösung sind strided Convolutions.

**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 halbiert und gleichzeitig die Anzahl der Feature Channels verdoppelt. Der Faktor um den Auflösung reduziert wird, wird als der Outputstride bezeichnet. Eine Outputstride von 32 und größer ist geläufig.

**Bemerkung 4.2.** In FCN werden neben strided Convolutions auch Pooling Layers genutzt um die Auflösung zu verkleinern. Bei einem Pooling-Layer wird auf Fenstern Durchschnitt's Werte oder Maxima berechnet der Channels berechnet.

### 4.2.2. Encoder

Der Encoder von Deeplab besteht großteils aus dem Backbone Model. Dieses liegt in modifizierter Form vor. Das Problem welches auftritt, wenn das unveränderte Backbone Model genutzt wird ist, dass der Outputstride groß ist. Um auf die originale Auflösung zukommen müssen die Features dementsprechend hochskaliert werden, was zu einer groben Akkuratheit der Pixel Klassifikation führt. Das Vorgehen besteht darin den Outputstride des Backbone-Modells niedrig zu halten.

#### Atrous Convolutions

Ein niedriger Outputstride führt zu unerwünschten Folgen. Convolutions in späteren Schichten haben die die Eigenschaft, dass durch die Verringerung der Auflösung, größere Bildbereiche durch Convolution-Fenster wahrgenommen nehmen. 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 festen Outputstride werden strided Convolution durch entsprechenden Atrous Convolution ersetzt [3].

**Bemerkung 4.3.** In der Implementation werden vortrainierte Parameter von dem Backbone Model genutzt werden (vgl. [3, 11]). 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 . [9]*

## Seperable Convolutions

### Atrous Spatial Pyramid Pooling

Eine Innovation die spätere Deeplab-Architekturen anwenden, ist das Atrous Spatial Pyramid Pooling (ASPP). Ein generelles Vorgehen in Segmentierungs-Modellen sind Bild-Pyramiden Module. Sie sollen die Intrinsische Lokalität von Convolution entgegen wirken, indem mehrere Bildskalen gleichzeitig betrachtet werden (vlg. [5]). Deeplab nutzt dabei Atrous-Convolutions um Multiskalen-Daten zu erfassen [3]. Dafür werden mehrere separable Atrous Convolution mit unterschiedlichen Raten und Padding in parallel angewendet auf eine Feature Map angewandt (siehe 4.1). Zudem wird eine 1x1 Convolution und ein 2x2 Average Pooling Layer genutzt.

Das Padding in den Atrous Convolution wird dabei so gewählt, dass die Auflösung der Feature Map erhalten bleibt. Der Output des Pooling Layers wird auf die dieselbe Auflösung hoch interpoliert. Somit haben alle Outputs die gleiche Auflösung und können verkettet werden. Durch eine  $1 \times 1$  Convolution werden die Multiskalen Feature-Channel kombiniert und projiziert [3]

