HAW HAMBURG

MASTER THESIS
Robin Grotkasten

Untersuchung der Anwendbarkeit von Machine Learning zur Selbstlokalisation auf einer Miniaturdrohne

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Robin Grotkasten

Untersuchung der Anwendbarkeit von Machine Learning zur Selbstlokalisation auf einer Miniaturdrohne

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang *Master of Science Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Tim Tiedemann
Zweitgutachter: Prof. Dr. Thomas Lehmann

Eingereicht am: 09. Februar 2024

Robin Grotkasten

Thema der Arbeit

Untersuchung der Anwendbarkeit von Machine Learning zur Selbstlokalisation auf einer Miniaturdrohne

Stichworte

Drohne, UAV, MAV, maschinelles Lernen, Sensorfusion, inertiale Navigation, RNN, GRU

Kurzzusammenfassung

Drohnen wie Quadrocopter benötigen zur Stabilisierung und Navigation eine präzise Schätzung der aktuellen Position und Orientierung im Raum. Die vorhandenen Sensordaten unterschiedlicher Sensoren müssen hierbei bestmöglich fusioniert werden. Konventionelle Verfahren zur Sensorfusion, wie das Extended Kalman Filter (EKF), müssen hierfür aufwendig für spezielle Anwendungsfälle parametriert werden. Bei einer rein inertialen Navigation (Koppelnavigation) führt die inhärente Sensordrift zu schnell ansteigenden Positionsfehlern. In dieser Arbeit wird untersucht, inwieweit Machine Learning zur Selbstlokalisation auf einer Miniaturdrohne genutzt werden kann. Herausforderung ist hierbei die geringe Rechenleistung, die auf einer leichten Drohne zur Verfügung steht. Es werden künstliche neuronale Netze mit unterschiedlicher Architektur trainiert und deren Performance mit konventionellen Filterverfahren verglichen. In diesem Rahmen wird ein rein inertialer Ansatz und ein Ansatz mit Unterstützung durch einen Optical-Flow- (OF) und Time-of-Flight-Sensor (ToF) untersucht.

Robin Grotkasten

Title of Thesis

Investigation of the applicability of machine learning for self-localization on a micro air vehicle (MAV)

Keywords

drone, UAV, MAV, machine learning, sensorfusion, inertial navigation, RNN, GRU

Abstract

Drones such as quadrocopters require a precise estimation of the current position and orientation in space for stabilization and navigation. The available sensor data from different sensors must be fused in the best possible way. Conventional methods for sensor fusion, such as the Extended Kalman Filter (EKF), require complex parameterization for special applications. In the case of purely inertial navigation (dead reckoning), the inherent sensor drift leads to rapidly increasing position errors. This thesis investigates the extent to which machine learning can be used for self-localization on a miniature drone. The challenge here is the low available computing power on a lightweight drone. Artificial neural networks with different architectures are trained and their performance is compared with conventional filtering methods. In this context, a purely inertial approach and an approach supported by an optical flow and time-of-flight sensor are investigated.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

ARKit Augmented Reality Kit.

BPTT Backpropagation Through Time.

BROAD Berlin Robust Orientation Estimation Assessment Dataset.

CNN Convolutional Neural Network.

DCM Direction Cosine Matrix.

DL Deep Learning.

DO IONet Direct-Orientation Inertial Odometry Network.

DOF Degrees of Freedom.

DPMAE Delta Position Mean Absolute Error.

EKF Extended Kalman Filter.

EuRoC MAV European Robotics Challenge Micro Aerial Vehicle.

GNSS Global Navigation Satellite System.

GPS Global Positioning System.

GRU Gated Recurrent Unit.

IFOG Interferometer Fiber-Optic Gyroscope.

IMU Inertial Measurement Unit.

INS Inertial Navigation System.

IO Inertial Odometry.

IONet Inertial Odometry Network.

KF Kalman Filter.

KNN Künstliches Neuronales Netz.

KNNs Künstliche Neuronale Netze.

LiDAR Light Detection and Ranging.

LIO Laser Inertial Odometry.

LSTM Long Short-Term Memory.

MAE Mean Absolute Error.

MARG Magnetic, Angular Rate, and Gravity.

MAV Micro Air Vehicle.

MEMS Micro-Electro-Mechanical Systems.

ML Machine Learning.

MSDF Multi-Data Sensor Fusion.

MSE Mean Square Error.

NLP Natural Language Processing.

OF Optical Flow.

OxIOD Oxford Inertial Odometry Dataset.

PDR Pedestrian Dead Reckoning.

QE Quaternion Error.

RIANN Robust IMU-based Attitude Neural Network.

RIO Radar Inertial Odometry.

RLG Ring Laser Gyroscope.

RMSE Root Mean Square Error.

RNN Recurrent Neural Network.

SINS Strapdown Inertial Navigation System.

SLAM Simultaneous Localization and Mapping.

SLERP Spherical Linear Interpolation.

TA Time-Aware.

TBPTT Truncated Backpropagation Through Time.

TCN Temporal Convolutional Network.

ToF Time-of-Flight.

TUM VI Technische Universität München Visual Intertial.

UAV Unmanned Aerial Vehicles.

UKF Unscented Kalman Filter.

UWB Ultra-Wideband.

VIO Visual Inertial Odometry.

WiFi Wireless Fidelity.

1 Einleitung

1.1 Problemstellung und Motivation

Unmanned Aerial Vehicles (UAV), auch als Drohnen[1] bekannt, sind unbemannte Luftfahrzeuge, die unter anderem zur Luftfotografie, zur Vermessung und Kartierung, zur Überwachung von Feldern in der Landwirtschaft oder zur schnellen Lieferung von Medikamenten eingesetzt werden [28][36].

Um all diese Aufgaben autonom bzw. teilweise autonom zu erledigen, benötigen Drohnen präzise Informationen über die eigene Position und Orientierung im Raum [3][37]. Für die Positionsbestimmung kann heutzutage in vielen Umgebungen Global Navigation Satellite System (GNSS) eingesetzt werden. Für abgeschirmte Bereiche, in denen GNSS nur bedingt oder gar nicht zur Verfügung steht, ist jedoch die Möglichkeit einer Selbstlokalisation mithilfe von *Onboard*-Sensoren notwendig. Beispiele hierfür sind der Einsatz von Drohnen in Gebäuden, in Tunneln, unter Brücken oder in dichtbebauten städtischen Gebieten [1].
Die wichtigsten Sensoren auf einer Drohne zur Selbstlokalisation sind hierbei Drehraten-, Kompass- und Beschleunigungssensoren, die in einer Inertial Measurement Unit (IMU) vereint werden. Gerade für die Bestimmung der Orientierung, d.h. Neigung (*Attitude*) und Ausrichtung (*Heading*) sind Inertialsensoren unabkömmlich.
Eine Drohne ist wie ein umgekehrtes Pendel ein instabiles System und benötigt für eine stabile Lage in der Luft eine genaue Kenntnis der aktuellen Orientierung im Raum, um diese ständig zu korrigieren. Hierfür werden mithilfe der IMU Lageänderungen erfasst und mehrere hundert Mal in der Sekunde Orientierungsänderungen ermittelt. Mithilfe von Reglern wird dann die gewünschte Fluglage durch individuelle Anpassungen der Drehzahl der Rotoren eingeregelt. Durch die Entwicklung von modernen Smartphones

[1] In dieser Arbeit werden mit dem Begriff „Drohnen“ Multikopter bezeichnet. Ein Multikopter ist charakterisiert durch mehrere in der Ebene angeordnete Rotoren. Je nach Anzahl der Rotoren wird hierbei beispielsweise in Quadrocopter (4 Rotoren), Hexacopter (6 Rotoren) oder Octocopter (8 Rotoren) unterschieden.

sind IMUs in kleiner Bauweise (Micro-Electro-Mechanical Systems (MEMS)) mittlerweile sehr günstig geworden. Jedoch sind sie deutlich ungenauer als die komplexeren Ringlaserkreisel, die unter anderem in der kommerziellen Luftfahrt eingesetzt werden. Für die Bestimmung der Position(-sänderung) können sogenannte Inertial Navigation System (INS)-Systeme genutzt werden. Diese nutzen den Beschleunigungssensor einer IMU, um Bewegungsänderungen im Raum zu berechnen. Durch die doppelte Integration der Beschleunigung in Strapdown Inertial Navigation System (SINS)-Systemen führen kleine Ungenauigkeiten jedoch schnell zu drastischen Positionsfehlern [37].
Für die Schätzung der Position bzw. Positionsänderung werden deshalb fast immer ergänzend weitere Sensoren wie Kamera, Optical Flow (OF), Laserscanner und GNSS verwendet, um eine Langzeitstabilität der Positionsschätzung zu ermöglichen. Solche Systeme, die unterschiedliche Sensoren zur Schätzung eines Gesamtzustands nutzen, werden auch als Multi-Data Sensor Fusion (MSDF) bezeichnet [3].

Es gibt einige etablierte Filter, die eingesetzt werden, um die Daten der Sensoren zu fusionieren. Diese Filter nutzen mathematische Modelle, um Rauschen, Bias und Kovarianz der Systemgrößen zu schätzen. Für die Positionsschätzung alleine können lineare Filter eingesetzt werden, wie das Kalman Filter (KF) oder gh-Filter. Für die Orientierungsschätzung sind nicht-lineare Filter notwendig, hierzu zählen z.B. dass Extended Kalman Filter (EKF), Unscented Kalman Filter (UKF), Mahony oder Madgwick Filter. In der Arbeit [6] konnte gezeigt werden, dass Filter mit vielen einstellbaren Parametern potentiell kleinere Fehler in der Schätzung der Positions(-änderung) und Orientierung(-sänderung) machen als Filter mit wenigen Parametern, jedoch müssen diese für jeden Anwendungsfall angepasst werden. Für schnelle und plötzliche Bewegungsänderungen dürfen die Daten des Beschleunigungssensors z.B. nur mit geringer Gewichtung für die Berechnung der Orientierung (genau: *Attitude*) genutzt werden.

Viele datenintensive Aufgaben, die ein hohes Abstraktionslevel benötigen, können heutzutage von Machine Learning (ML) Ansätzen übernommen werden. ML-Verfahren werden z.B. eingesetzt, wenn eine analytische Bestimmung eines Zusammenhangs zu komplex oder aufwendig ist. ML-Verfahren können durch statistische Verfahren mithilfe von Beispieldaten Zusammenhänge abstrahieren. Beispiele für solche Aufgabenbereiche sind z.B. Objekterkennung, Klassifizierung, Bildgenerierung oder Übersetzungstools. Eine besondere Art ist das *Sequential Modeling*. Hierbei werden Zeitreihen durch frühere Beobachtungen prognostiziert. Spracherkennung, Natural Language Processing (NLP) und Musikgenerierung fallen auch in diesen besonderen Anwendungsfall von neuronalen Netzen [3]. In einigen Papern werden künstliche neuronale Netze bereits verwendet, um

die Orientierung(-sänderung) und Position(-sänderung) aus IMU-Daten zu bestimmen. Hierbei übertreffen die Ergebnisse der ML-Verfahren bereits teilweise die konventionellen Filterverfahren in Langzeitstabilität und Toleranz gegenüber Sensorrauschen [3][54].

In dieser Arbeit soll untersucht werden, ob ML-Verfahren auch zur Sensorfusion auf einer Miniaturdrohne verwendet werden können, um damit eine Selbstlokalisation im Raum durchzuführen. Die beschränkte Hardwareleistung, die auf einer leichten Drohne zur Verfügung steht und die hohe Anforderung an Echtzeitfähigkeit sind hierbei Herausforderung und Motivation der Arbeit. In diesem Rahmen sollen ein rein inertialer Ansatz und ein Ansatz, der zusätzlich ergänzende Sensoren zur Langzeitstabilität nutzt, untersucht werden.

Es sollen folgende Leitfragen für diese Arbeit gelten:

- Reicht die Leistung einer Miniaturdrohne für den Einsatz von künstlichen neuronalen Netzen zur *Pose*-Schätzung?
- Können die erzielten Ergebnisse aus der Referenzliteratur reproduziert werden?
- Kann ein Künstliches Neuronales Netz (KNN) gegenüber konventionellen Verfahren zur Bestimmung von Position(-sänderung) und Orientierung(-sänderung) besser performen?
- Wie genau ist die geschätzte Pose gegenüber *Ground Truth*?
- Wie lange kann eine Schätzung der Position und Orientierung mit dem entwickelten Verfahren sinnvoll genutzt werden?
- Können die Ergebnisse durch die Nutzung eines OF- und Time-of-Flight (ToF)-Sensors verbessert werden?

1.2 Zielsetzung

In dieser Arbeit sollen Künstliche Neuronale Netze (KNNs) mit unterschiedlichen Netzarchitekturen zur Bestimmung von Position(-sänderung) und Orientierung(-sänderung) trainiert werden. Es sollen Variationen der Hyperparameter, Ausgangsgrößen und Netzarchitektur zur Verbesserung der Ergebnisse erprobt werden, um so systematisch das Netz mit den besten Parametern zu ermitteln. Die trainierten Netze sollen auf einer Miniaturdrohne in Echtzeit ausgeführt werden können. Ziel ist es weiterhin, ein universelles

Netz zu trainieren, welches im Gegensatz zu konventionellen Verfahren nicht aufwendig parametriert werden muss und besser performt als die konventionellen Filterverfahren. Für die Verifizierung und Bewertung der Performance sollen ungesehene (engl. *unseen*) Testdatensätze genutzt werden und die Ergebnisse der *Pose*-Schätzungen der konventionellen Verfahren und des eigenen Ansatzes gegenübergestellt werden.

1.3 Abgrenzung

In den meisten aktuellen Papern spielt die Ausführungszeit des trainierten KNN auf der verwendeten Hardware keine primäre Rolle. Die Netze werden auf einem Desktop-Rechner mit starker Grafikkarte trainiert und evaluiert. In dieser Arbeit soll das trainierte KNN für eine Echtzeitanwendung auf einer Miniaturdrohne genutzt werden. Hierbei steht ein Mikrocontroller (STM32H743) auf der Miniaturdrohne zur Verfügung. Das entwickelte Netz muss daher effizient in der Ausführung sein.
In den meisten Papern werden lediglich IMU-Daten zur inertialen Navigation verwendet. In dieser Arbeit soll ergänzend ein weiterer Ansatz untersucht werden, bei dem OF- und ToF-Sensordaten als Stützinformationen genutzt werden.
Es sollen keine Pedestrian Dead Reckoning (PDR)-Systeme in dieser Arbeit untersucht werden, welche bestimmte Muster im menschlichen Gang nutzen, um die Schätzungen zu verbessern.

1.4 Aufbau der Arbeit

In Kapitel 2 werden die wichtigsten theoretischen Grundlagen zum Verständnis der Arbeit vermittelt. Kapitel 3 gibt einen Überblick über den aktuellen Stand der Technik und aktuelle Veröffentlichungen zu dem Themengebiet. In Kapitel 4 wird anhand der Ergebnisse der Literaturrecherche ein eigenes Systemdesign entwickelt. In Kapitel 5 werden sowohl die Plattform zur Software-Entwicklung als auch die Hardware-Plattform (die Miniaturdrohne) vorgestellt. In Kapitel 6 werden die genutzten quelloffenen Trainingsdaten und das Vorgehen zur Erstellung von eigenen Trainingsdaten vorgestellt. In Kapitel 7 werden Versuche zur Optimierung der Performance der Netzmodelle durchgeführt. In Kapitel 8 werden die Verfahren untereinander und mit den konventionellen Verfahren verglichen. In Kapitel 9 wird eine Zusammenfassung der Arbeit und ein Ausblick gegeben.

2 Theoretische Grundlagen

In diesem Abschnitt sollen die wichtigsten theoretischen Grundlagen vorgestellt werden, die für das Verständnis der weiteren Kapitel notwendig sind. Für Details wird auf die einschlägige Literatur hingewiesen. Wie in Kapitel 1 dargestellt, soll in dieser Arbeit untersucht werden, ob ML auf einer Drohne zur Selbstlokalisation genutzt werden kann. In Abschnitt 2.1 soll daher ein Grundverständnis für die Beschreibung der Position und Orientierung eines Körper im Raum und die Referenzkoordinatensysteme gegeben werden. Es sollen dann die auf der Miniaturdrohne verwendeten Sensoren in Abschnitt 2.2 und deren Fehlercharakteristik in Abschnitt 2.3 vorgestellt werden. In Abschnitt 2.4 wird die inertiale Navigation mithilfe der Daten einer IMU in mehreren Schritten gezeigt. Der Abschnitt 2.5 soll einen Einblick in das maschinelle Lernen (engl. *machine learning*) geben. In Abschnitt 2.6 werden die konventionellen Filter zur Schätzung von Position und Orientierung vorgestellt.

2.1 Position und Orientierung im Raum

Die Position und Orientierung eines starren Körpers wird zusammengefasst unter dem Begriff *Pose*. Es gibt unterschiedliche Ansätze, um eine Pose eines starren Körpers zu beschreiben. Es sind minimal sechs Koordinaten nötig, um einen starren Körper im euklidischen Raum zu lokalisieren [48]. Position und Orientierung können generell unabhängig voneinander beschrieben werden.

2.1.1 Koordinatensysteme

Für die Beschreibung der Position im Raum ist zunächst einmal die Wahl des Referenz-Koordinatensystems elementar. Die Position wird dann relativ zum Ursprung des Koordinatensystems beschrieben. Es wird in unterschiedliche Koordinatensysteme unterschieden, die im Folgenden kurz vorgestellt werden [55].

- **Körperfestes Koordinatensystem (b-Frame)**: Das körperfeste Koordinatensystem ist fest mit dem Körper verbunden. Die Achsen zeigen in die Fahrzeuglängsrichtung (x), nach rechts (y) und nach unten (z). Der Ursprung befindet sich im Fahrzeug. In dieser Arbeit soll davon ausgegangen werden, dass die IMU exakt orthogonal zu diesem Koordinatensystem ausgerichtet ist. Das Sensorkoordinatensystem fällt daher mit dem körperfesten Koordinatensystem zusammen.
- **Navigationskoordinatensystem (n-Frame)**: Das Navigationskoordinatensystem hat den gleichen Ursprung wie das körperfeste Koordinatensystem. Die z-Achse ist parallel zum Vektor der Erdbeschleunigung. Die x- und y-Achse weisen in Richtung Nord bzw. Ost.
- **Inertialkoordinatensystem (i-Frame)**: Die Erde wird durch ein Rotationsellipsoid modelliert. Der Ursprung des Koordinatensystems befindet sich in diesem Ellipsoid. Die z-Achse fällt mit der Rotationsachse zusammen. Die x- und y-Achse liegen in der Äquatorebene. Eine IMU misst Beschleunigungen und Drehraten des körperfesten Koordinatensystems bezüglich des Inertialkoordinatensystems.
- **Erdfestes Koordinatensystem (e-Frame)**: Das erdfeste Koordinatensystem ähnelt dem Inertialkoordinatensystem, allerdings rotiert es mit der Winkelgeschwindigkeit Ω, sodass die x-Achse in der Schnittgeraden von Äquatorebene und Nullmeridian liegt.

2.1.2 Position im Raum

Die Position im Raum kann über einen dreidimensionalen Vektor ausgedrückt werden:

$$\vec{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{2.1}$$

Wenn von einem konstantem Geschwindigkeitsvektor $\vec{v}_t$ über ein Zeitintervall Δt ausgegangen wird, kann die neue Position $\vec{p}_{t+1}$ folgendermaßen berechnet werden:

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}_t \Delta t \tag{2.2}$$

2.1.3 Orientierung im Raum

Für die Beschreibung der Orientierung und Rotationen im Raum existieren mehrere Möglichkeiten. Diese sollen in diesem Abschnitt vorgestellt werden.

Euler-Winkel/Tait-Bryan-Winkel

Eine einfache Art, die Orientierung eines Koordinatensystems gegenüber einem anderen auszudrücken, ist die Beschreibung durch Euler-Winkel. Hierbei wird das körpereigene Koordinatensystem nacheinander um drei Koordinatenachsen gedreht. Die Drehung wird über einen Vektor aus drei Winkeln (ψ, θ, ϕ) beschrieben.

Bei den eigentlichen Euler-Winkeln wird die erste und dritte Drehung um die gleiche Koordinatenachse durchgeführt, z.B. $z - x' - z''$. Weiter verbreitet ist aber die Nutzung der sogenannten Tait-Bryan-Winkel (Kardan-Winkel). Hierbei werden alle Drehungen um verschiedene Koordinatenachsen durchgeführt, z.B. $z - y' - x''$. Die Reihenfolge der Rotationen ist elementar, weil Rotationen im 3D-Raum nicht kommutativ sind [55].

Bei der Rotation muss in intrinsische und extrinsische Drehungen unterschieden werden:

- **Intrinsische Drehung**: es wird im körpereigenen Koordinatensystem, d.h. um die körpereigenen Achsen gedreht
- **Extrinsische Drehung**: es wird im ursprünglichen Koordinatensystem gedreht (z.B. im Navigationskoordinatensystem)

Generell können intrinsische und extrinsische Drehungen durch Umkehrung der Drehreihenfolge ineinander umgewandelt werden.

In der Industrie (Luftfahrt: DIN 9300) wird oft nach einer genormten Drehfolge, der sogenannten Gier-Nick-Roll-Drehung (engl. yaw-pitch-roll), gedreht.

- In der intrinsischen Drehung entspricht dies der Reihenfolge (Gier-Nick-Roll) $z - y' - x''$.
- In der extrinsischen Drehung entspricht dies der Reihenfolge (Roll-Nick-Gier) $x - y - z$.

Die intrinsische Drehung nach DIN 9300 wird folgendermaßen durchgeführt:

- **Gierwinkel ψ (yaw:)**: Es erfolgt die erste Drehung um die z-Achse des körpereigenen Koordinatensystems.
- **Nickwinkel θ (pitch)**: Es wird um die neue y-Achse (y') des gedrehten Koordinatensystems gedreht.
- **Rollwinkel ϕ (roll)**: Es wird um die neue x-Achse (x'') des gedrehten Koordinatensystems gedreht.

Die Nutzung von Euler-Winkeln hat mehrere Nachteile. Zum einen kann die Berechnung nur aufwendig über Rotationsmatrizen durchgeführt werden (siehe Gleichung 2.3). Des Weiteren können Singularitäten entstehen, wenn mit einem Nick-Winkel von $\pm 90°$ gedreht wird. In diesem Fall fällt einer der drei Freiheitsgrade weg und Drehungen um die Roll- und Gierachse haben den gleichen Effekt auf den Körper. Dieses Phänomen wird auch als *Gimbal-Lock* bezeichnet [55].

Richtungskosinusmatrix

Eine Richtungskosinusmatrix (engl. Direction Cosine Matrix (DCM)) ist eine Transformationsmatrix, die die Referenzachsen des Ursprungskoordinatensystems A in die rotierten Achsen des Koordinatensystems B projiziert. Es wird als DCM bezeichnet, da jedes Element den Cosinus des vorzeichenlosen Winkels zwischen den Raumachsen beschreibt (siehe Gleichung 2.3).

$$
{}^{A}_{B}R = \begin{bmatrix} \cos\left(\theta_{x^A,x^B}\right) & \cos\left(\theta_{y^A,x^B}\right) & \cos\left(\theta_{z^A,x^B}\right) \\ \cos\left(\theta_{x^A,y^B}\right) & \cos\left(\theta_{y^A,y^B}\right) & \cos\left(\theta_{z^A,y^B}\right) \\ \cos\left(\theta_{x^A,z^B}\right) & \cos\left(\theta_{y^A,z^B}\right) & \cos\left(\theta_{z^A,z^B}\right) \end{bmatrix} \tag{2.3}
$$

Die Rotation um einzelne Koordinatenachsen wird durch folgende Rotationsmatrizen beschrieben:

$$
R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) \\ 0 & -\sin(\theta) & \cos(\theta) \end{bmatrix} \tag{2.4}
$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.5}$$

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

Eine DCM ist stabil, aber langsam in der Berechnung, da insgesamt neun Werte für die Beschreibung einer DCM notwendig sind [32].

Die Umwandlung der Euler-Winkel in eine DCM wird durch eine Verkettung von Rotationen um die einzelnen Koordinatenachsen beschrieben:

$$R_{xyz} = R_x(\phi)R_y(\theta)R_z(\psi) \tag{2.7}$$

Orientierungsvektor

Eine Rotation kann auch durch einen Orientierungsvektor ausgedrückt werden. Hierbei beschreibt der Vektor $\vec{r}$ eine Achse im Raum, um die eine Rotation stattfindet. Die Länge des Vektors $\|\vec{r}\|$ beschreibt hierbei den Winkelbetrag, um den rotiert wird. Mit dem Orientierungsvektor wird ein Koordinatensystem durch eine einzige Drehung in ein anderes Koordinatensystem überführt.

$$\vec{r} = \begin{bmatrix} r_x \\ r_y \\ rz \end{bmatrix} \tag{2.8}$$

Quaternion

Verwandt zum Orientierungsvektor ist auch die Darstellung über Quaternionen. Eine Quaternion erweitert den reellen Zahlenbereich, ähnlich den komplexen Zahlen. Quaternionen ermöglichen eine Beschreibung des dreidimensionalen euklidischen Raums durch vier Werte. Eine Quaternion kann folgendermaßen dargestellt werden:

$$q = q_0 + q_1 i + q_2 j + q_3 k \tag{2.9}$$

Eine Quaternion besteht aus einem skalaren Part q_0 und einem Vektorpart $q_1 i + q_2 j + q_3 k$.

Bei der Nutzung von Quaternionen für die Beschreibung von Rotationen im euklidischen Raum wird nur ein eingeschränkter Bereich des Zahlenraums von Quaternionen genutzt. Es werden lediglich Einheitsquaternionen genutzt, die immer eine Länge von eins haben. Es gilt demnach:

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1 \tag{2.10}$$

Eine Quaternion ist ähnlich dem Orientierungsvektor aufgebaut. Wenn ein Koordinatensystem A in ein anderes Koordinatensystem B überführt werden soll, kann das Koordinatensystem A durch eine Rotation mit dem Winkel θ um den Vektor ${}^A\vec{r} = [r_x, r_y, r_z]$ gedreht werden (siehe Abbildung 2.1). Die Quaternion, welche diese Rotation ausdrückt, ist folgendermaßen definiert [33]:

$${}^A_B q = \begin{bmatrix} cos(\frac{\theta}{2}) \\ -r_x \cdot sin(\frac{\theta}{2}) \\ -r_y \cdot sin(\frac{\theta}{2}) \\ -r_z \cdot sin(\frac{\theta}{2}) \end{bmatrix} \tag{2.11}$$

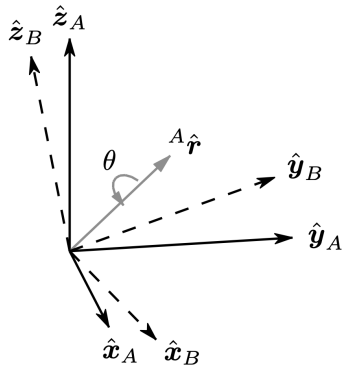


Abbildung 2.1: Rotation eines Koordinatensystems um einen Vektor [33]

Berechnungen mit Quaternionen

Die Berechnungen mit Quaternionen haben einige Eigenheiten, die hier kurz erläutert werden sollen. Bei einer Multiplikation von Quaternionen muss das Hamilton Produkt genutzt werden. Das Hamilton Produkt wird hier durch das Symbol $\otimes$ gekennzeichnet. In diesem Beispiel findet eine Verkettung von zwei Quaternionen statt: ${}^{B}_{C}q$ und ${}^{A}_{B}q$ werden zur Gesamtrotation ${}^{A}_{C}q$ verkettet.

$$ {}^{A}_{C}q = {}^{B}_{C}q \otimes {}^{A}_{B}q \tag{2.12} $$

Das Hamilton Produkt ist folgendermaßen definiert:

$$ a \otimes b = \begin{bmatrix} a_0b_0 - a_1b_1 - a_2b_2 - a_3b_3 \\ a_0b_1 + a_1b_0 + a_2b_3 - a_3b_2 \\ a_1b_2 - a_1b_3 + a_2b_0 + a_3b_1 \\ a_2b_3 + a_1b_2 - a_2b_1 + a_3b_0 \end{bmatrix} \tag{2.13} $$

Es ist nicht kommutativ, d.h. es gilt $a \otimes b \neq b \otimes a$.

Eine konjugierte bzw. inverse Quaternion meint eine Rotation des gleichen Winkels in entgegengesetzte Richtung. Bei der inversen Quaternion wird der Imaginärteil der Quaternion negiert.

$$ {}^{B}_{A}q* = {}^{A}_{B}q = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^T \tag{2.14} $$

Eine Rotation eines Vektors ${}^{A}\vec{p}$ mithilfe einer Quaternion ${}^{A}_{B}q$ wird folgendermaßen durchgeführt:

$$ \begin{bmatrix} 0 \\ {}^{B}\vec{p} \end{bmatrix} = {}^{A}_{B}q \otimes \begin{bmatrix} 0 \\ {}^{A}\vec{p} \end{bmatrix} \otimes {}^{A}_{B}q* \tag{2.15} $$

Alternativ kann die Quaternion ${}^{A}_{B}q$ in eine Rotationsmatrix $R({}^{A}_{B}q)$ überführt werden und eine Rotation eines Vektors ${}^{A}\vec{p}$ durchgeführt werden:

$$ {}^{B}\vec{p} = R({}^{A}_{B}q) \cdot {}^{A}\vec{p} \tag{2.16} $$

Hierbei kann eine Quaternion q allgemein in eine Rotationsmatrix $R(q)$ überführt werden:

$$R(q) = \begin{bmatrix} 2q_0^2 - 1 + 2q_1^2 & 2(q_1q_2 + q_0q_3) & 2(q_1q_3 - q_0q_2) \\ 2(q_1q_2 - q_0q_3) & 2q_0^2 - 1 + 2q_2^2 & 2(q_2q_3 + q_0q_1) \\ 2(q_1q_3 + q_0q_2) & 2(q_2q_3 - q_0q_1) & 2q_0^2 - 1 + 2q_3^2 \end{bmatrix} \tag{2.17}$$

Attitude und Heading

Die Orientierung kann anders als die Euler-Winkel auch in Neigung (*attitude* bzw. *inclination*) und Ausrichtung (*heading*) zerlegt werde. Nach [27] können diese unabhängig voneinander betrachtet werden (siehe Abbildung 2.2). Hierbei meint *Heading* eine Rotation um die vertikale Achse des Navigationskoordinatensystems und *Inclination* eine Rotation um eine horizontale Achse. Im Gegensatz zu anderen Verfahren, sind die beiden Winkelangaben kommutativ (nicht jedoch die Rotationsachse für die *Inclination*) [27]. In dieser Arbeit soll der Begriff *Attitude* synonym für *Inclination* genutzt werden, hierbei ist immer die Neigung ohne Berücksichtigung der *Heading* gemeint.

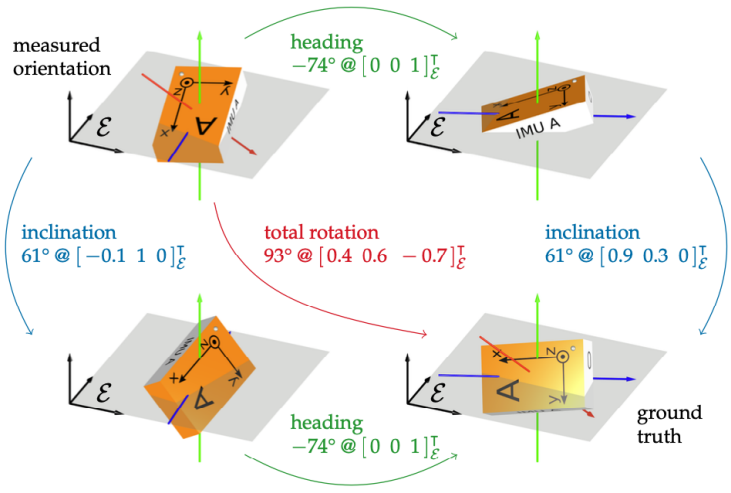


Abbildung 2.2: Zerlegung in *Heading* und *Inclination* [27]

2.2 Sensoren

Micro-Electro-Mechanical Systems (MEMS) sind miniaturisierte Systeme, deren Komponenten im Bereich von 1 und 100 μm liegen. Diese MEMS-Systeme bestehen aus Sensoren und Aktoren, die auf einem Substrat in einem Chip realisiert werden. Sie basieren damit auf der Halbleitertechnik. Es können hierbei aber auch sämtliche andere Werkstoffe wie Metalle, Keramik, Kunststoffe etc. bei der Herstellung verwendet werden.

Vorteile von MEMS gegenüber konventionellen Makrosystemen sind die kleine Bauweise und Kostenersparnis bei der Herstellung sowie ein geringer Energie- und Leistungsbedarf [32].

Einen großen Nutzen haben MEMS-Systeme bei der Herstellung von Inertialsensoren, die sich heutzutage in jedem Smartphone und Drohnen wiederfinden.

2.2.1 Drehratensensor

Drehratensensoren (auch als Kreisel oder Gyroskop bezeichnet) werden zur Messung von Drehraten genutzt. In der kommerziellen Luftfahrt werden unter anderem Faserkreisel (engl. Interferometer Fiber-Optic Gyroscope (IFOG)) oder Ringlaserkreisel (engl. Ring Laser Gyroscope (RLG)) verwendet. In Drohnen und Smartphones werden dagegen die kostengünstigeren MEMS-Kreisel verwendet.

MEMS-Kreisel, die die Coriolis-Kraft ausnutzen, arbeiten folgendermaßen: Es werden zwei Probemassen durch elektrostatische Anregung in gegenphasige Schwingung in x-Richtung versetzt. Liegt eine Drehrate $\omega\vec{e}_y$ vor, wird aufgrund der Coriolis-Kraft eine orthogonale Beschleunigung $\vec{a}_c$ auf die Probemasse in z-Richtung verursacht, die ebenfalls ein Schwingen der Masse zur Folge hat. Über eine kapazitive Messung kann die Auslenkung der Probemasse detektiert werden und ein Rückschluss auf die Drehgeschwindigkeit erfolgen. Ein Prinzipbild ist in Abbildung 2.3 zu sehen. [55]

$$\vec{a}_c = 2\omega\vec{e}_y \times v_a(t)\vec{e}_x = -2v_a(t)\omega\vec{e}_z \tag{2.18}$$

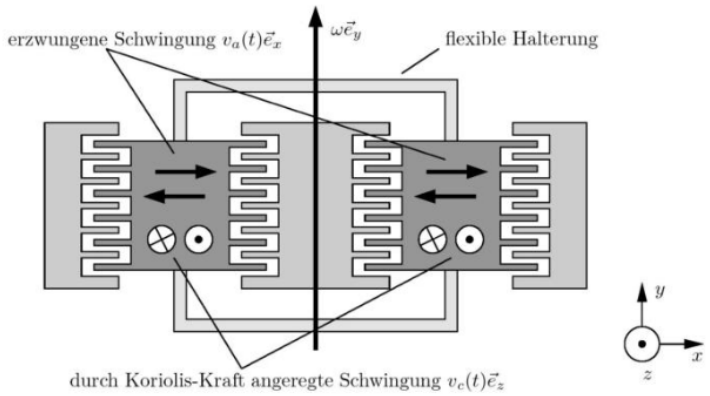


Abbildung 2.3: Prinzipaufbau eines MEMS-Drehratensensors (Quelle: [55])

2.2.2 Beschleunigungssensor

Ein Beschleunigungssensor misst die auf ihn wirkende Beschleunigung. Hierfür wird in einem MEMS-Sensor eine Probemasse in einer Federaufhängung realisiert. Bei einer Beschleunigung wirkt eine Trägheitskraft auf den Körper, die zu einer Auslenkung führt. Diese kann über unterschiedliche Methoden (piezoelektrisch, kapazitiv, piezoresistiv etc.) gemessen werden und in eine Beschleunigung umgerechnet werden.

Für die Trägheitskraft gilt der Zusammenhang:

$$F = m \cdot a \tag{2.19}$$

Für die Federkraft gilt:

$$F = k \cdot \Delta l \tag{2.20}$$

Ein Gleichsetzen der Kräfte ergibt die gemessene Beschleunigung:

$$a = \frac{k}{m} \cdot l \tag{2.21}$$

2.2.3 Magnetometer

Ein Magnetometer kann die magnetische Feldstärke in alle drei Raumrichtungen bestimmen. Ein Magnetometer nutzt oftmals den Hall-Effekt. Hierbei wird ein stromdurchflossener Leiter einem senkrecht dazu verlaufenden Magnetfeld ausgesetzt. Die induzierte Spannung (Hall-Spannung) gibt Rückschluss auf die Stärke des Magnetfelds. Bei einem konstanten Strom ist die Hall-Spannung direkt proportional zum Magnetfeld.
Mithilfe eines Magnetometers kann durch Messung des Erdmagnetfelds die Nordrichtung bestimmt werden und so eine absolute Bestimmung der *Heading* ermöglicht werden. Die Nutzung eines Magnetometers soll in dieser Arbeit nicht erfolgen, da das Magnetfeld in Innenräumen durch elektrische Geräte stark gestört werden kann (siehe Unterabschnitt 2.3.3).

2.2.4 Optischer Flusssensor

Der optische Fluss (engl. Optical Flow (OF)) beschreibt die Bewegung von Pixeln und Objekten in einer Bildsequenz. Er kann durch ein Vektorfeld beschrieben werden. Für die Bestimmung des optischen Flusses existieren einige Verfahren wie das Lucas-Kanade- [31] oder das Horn-Schunck-Verfahren [20]. Fertige optische Flusssensoren wie der PMW3901 [51] liefern bereits einen gemittelten Bildfluss (δ_x, δ_y). Das auf dem PMW3901 verwendete Verfahren ist nicht veröffentlicht.

Auf einer Drohne kann ein OF-Sensor zur Schätzung der aktuellen Geschwindigkeit über Grund herangezogen werden. Hierbei ist eine ausgeleuchtete Umgebung mit möglichst statischem Grund eine Voraussetzung für gute Ergebnisse.

Die Bewegungsgeschwindigkeit (v_x, v_y) der Drohne ist nach [56] direkt abhängig von den beiden gemittelten Bildflüsse (δ_x, δ_y) des optischen Flusssensors, der orthogonalen Distanz d und den aktuell vorliegenden Neigungsraten (ω_y, ω_x) der Drohne (siehe Gleichung 2.22 und Gleichung 2.23). Die Geschwindigkeiten (v_x, v_y) beziehen sich hierbei immer auf das körpereigene Koordinatensystem. f ist hierbei ein sensorspezifischer Faktor.

$$v_x = -d(\frac{\delta_x}{f} + \omega_x) \tag{2.22}$$

$$v_y = -d(\frac{\delta_y}{f} + \omega_y) \tag{2.23}$$

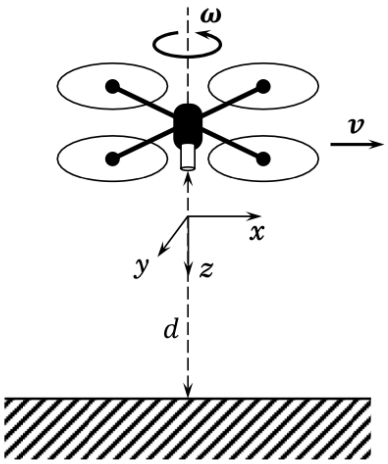


Abbildung 2.4: Darstellung der relevanten Größen zur Ermittlung der Bewegungsgeschwindigkeit der Drohne (Quelle: [56])

2.2.5 Time-of-Flight (ToF) Sensor

Ein Time-of-Flight (ToF)-Sensor kann mithilfe eines Laufzeitverfahrens die Distanz zur Umgebung messen. Hierbei werden Lichtpulse erzeugt und für jeden Bildpunkt eine Distanz ermittelt. Bei einfachen ToF-Sensoren wie dem ST VL53L0X wird lediglich eine einfache Entfernung bis zu 2 m ermittelt. Bei einer Neigung der Drohne um die Horizontal wird bei gleicher Höhe eine höhere Distanz gemessen. Um aus der gemessenen Distanz die Höhe zu ermitteln, muss zunächst die Neigung gegen die Horizontale ermittelt werden. Die Neigung α kann aus der Orientierung (Quaternion) direkt ermittelt werden (siehe Gleichung 2.24). Im Anschluss wird die gemessene Distanz in die Höhe zum Grund umgerechnet. Dies funktioniert natürlich nur, wenn freie Sicht zum Boden ist und der Winkel nicht größer als 90 ° ist.

$$\alpha = 2 \arccos \sqrt{q_{e,w}^2 + q_{e,z}^2} \tag{2.24}$$

$$p_z = d_{tof} \cdot cos(\alpha) \tag{2.25}$$

2.2.6 Barometer

Ein Barometer ist ein Sensor zur Messung des aktuellen Luftdrucks in der Umgebung. Da der Luftdruck mit der Höhe abnimmt, kann dieser als Referenz für die Höhenbestimmung genutzt werden. Da sich der aktuelle Umgebungsdruck durch Wetterereignisse über die Zeit ändern kann, sollte immer eine relative Änderung des Drucks betrachtet werden und der Sensor nicht alleinige Quelle für Höheninformationen dienen. In dieser Arbeit soll kein Barometer genutzt werden, da der ToF genaue Ergebnisse für einen Arbeitsbereich bis 2 m liefert.