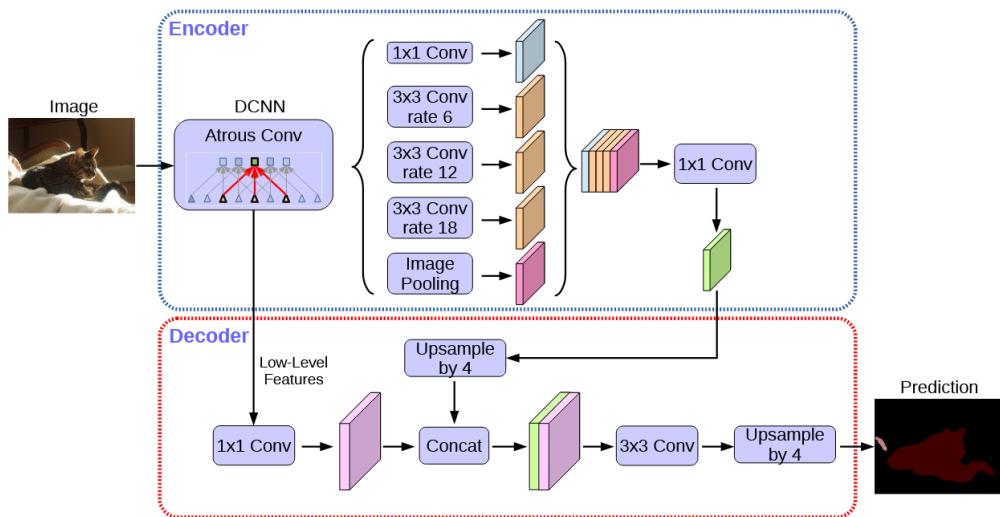


Abbildung 4.1.: DeepLabv3+ Architecture (Original aus [4])

### 4.2.3. Decoder

In früheren Deeplab Modellen wurde die Ergebnisse des ASPP Modules Bilinear auf die originale Auflösung hoch interpoliert. Eine Verfeinerung der Vorhersagen wurde mithilfe von CRF's erreicht. Folgende Deeplab Modelle nutzen nicht mehr CRF's sondern setzen auf einen Decoder für eine Bildregionen-Verfeinerung [4, 3].

In dem Decoder werden vorherige Featuremaps (Low-Level-Features) genutzt um die hoch interpolierten Outputs des ASPP-Moduls (High-Level-Features) zu verfeinern. Dazu werden die High- und Low-Level Features verkettet und mithilfe von  $3 \times 3$  Convolutionen fusioniert. Auf den erzeugten Feature's wird eine per Pixel Klassifikation durchgeführt mithilfe einer  $1 \times 1$  Convolution (siehe Abb. 4.1).

Ziel dieses Prozesses ist es, eine präzisere Klassifikation insbesondere an den Objekt- und Regionsrändern zu ermöglichen.

## 4.3. Trainingskonfiguration

Für das Training des Model wird eine modulare Implementation von dem Deeplabv3plus Model in Pytorch genutzt [9]. Das Repository erlaubt es vortrainierten Parameter einfach für verschiedene Backbone Modelle zu laden. Dabei stehen die Modelle ResNet, MobilenetV2, Xception, und Hrnet zu Verfügung.

Als Backbone-Model für die Experimente wurde das ResNet-50 gewählt. Es handelt sich um ein neuronales Netz, das auf Standard-Convolutionen basiert. Im Gegensatz dazu setzen Modelle wie MobileNetV2 oder Xception auf Depthwise-Separable-Convolutionen, was zu deutlich weniger Parametern und geringerem Rechenaufwand führt. Die ResNet Architektur wird auch als Backbone in dem Paper [3, 11] 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 [6] trainiert.

### 4.3.1. Hyperparameter

Die Konfiguration des Training bietet eine Vielzahl von zu wählenden Parameter, die sogenannten Hyperparameter. Sie müssen vor dem Training eines Modells festgelegt

werden.

### Loss

Für die Bildsegmentierung, sowie allgemein Klassifikation-Probleme ist der Cross-Entropy-Loss eine gängige Fehlerfunktion (vlg. [3, 5]).

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

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

Der Gewichtungswert wird für das manuelle Ausgleichen von Klassenungleichgewichten genutzt.

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

**Definition 10.** Sei die Notation gleich zu der in 4.1. Dann ist der Focal-Loss gegeben durch (vgl. [12]).

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

Der Focal-Loss stellt eine adaptive Gewichtung des Cross-Entropy-Losses da, bei dem Pixel mit hoher vorhergesagter Confidence geringer gewichtet werden als Pixel mit niedriger Confidence. Somit wird automatisch der Fokus auf schwer zu klassifizierende Pixel gelegt wird.

### Weight-Decay

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

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

$W$  stellt den Vektor mit allen trainierbaren Parametern des Modells da.  $\lambda$  ist der Weight-Decay Faktor. Er ist ein Strafterm für die Komplexität des Modells.

### Optimierer

Für das Training wurde der AdamW-Optimierer eingesetzt. AdamW ist ein Optimierer auf Basis von Adam, bei dem das Weight-Decay korrekt vom Gradientenupdate entkoppelt ist [13]. Er stellt einen robusten Optimierer für Tiefe CNN's da. Im Gegensatz zu dem häufig genutzten Moment basierte Stochastic-Gradient-Decent Optimizer, ist ein geringe Wahl von Hyperparameter nötig.

## Backbone Training

Aufgrund der Größe des Datensatzes ist es nicht sinnvoll ein Tiefes Convolutionelles Modell auf ihm neu zu trainieren. Aus diesem Grund wird versucht die vortrainierte Parameter unverändert zu lassen. Dafür wird für das die Lernrate für die Parameter des Backbone-Modells verringert (vlg. [3]).

Eine andere Herangehensweise ist es die Parameter aus dem Training auszuschließen. Dabei muss beachtet werden nur die Parameter zu entfernen die auch unverändert auf der jeweiligen Feature-Map operieren. Wie schon in der Bemerkung 4.3 erwähnt, werden bestimmte Parameter des Backbone-Modells in Atrous-Convolutions verwendet. Aufgrund des veränderten Control-Flow des Modells sollten sie auch mittrainiert werden.

## Outputstride

Für die Konfiguration des Backbone-Models hat die Wahl des Outputstride ein starken Einfluss. Ist großer Outputstride erhöht den Rechenaufwand und Speicherverbrauch [3]. Ist er zu klein sind mehr Interpolationsschritte nötig. Der Outputstride wird aus praktischen Gründen auf 16 festgelegt (vlg. [3]).

### 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 [27]. Dabei wird versucht eine bestimmte Metrik zu optimieren. Die Implementierung beider Algorithmen sowie die Steuerung des gesamten Optimierungsprozesses erfolgt mithilfe des Hyperparameter-Optimierungsframeworks Optuna [2].

Die Parameterräume für die genutzten Parameter sind in der Tabelle 4.2 angegeben.

Parameter	Range	Skalierung
Lernrate	$[1^{-2}, 1^{-5}]$	logarithmisch
Batch-Size	$[3, 15]$	uniform
Background-Weight	$[0.1, 1]$	uniform
Weight-Decay	$[1^{-2}, 1^{-6}]$	logarithmisch
Loss	[Focal, Cross-Entropy]	-
Backbone-Training	[True, False]	-

Abbildung 4.2.: Suchraum der Hyperparameter für Optuna

## 4.4. Trainingsergebnisse

### 4.4.1. Metriken

Die Bewertung der semantischen Segmentierung erfordert Metriken, die eine Vielzahl paralleler Vorhersagen berücksichtigen, da jeder Pixel einem eigenen Klassifikationsproblem entspricht. In der verwendeten Notation bezeichnet  $C$  die Anzahl der Klassen.  $n_i^j$  beschreibt die Anzahl der Pixel, die zu der Klasse  $i$  gehören und als Klasse  $j$  vorhergesagt wurde.  $t_i$  steht für die Gesamtzahl der Pixel der Klasse  $i$ , während  $G$  die Gesamtzahl aller Vorhersagen beschreibt.[5]

#### Accuracy

Eine der gebräuchlichsten Metriken für Klassifikationsaufgaben ist die Accuracy. Im Kontext der semantischen Segmentierung wird die Pixel Accuracy verwendet. Sie beschreibt das Verhältnis der korrekt klassifizierten Pixel zur Gesamtzahl aller Pixel im Bild.

$$Pixel\text{Accuracy} : \frac{1}{G} \sum_i n_i^i \quad (4.4)$$

Die Mean Accuracy beschreibt dagegen die Akkurateit der Vorhersagen in den einzelnen Klassen (vlg.[5]).

$$Mean\text{Accuracy} : \frac{1}{C} \sum_i \frac{n_i^i}{t_i} \quad (4.5)$$

#### IOU

In der semantischen Segmentierung ist nicht immer eine pixelgenaue Vorhersage erforderlich. Häufig steht vielmehr die korrekte Identifikation ganzer Bildregionen im Vordergrund. Der Intersection over Union (IoU) dient dabei als Maß zur Bewertung der Übereinstimmung zwischen vorhergesagten und tatsächlichen Segmenten. Er berücksichtigt sowohl False Positives als auch False Negatives und hat sich als Standardmetrik für die Evaluation semantischer Segmentierungsverfahren etabliert.