2.3 Fehlerarten der Sensoren

Die Messdaten von Sensoren sind oftmals fehlerbehaftet. Es gibt unterschiedliche Fehlerarten. Diese sollen in diesem Abschnitt für die unterschiedlichen Sensorarten beschrieben werden.

2.3.1 Drehratensensor

Ein Fehlermodell für einen Drehratensensor kann folgendermaßen aussehen:

$$\tilde{\vec{\omega}}_{ib}^{b} = M_{Gyro} \cdot \vec{\omega}_{ib}^{b} + \vec{b}_{\omega} + \vec{n}_{\omega} \tag{2.26}$$

Hierbei ist $\vec{\omega}_{ib}^{b}$ der reale Drehratenvektor. Der gemessene fehlerbehaftete Drehratenvektor ist hierbei $\tilde{\vec{\omega}}_{ib}^{b}$.

Der Fehler setzt sich hierbei aus drei Einflussfaktoren zusammen [55]:

- Eine IMU besteht meist aus drei orthogonal angeordneten Drehratensensoren. Da in der Fertigung meist eine kleine Winkelabweichung auftritt, kann die Missweisung über die Misalignment-Matrix M_{Gyro} korrigiert werden.
- $\vec{b}_{\omega}$ ist der Bias des Sensors, d.h. der Nullpunktfehler des Sensors. Dies kann ein konstanter Anteil sein, der während des Betriebs konstant bleibt und ein veränderlicher Anteil, der während des Betriebs driftet.

- $\vec{n}_\omega$ bezeichnet das sensorinhärente Rauschen des Sensors, welches als weiß, normalverteilt und mittelwertfrei angenommen wird.

Oftmals führt der Hersteller bereits werksseitig eine Korrektur von *Gain* und *Offset* des Sensors durch. Durch die Erhitzung beim Löten und Auftragen auf einem Breakoutboard können die Eigenschaften des Sensors sich auch wieder geringfügig verändern [11].

2.3.2 Beschleunigungssensor

Ein Fehlermodell für einen Beschleunigungssensor sieht ähnlich dem Modell des Drehratensensors aus:

$$\tilde{\vec{f}}_{ib}^{b} = M_{Acc} \cdot \vec{f}_{ib}^{b} + \vec{b}_a + \vec{n}_a \tag{2.27}$$

Hierbei ist $\vec{f}_{ib}^{b}$ der reale Beschleunigungsvektor. Der gemessene fehlerbehaftete Beschleunigungsvektor ist hierbei $\tilde{\vec{f}}_{ib}^{b}$.

Der Fehler setzt sich hierbei aus drei Einflussfaktoren zusammen [55]:

- Analog zu den Drehratensensoren, enthält eine IMU ebenfalls drei orthogonal angeordnete Beschleunigungssensoren. Da in der Fertigung meist eine kleine Winkelabweichung auftritt, kann die Missweisung über die Misalignment-Matrix M_{Acc} korrigiert werden.
- $\vec{b}_a$ ist der Bias des Sensors. Der Bias kann einen Anteil aufweisen, der von einwirkenden Vibrationen abhängig ist und damit zeitlich veränderlich ist.
- $\vec{n}_a$ bezeichnet das sensorinhärente Rauschen des Sensors.

2.3.3 Magnetometer

Ein Magnetometer steht hauptsächlich unter dem Einfluss von *Hard-Iron-* und *Soft-Iron-Effekten*. *Hard-Iron-Effekte* werden hierbei normalerweise durch ferromagnetische Materialien mit permanentem magnetischen Feld ausgelöst. *Soft-Iron-Effekte* können auch durch ferromagnetische Materialen entstehen, deren Magnetfeld sich aber durch äußere Einflüsse verändern kann. Dies macht eine Kompensation von *Soft-Iron-Effekte* schwieriger als *Hard-Iron-Effekte*. [11]

Zusätzlich können störende Magnetfelder auch durch elektrische Geräte in der Nähe verursacht werden. Aufgrund dessen soll ein Magnetometer in dieser Arbeit nicht verwendet werden.

2.4 Inertiale Navigation

Die Inertiale Navigation meint die Bestimmung der *Pose* eines Objekts lediglich mit Daten einer IMU. Ein solches System wird auch als Inertial Navigation System (INS) bezeichnet. Bei einem INS handelt es sich um eine sogenannte Koppelnavigation (engl. *Dead Reckoning*), d.h. eine näherungsweise Bestimmung von Geschwindigkeit und Bewegungsrichtung. Die Grundlage hierfür ist die ständige Messung von Beschleunigung und Drehraten. Diese Trägheitsnavigation ist meist nur über kurze Zeit stabil, da sich Fehler über die Zeit aufaddieren und zu einer falschen Ortsbestimmung führen können. Deshalb werden diese Systeme oft in Kombination mit langzeitstabilen Sensoren verwendet. Es kann z.B. GNSS, eine Kamera, ein Laserscanner oder Ähnliches für eine Stützung der Daten genutzt werden.

Früher wurden für INS mechanische Kreiselsysteme eingesetzt. Die Beschleunigungssensoren befanden sich dabei auf einer kardanisch gelagerten, stabilisierten Plattform. Damit konnten die Beschleunigungsmessungen in einem raumfesten Koordinatensystem stattfinden. Drehratensensoren wurden hierbei nicht benötigt bzw. nur zur Korrektur von kleinsten Bewegungsänderungen der Aufhängung durch z.B. Reibung genutzt. Die Orientierung des Systems konnte an der Stellung des Kardanrahmens direkt abgelesen werden. [55]

Heutzutage werden sogenannte SINS eingesetzt. Hierbei sind die Inertialsensoren fest mit den Fahrzeugachsen verbunden. Ein solches System besteht meist aus jeweils drei orthogonal angeordneten Drehraten- und Beschleunigungssensoren und wird als IMU bezeichnet. Im Unterschied zu den mechanischen Kreiselsystemen werden Beschleunigungen im körperfesten Koordinatensystem (b-Frame) gemessen. Für die Berechnungen von Positionsänderungen im Inertialkoordinatensystem müssen die Beschleunigungen unter Berücksichtigung der aktuellen Orientierung zunächst transformiert werden.

SINS wurden erst durch die Entwicklung von Ringlaserkreiseln möglich. Ringlaserkreisel-Systeme nutzen einen Laserstrahl in einem Spiegelsystem, bei dem eine stehende Welle erzeugt wird. Bei der Rotation des Systems verändert sich der Nullpunkt der Welle, dies

kann über einen Detektor gemessen werden. Ringlaserkreisel werden in der kommerziellen Luftfahrt sowie im Militär verwendet. In der privaten Luftfahrt werden oft weiterhin mechanische Kreiselsysteme eingesetzt.

Im Gegensatz zu den Ringlaserkreiseln werden für Strapdown Anwendungen auch sogenannte MEMS-Sensoren verwendet. Diese sind in der Herstellung deutlich günstiger und werden aufgrund ihres geringen Gewichts in nahezu jedem Smartphone sowie Drohnen verbaut. MEMS-Sensoren werden in Abschnitt 2.2 näher beschrieben.

2.4.1 Strapdown Inertial Navigation System (SINS)

Ein SINS dient dazu, die Position und Orientierung eines Objektes zu tracken. Im Folgenden wird die Funktion eines SINS in drei Schritten dargestellt. Die Positionsänderung durch Erdrotation und Corioliskraft wird hierbei vernachlässigt.

1. Orientation Update

Ziel ist es zunächst, mithilfe der Drehraten aus der bekannten Start-Orientierung laufend neue Schätzungen für die aktuelle Orientierung zu berechnen. Die Änderung des Orientierungsvektors kann durch die sogenannte Bortzsche Orientierungsvektordifferentialgleichung ausgedrückt werden. Diese Gleichung kann für hohe Update-Raten und eine konstante Drehgeschwindigkeit vereinfacht gelöst werden (siehe Gleichung 2.28). Die genaue Herleitung wird in [55] näher beschrieben.

$$\Delta q \approx \begin{bmatrix} cos(\frac{\theta}{2}) \\ \vec{u} \cdot sin(\frac{\theta}{2}) \end{bmatrix} \tag{2.28}$$

Der Drehratensensor misst die Drehraten im körpereigenen Koordinatensystem. Hierbei werden die Drehraten des Gyroskop im Vektor $\vec{\omega} = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$ zusammengefasst. Die Rotationsänderung um die Achse $\vec{u} = \frac{\vec{\omega}}{\|\vec{\omega}\|}$ kann mit dem Winkel $\theta = \|\vec{\omega}\|\Delta t$ ausgedrückt werden.

Unter Berücksichtigung der letzten bestimmten Orientierung kann eine neue Schätzung der Orientierung berechnet werden. Da Δq im intrinsischen Koordinatensystem (körperfestes Koordinatensystem) vorliegt, gilt folgender Zusammenhang:

$$q_{t+1} = q_t \otimes \Delta q \tag{2.29}$$

2. Velocity Update

Der Beschleunigungssensor misst den Beschleunigungsvektor $\vec{a}_t$ im körpereigenen Koordinatensystem. Dieser Beschleunigungsvektor muss zunächst mithilfe der im 1. Schritt ermittelten Orientierung in das Navigationskoordinatensystem transformiert werden. Anschließend kann durch Subtraktion die Erdbeschleunigung $\vec{g} = \begin{bmatrix} 0 & 0 & 9.81 \end{bmatrix}^T m/s^2$ aus dem Beschleunigungsvektor eliminiert werden. Durch numerische Integration des daraus entstehenden neuen Beschleunigungsvektors kann die Geschwindigkeit $\vec{v}_{t+1}$ bestimmt werden. Es wird von einer konstanten Beschleunigung über das Messintervall ausgegangen. [11][55]

$$\vec{v}_{t+1} = \vec{v}_t + (R(q_t) \cdot \vec{a}_t - \vec{g}_n)\Delta t \tag{2.30}$$

$R(q_t)$ meint hierbei die Richtungskosinusmatrix für die entsprechende Quaternion q_t.

3. Position Update

Die Position $\vec{p}_t$ im Inertialkoordinatensystem kann über den ermittelten Geschwindigkeitsvektor $\vec{v}_t$ aktualisiert werden. Es wird von einer konstanten Geschwindigkeit Δt über das Messintervall ausgegangen.

$$\vec{p}_{t+1} = \vec{p}_t + \vec{v}_t \Delta t \tag{2.31}$$

Das vorgestellte Verfahren stellt die einfachste Variante eines SINS dar. In der Praxis müssen die Sensordaten gefiltert werden. Ein SINS kann z.B. in Kombination mit einem Madgwick oder Mahony-Filter verwendet werden für eine präzisere Bestimmung der Orientierung und einem Kalman-Filter für die Translation.
Außerdem werden in dem vorgestellten Verfahren lediglich Drehraten für die Bestimmung

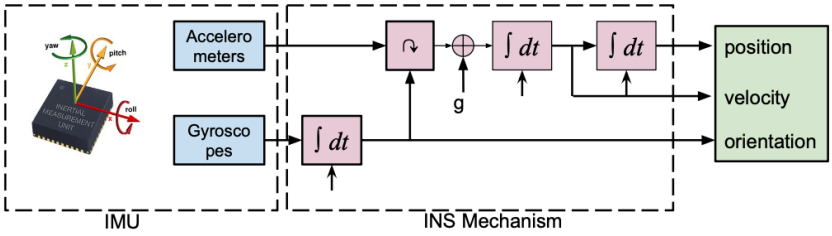


Abbildung 2.5: Funktionsweise eines Strapdown Inertial Navigation System (Quelle: [7])

der Orientierung genutzt. Mithilfe des Beschleunigungssensors und der Bestimmung des Gravitationsvektors können Roll- und Pitch-Winkel langzeitstabilisiert werden [38]:

$$\theta = \arctan\left(\frac{a_y}{a_z}\right) \tag{2.32}$$

$$\phi = \arctan\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right) \tag{2.33}$$

Diese Bestimmung funktioniert jedoch nur für einen Körper, der ausschließlich der Erdbeschleunigung unterliegt. Da eine Drohne im Flug normalerweise immer einer weiteren Beschleunigung unterliegt, können die ermittelten Neigungswinkel jedoch über lange Sicht zur Stabilisierung der Orientierung beitragen.

2.5 Machine Learning

Machine Learning (ML) ist ein Forschungsgebiet der Informatik und wird genutzt, um komplexe Zusammenhänge, die analytisch schwer oder gar nicht zu beschreiben sind, zu lösen.

Hierbei werden meist sogenannte Künstliche Neuronale Netze (KNNs) genutzt. Diese werden mithilfe von Beispieldaten für spezielle Anwendungsfälle trainiert. Beim Training werden die Gewichte im Modell mithilfe einer Fehlerfunktion (engl. *Loss*) angepasst, um den Beispieldaten bestmöglich zu entsprechen. Hierbei wird Wissen abstrahiert, sodass das neuronale Netz auch mit neuen Daten umgehen kann, die es bisher nicht gesehen hat (engl. *unseen data*).

In der Bildverarbeitung wird ML z.B. zur Objektklassifizierung oder Bildgenerierung verwendet. In der Medizin kann ML zur Erkennung von Krebszellen in MRT- oder CT-

Bildern genutzt werden. In der Sprachverarbeitung kann ML genutzt werden, um Dokumente zusammenzufassen, Texte zu übersetzen oder Sprache zu erkennen. Diese Form von Anwendungsfällen werden unter dem Begriff Natural Language Processing (NLP) zusammengefasst.

Die Anwendungsfälle für ML sind in den letzten Jahren stark gestiegen. [18]

Ein KNN ist meist eine Kombination von verschiedenen Layern, die zu einem Netz verbunden werden. Wenn ein Netz sehr viele Schichten aufweist, wird vom sogenannten Deep Learning (DL) gesprochen.

Da das Thema KNN zu umfangreich ist, um es hier komplett darzustellen, soll hier auf gängige Literatur wie [15][18][24] verwiesen und nur ein kleiner Einblick in die relevanten Bereiche für diese Arbeit gegeben werden. Im Folgenden sollen zwei besondere Formen von KNNs (Recurrent Neural Network (RNN) und Temporal Convolutional Network (TCN)) näher erläutert werden:

2.5.1 Recurrent Neural Networks (RNN)

Wenn es um die Interpretation und Prädiktion von zeitbasierten Daten geht, wird oftmals eine besondere Art von KNNs verwendet. Es handelt sich um Recurrent Neural Network (RNN). Diese haben rückwärts gerichtete Verbindungen (engl. *Recurrent Connections*) in ihren *Hidden Layern*, sodass der interne Zustand des Neurons auf den Eingang des Layers zurückgeführt wird. Da der Output eines Neurons damit nach n Zeitschritten direkt abhängig ist von allen bisherigen Inputs, hat das Netz eine Art Gedächtnis (engl. *Memory*). Zur Erklärung kann man ein RNN-Layer entrollen (engl. *Unrolling*), um das Verhalten besser zu verstehen. In Abbildung 2.6 ist eine ausgerollte Darstellung eines einzelnen Neurons über die Zeit zu sehen. Über mehrere Zeitschritte werden Ausgangssignale generiert, der interne Zustand wird weitergereicht [15]. Die Länge der Eingangssequenz muss nicht fest definiert sein, da RNN-Netze mit einer beliebigen Sequenzlänge umgehen können. Theoretisch können RNNs Informationen auch beliebig lange halten, in der Realität sieht dies jedoch anders aus.

RNNs werden mithilfe des Backpropagation Through Time (BPTT) trainiert, d.h. dass das neuronale Netz zum Training vollständig entrollt wird und wie ein gewöhnliches *Dense*-Netz trainiert werden kann.

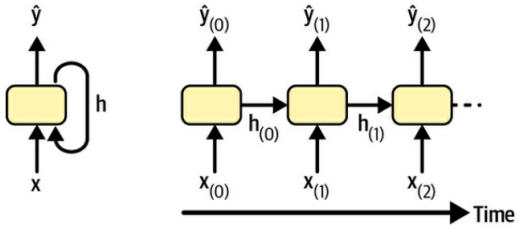


Abbildung 2.6: Ein RNN-Neuron (links) und die entrollte Darstellung (rechts) (Quelle: [3])

Beim Training von RNNs ergeben sich ähnlich wie bei sehr tiefen Netzen einige Herausforderungen. Das *Exploding Gradient Problem* beschreibt die Problematik, dass bei langen Sequenzen während der *Backpropagation* der Fehlergradient sehr stark ansteigen kann, dies führt zu einem instabilen Trainingsvorgang. Aufgrund dessen kann ein einfaches RNN nur für bis zu zehn Zeitschritten trainiert werden. Die Nutzung von gesättigten Aktivierungsfunktionen wie der tanh-Funktion und kleinen Lernraten wirken diesem Phänomen entgegen. Mit Long Short-Term Memory (LSTM)- und Gated Recurrent Unit (GRU)-Zellen werden die einfachen RNNs soweit verbessert, dass Informationen auch über sehr viel mehr Zeitschritte gehalten und die Netze mit lagen Sequenzen trainiert werden können. [3][15]

Bei RNN-Netzen gibt es unterschiedliche Möglichkeiten für den Umgang mit Sequenzen:

- Ein Netz, welches eine Sequenz als Input und Output generiert, wird als ***Sequence-to-sequence*** bezeichnet (Abbildung 2.7 oben links).
- Ein Netz, welches eine Sequenz als Input hat, jedoch nur das Ergebnis des letzten Schritts ausgibt, wird als ***Sequence-to-vector*** bezeichnet (Abbildung 2.7 oben rechts).
- Ein Netz, welches ein statisches Eingangssignal (gleichbleibender Eingangsvektor über die Sequenz) hat und daraus eine Sequenz erstellt, wird als ***Vector-to-sequence*** bezeichnet (Abbildung 2.7 unten links).
- Eine besondere Art sind *Encoder-Decoder* Architekturen. Hierbei wird eine Kombination aus einem ***Sequence-to-vector*** (*Encoder*) und ein ***Vector-to-sequence*** (*Decoder*) genutzt (Abbildung 2.7 unten rechts).

In dieser Arbeit sind vor allem die beiden Fälle **Sequence-to-vector** und **Sequence-to-sequence** von Relevanz.

Bei RNNs wird grundlegend in *Stateful* und *Stateless* RNNs unterschieden.

- Bei einem zustandslosen RNN (*Stateless*) wird der interne Zustand zu Beginn eines Batch (*Initial State*) resettet, d.h. meistens auf Null gesetzt (*Zero-State*).
- Bei einem zustandsbehafteten RNN (*Stateful*) wird der letzte Zustand des vorherigen Batch-Laufs in den neuen Batch übertragen.

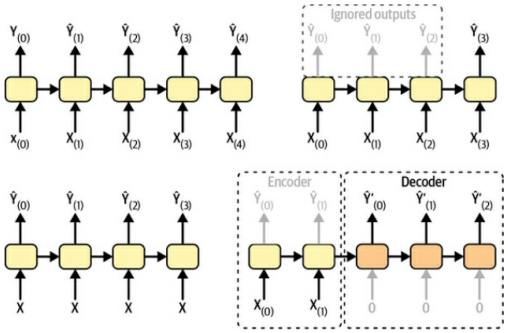


Abbildung 2.7: Unterschiedliche Möglichkeiten für den Umgang mit Sequenzen (Quelle: [3])

2.5.2 Long Short-Term Memory (LSTM)

Die LSTM-Zelle wurde 1997 von Sepp Hochreiter und Jürgen Schmidhuber vorgestellt und später weiter verbessert. Die Zelle hat gegenüber einer einfachen Recurrent Neural Network (RNN)-Zelle einige Vorteile: Informationen können deutlich länger gespeichert werden und beim Training konvergiert das neuronale Netz schneller. Der Aufbau einer LSTM-Zelle ist in Abbildung 2.8 zu sehen. Der Zustand der Zelle teilt sich in die beiden Vektoren $h(t)$ und $c(t)$ auf. Hierbei kann $h(t)$ als der Kurzzeit-Zustand und $c(t)$ als der Langzeit-Zustand interpretiert werden. Der Langzeit-Zustand $c(t)$ wandert durch zwei *Gates* (*Forget-Gate* und *Input-Gate*). Mithilfe der beiden *Gates* können abhängig von der Ansteuerung der Gates, Informationen dem Langzeit-Zustand hinzugefügt oder entfernt werden. Der Kurzzeit-Zustand $h(t)$ und der Ausgangsvektor $y(t)$ sind identisch und werden vom Zustand $c(t)$ kopiert und durch das *Output-Gate* gefiltert. Gesteuert werden die drei Gates über *Fully Connected Layer*, die als Eingangsgrößen den letzten Eingangsvektor $x(t)$ und den Kurzzeit-Zustand $h(t-1)$ des letzten Zeitschritts bekommen.

Die *Fully Connected Layer* weisen eine logistische Aktivierungsfunktion auf. Mithilfe des Layers, welches $g(t)$ ausgibt, können Informationen zum Langzeit-Zustand hinzugefügt werden. [3][15]

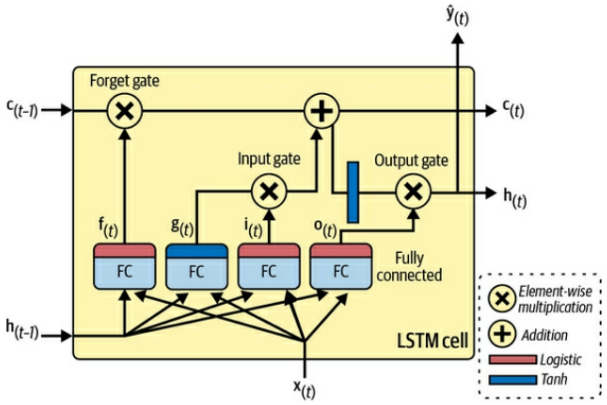


Abbildung 2.8: LSTM Zelle (Quelle: [3])

2.5.3 Gated Recurrent Unit (GRU)

Eine Gated Recurrent Unit (GRU)-Zelle ist eine vereinfachte Version einer LSTM-Zelle und performt aber in der Regel genauso gut. Im Gegensatz zu einer LSTM-Zelle hat eine GRU-Zelle nur einen Zustandsvektor $h(t)$ und einen einzelnen Gate Controller z(t), der das *Forget-* und *Input-Gate* steuert. Es gibt außerdem kein *Output-Gate*. Der *Statevektor* ist mit dem *Outputvektor* identisch. Durch den schlankeren Aufbau einer Zelle, kann ein GRU Netz effizienter trainiert und ausgeführt werden. [3][15]

2.5.4 Temporal Convolutional Network (TCN)

Ein Temporal Convolutional Network (TCN) oder auch *1D Convolutional Network* ist ein zustandsloses *Feed-Forward* Netz und eine Variante eines Convolutional Neural Network (CNN). Es wird speziell für Zeitreihen *Forecasting* oder NLP verwendet. Als Eingangsvektor kann ähnlich zu einem RNN ein *Sliding Window* mit fester Länge verwendet werden. Ein TCN hat üblicherweise einen kausalen Informationsfluss, d.h. es gibt keinen Informationsfluss von der Zukunft in die Vergangenheit. Im Gegensatz zu einem CNN für die Bildverarbeitung wird ein eindimensionaler Kern verwendet. Dieser Filterkern mit entsprechender *Kernel Size* bewegt sich mit der Schrittweite (engl. *stride*) über die

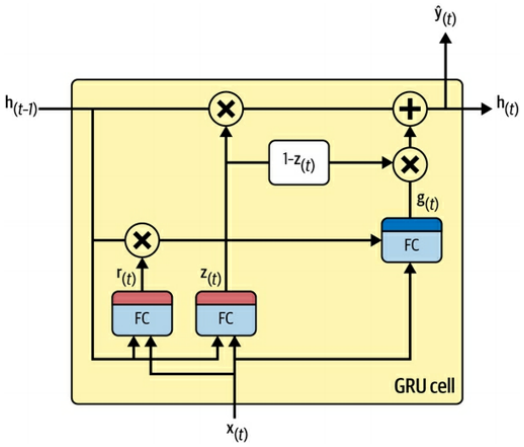


Abbildung 2.9: GRU Zelle (Quelle: [3])

Sequenz und wird sukzessive auf die Eingangssequenz angewandt. Die Eingangssequenz kann hierbei auf eine Ausgangssequenz mit gleicher Länge abgebildet werden. Mit der *Dilation*-Technik können auch sehr lange Sequenzen verarbeitet werden und das Netz einen größeren Kontext der Eingangssequenz berücksichtigen. Hierbei wird das Wahrnehmungsfeld (engl. *receptive field*) mit jeder Schicht exponentiell größer (siehe Abbildung 2.10). [15]

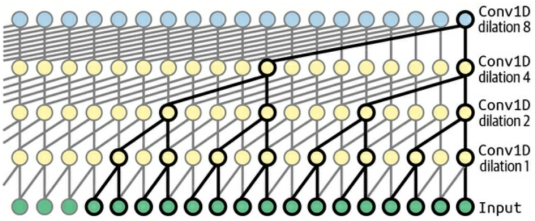


Abbildung 2.10: TCN mit Dilation (Quelle: [15])

2.6 Konventionelle Filter

In diesem Abschnitt sollen einige konventionelle Filterverfahren vorgestellt werden, die zur Bestimmung der Position(-sänderung) und Orientierung(-sänderung) genutzt werden können. Diese sollen später als Referenz für die trainierten KNNs dienen.

2.6.1 Kalman Filter (KF)

Das KF wurde in den 1960ern von Rudolf E. Kalman entwickelt und ist ein leistungsstarkes mathematisches Verfahren der Signalverarbeitung und Zustandsschätzung. Das KF wird in verschiedenen Bereichen wie z.B. in der Navigation, Robotik und im Finanzwesen verwendet.

Das Kalman Filter (KF) wird bei Anwendungen genutzt, in denen die Messungen rauschanfällig sind oder nicht den gewünschten Zustand des Systems widerspiegeln. Um den aktuellen Zustand des Systems zu berechnen, werden aktuelle Messungen und vorherige Schätzungen kombiniert. Die Unsicherheiten sowohl der Messungen, als auch Vorhersagen werden hierbei bestmöglich berücksichtigt. In dieser Arbeit soll die Implementierung des KF aus [40] genutzt werden.

Die Arbeitsweise des KF kann in zwei Schritte untergliedert werden:

- **Prädiktion (Prediction):** Zuerst wird eine Vorhersage für den Zustand des Systems $\hat{x}$ gemacht, basierend auf dem vorherigen geschätzten Zustand x und dem bekannten Systemverhalten (der sogenannten Dynamikmatrix F). Diese Vorhersage ist mit Unsicherheiten behaftet, da sie lediglich das Dynamikmodell berücksichtigt und nicht auf Messungen basiert. Parallel zum Systemzustand wird mithilfe der Dynamikmatrix F auch eine neue Schätzung für die Fehler-Kovarianzmatrix $\hat{P}$ berechnet. Die Kovarianzmatrix gibt die Unsicherheit für den neuen geschätzten Zustand an. Q ist hierbei die Prozessrauschkovarianzmatrix. Die Hut Notation $\hat{}$ wird hier genutzt, um einen geschätzten Wert zu kennzeichnen.

$$\hat{x} = Fx + Bu \tag{2.34}$$

$$\hat{P} = FPF^T + Q \tag{2.35}$$

- **Korrektur (Correction):** In diesem Schritt nutzt das Filter die aktuellen Messungen z, um die Schätzung der Prädiktion $\hat{x}$ zu korrigieren. Es wird eine verbesserte Schätzung x generiert. Außerdem wird eine sogenannte Kalman-Gain-Matrix K berechnet. Diese gibt an, wie stark die Messungen gegenüber der Schätzung aus der Prädiktion gewichtet werden sollen. Hierbei ist x der korrigierte Systemzustand.

$$K = P\hat{H}^T(H\hat{P}H^T + R)^{-1} \tag{2.36}$$

$$x = \hat{x} + K(z - H\hat{x}) \tag{2.37}$$

$$P = (I - KH)\hat{P} \tag{2.38}$$

x Systemzustand
u deterministische Störung
z neue Beobachtungen (Messwerte)
F Übergangsmatrix (Dynamikmatrix)
B Steuermatrix
P Fehler-Kovarianzmatrix
Q Prozesssrauschkovarianzmatrix
K Kalman-Gain-Matrix
H Beobachtungsmatrix (Messmatrix)
R Messrauschkovarianzmatrix
I Einheitsmatrix

2.6.2 Extended Kalman Filter (EKF)

Das Extended Kalman Filter (EKF) ist eine Erweiterung des KF, welche es ermöglicht, auch nichtlineare Zusammenhänge zu berechnen. Die translatorischen Bewegungen der Drohne sind linear. Die Rotationen der Drohne sind jedoch nicht-linear. Das EKF löst die Problematik der Nichtlinearitäten durch eine Linearisierung am Punkt der aktuellen Schätzung. Hierfür wird die Jacobi-Matrix für die nicht-lineare Dynamikmatrix und die nicht-lineare Beobachtungsmatrix genutzt (siehe Gleichung 2.39 und Gleichung 2.40). In dieser Arbeit soll die Implementierung des EKF aus [40] genutzt werden.

$$F = \left.\frac{\partial f(x_t, u_t)}{\partial x}\right|_{x_t, u_t} \tag{2.39}$$

$$\bar{H} = \left.\frac{\partial h(\bar{x}_t)}{\partial x}\right|_{\bar{x}_t} \tag{2.40}$$

2.6.3 Mahony-Filter

Mahony et al. [34] stellen in ihrem Paper von 2008 ein Filter zur Bestimmung der Orientierung mithilfe einer IMU vor. Es handelt sich hierbei um die Erweiterung eines Komplementärfilters. Das Filter kann mit Daten eines Gyroskops, Beschleunigungssensors und optional mit einem Magnetometer genutzt werden. Durch ein PI-Glied als Kompensator werden die gemessenen Drehraten durch die Daten vom Beschleunigungssensor um Drift und Bias korrigiert. Die Berechnung findet mit Quaternionen statt. Der Mahony Filter ist eines der populärsten Filter zur Bestimmung der Orientierung mithilfe von Daten einer IMU. Er wird unter anderem in der *OpenSource Flighcontroller*-Software INAV verwendet [16]. In dieser Arbeit soll die Implementierung des Mahony-Filters aus [40] genutzt werden.

2.6.4 Madgwick-Filter

Madgwick et al. [33] beschreiben 2010 in ihrer Veröffentlichung einen effizienten Algorithmus zur Bestimmung der Orientierung mithilfe einer IMU. Dieser basiert auf einem Gradientenabstiegsverfahren. Es wird in einen IMU-basierten Ansatz für die Nutzung von Drehraten- und Beschleunigungssensoren und einen Magnetic, Angular Rate, and Gravity (MARG)-basierten Ansatz, der zusätzlich die Daten des Magnetometers nutzt, unterschieden. Die Berechnung findet auf Grundlage von Quaternionen statt. Magnetische Störungen, Verzerrungen und Sensordrift werden durch das Gradientenabstiegsverfahren kompensiert. Der prinzipielle Aufbau des Madgwick-Filters ist in Abbildung 2.11 zu sehen. In dieser Arbeit soll die Implementierung des Madgwick-Filters aus [40] genutzt werden.

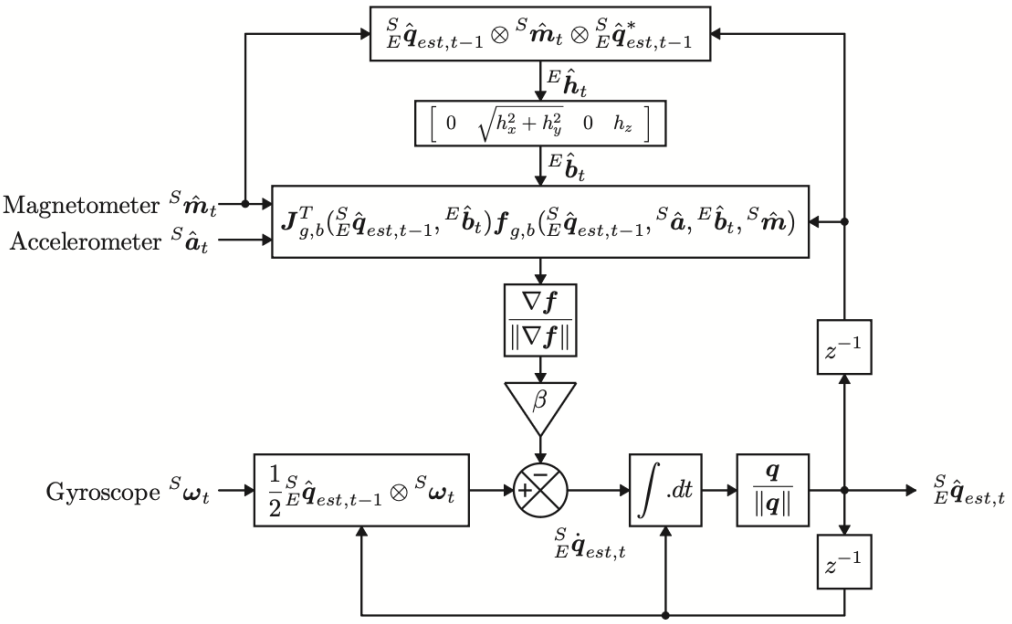


Abbildung 2.11: Madgwick-Filter schematisch (Quelle: [33])

3 Verwandte Arbeiten

In diesem Abschnitt soll der aktuelle Stand der Technik zu dem Thema dieser Arbeit anhand von aktuellen Veröffentlichungen beschrieben werden. In Abschnitt 3.1 werden zunächst konventionelle Filterverfahren zur Schätzung der Position und Orientierung und in Abschnitt 3.2 Verfahren, die auf ML basieren, vorgestellt.

Bei einem regulären SINS basiert die Bestimmung der Pose rein auf IMU Daten. Die aktuelle Orientierung wird durch die Drehraten der Gyros berechnet. Die Position wird dann unter Berücksichtigung der aktuellen Orientierung durch doppelte Integration der Beschleunigungswerte abzüglich der Erdbeschleunigung berechnet. Durch die doppelte Integration der Beschleunigungswerte und eine fehlerhafte Orientierung können über die Zeit starke Driftfehler entstehen [43]. Deshalb werden oftmals Stützinformationen für das SINS benutzt, um die Langzeitstabilität zu gewährleisten. Dies kennzeichnet sogenannte hybride Systeme.
Im Outdoorbereich werden am häufigsten Daten vom GNSS verwendet. Im Indoorbereich gestaltet sich die Lokalisierung schwieriger. Mithilfe der Signalstärke von Funkwellen, wie z.B. Wireless Fidelity (WiFi) oder Bluetooth, kann eine Bestimmung der Position im Raum stattfinden [29][47][42]. Eine andere bekannte Methode ist die Nutzung von Ultra-Wideband (UWB). In anderen Arbeiten werden die Feldstärken des Erdmagnetfeldes oder Anomalien von elektromagnetischen Störungen genutzt [52].
Auch Kameradaten können als ergänzende Information genutzt werden. Solche kombinierten Systeme mit INS werden als Visual Inertial Odometry (VIO) bezeichnet [41]. Im konventionellen Ansatz werden Feature Extraktionen oder Optical Flow Berechnungen genutzt, um die Daten der IMU zu verbessern. Diese Verfahren sind aufgrund der Bildverarbeitung jedoch rechenintensiv [37].
Neben VIO-Systemen ist auch der Einsatz von Radar Inertial Odometry (RIO)- oder Laser Inertial Odometry (LIO)-Systeme möglich. Hierbei wird ein Radar oder Laser zur Wahrnehmung der Umgebung verwendet [46]. Die Hardware dieser Systeme ist jedoch deutlich größer und schwerer als bei VIO oder Inertial Odometry (IO)-Systemen.

Für Fußgänger werden oftmals sogenannte PDR-Algorithmen genutzt. Hierbei wird die IMU z.B. am Fuß einer Person befestigt. PDR-Systeme nutzen die periodische Bewegung des menschlichen Gangs, um z.B. *Zero-Velocity* Phasen zu detektieren und die Sensoren zu kalibrieren. Die Drift ist bei diesen Systemen deutlich kleiner als bei SINS-Systemen [7][23][26]. Diese Systeme werden nur für 2D-Positionsbestimmung genutzt. Für die Nutzung auf einer Drohne ist ein PDR-System daher ungeeignet.

3.1 Konventionelle Ansätze

In diesem Abschnitt sollen Veröffentlichungen zu konventionellen Verfahren zur Bestimmung der Position und Orientierung vorgestellt werden.

Das KF bzw. dessen Erweiterung der EKF oder UKF, gelten als universelles Werkzeug zur Schätzung der Pose mithilfe von IMU Daten [35]. Das EKF und UKF sind die einzigen in diesem Abschnitt vorgestellten Filter, die sowohl Position als auch Orientierung gleichermaßen berechnen können. Beide Verfahren wurden bereits detailliert in Unterabschnitt 2.6.1 und Unterabschnitt 2.6.2 beschrieben.

Mahony et al. [34] stellen in ihrem Paper von 2008 ein Filter zur Bestimmung der Orientierung mithilfe einer IMU vor. Details wurden im Unterabschnitt 2.6.3 beschrieben.

Madgwick et al. [33] beschreiben 2010 in ihrer Veröffentlichung einen effizienten Algorithmus zur Bestimmung der Orientierung mithilfe einer IMU. Details wurden bereits in Unterabschnitt 2.6.4 beschrieben.

Cirillo et al. [11] vergleichen in ihrem Paper von 2016 einen EKF mit dem Mahony und Madgwick Filter. Hierbei wird eine IMU in einen KUKA Youbot Roboterarm gespannt. Anschließend werden diverse Rotationen mit dem Roboterarm abgefahren und die Koordinaten als *Ground Truth* gespeichert. Die Filter werden hierbei auf einem STM32F3 ausgeführt. Die drei Filter performen ähnlich gut. Die Ausführungszeit für das EKF-Filter ist auf dem Mikrocontroller jedoch deutlich höher.

Ludwig et al. [32] verglichen in ihrer Arbeit von 2018 ebenfalls den Mahony, Madgwick und EKF auf einer Drohne bezüglich der geschätzten Orientierung. Für *Ground Truth* wurde ein Vicon-System verwendet. Als Vergleichsmetrik wird der Root Mean Square Error (RMSE) bzw. der Mean Absolute Error (MAE) der geschätzten Euler Winkel

herangezogen. Die Performance aller drei Filter ist ähnlich gut. Das Mahony-Filter zeigt jedoch die besten Ergebnisse und hat die geringste Ausführungszeit. Laut Angaben der Autoren musste das Mahony-Filter allerdings exakt für das Experiment parametriert werden. Das EKF zeigt die längste Ausführungszeit und hat am schlechtesten performt.

3.2 Machine-Learning Ansätze

Es existieren einige Verfahren, die ML zur Unterstützung von konventionellen Filtern nutzen. In [4] wird ein RNN genutzt, um Bewegungen zu erkennen und das beste Kalman-Filter für den aktuellen Systemzustand zu wählen. In [10] und [44] wird ein *Feedforward-Netz* zur Vorverarbeitung der Sensordaten für ein Kalman Filter bzw. der Nachverarbeitung der berechneten Größen des Kalman Filters genutzt. In [8] wird ein bidirektionales Netz zur Schätzung von Geschwindigkeit und Ausrichtung (*Heading*) genutzt.

Es existieren jedoch auch *End-to-End-Ansätze*, bei denen IMU-Daten als Input verwendet werden und Orientierung und Position direkt bestimmt werden. Einige Verfahren fokussieren sich lediglich auf die Bestimmung der Orientierung. Diese werden in Unterabschnitt 3.2.1 beschrieben. Verfahren, die eine vollständige *Pose*-Schätzung vornehmen, werden in Unterabschnitt 3.2.2 beschrieben.

3.2.1 Orientierung

Liu et al. [30] entwickelten 2020 ein neuronales Netz basierend auf GRU-Layern, um die Orientierung mithilfe der IMU Sensordaten (Drehraten-, Beschleunigungssensor, Magnetometer) und einem OF-Sensor zu schätzen. Laut den Autoren enthält der gemittelte Bildfluss des OF-Sensors Informationen, die die Schätzung der Orientierung verbessern können. Im Vergleich zu einem EKF liefert das entwickelte neuronale Netz bessere Schätzungen für die Orientierung und wird auf der verwendeten Hardware STM32F103 schneller ausgeführt als das EKF. Auch ohne die ergänzende Nutzung des OF-Sensors ist der Orientierungsfehler gegenüber der verwendeten *Ground Truth* besser.