Der Class IoU beschreibt das Verhältnis von richtig vorhergesagten Pixel der Klasse

## 4. Semantische Bild Segmentierung

---

$c$  zu der Anzahl die insgesamt als  $c$  vorhergesagt wurden.

$$ClassIoU(c) = \frac{n_c^c}{t_c + \sum_j^C n_c^j - n_c^c} \quad (4.6)$$

Der Mean IoU setzt sich als Durchschnitt über die einzelnen Class IoU zusammen. Bei dem Frequency-weighted-IoU bietet ein Maß der generellen Genauigkeit der Bildvorhersage Regionen.[5]

$$MeanIoU : \frac{1}{C} \sum_i ClassIoU(i) \quad (4.7)$$

$$FrequencyWeightedIoU : \frac{1}{G} \sum_i t_i ClassIoU(i) \quad (4.8)$$

Als Objective-Funktion der Hyperparameter-Optimierung wird Mean-IoU. Aufgrund der Imbalance zwischen Hintergrund und den restlichen Klassen, kann

Metriken nicht gut die Akkuratheit auf den Rest der Labels. Da der IoU die Standard-Metrik für die semantische Segmentation darstellt wird, ist das Ziel des Optimierers den Mean-IoU zu maximieren.

### 4.4.2. Ergebnisse

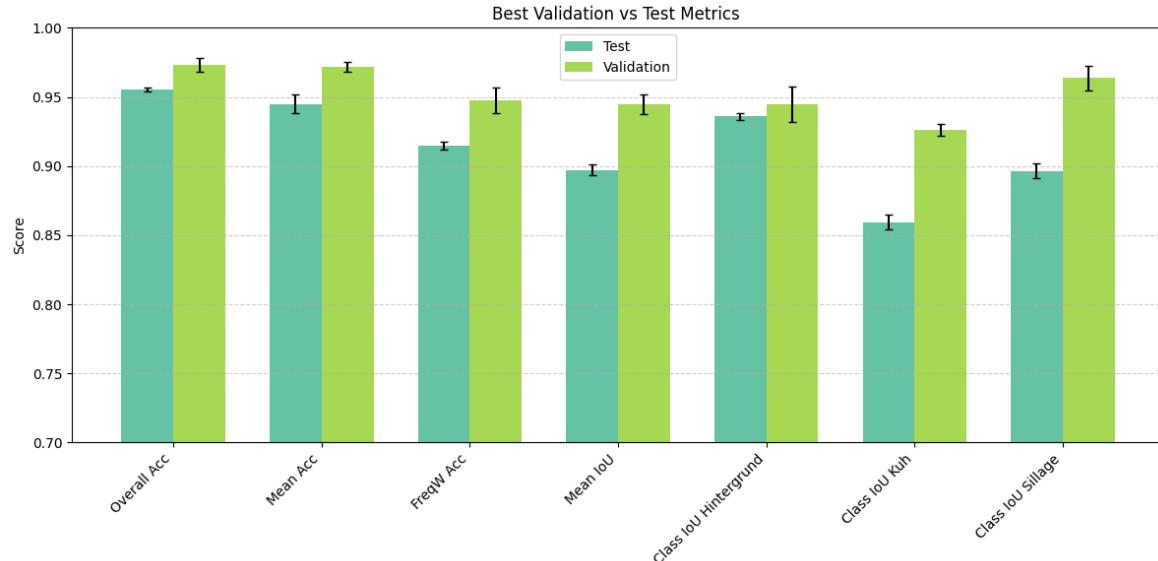


Abbildung 4.3.: Optimierung verlauf