Weber et. al [53] entwickelten 2020 einen *End-to-End* Ansatz zur Schätzung der Orientierung (nur *Attitude*) mithilfe eines KNN. Als Eingangsgröße werden Drehraten und Beschleunigungswerte einer 6-Degrees of Freedom (DOF) IMU genutzt. Als Ausgangsgröße wird die Orientierung als Quaternion direkt geschätzt. Diesen Ansatz entwickelten

sie weiter und veröffentlichen 2021 ein weiteres Paper [54]. Hierin stellen sie ein universelles neuronales Netz Robust IMU-based Attitude Neural Network (RIANN) zur Schätzung der Orientierung vor. Es wird bewusst nur eine Neigungsschätzung (*Attitude*) vorgenommen, da laut den Autoren ohne ein Magnetometer keine langzeitstabile Schätzung der Ausrichtung (*Heading*) möglich ist. Das entwickelte Netz soll laut den Autoren für unterschiedliche Anwendungsfälle direkt einsatzfähig sein. Es muss im Gegensatz zu konventionellen Filtern nicht parametriert werden und soll mit IMUs unterschiedlicher Hersteller arbeiten. Laut den Autoren gab es vor dieser Arbeit keinen *Attitude Estimator*, der ähnlich robust für unterschiedliche Hardware und Samplingraten funktioniert. Als Netzarchitektur wird in der ersten Arbeit ein RNN-Netz mit einem TCN-Netz verglichen (siehe Abbildung 3.1). Das RNN-Netz performt besser, d.h. die Schätzungen der Orientierung liegen näher an der *Ground Truth*. In ihrer zweiten Arbeit verbessern sie die Netzarchitektur durch die Nutzung von GRU-Layern statt einfacher RNN-Layern. Im Gegensatz zu [3] und [30] nutzen sie ein zustandsbehaftetes RNN-Netz (*Stateful*). Zur Stabilisierung des Trainings werden mithilfe von Data Augmentation virtuelle IMU Rotationen durchgeführt und künstliche Messfehler (Messrauschen und Gyro Bias) erzeugt. Es wird der BROAD und TUM VI Datensatz zum Trainieren verwendet. Im Vergleich zum Mahony- und Madgwick-Filter ist der Orientierungsfehler von RIANN für sämtliche Testdatensätze deutlich geringer.

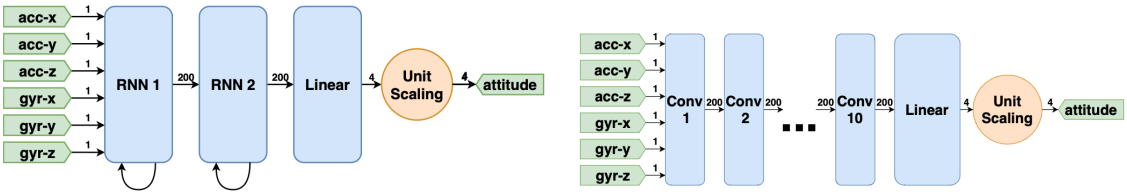


Abbildung 3.1: Neuronale Netzansätze zur Bestimmung der Orientierung (Quelle: [53])

Golroudbari et al. [3] bauen auf den Ansätzen von [53] auf und stellten 2023 ebenfalls einen Ansatz vor, um mithilfe von CNN- und LSTM-Netzen aus 6-DOF IMU Daten eine Schätzung der Orientierung (nur *Attitude*) zu machen. Anders als bei Weber et al. [53] wird ein zustandsloses RNN (*Stateless*) genutzt, d.h. für jede neue Schätzung wird eine Sequenz von 200 IMU-Zeitschritten benötigt. Es werden 100 Zeitschritte aus der Vergangenheit und 100 Zeitschritte aus der Zukunft verwendet.
Es werden drei Netz-Modelle vorgestellt, die sich durch unterschiedliche Kombinationen von LSTM- und CNN-Layern kennzeichnen. In allen Fällen wird die *Attitude* direkt als Quaternion geschätzt. Die Netze werden in über 500 Epochen mit den Datensätzen (BROAD und OxIOD) trainiert. Das Netz wird mit konventionellen Verfahren (Madg-

wick, Mahony und EKF), sowie dem RIANN-Netz von [54] verglichen und hat gegenüber den anderen Verfahren in fast allen Fällen den geringsten Orientierungsfehler (MAE, RMSE).

3.2.2 Position und Orientierung

Laut eigenen Angaben stellen Chen et al. [7] in ihrem Paper von 2018 das erste KNN Framework vor, welches Inertial Odometry ausschließlich auf Basis von inertialen Daten ermöglicht. Das Framework wird Inertial Odometry Network (IONet) genannt. Hierbei werden die Änderungen von Position und Orientierung geschätzt. Laut den Autoren soll durch die Nutzung von Sequenzen eine bestmögliche Schätzung ermöglicht werden. Das Verfahren fokussiert sich wie PDR-Verfahren allerdings lediglich auf eine 2D-Trajektorie. Durch die Nutzung von sequentiellen Daten kann der Drift Fehler minimiert werden. Als Trainingsdaten werden Sequenzen mit einem Vicon-System als *Ground Truth* erstellt. Im Vergleich zu einem SINS- und PDR-System performt IONet besser.

Lima et al. [37] stellen in ihrem Paper von 2019 einen *End-to-End* Ansatz vor, um mit einem KNN eine 6-DOF Odometrie im Raum zu ermöglichen. Laut eigenen Angaben ist es das erste *End-to-End* Verfahren zur Schätzung von Position und Orientierung. Es basiert auf einem zustandslosen RNN aus zwei LSTM-Layern und einem CNN-Layer. Wie in [3] wird ein bidirektionales LSTM verwendet, sodass die Änderung der Pose mithilfe von 100 IMU Zeitschritten aus der Vergangenheit und 100 IMU Zeitschritten aus der Zukunft geschätzt wird. Demnach liegt eine neue Schätzung der Änderung der *Pose* immer mit einer Verzögerung von 100 Zeitschritten vor. Hierbei werden die Änderungen der Orientierung als Quaternion und die Positionsänderung als dreidimensionaler Vektor geschätzt. Hieraus kann die Position und Orientierung durch Integration ermittelt werden. Die Architektur ist in Abbildung 3.2 dargestellt. Als Trainingsdaten werden die Datensätze OxIOD und EuRoC MAV verwendet.

AbdulMajuid et. al [1] stellen 2021 in ihrem Paper ein RNN-Netz zur Schätzung von Orientierung und Position vor. Das Netz wird mithilfe von aufgezeichneten Logs von Drohnen mit einer Pixhawk Flugsteuerung trainiert. Hierbei werden die Schätzungen des internen EKFs als *Ground Truth* verwendet. Das EKF nutzt Global Positioning System (GPS), Barometer und Magnetometer. Insgesamt wurden 83 Flüge mit insgesamt 45 Stunden Flugzeit für das Training verwendet. Das Netz schätzt die Änderung der Orientierung und Position und die Geschwindigkeit. Die absolute Orientierung und Position

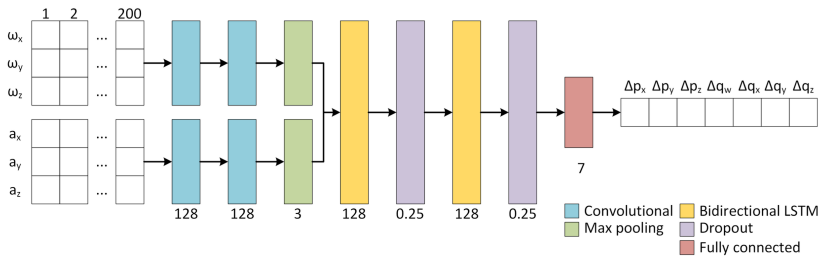


Abbildung 3.2: Neuronales Netz zur Schätzung von Position und Orientierung (Quelle: [37])

wird durch Integration in einem zweiten Schritt berechnet. Als Eingangsdaten dienen IMU, Magnetometer und Barometer-Daten. Hiermit unterscheidet sich die Arbeit von [54] und [53], wo eine absolute Orientierung geschätzt wird. Als *Loss*-Funktion wird ein gewichteter MAE verwendet. Ein Netzaufbau mit vier LSTM Layern und jeweils 200 Neuronen pro Layer erreicht die besten Ergebnisse. Die Architektur ist in Abbildung 3.4 dargestellt. Eine *Window Size* von 200 (40 Sekunden) hat die besten Ergebnisse geliefert. Das Netz ermöglicht laut den Autoren eine gute Positionsschätzung. Es werden jedoch keine Vergleiche mit anderen Verfahren durchgeführt. Aufgrund der hohen Latenz ist es aber nicht sinnvoll für eine *Attitude Estimation* einsetzbar.

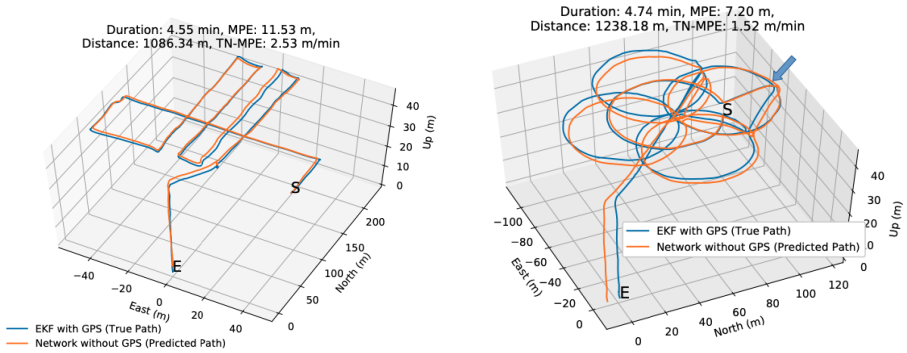


Abbildung 3.3: Validation Flights (Quelle: [1])

Zeinali et al. [58] bauten unter anderem auf der Arbeit von AbdulMajuid et. al [1] auf und stellten 2022 in ihrem Paper die Architektur IMUNet vor. Diese Architektur wurde speziell für mobile Endgeräte entwickelt, um eine bestmögliche Schätzung der Pose mithilfe einer IMU zu machen. Die *Ground Truth* Daten wurden mithilfe eines Simultaneous Localization and Mapping (SLAM)-Verfahrens (*Google ArCore API*) auf zwei

Smartphones erzeugt. Neben dem IMUNet werden auch andere modifizierte mobile Netzarchitekturen (ResNet18, MobileNet, MobileNet v2 ...) mit den Daten trainiert und untereinander verglichen. Das IMUNet ist deutlich schneller bei der Ausführung als die anderen Referenznetze und zeigt mit den geringsten Positionsfehler.

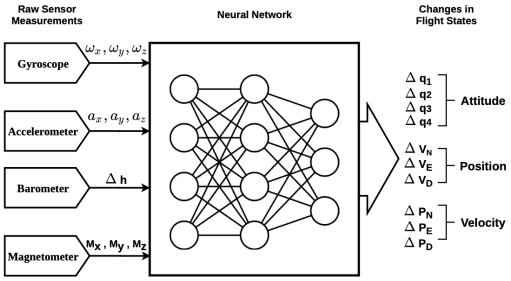


Abbildung 3.4: Neuronales Netz zur Schätzung von Position und Orientierung (Quelle: [1])

Seo et al. [46] entwickelten 2023 in ihrem Paper das Direct-Orientation Inertial Odometry Network (DO IONet). Sie bauen hierbei auf den Erkenntnissen von Lima et al. [37] auf. Im Gegensatz zu Lima et al. [37] wird nicht die Änderung der Orientierung, sondern die absolute Orientierung als Quaternion geschätzt. Hierfür werden ergänzend zu Lima et. al [37] auch Magnetometer-Daten verwendet. Laut den Autoren soll hierdurch die Drift der Orientierung minimiert werden. Die Architektur des KNN (siehe Abbildung 3.5) entspricht weitgehend der von Lima et al. [37]. Es werden zwei CNN Schichten und zwei LSTM Schichten verwendet. Das Netz wird mithilfe des OxIOD Datensatzes trainiert und zeigt im Vergleich zu Lima et al. [37] eine deutlich geringere Drift bei der Orientierung. Wie bei Lima et al. [37] handelt es sich bei dem Netz um ein *Stateless* RNN, d.h. es wird eine Sequenz von 200 Messwerten der IMU benötigt, um eine Schätzung zu erstellen.

3.2.3 Rückschlüsse für die eigene Arbeit

Für das eigene Systemdesign sollen folgende Rückschlüsse aus der Literaturrecherche gezogen werden:

- In vielen Veröffentlichungen ([1][3][37][46]) werden *Stateless* RNN Ansätze gewählt, d.h. es werden gleitende Fenster (engl. *Sliding Windows*) verwendet, um Schätzungen der *Pose* vorzunehmen. Hierbei werden Daten mehrfach redundant verarbeitet.

Wie im Abschnitt 4.5 dargestellt, ist dieser Ansatz für die beschränkte Hardwareleistung auf einer Drohne nicht sinnvoll.

- Eine Nutzung von bidirektionalen RNNs, wie in [3][37][46], ist nur für Offline-Ansätze denkbar, da hierbei Daten aus der Vergangenheit und Zukunft benötigt werden. Dies führt dazu, dass eine Schätzung erst mit einer gewissen Verzögerung vorliegt. Da in dieser Arbeit eine Echtzeitanwendung betrachtet wird, ist dieser Ansatz nicht denkbar.

- Bei einigen Papern ([53][54]) wird lediglich die *Attitude* geschätzt. Die Drohne benötigt jedoch eine vollständige Schätzung der Orientierung inklusive Ausrichtung (*Heading*).

- In einigen Papern wird die Orientierung direkt geschätzt ([3][53][54][46]), in anderen Papern wird die Orientierungsänderung geschätzt ([1][37]). Beide Ansätze sollen in dieser Arbeit verglichen werden.

- In den Papern [37][46] wird der Oxford Inertial Odometry Dataset (OxIOD)-Datensatz zum Training verwendet. Die aufgezeichneten Trajektorien des OxIOD Datensatzes ähneln sich stark untereinander. Es wird vermutet, dass neuronale Netze hier allgemein gut performen, weil sie die ovale Form beim Training abstrahieren können. Da dieser Datensatz im Allgemeinen eine gute Datenqualität aufweist, soll dieser auch in dieser Arbeit verwendet werden. Die Ergebnisse müssen bei der Interpretation jedoch kritisch hinterfragt werden.

- Der OxIOD Datensatz beinhaltet neben der tatsächlich gemessenen Beschleunigung auch Sensorwerte, die bereits um die Erdbeschleunigung korrigiert wurden. Diese werden jedoch von den wenigsten Sensoren in dieser bereinigten Form bereitgestellt. In einigen Papern wie z.B. [37] fallen die Ergebnisse daher deutlich besser aus, als dies mit unbereinigten Daten möglich wäre. In dieser Arbeit sollen daher die tatsächlichen, d.h. die unbereinigten Sensordaten verwendet werden.

- Verfahren, die auf PDR basieren [7] bzw. *Zero-Velocity-Phasen* erkennen, können in dieser Arbeit nicht verwendet werden, da diese auf Muster im menschlichen Gang fokussiert sind.

- In fast allen Papern liegt der Fokus auf der Verarbeitung von IMU-Daten. In dieser Arbeit sollen ergänzend Daten von OF-Sensor und ToF-Sensor verwendet werden.

(In der Arbeit von [30] werden zwar OF-Sensordaten verwendet, jedoch werden diese nur zur Schätzung der Orientierung, nicht aber der Position genutzt.)

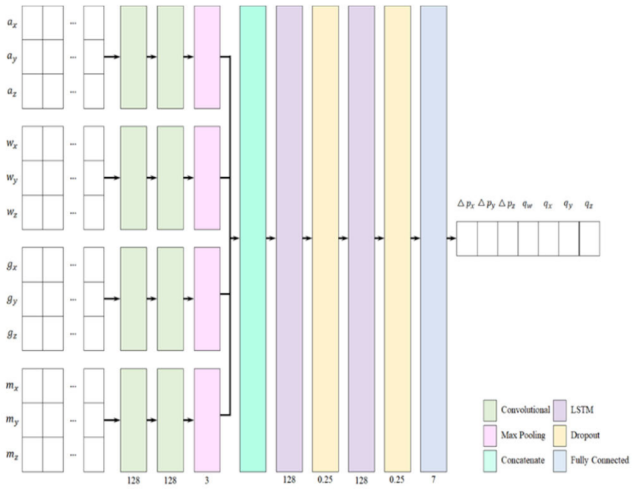


Abbildung 3.5: DO IONet Aufbau des neuronalen Netzes (Quelle: [46])

4 Systemdesign

In Unterabschnitt 3.2.3 wurden bereits einige Rückschlüsse aus der Literaturrecherche gezogen, diese sollen nun in diesem Kapitel in das Systemdesign einfließen. In Abschnitt 4.1 sollen zunächst Rahmenbedingungen für das Systemdesign festgelegt werden und in Abschnitt 4.2 der allgemeine Ansatz skizziert werden. In Abschnitt 4.3 und Abschnitt 4.4 sollen die Eingangs- und Ausgangssignale für das neuronale Netz festgelegt werden. Der Abschnitt 4.6 enthält Überlegungen zur *Loss*-Berechnung. Und in Abschnitt 4.7 sollen *Hyperparameter* für das Training festgelegt werden.

4.1 Rahmenbedingungen

Für das Systemdesign sollen folgende Rahmenbedingung angenommen werden:

- Das entwickelte KNN soll auf der Miniaturdrohne in einer passablen Zeit ausgeführt werden können (Looptime: $< 5\,ms$ entspricht $> 200\,Hz$).
- Es soll keine externe Rechenpower für die Ausführung des Netzes genutzt werden. Sämtliche Berechnungen müssen auf der Drohne stattfinden können.
- Es sollen nur auf der Drohne vorhandene Sensoren (Gyro, Beschleunigungssensor, ToF-Sensor, OF-Sensor) verwendet werden. Externe Trackingsysteme (Vicon, UWB, ...) sollen nicht zur *Pose*-Bestimmung genutzt werden.
- Da die Drohne auch im Indoor-Bereich genutzt werden soll, soll kein GNSS (z.B. GPS) genutzt werden.
- In dieser Arbeit soll von einer maximalen Flughöhe von 2 m ausgegangen werden, da dies der maximale Arbeitsbereich des ToF-Sensors ist.

- Es wird von einer statischen Umgebung ausgegangen, in der sich keine dynamischen, bewegten Objekte befinden, sodass der durch den OF-Sensor ermittelte optische Fluss nicht durch bewegte Objekte verfälscht wird.

4.2 Allgemeiner Ansatz

Wie in der aufgezeigten Literatur in Kapitel 3 ist ein KNN mithilfe von Trainingssequenzen und einer *Ground Truth* generell in der Lage, die Bestimmung der Position(-sänderung) und Orientierung(-sänderung) aus den Sensordaten zu abstrahieren. Im Gegensatz zu den konventionellen Verfahren lernt das neuronale Netz hierbei die notwendigen physikalischen Zusammenhänge selbst. Die referenzierten Arbeiten nutzen diverse Ansätze mit unterschiedlichen Schwerpunkten. In dieser Arbeit sollen die bereits gewonnen Erkenntnisse für das eigene Systemdesign verwendet werden.
Ein naiver Ansatz wäre die Nutzung von *Dense-Layern* zur Schätzung der Position und Orientierung aus den Sensordaten. Eine absolute Schätzung von Position und Orientierung nur mithilfe eines Samples der Sensordaten ist jedoch nicht möglich, da die Sensoren (Drehraten- und Beschleunigungssensor) lediglich Änderungen messen können. Demnach ist immer der vorherige Zustand (d.h. die letzte Orientierung) bei jeder Iteration zu berücksichtigen. Bei dem *Dense*-Ansatz ließe sich dies nur durch eine Rückführung von Ausgangsgrößen auf den Eingang des Netzes realisieren (*Autoregression*). Ein solches Netz ist jedoch schwierig zu trainieren und eine Parallelisierung der Berechnungen schwer möglich [53]. Daher soll dieser Ansatz hier verworfen werden.

In vielen Arbeiten ([53][3][37][46]) werden TCN-Layer alleine oder teilweise in Kombination mit anderen Layern verwendet. Bei TCN-Netzen handelt es sich um Faltungsnetze. Die Funktionsweise eines TCN-Netzes wurde in Unterabschnitt 2.5.4 bereits näher erläutert. In den genannten Arbeiten werden gleitende Fenster (*Sliding Windows*) benutzt, die meist 200 Messwert-Sample für einen Iterationsschritt nutzen. Die Ergebnisse der Arbeiten zeigen, dass ein TCN-Netz zur Schätzung der Orientierung(-sänderung) und Positions(-änderung) in der Lage ist. Das gleitende Fenster (*Sliding Windows*) führt jedoch zu einer redundanten Verarbeitung von Messwert-Samplen und damit zu einer langen Ausführungszeit auf dem Mikrocontroller der Drohne. Da mit dem Ansatz in den referenzierten Arbeiten aber sehr gute Ergebnissen erreicht wurden, soll dieser Ansatz trotz des hohen Rechenbedarfs als Referenz evaluiert werden. In Unterabschnitt 4.5.2 wird ein entsprechendes Netz mit dieser Architektur vorgestellt.

In den Arbeiten von [30][37][46][53] wurden RNNs als die optimale Netzvariante für die Schätzung einer *Pose* mithilfe von Sensordaten ermittelt. RNNs enthalten einen Speicher, um Zustände über mehrere Schritte hinweg zu speichern. Die Funktionsweise von RNNs wurde bereits in Unterabschnitt 2.5.1 näher erläutert. Die *Stateless*-Variante des Netzes ist aufgrund der mehrfachen Verarbeitung der Sensordaten in *Sliding Windows*, ähnlich wie ein TCN, sehr rechenintensiv. In [54] wurde jedoch auch schon ein *Stateful*-Ansatz erfolgreich für die Bestimmung der Neigung (*Attitude*) genutzt. Dieser Ansatz soll nun auf eine komplette Schätzung von Position(-sänderung) und Orientierung(-sänderung) adaptiert werden. In Unterabschnitt 4.5.1 wird ein entsprechendes Netz mit dieser Architektur vorgestellt.

In der überwiegenden Anzahl der referenzierten Arbeiten wird ein *Fully-Connected Dense Layer* als Ausgangsschicht verwendet. Dieses Layer beinhaltet die Anzahl der zu ermittelten Größen für die *Pose*-Schätzung. In dieser Arbeit soll auch ein *Dense-Layer* als letzte Schicht genutzt werden. Die genaue Ausgestaltung der Ausgangsgrößen wird in Abschnitt 4.4 diskutiert. Als Aktivierungsfunktion für die RNN-Layer soll eine *tanh*-Funktion verwendet werden, da lediglich mit dieser eine GPU-Hardwarebeschleunigung (cuDNN) in Keras möglich ist [13].

4.3 Eingangsgrößen

Die Eingangsgrößen dienen dem neuronalen Netz als Grundlage für die Schätzung der *Pose*. Wie bei den konventionellen Filtern, sollen hierfür die zur Verfügung stehenden Sensordaten genutzt werden. In allen referenzierten Arbeiten wird ein *End-to-End* Ansatz für die Sensordaten genutzt, d.h. es findet keine Vorverarbeitung (ausgenommen eine Standardisierung) der Rohdaten statt. Auch in dieser Arbeit sollen die Rohdaten der Sensoren als Eingangsvektor in das KNN geleitet werden. Hierbei werden die Rohdaten aller Achsen des Drehratensensors (ω_x, ω_y, ω_z) und Beschleunigungssensors (a_x, a_y, a_z) separat ins neuronale Netz geleitet. Da die Magnetometer-Daten im Indoorbereich jedoch unzuverlässig vorliegen, soll auf diese gänzlich verzichtet werden. Für die späteren Versuche soll in dieser Arbeit ein Ansatz, der rein auf inertialen Sensordaten basiert (Variante IMU) und ein Ansatz, der zusätzlich die Daten von ToF- und OF-Sensor verwendet (Variante OF), untersucht werden (siehe auch Abbildung 4.1) Die Daten des optischen Flusssensors (δ_x, δ_y) und ToF-Sensors (h) werden dafür ergänzend in das neuronale Netz geleitet.

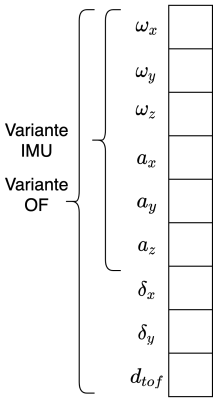


Abbildung 4.1: Eingangsgrößen für neuronales Netz

4.4 Ausgangsgrößen

Ziel des neuronalen Netzes ist es, die Orientierung und Position relativ zum Startpunkt bestmöglich zu schätzen. Für die Repräsentation von Positions(-änderung) und Orientierung(-sänderung) sollen hier unterschiedliche Ansätze erläutert werden, die später in den Versuchen getestet und verglichen werden. In Abbildung 4.2 sind die verschiedenen Varianten für die Ausgangsgrößen dargestellt.

Orientierungs(-änderung)
Für die Schätzung der Orientierung werden in der Literatur unterschiedliche Ansätze gewählt. Weber et al. [54] nutzen ein *Stateful*-RNN und machen mit dem RIANN-Netz eine direkte Schätzung der *Attitude*. Lima et al. [37] nutzen ein neuronales Netz, welches eine Schätzung der Änderung der Orientierung berechnet und Seo et al. [46] nutzen eine absolute Orientierungsschätzung, jedoch unter zu Hilfenahme von Magnetometer-Daten.

In dieser Arbeit sollen die unterschiedlichen Varianten für diesen Anwendungsfall verglichen werden und ein eigener verbesserter Ansatz erprobt werden. Hierfür wird in der ersten Variante die Orientierung (q_w, q_x, q_y, q_z) direkt geschätzt (Variante DIRECT). In einer zweiten Variante soll die Änderung der Orientierung (Δq_w, Δq_x, Δq_y, Δq_z) geschätzt werden (Variante DELTA). Darüber hinaus soll ein hybrider Ansatz getestet werden, der es ermöglicht, die *Attitude* absolut und die *Heading* als Änderung zu schätzen. Dies ist sinnvoll, da durch die Daten des Beschleunigungssensors eine absolute Schätzung der *Attitude* möglich ist, eine Schätzung der *Heading* aufgrund der fehlenden Magnetometer-Daten jedoch nicht. Hierfür wird eine Quaternion (q_w, q_x, q_y, q_z), welche

lediglich die Neigungsinformation (*Attitude*) enthält und eine einfache Winkeländerung $\Delta\theta$, welche die Änderung der *Heading* repräsentiert, geschätzt. In Abbildung 4.2 werden die drei Varianten dargestellt.

Positions(-änderung)
Für die Schätzung der Position wird in der überwiegenden Anzahl der referenzierten Arbeiten eine Änderung der Position gegenüber dem letzten Zeitschritt im körpereigenen Koordinatensystem geschätzt. Dies soll bei dem eigenen Systemdesign genauso gemacht werden.
Die Positionsänderung muss dann unter Berücksichtigung der aktuellen Orientierung ins Navigationskoordinatensystem umgerechnet werden, um die Position relativ zum Startpunkt zu berechnen. In Abbildung 4.2 werden bei allen dargestellten Varianten die Positionsänderungen (Δp_x, Δp_y, Δp_z) ausgegeben.

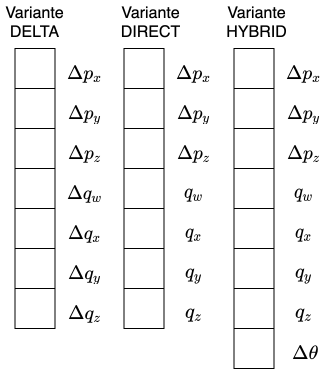


Abbildung 4.2: Ausgangsgrößen für neuronales Netz

4.5 Netzgröße und Laufzeit

Die Netzgröße eines neuronalen Netzes gibt die mögliche Kapazität zur Modellierung von komplexen Funktionen wieder. Ein größeres Netz ist daher grundlegend besser in der Lage, komplexe Probleme zu lösen und anspruchsvolle Daten zu modellieren. Jedoch kann ein zu großes Netz auch zu *Overfitting* führen, sodass das Netz zu stark an die Trainingsdaten angepasst ist und schlecht auf neue Daten generalisiert.

In den referenzierten Arbeiten [37][46][53][54] hat sich die Nutzung von zwei RNN-Layern als guter Kompromiss herauskristallisiert, daher sollen auch in dieser Arbeit zwei RNN-Layer gewählt werden.

Die Anzahl der Neuronen pro Layer schwankt in den referenzierten Arbeiten zwischen 128 und 200. In der Arbeit von Weber et al. [53] wird eine Neuronenanzahl pro RNN-Layer von 200 als optimal ermittelt. Aber bereits mit 20 Neuronen pro Layer konnten in der gleichen Arbeit ähnliche Ergebnisse wie bei dem verwendeten Referenzfilter erreicht werden.

Die Netzgröße wird in dieser Arbeit aufgrund der verwendeten Hardware jedoch stark eingeschränkt (siehe Abschnitt 4.5). Wie in Abschnitt 4.1 beschrieben, muss das KNN auf der Drohne in einer passablen Zeit ausgeführt werden können, damit die Drohne ein sauberes Flugverhalten zeigt. Es wird davon ausgegangen, dass eine *Looptime* von maximal 5 ms (200 Hz) noch passabel ist.

Um unnötiges Trainieren von Netzvarianten zu vermeiden, die später nicht auf der Drohne ausgeführt werden können, wurden die Laufzeiten von unterschiedlichen Netzgrößen auf dem STM32H743 Mikrocontroller ermittelt. Hierbei wurden untrainierte neuronale Netze auf dem Mikrocontroller der Drohne ausgeführt und die Laufzeit für eine einzelne Ausführung gemessen. Es wird dabei davon ausgegangen, dass ein untrainiertes Netz eine ähnliche Ausführungszeit hat wie ein trainiertes Netz, da sich lediglich die Gewichte unterscheiden, die Rechenoperationen aber identisch sind.

Es werden *Stateless* und *Stateful* Netzvarianten mit 2 GRU-Layern und einem abschließenden *Dense-Layer* getestet. Hierbei werden 9 Eingangs- und 7 Ausgangsgrößen verwendet, dies entspricht in etwa der später in den Versuchen (siehe Kapitel 7) zu testenden Spezifikation. Die Neuronenanzahl wird für die *Stateful*-Layer zwischen 50, 100 und 200 variiert. Die Sequenzlänge wird auf 1 festgesetzt, da ein *Stateful*-Netz einen internen Zustand über den kompletten Betrieb speichert, so ist pro *Loop* nur ein neues Messwert-Tupel erforderlich, um eine neue Schätzung der *Pose* zu generieren. Bei dem *Stateless*-Netz wird die Neuronenanzahl auf 200 festgesetzt. Systembedingt benötigt ein *Stateless*-Netz immer eine Sequenz mit Sensormesswerten, um ein Ergebnis zu liefern. Die Sequenzlänge wird zwischen 50, 100 und 200 variiert.

Ergebnis

Eine Übersicht der ermittelten Laufzeiten für unterschiedliche Netzgrößen ist in Tabelle 4.1 dargestellt. Als Referenz wurde zunächst die Ausführung der Flugsteuerung ohne Nutzung eines neuronalen Netzes ermittelt. Diese liegt bei 1,4 ms. Die Ausführungszeit für das *Stateful*-Netz liegt je nach Neuronenanzahl zwischen 1,9 ms und 7,2 ms. Die Ausführungszeit für das *Stateless*-Netz liegt je nach Sequenzlänge zwischen 277,8 ms und 1104 ms. Aus den gemessenen Laufzeiten lässt sich schlussfolgern, dass die Nutzung von

einem *Stateless*-RNN auf der Drohne mit der zur Verfügung stehenden Hardware nicht möglich ist, da die Ausführungszeit bei 200 Neuronen pro Layer und einer Sequenzlänge von 50 bereits bei 277,8 ms liegt und damit deutlich über der angenommenen maximalen Ausführungszeit von 5 ms. Dagegen liegt die Ausführungszeit bei einem *Stateful RNN* mit 100 Neuronen pro Layer gerade mal bei 3 ms. Die Nutzung von *Stateful* RNNs mit 100 Neuronen pro Layer wäre daher auf der Drohne möglich. Ggf. kann die Neuronenanzahl weiter erhöht werden, um näher an die Ausführungszeit von 5 ms zu gelangen.

Tabelle 4.1: Betrachtung von Ausführungszeiten unterschiedlicher Netzarchitekturen auf dem STM32H743 Mikrocontroller

Modell	**Layer**	**Input**	**Neuronen pro Layer**	**Sequenzlänge**	**Zeit [ms]**
Ohne NN	-	-	-	-	1,4
Stateful RNN	2x GRU	9	50	1	1,9
Stateful RNN	2x GRU	9	100	1	3,0
Stateful RNN	2x GRU	9	200	1	7,2
Stateless RNN	2x GRU	9	200	50	277,8
Stateless RNN	2x GRU	9	200	100	550,3
Stateless RNN	2x GRU	9	200	200	1104,0

Es sollen nun zwei Netzvarianten für die folgenden Versuche vorgestellt werden.

4.5.1 Netzvariante 1: GRU

In der ersten Netzvariante soll ein RNN-Netz mit zwei GRU-Layern untersucht werden, welches *Stateful* arbeitet, d.h. der interne Zustand des Netzes wird nach einer Schätzung nicht zurückgesetzt. Es werden GRU-Layer anstatt von LSTM-Layern genutzt, da diese effizienter arbeiten, aber ähnlich gut performen wie LSTM-Layer (siehe Unterabschnitt 2.5.3). Aufgrund des *Stateful*-Ansatzes wird jedes neue *Messwert-Sample* dem neuronalen Netz zugeführt und daraus direkt eine neue *Pose*-Schätzung erzeugt. Dieser Ansatz ist mit Ausnahme von [54] nicht in der Literatur zu finden. Die Annahme ist, dass ein solches Netz ähnlich gut performt wie ein *Stateless*-Ansatz, jedoch deutlich ressourcenschonender ist. Im Anhang ist das Netzmodell detailliert dargestellt (siehe Abbildung 10.1 und Abbildung 10.2).

In den Versuchen (siehe Kapitel 7) sollen die Parameter des Netzes variiert werden, um die bestmögliche Ausgestaltung des Netzes zu finden. Dies wird in Abbildung 4.3 durch Varianten dargestellt. Der Parameter x meint eine Variation der Neuronenanzahl

pro Layer. Die Neuronenanzahl hat direkten Einfluss auf die Ausführungszeit (wie in Abschnitt 4.5 beschreiben). Es wird später in den Versuchen eine Neuronenanzahl von 50, 100 und 200 getestet.

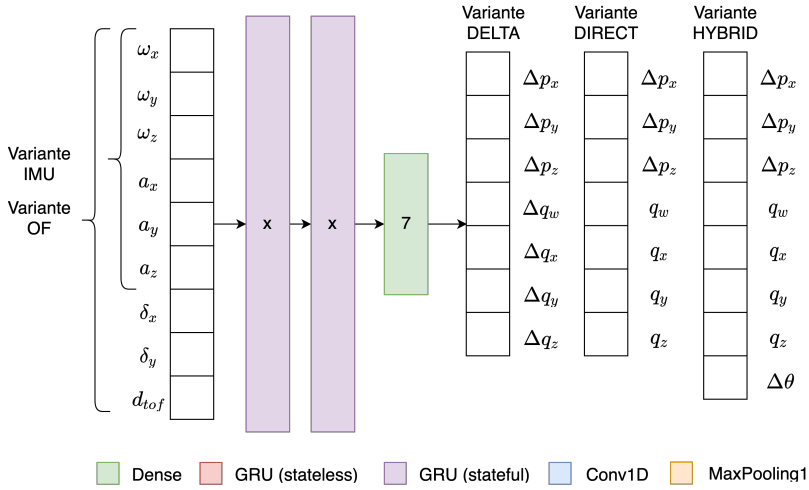


Abbildung 4.3: Netzvariante 1: GRU (Die Neuronenanzahl pro Layer (x) soll variiert werden)

4.5.2 Netzvariante 2: TCN

In der zweiten Netzvariante soll eine Kombination aus TCN- und GRU-Layern getestet werden. Dieser Ansatz ist an Lima et al. [37] und Seo et al. [46] angelehnt. Die ersten beiden Layer sind TCN-Layer mit einer Neuronenanzahl von 128 und einer *Kernel Size* von 11. Es folgt ein *MaxPooling-Layer* und zwei GRU-Layer, die als *Stateless* genutzt werden. Das Netz wird mit Sequenzen mit festgelegter Länge von z.B. 200 Messwert-Tupeln genutzt. Nach jeder Schätzung mithilfe eines *Sliding Windows* wird der interne Zustand der LSTM-Layer zurückgesetzt. (siehe Abbildung 10.3 und Abbildung 10.4)

In den Versuchen (siehe Kapitel 7) sollen später Parameter des Netzes variiert werden, um die bestmögliche Ausgestaltung des Netzes zu finden. Dies wird in der Abbildung 4.4 durch Varianten dargestellt. Der Parameter x meint eine Variation der Neuronenanzahl pro Layer. Wie bei Netzvariante 1 wird eine Neuronenanzahl von 50, 100 und 200 später in den Versuchen getestet.

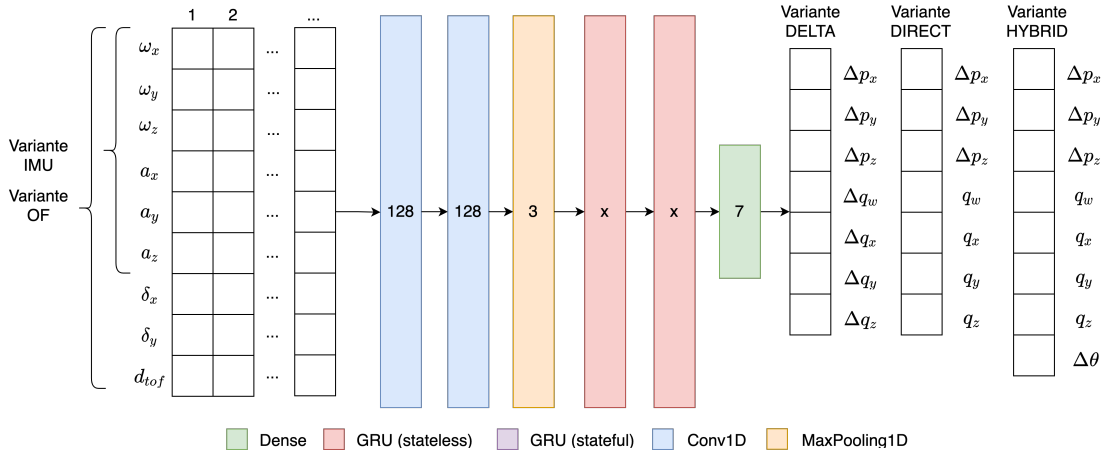


Abbildung 4.4: Netzvariante 2: TCN (Die Neuronenanzahl pro Layer (x) soll variiert werden)

4.6 Loss

Die *Loss*-Funktion gibt an, wie weit eine Prädiktion von dem wahren bzw. gewünschten Wert entfernt ist. Mithilfe eines Optimierungsverfahrens werden die Gewichte im neuronalen Netz nach jedem *Batch* schrittweise angepasst. Ziel beim Trainingsvorgang ist es, den *Loss* zu minimieren. Die Wahl der *Loss*-Funktion ist wichtig für den Erfolg des Trainings. Im Folgenden werden sinnvolle *Loss*-Funktionen für Position und Orientierung diskutiert und eine Bündelung der *Loss*-Funktionen zu einem Gesamt-*Loss* beschrieben.

4.6.1 Position

Wie bereits zuvor beschrieben, soll das neuronale Netz die Positionsänderung im körpereigenen Koordinatensystem schätzen. Wie in [37], soll der Delta Position Mean Absolute Error (DPMAE) verwendet werden. Hiermit ist der *L1-Norm* der Differenz der berechneten Positionsänderung $\Delta\hat{p}$ zur tatsächlichen Positionsänderung (*Ground Truth*) Δp gemeint.

$$e_{DPMAE} = \|\Delta\hat{p} - \Delta p\|_1 \tag{4.1}$$

4.6.2 Orientierung

Übliche *Loss*-Funktionen wie der MAE oder Mean Square Error (MSE) können für die Orientierung nicht direkt verwendet werden. Eine Berechnung des MAE bzw. MSE der einzelnen Komponenten einer Quaternion wäre nicht zielführend, da die Komponenten der Quaternion keinen linearen Zusammenhang haben. Daher sollen hier alternative *Loss*-Funktionen betrachtet werden.