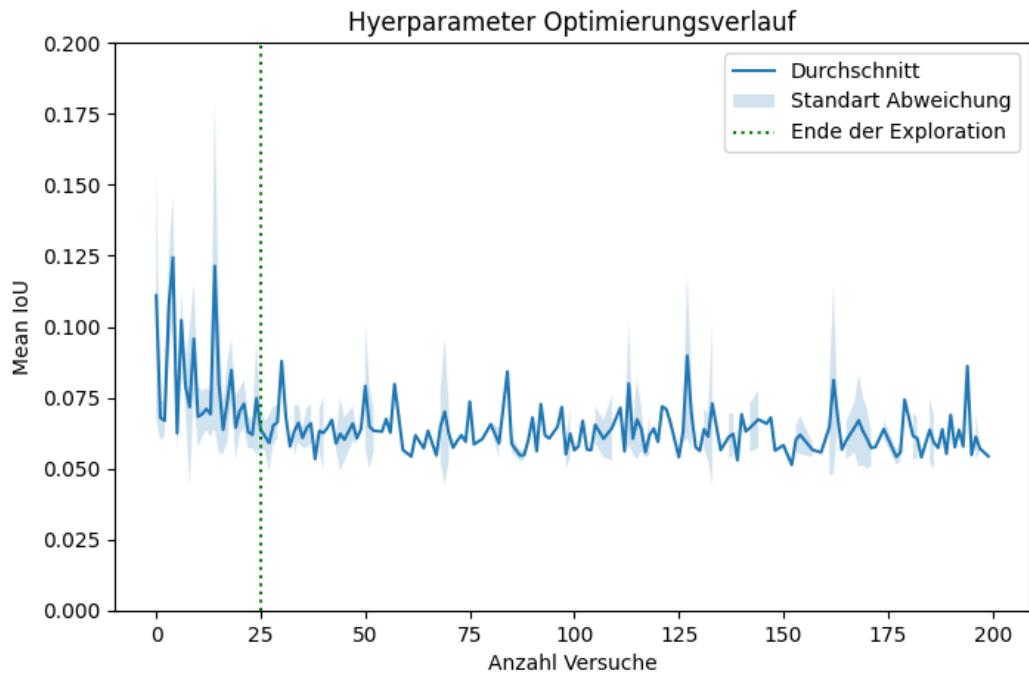
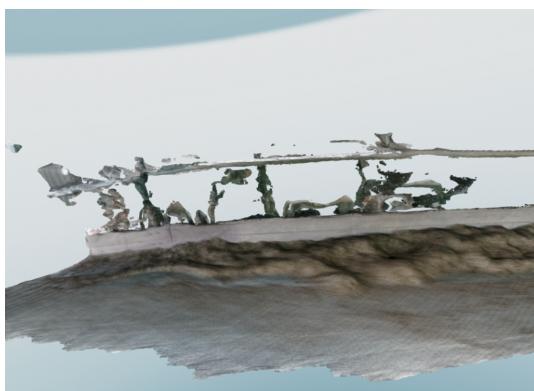


Abbildung 4.4.: Optimierung verlauf

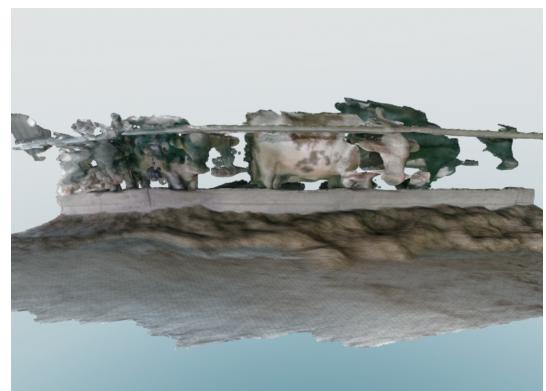
Tabelle 4.1.: Test Metriken

Mean IoU [%]	FreqW IoU [%]	Mean Acc [%]	Pixel Acc [%]	Cow IoU [%]
--------------	---------------	--------------	---------------	-------------

---



(a) Stall Rekonstruktion mit Segmentierungsmasken



(b) Stall Rekonstruktion unmaskiert

Abbildung 4.5.: Vergleich Rekonstruktion mit Segmentierungsmasken

## 5. Fazit und Ausblick

Das Ziel dieser Arbeit war es, ausgewählte Rekonstruktions- und Tracking-Algorithmen im Kontext von Stallumgebungen zu verbessern. Hierfür wurden die Verfahren der visuellen Odometrie sowie des TSDF-Trackings durch die Einbeziehung von Masken erweitert. Die Anpassungen führten insbesondere in dynamischen Umgebungen zu deutlichen Verbesserungen. Besonders hervorzuheben ist dabei das TSDF-Tracking in Kombination mit dem Point-to-Point-Fehler und der Maskierung, das insgesamt robustere Ergebnisse erzielen konnte.