Für die Orientierung bzw. Orientierungsänderung soll nach [3] der Quaternion Error (QE) bzw. der Winkel zwischen tatsächlicher Orientierung q und geschätzter Orientierung $\hat{q}$ als *Loss* genutzt werden. Hierbei muss berücksichtigt werden, dass das Argument für den *Arccos* auf den Bereich [-1,1] beschränkt wird, da der Gradient sonst extreme Werte annehmen würde [3].

$$\Delta q = \hat{q} \otimes q^* \tag{4.2}$$

$$e_{QE} = 2 \arccos |\Delta q_w| \tag{4.3}$$

Alternativ kann anstatt des *Arccos* eine nicht trigonometrische Variante genutzt werden. So wird der arccos(x) durch die Funktion 1-x ersetzt. Die Problematik von extremen Gradienten bei Argumenten nahe 1 kann hierdurch umgangen werden. Diese Variante soll als SAFE bezeichnet werden.

$$e_{QE,safe} = 1 - |\Delta q_w| \tag{4.4}$$

4.6.3 Attitude

Wie in Abschnitt 4.4 beschrieben, soll auch eine HYBRID-Variante erprobt werden, hierbei muss die *Attitude* isoliert von der Orientierung bewertet werden. Die *Attitude* wird als Quaternion repräsentiert. Die folgende Funktion ermöglicht unabhängig von der aktuellen *Heading* eine Bewertung einer geschätzten Quaternion $\hat{q}$ zur realen Quaternion q.

$$\Delta q = \hat{q} \otimes q^* \tag{4.5}$$

$$e_a = 2 \arccos\left(\sqrt{\Delta q_w^2 + \Delta q_z^2}\right) \tag{4.6}$$

Wie bei der Orientierung kann auch hier eine vereinfachte Variante genutzt werden, bei der der Arccos(x) durch 1-x ersetzt wird. Es ergibt sich folgende Funktion:

$$e_{a,safe} = 1 - \sqrt{\Delta q_w^2 + \Delta q_z^2} \tag{4.7}$$

4.6.4 Heading

Für die hybride Variante muss die Winkeländerung der *Heading* als *Loss* bewertet werden. Es soll daher ein einfacher Betrag über die Winkelabweichung gebildet werden:

$$e_\theta = |\Delta\hat{\theta} - \Delta\theta| \tag{4.8}$$

4.6.5 Multi-task Learning

Für das Training muss das Netz nach zwei bzw. drei unterschiedlichen *Loss*-Funktionen optimiert werden. Dies wird auch als *Multi-task Learning* bezeichnet. Die einfache Addition der *Loss*-Funktionen birgt den Nachteil, dass durch unterschiedliche Skalierungen die *Loss*-Funktion mit der größeren Varianz beim Optimieren einen größeren Einfluss auf die Anpassungen der Gewichte hätte. Eine sinnvolle Skalierung der Loss-Funktionen ist daher sinnvoll. In [37] wird eine Funktion vorgestellt, bei der die Varianzen σ_i^2 der unterschiedlichen *Loss*-Funktionen L_i als Gewichte trainiert werden. Hiermit soll eine automatisierte Gewichtung der *Loss*-Funktionen während des Trainingsvorgangs ermöglicht werden. Diese werden im Gesamt-*Loss* L_{total} gebündelt. Diese Funktion wird später in allen Versuchen verwendet.

$$L_{\text{total}} = \sum_{i=1}^{n} e^{-\log \sigma_i^2} L_i + \log \sigma_i^2 \tag{4.9}$$

4.7 Hyperparameter

Es soll der *Adam-Optimizer* verwendet werden. Die genutzten Lernraten schwanken in der Literatur zwischen 0,0001 und 0,001. In dieser Arbeit soll eine Lernrate von 0,0001 verwendet werden. Die *Batch Size* variiert in den Arbeiten zwischen 32 und 1024. Es soll eine *Batch Size* von 32 für alle Versuche verwendet werden.

5 Plattform

In Abschnitt 5.1 werden die in dieser Arbeit verwendeten Frameworks zur Datenverarbeitung und zum Trainieren der neuronalen Netze genannt. Abschnitt 5.2 enthält eine Vorstellung der Miniaturdrohne.

5.1 Software

Die trainierten neuronalen Netze sollen auf einem STM32H743 Mikrocontroller ausgeführt werden. Um dies zu ermöglichen, muss eine Konvertierung der neuronalen Netze in C-Code durchgeführt werden. Die offizielle Entwicklungsumgebung des Herstellers STM32CubeIDE bietet dafür über das Framework X-CUBE-AI eine Unterstützung für sämtliche Formate an. Da eine native Unterstützung von *Stateful* RNN-Netzen jedoch nur für das *keras*-Format vorhanden ist, soll die *high-level API* Keras mit TensorFlow Backend für das Trainieren der neuronalen Netze verwendet werden. Das Training wird in Google Colab mit einer NVIDIA Tesla T4 Grafikkarte durchgeführt. Hierbei wurde teilweise Quellcode von Lima et. al [25] genutzt. Es wird Python in der Version 3.10.12 genutzt. Die wichtigsten genutzten Python Bibliotheken sind:

- keras 2.15.0
- tensorflow 2.15.0
- ahrs 0.3.1
- filterpy 1.4.5
- pandas 1.5.3
- scipy 1.11.4
- seaborn 0.13.1

5.2 Hardware

In dieser Arbeit wird eine Miniaturdrohne genutzt, die in einer vergangenen Arbeit [17] aufgebaut wurde. Die Drohne hat eine Größe von ca. 11x11 cm und ein Gewicht von 96 g.

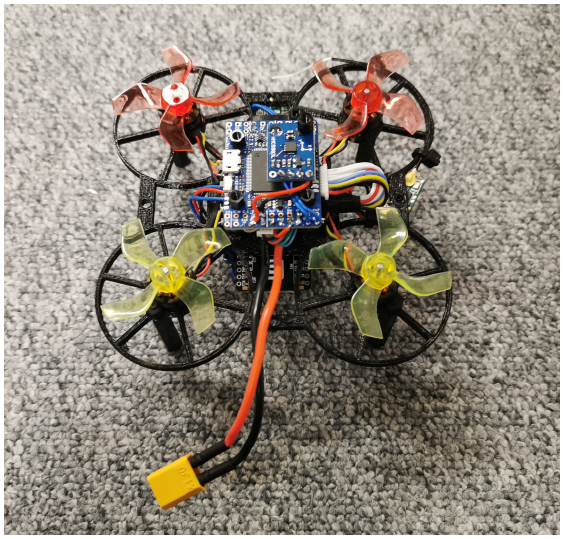


Abbildung 5.1: Miniaturdrohne

Auf der Drohne wird ein Flight Controller (Matek H743-MINI) verwendet. Dieser hat einen 32-bit Mikrocontroller vom Typ STM32H743VIT6 [49]. Es befinden sich zwei 6-achsige MEMS-IMUs auf dem *Breakout Board*. Dies ist zum einen der TDK InvenSense MPU-6000 [22] und der TDK InvenSense ICM-20602 [21]. Es ist außerdem ein Barometer der Firma Infineon (DPS310) verbaut.

Auf der Drohne wird ein OF-Sensor vom Typ Matek 3901-L0X Optical Flow verwendet. Dieser vereint den Sensor PMW3901 und den ToF-Sensor VL53L0X. Der VL53L0X hat einen Arbeitsbereich von 8-200 cm. Das auf dem OF-Sensor verwendete Verfahren zur Bestimmung des optischen Flusses ist nicht bekannt.

Es wird eine selbstentwickelte Flugsteuerung-Software verwendet. Diese basiert auf einer kaskadierten PID-Regelung für Orientierung- und Positionsregelung. Der Quellcode wurde in C mit der Software STM32CubeIDE entwickelt.

6 Datensätze

Für das Training der KNNs sind Trainingsdaten nötig. Hierbei sind Datensätze mit Sensordaten und der entsprechenden *Ground Truth* nötig. Generell muss in synthetische und reale Datensätze unterschieden werden. Bei den synthetischen Datensätzen werden die Sensordaten und *Ground Truth* simuliert. Bei realen Datensätze werden die Trainingsdaten tatsächlich durch Messung ermittelt. [3]
Für diese Arbeit soll eine Kombination aus selbst erstellten Datensätzen und öffentlich zur Verfügung stehenden Datensätzen aus dem Internet genutzt werden.
In Abschnitt 6.1 soll eine Auswahl aus bekannten quelloffenen Datensätzen vorgenommen werden. Da die quelloffenen Datensätze keine Sensordaten von OF- oder ToF-Sensoren enthalten, sollen diese theoretisch berechnet werden. Die Berechnung wird in Abschnitt 6.2 beschrieben.
In Abschnitt 6.3 wird das Vorgehen für die Aufzeichnung von eigenen Datensätzen vorgestellt.
Alle zur Verfügung stehenden Datensätze sollen in Trainings- und Testdatensätze aufgeteilt werden. In Tabelle 10.2 und Tabelle 10.1 sind die Datensätze aufgelistet, die für das Training verwendet werden. In Tabelle 10.4 und Tabelle 10.3 sind alle Datensätze, die für die Validierung verwendet werden.

6.1 Externe Trainingsdaten

Es existieren eine Reihe von frei verfügbaren Datensätzen, die aufgezeichnete Sensordaten einer IMU und die getrackte Position und Orientierung (*Ground Truth*) enthalten. In Tabelle 6.1 wird ein Überblick über bekannte Datensätze und verwendete Sensoren sowie *Ground Truth* Genauigkeit gegeben. Bei einigen der Datensätze sind neben Daten vom Drehraten- und Beschleunigungssensor (*Inertial*) auch Daten vom Kompass-Sensor (*Magnetometer*) oder Kameradaten (*Visual*) enthalten, die in dieser Arbeit jedoch nicht verwendet werden sollen.

In dieser Arbeit werden folgende Datensätze genutzt:

- Berlin Robust Orientation Estimation Assessment Dataset (BROAD) [27]
- OxIOD [9]
- European Robotics Challenge Micro Aerial Vehicle (EuRoC MAV) [5]

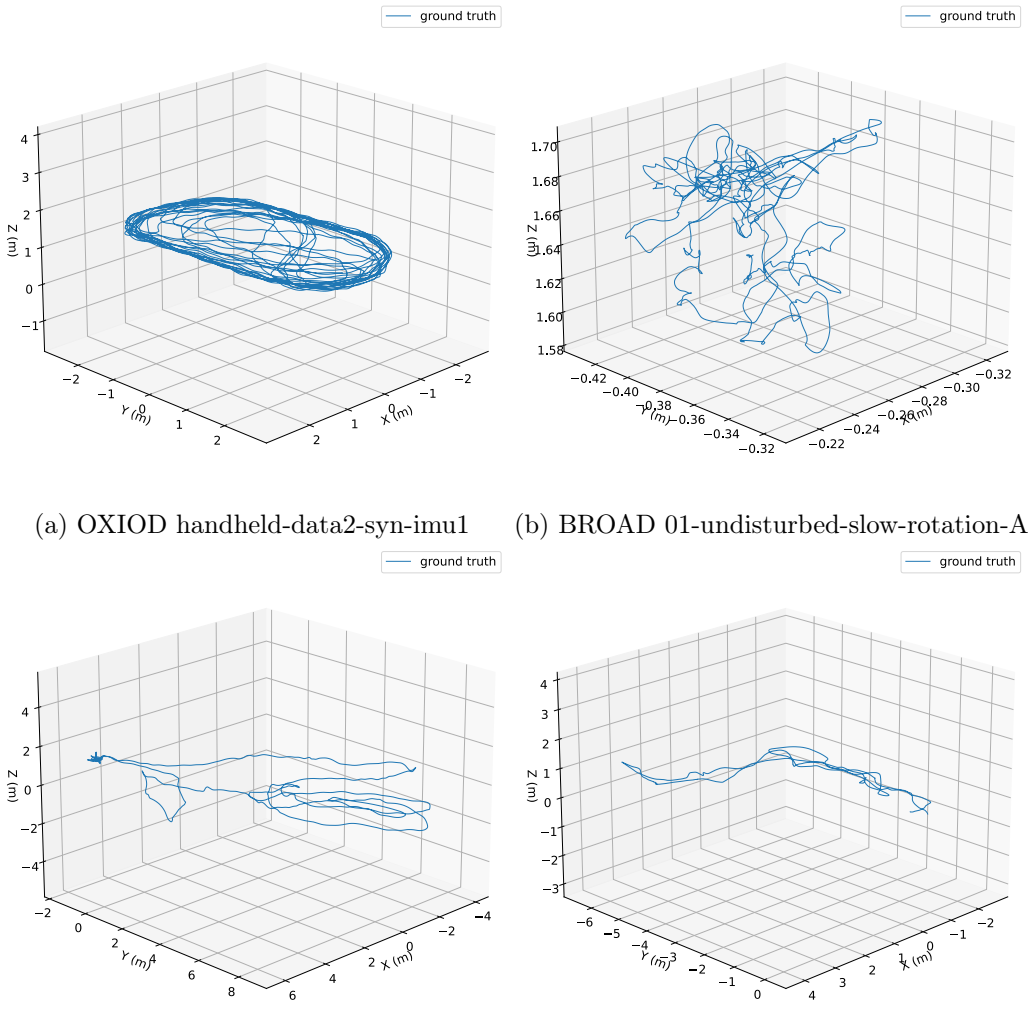
Die Datensätze BROAD und OxIOD zeichnen sich durch eine große Vielfalt unterschiedlicher Bewegungsformen und eine hohe Genauigkeit der *Ground Truth* von 0,5 mm bzw. 0,6 mm aus. Der EuRoC MAV enthält Messdaten eines Micro Air Vehicle (MAV) während des Flugs und ist damit optimal für den vorgesehenen Einsatzzweck einer Selbstlokalisation auf einer Miniaturdrohne geeignet. Durch die Kombination unterschiedlicher Datensätze soll eine breite Palette von unterschiedlichen Bewegungsformen und Varianzen in den Messdaten für das Training zur Verfügung stehen. In Abbildung 6.3 sind die Varianzen und Mittelwerte der Daten für Drehraten und Beschleunigungen gegenübergestellt.

Keiner der Datensätze enthält jedoch Daten eines OF- oder ToF-Sensors. Um die Datensätze dennoch für das Training zu verwenden, sollen OF- und ToF-Messungen synthetisch aus der *Ground Truth* berechnet werden (siehe Abschnitt 6.2).

In Abbildung 6.1 und Abbildung 6.2 sind die Trajektorien und der Verlauf der Beträge der Sensordaten von Beschleunigungs-, Drehraten- und Kompasssensor exemplarisch für jeweils einen Datensatz dargestellt.

Tabelle 6.1: Übersicht der Datensätze (v: visuell, i: inertial, m: magnetometer)

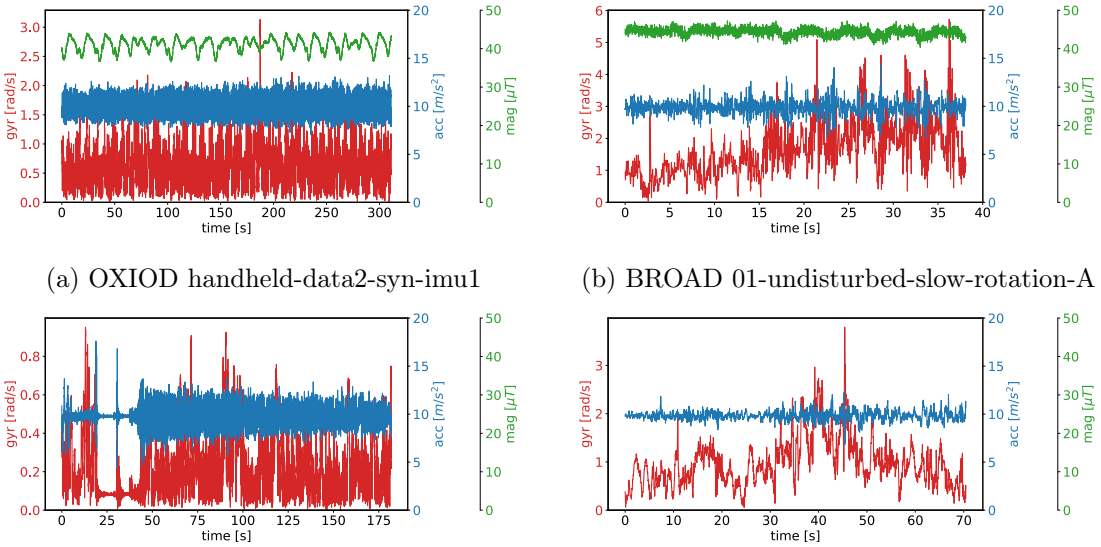
Datensatz	**Jahr**	**Sensor**	**[Hz]**	**v**	**i**	**m**	**GT präz.**	**Länge**
KITTI [14]	2013	OXTS RT 3003	10	x	x		10 cm	39,2 km
EuroC-MAV [5]	2016	ADIS16448	200	x	x		1 mm	0,9 km
RepoIMU [50]	2016	Xsens Mit	100		x		?	?
RIDI [57]	2017	Google Tango	200		x		?	100 min
TUM-VI [45]	2018	BMI160	200		x	x	1 mm	20 km
OxIOD [9]	2018	iPhone 7	100		x	x	0,5 mm	42 km
RoNIN [19]	2019	Android	200		x	x	?	42,7 h
BROAD [27]	2021	myon aktos-t	286		x	x	0,6 mm/0,2 °	141 min
Sassari [6]	2021	diverse	100		x	x	0,1 mm/0,5 °	2,8 min
PX4 Dataset	-	PX4	5		x	x	?	?



(a) OXIOD handheld-data2-syn-imu1 (b) BROAD 01-undisturbed-slow-rotation-A

(c) EuRoC MH-01-easy-mav0-imu0 (d) ARKit timeseries-1 (eigener Datensatz)

Abbildung 6.1: Beispiel Ground Truth Trajektorien aus unterschiedlichen Trainingsdatensätzen



(a) OXIOD handheld-data2-syn-imu1

(b) BROAD 01-undisturbed-slow-rotation-A

(c) EuRoC MH-01-easy-mav0-imu0

(d) ARKit timeseries-1 (eigener Datensatz)

Abbildung 6.2: Beispiel Sensordaten aus unterschiedlichen Trainingsdatensätzen

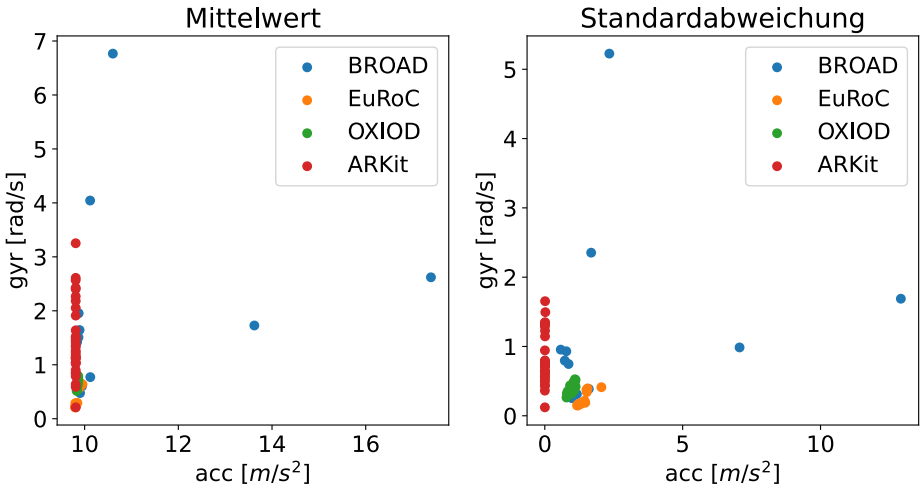


Abbildung 6.3: Standardabweichung und Mittelwert für Beschleunigung und Drehraten der Trainingsdatensätze

6.2 Sensordaten aus Trajektorie ermitteln

Wie im vorherigen Abschnitt beschrieben, stehen für die externen Datensätze keine Sensordaten für OF- und ToF-Sensor zur Verfügung. In diesem Abschnitt soll beschrieben werden, wie aus der bekannten Position und Orientierung (*Ground Truth*) theoretische Sensordaten für OF- und ToF-Sensor hergeleitet werden können.