Als nächster potenzieller Schritt erscheint die Betrachtung von Methoden zur Fusion mehrerer Tracking-Ansätze sinnvoll, um das TSDF-Tracking weiter zu stabilisieren und langfristig zu robustifizieren. Eine fortbestehende Herausforderung liegt dabei in der Szenenerfassung bei langen Kameraverläufen. Aufgrund der baulichen Struktur des Stalls stehen nur wenige stabile Referenzpunkte über die Zeit zur Verfügung, was zu einer Krümmung der rekonstruierten Szenen führen kann.

Ein zweiter Schwerpunkt der Arbeit war die automatische Erstellung geeigneter Masken durch ein trainiertes Segmentierungsmodell. Die Segmentierung lieferte adäquate Resultate durch das grobe Erkennen von Kühen und ermöglicht somit eine partielle Entfernung ihres Einflusses auf die Rekonstruktionsmodelle.

Für die Segmentierung ergibt sich als weiterer Arbeitsschritt die Diversifizierung des Datensatzes, um die Ergebnisse auf andere Umgebungen übertragen zu können. Darüber hinaus bietet sich eine Portierung auf ressourcenschonendere Architekturen wie MobileNet an, um die Nutzung auch in rechenbeschränkten Systemen zu ermöglichen. Schließlich können die entwickelten Segmentierungsmasken in Zukunft auch für zusätzliche Anwendungen genutzt werden, etwa für das Finale Ziel der präziseren Futtermengenmessung.

## **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 ± 7.96%	0.0195 ± 6.34%	0.0184 ± 7.47%	0.0474 ± 0.83%	0.0124 ± 3.66%	0.0065 ± 1.41%	0.0879 ± 25.90%	0.0323 ± 21.51%	—	—
Intensität maskiert	0.0209 ± 3.64%	0.0194 ± 2.68%	0.0180 ± 3.27%	0.0471 ± 0.68%	0.0165 ± 19.65%	0.0075 ± 10.14%	0.0940 ± 37.86%	0.0343 ± 34.52%	—	—
Hybrid default	0.0115 ± 1.12%	0.0144 ± 0.36%	0.0318 ± 6.39%	0.0470 ± 2.81%	0.0101 ± 12.80%	0.0061 ± 5.66%	0.0287 ± 9.87%	0.0158 ± 3.96%	—	—
Hybrid maskiert	0.0114 ± 0.58%	0.0144 ± 0.24%	0.0329 ± 7.02%	0.0486 ± 4.72%	0.0075 ± 1.26%	0.0056 ± 0.68%	0.0280 ± 2.91%	0.0141 ± 1.17%	—	—
P2P default	0.0044 ± 0.04%	0.0109 ± 0.02%	0.0075 ± 0.07%	0.0201 ± 0.02%	0.0181 ± 7.50%	0.0094 ± 6.75%	0.0296 ± 9.46%	0.0177 ± 7.61%	—	—
P2P maskiert	0.0045 ± 0.05%	0.0109 ± 0.02%	0.0078 ± 0.08%	0.0201 ± 0.03%	0.0041 ± 0.22%	0.0057 ± 0.03%	0.0079 ± 0.52%	0.0126 ± 0.04%	0.0611 ± 51.08%	0.0399 ± 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]								
Intensität default	1.0399 ± 18.94%	0.6811 ± 21.08%	0.6808 ± 16.23%	1.0558 ± 12.88%	0.0683 ± 3.91%	0.0215 ± 3.68%	2.7599 ± 33.63%	1.0619 ± 52.83%	—	—
Intensität maskiert	1.2910 ± 32.83%	0.9918 ± 26.01%	0.7712 ± 9.00%	1.0139 ± 14.92%	0.0425 ± 25.70%	0.0173 ± 13.49%	2.8556 ± 31.88%	1.1483 ± 56.38%	—	—
Hybrid default	0.0578 ± 2.17%	0.1179 ± 1.29%	1.0401 ± 38.40%	0.8918 ± 20.38%	0.1712 ± 87.18%	0.0543 ± 83.24%	2.0014 ± 24.41%	0.9059 ± 14.65%	—	—
Hybrid maskiert	0.0544 ± 1.36%	0.1178 ± 0.72%	1.0982 ± 43.08%	0.9033 ± 19.95%	0.0310 ± 1.87%	0.0150 ± 0.77%	0.2893 ± 3.71%	0.0655 ± 17.25%	—	—
P2P default	0.0330 ± 0.03%	0.0336 ± 0.02%	0.0766 ± 0.10%	0.1021 ± 0.02%	0.8969 ± 19.69%	0.2534 ± 35.27%	1.2095 ± 17.29%	0.8985 ± 10.30%	—	—
P2P maskiert	0.0329 ± 0.02%	0.0336 ± 0.02%	0.0759 ± 0.07%	0.1022 ± 0.02%	0.0466 ± 0.03%	0.0136 ± 0.04%	0.1053 ± 0.05%	0.0320 ± 0.05%	1.1831 ± 56.88%	0.8706 ± 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] Gabriela Csurka, Riccardo Volpi und Boris Chidlovskii. *Semantic Image Segmentation: Two Decades of Research*. 2023. arXiv: 2302.06378 [cs.CV].
- [6] 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.
- [7] Slimane Djema u. a. *Dense Visual Odometry Using Genetic Algorithm*. 2023. arXiv: 2311.06149 [cs.R0].
- [8] Wei Dong u. a. *ASH: A Modern Framework for Parallel Spatial Hashing in 3D Perception*. 2023. arXiv: 2110.00511 [cs.CV].
- [9] Gongfan Fang. *DeepLabv3Plus-Pytorch*. <https://github.com/VainF/DeepLabV3Plus-Pytorch>. Accessed: 2025-06-02. 2022.
- [10] Christian Forster u. a. „On-Manifold Preintegration for Real-Time Visual–Inertial Odometry“. In: *IEEE Transactions on Robotics* 33.1 (Feb. 2017), S. 1–21. doi: 10.1109/tro.2016.2597321.
- [11] 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.
- [12] Tsung-Yi Lin u. a. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].
- [13] Ilya Loshchilov und Frank Hutter. *Decoupled Weight Decay Regularization*. 2019. arXiv: 1711.05101 [cs.LG].
- [14] 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.

- 
- [15] 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>.
  - [16] Shervin Minaee u. a. *Image Segmentation Using Deep Learning: A Survey*. 2020. arXiv: 2001.05566 [cs.CV].
  - [17] 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.
  - [18] 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.
  - [19] OpenCV Team. *Feature Matching*. Accessed: 2025-07-08. OpenCV. 2024.
  - [20] 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.
  - [21] PyTorch Developers. *torch.nn.Conv2d - Pytorch Documentation*. Techn. Ber. Zugriff: 1-05-2025. Pytorch, 2025.
  - [22] Nikhila Ravi u. a. *SAM 2: Segment Anything in Images and Videos*. 2024. arXiv: 2408.00714 [cs.CV].
  - [23] 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.
  - [24] 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.
  - [25] 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.
  - [26] 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.
  - [27] Shuhei Watanabe. *Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance*. 2023. arXiv: 2304.11127 [cs.LG].
  - [28] Yuxin Wu u. a. *Detectron2*. <https://github.com/facebookresearch/detectron2>. 2019.
  - [29] Qian-Yi Zhou, Jaesik Park und Vladlen Koltun. „Open3D: A Modern Library for 3D Data Processing“. In: *arXiv:1801.09847* (2018).

## Literatur

---

- [30] 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