6.2.1 ToF-Sensor

Der ToF-Sensor (VL53L0X) ist unten an der Drohne befestigt und misst mit Blickfeld nach unten orthogonal den Abstand zum nächstliegenden Objekt. Er hat einen maximalen Messbereich von bis zu 2 m. Zur Berechnung der theoretisch gemessenen Distanz d_{tof} des Sensors kann die in Unterabschnitt 2.2.5 vorgestellte Formel umgestellt werden. Um den realen Daten des ToF-Sensors bestmöglich zu entsprechen, soll bei einem theoretisch berechneten Abstand über 2 m eine Begrenzung auf den maximalen Arbeitsbereich des Sensors stattfinden. Außerdem soll bei einer Neigung der Drohne über 90° auch von einer Distanz von 2 m ausgegangen werden (siehe Gleichung 6.2).

$$\alpha = 2 \arccos \sqrt{q_{e,w}^2 + q_{e,z}^2} \tag{6.1}$$

$$d_{tof} = \begin{cases} \frac{p_z}{cos(\alpha)} & \alpha < 90° \\ 2.0m & \alpha \geq 90° \end{cases} \tag{6.2}$$

6.2.2 OF-Sensor

Für die Berechnung des gemittelten optischen Flusses soll die Formel aus Unterabschnitt 2.2.4 verwendet werden. Für die Ermittlung der Bewegungsgeschwindigkeiten (v_x, v_y) wird die Geschwindigkeit im Navigationskoordinatensystem unter Berücksichtigung der aktuellen Orientierung ins körpereigene Koordinatensystem überführt. Die gemittelten Bildflüsse (δ_x, δ_y) ergeben sich dann folgendermaßen:

$$\delta_x = -f(\frac{v_x}{d} + \omega_y) \tag{6.3}$$

$$\delta_y = -f(\frac{v_y}{d} + \omega_x) \tag{6.4}$$

6.2.3 Vergleich zu realen Messwerten

Die theoretischen Sensordaten weisen starke Peaks auf, die so in den realen Sensordaten nicht vorkommen würden. Um dem realen Verhalten bestmöglich zu entsprechen, werden die theoretischen Sensordaten geglättet. Hierfür wird ein *Butterworth* Filter verwendet. Das verwendete Filter wird mithilfe von *Forward-backward Filtering* zweifach auf die Sensordaten angewandt, hierdurch wird ein Zeitversatz (*Zero-Phase-Filter*), wie bei einem regulären kausalen *Low-Pass-Filter* vermieden. *Forward-backward Filtering* funktioniert nur, wenn die zukünftigen Messwerte schon bekannt sind, d.h. nicht in Echtzeit. Für die Vorbereitung der Trainingsdaten ist dies jedoch nutzbar. Die Parameter des Filters wurden durch Probieren ermittelt.

In Abbildung 6.4 wird exemplarisch für einen aufgezeichneten Datensatz (realer Datensatz, siehe Abschnitt 6.3) der Verlauf für die berechneten OF- und ToF-Daten mit den realen Daten gegenübergestellt. Es lässt sich erkennen, dass die theoretischen Sensordaten dem Verlauf der tatsächlichen Sensordaten grundsätzlich entsprechen. Die tatsächlich gemessenen OF-Sensordaten weisen jedoch mehr Unregelmäßigkeiten auf als die berechneten, geglätteten Werte. Es wird vermutet, dass die optische Bestimmung des optischen Flusses je nach Kontrast des Untergrundes zu Unregelmäßigkeiten führt.

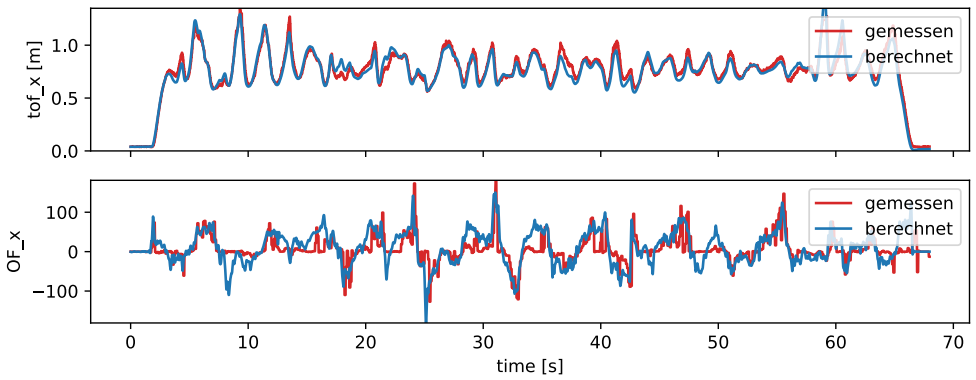


Abbildung 6.4: Vergleich berechnete und gemessene Sensordaten

6.3 Eigene Trainingsdaten

Für das Trainings der KNNs sollen auch eigene Trainingsdatensätze mithilfe der Miniaturdrohne aufgezeichnet werden. Herausforderung ist hierbei die Erstellung einer präzisen *Ground Truth* zu den Sensordaten. Bei den öffentlich zur Verfügung stehenden Datensätzen werden meist präzise Kamerasysteme (z.B. Vicon) oder 3D Laserscanner verwendet. Für die Erstellung der *Ground Truth* soll in dieser Arbeit ein iPhone 15 Pro mit Augmented Reality Kit (ARKit) [2] verwendet werden. ARKit ist ein Framework von Apple. Dieses wurde im Jahre 2017 veröffentlicht und kann auf dem iPhone oder iPad genutzt werden. ARKit nutzt VIO, um die *Pose* im Raum festzustellen. Hierbei kann z.B. die relative Pose der Kamera gegenüber der Startposition getrackt werden. Bei iPhone Pro Modellen ab Generation 12 wird auch das integrierte Light Detection and Ranging (LiDAR)-Modul für eine präzise Oberflächenbestimmung verwendet. Dies verbessert die *Pose*-Schätzung weiter. ARKit wird auch in [12] zur Erstellung einer *Ground Truth* verwendet. Generell ist die Nutzung von VIO auf Smartphones für die Erstellung einer *Ground Truth* in vielen aktuellen Arbeiten zu finden ([19][57][58]).
Mithilfe eines 3D-Druckers wurde eine Halterung gedruckt, um das iPhone statisch mit der Drohne zu verbinden (siehe Abbildung 6.5).

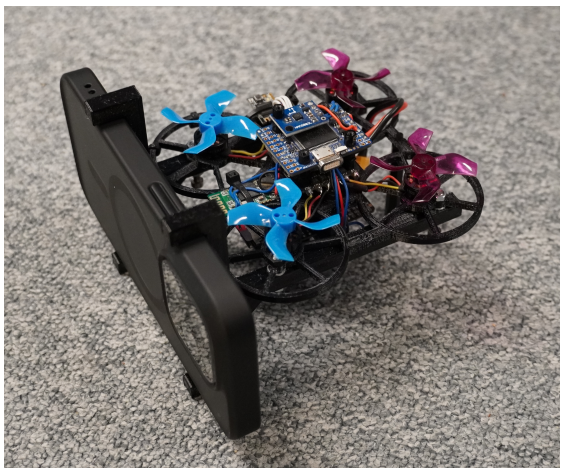


Abbildung 6.5: statische Verbindung iPhone und Miniaturdrohne

Es werden einige Sequenzen im Indoor-Bereich aufgezeichnet, indem die Drohne händisch durch den Raum getragen wird und währenddessen Rotationen durchgeführt werden. Die Rotoren sind hierbei ausgeschaltet. Das iPhone loggt die *Ground Truth* und die Miniaturdrohne loggt parallel die aktuellen Sensordaten von IMU-, OF- und ToF-Sensor. Das

iPhone zeichnet parallel auch IMU-Sensordaten auf, diese sollen nicht für das Training verwendet werden, jedoch um die Messdaten von Drohne und iPhone später exakt übereinander zu legen. (siehe Unterabschnitt 6.3.1) Für die Aufzeichnung der *Pose* auf dem iPhone wird der ARKit Data Logger [39] verwendet und minimal modifiziert. Dieser schreibt die aktuelle *Pose* mit Zeitstempel in eine csv-Datei. Die maximale Frequenz für die *Pose*-Bestimmung liegt bei 60 Hz. Für die Aufzeichnung der Sensordaten der IMU auf der Drohne wurde ein C-Programm entwickelt, welches mit einer Frequenz von 200 Hz die Sensordaten aufzeichnet.

Bei den aufgezeichneten Daten ist zu berücksichtigen, dass sich diese von einer tatsächlichen Trajektorie im Flug unterscheiden. Da die Rotoren ausgeschaltet sind, fallen Vibrationen weg. Außerdem kann die Rotations- und Bewegungsgeschwindigkeit von der tatsächlichen Bewegung während eines Flugvorgangs abweichen.

In Abbildung 6.1d und Abbildung 6.2d sind die Trajektorien und Sequenzen der Sensordaten von Beschleunigungs- und Drehratensensor exemplarisch für jeweils einen aufgezeichneten Datensatz dargestellt.

6.3.1 Zeitliche Angleichung

Die Drohne und das iPhone zeichnen die Messdaten unabhängig voneinander auf, d.h. es gibt keine technische zeitliche Synchronisation. Die Sensordaten müssen daher nachträglich exakt mit der *Ground Truth* übereinander gelegt werden. Da das iPhone auch über eine IMU verfügt, zeichnet dieses neben der *Pose* auch Sensordaten auf. Die Drehraten und Beschleunigungsmesswerte der iPhone IMU sollten sich nahezu identisch zu den Sensordaten der Drohne verhalten. Diese Eigenschaft kann genutzt werden, um die zeitliche Verschiebung τ zu ermitteln. Hierfür wird die Kreuzkorrelation der beiden Kurven

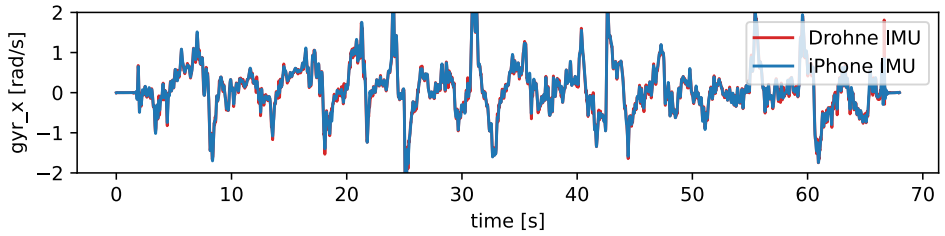


Abbildung 6.6: Sensordaten von iPhone und IMU übereinander gelegt

ermittelt und für die maximale Korrelation die Zeitverschiebung τ abgelesen. Für jede

aufgezeichnete Trajektorie wird eine individuelle Zeitverschiebung ermittelt. So werden in einer Nachverarbeitung die Sensordaten exakt zur *Ground Truth* zeitlich ausgerichtet. In Abbildung 6.6 sind die zeitlich ausgerichteten Daten von Drohne und iPhone zu sehen.

6.4 Angleichung der Datensätze

6.4.1 Koordinatensysteme

Damit ein gemeinsames Training und die Validierung mit Datensätzen aus unterschiedlichen Quellen überhaupt möglich wird, müssen die Orientierungen der Koordinatensysteme der unterschiedlichen Datensätze aneinander angeglichen werden. Die Sensordaten werden ebenso entsprechend angeglichen, sodass diese zur *Ground Truth* konsistent sind.

6.4.2 Resampling

Die Trainingsdaten aus den externen Quellen (Abschnitt 6.1) und die aufgezeichneten Datensätze (Abschnitt 6.3) liegen in unterschiedlichen Samplingraten vor. Bei einem konventionellen INS-Algorithmus geht die Zeitdifferenz dt zweier Messungen direkt in die Berechnung des nächsten Iterationsschrittes mit ein. Bei einem KNN wäre ein ähnlicher Ansatz denkbar, bei dem das neuronale Netz schon während des Trainings lernen muss, mit einer Zeitinformation als Eingangsgröße umzugehen. Hierbei handelt es sich um einen Time-Aware (TA)-Ansatz. Dieser setzt voraus, dass schon beim Training Trainingsdaten mit vielen unterschiedlichen Frequenzen zur Verfügung stehen, damit das Netz den Umgang mit dem Eingangssignal lernen kann. In [54] wird ermittelt, dass für einen TA-Ansatz Sequenzen mit über 100 unterschiedlichen Frequenzen benötigt werden, um eine gute Generalisierung für den kompletten Frequenzbereich zu erreichen.

Eine andere Möglichkeit ist das Resampling der Trainingsdaten auf eine einheitliche Frequenz. Wie in Abschnitt 4.1 dargelegt, soll auf der Flugsteuerung von einer maximalen Loop-Time von 5 ms ausgegangen werden, was einer Frequenz von 200 Hz entspricht. Für das Training sollen daher alle Trainingsdaten auf 200 Hz resampelt werden. Hierbei werden aus den bestehenden Messdaten neue Punkte interpoliert. Für den Positionsvektor

und die Daten des Beschleunigungssensors, Drehratensensors, OF und ToF-Daten kann eine lineare Interpolation stattfinden.
Für die Orientierung funktioniert eine lineare Interpolation nicht. Hierfür soll ein Spherical Linear Interpolation (SLERP) Algorithmus verwendet werden. Dieser Algorithmus findet Anwendung in der Computergrafik zur Animation von flüssigen 3D Rotationen. Ziel es dabei eine gleichmäßige Rotation von einem Quaternion A in ein Quaternion B zu ermöglichen. Es wird zunächst eine Rotationsachse bestimmt, um die kürzeste Rotation von Quaternion A zu B zu ermöglichen. Über einen Faktor t lässt sich jeder Zwischenwinkel zwischen den Quaternionen abfahren. Dies ermöglicht eine lineare Interpolation bezogen auf den Winkel.

6.5 Standardisierung

Die Sensordaten haben unterschiedliche Wertebereiche, dies kann sich negativ auf das Training auswirken. So hat z.B. die z-Achse des Beschleunigungssensors aufgrund der Erdbeschleunigung in einer horizontalen nicht beschleunigten Fluglage einen Wert von 9,81 m/s^2. Die Daten des Drehratensensors und Daten des Beschleunigungssensor haben ebenso andere Größenordnungen. Um das Training zu stabilisieren, sollen die Trainingsdaten standardisiert werden. Hierfür wird jede Eingangsgröße über alle Datensätze auf einen Mittelwert μ von 0 und eine Standardabweichung σ von 1 standardisiert. Es wird für das Eingangsfeature X über alle Sequenzen der Mittelwert μ_x und die Standardabweichung σ_x bestimmt. Die Standardisierung jedes Featurewertes x_i kann dann mit Gleichung 6.5 erfolgen. Der neue standardisierte Wert ist dann x'_i.

$$x'_i = \frac{x_i - \mu_X}{\sigma_X} \tag{6.5}$$

6.6 Data Augmentation

Mithilfe von *Data Augmentation* sollen die Daten durch domänenspezifische Informationen variiert werden, um mehr Trainingsdaten zu erhalten und eine bessere Generalisierung des neuronalen Netzes zu erreichen. Hierdurch wird auch die Gefahr für Overfitting verringert. In der Arbeit von [54] konnte eine signifikante Verbesserung der Orientierungsschätzung durch *Data Augmentation* erreicht werden. In dieser Arbeit sollen zwei

Möglichkeiten von *Data Augmentation* angewandt werden, die im Folgenden erläutert werden.

6.6.1 Zufällige Rotation

Bei der zufälligen Rotation werden die Sensordaten und *Ground Truth* durch zufällige Rotationen variiert. Hierbei wird die Orientierung q_{ori} der Drohne im körpereigenen Koordinatensystem um das zufällige Quaternion Δq_{rnd} rotiert (siehe Gleichung 6.6). Der Beschleunigungsvektor $\vec{a}$ und Drehratenvektor $\vec{\omega}$ liegen im Sensorkoordinatensystem vor und müssen um die verdrehte Orientierung Δq_{rnd} korrigiert werden. Es wird demnach mithilfe der konjugierten Quaternion Δq^*_{rnd} (d.h. Rotation um die gleiche Achse in entgegengesetzte Richtung) die Konsistenz zur Ground Truth wiederhergestellt (siehe Gleichung 6.7 und Gleichung 6.8). Die Positionsvektoren der *Ground Truth* bleiben unverändert, d.h. die Trajektorie verändert sich dabei nicht.

$$q'_{ori} = q_{ori} \otimes \Delta q_{rnd} \tag{6.6}$$

$$\vec{\omega}' = R(\Delta q^*_{rnd}) \cdot \vec{\omega} \tag{6.7}$$

$$\vec{a}' = R(\Delta q^*_{rnd}) \cdot \vec{a} \tag{6.8}$$

6.6.2 Rauschen und Bias

Wie in Abschnitt 2.3 beschrieben, sind Messdaten von Sensoren im Allgemeinen fehlerbehaftet. Für ein robusteres Training soll ein zusätzliches Rauschen und ein Bias simuliert werden. Hierbei werden ein normalverteiltes Rauschen mit einer Standardabweichung und ein Bias für die Drehraten zufällig erzeugt und auf die Sensorsignale addiert. In dieser Arbeit sollen für die späteren Versuche zwei unterschiedliche Stärken für das Rauschen unterschieden werden. In der Tabelle 6.2 sind die verwendeten Parameter für das Rauschen dargestellt. Hierbei wird der Bias als Intervall definiert, in dessen Grenzen ein Bias-Wert zufällig ausgewählt wird. Die Stärke des Rauschens für die Standardabweichung σ definiert. In Abbildung 6.7 ist das addierte Rauschen exemplarisch für die x-Achse der Drehraten dargestellt.

Tabelle 6.2: Rauschen

	Acc Bias $[m/s^2]$	**Gyro Bias** $[rad/s]$	**Acc Rauschen** $[rad/s]$	**Gyro Rauschen** $[rad/s]$
starkes Rauschen	[-0,25;0,25]	[-0,1;0,1]	$\sigma = 0{,}5$	$\sigma = 0{,}05$
schwaches Rauschen	[-0,025;0,025]	[-0,01;0,01]	$\sigma = 0{,}05$	$\sigma = 0{,}005$

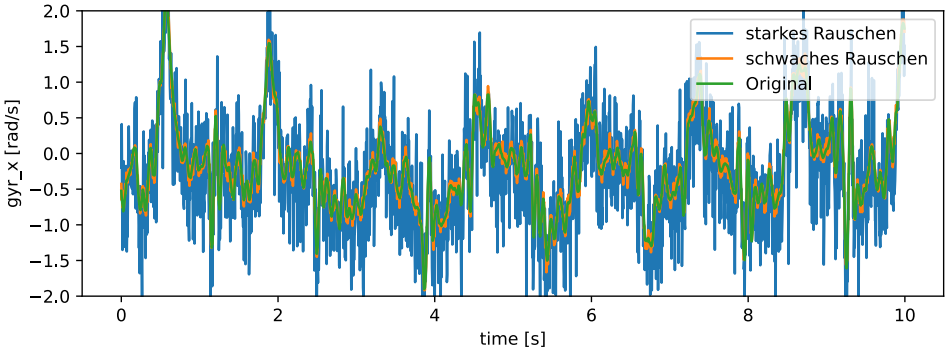


Abbildung 6.7: Rauschen

6.7 Sliding Windows

Um ein RNN-Netz zu trainieren, werden Sequenzen mit festgelegter Länge benötigt. Um möglichst viele Trainingsdaten zu genieren, wird oftmals die *Sliding Window*-Technik verwendet. Hierbei fährt bildlich vorgestellt ein Fenster mit einer Länge von z.B. 200 Zeiteinheiten über einen Datensatz und schneidet zusammenhängende Sequenzen heraus. So werden aus einem Datensatz überlappende Sequenzen generiert. Durch einen *stride*-Parameter kann die Schrittweite definiert werden, hiermit kann die tatsächliche Überlappung vergrößert bzw. verkleinert werden. Generell können RNN-Netze nicht mit unendlich langen Sequenzen trainiert werden. Bei sehr langen Sequenzen kann das schon in Unterabschnitt 2.5.1 vorgestellte *Exploding Gradient Problem* auftreten. Um ein *Stateful*-Netz dennoch mit langen Sequenzen trainieren zu können, kann Truncated Backpropagation Through Time (TBPTT) verwendet werden. Dies wird im Folgenden erklärt.

6.7.1 Truncated Backpropagation Through Time

Truncated Backpropagation Through Time (TBPTT) ist eine Technik, um ein neuronales Netz mit sehr langen Sequenzen zu trainieren. Das Verfahren funktioniert jedoch nur bei *Stateful*-RNNs, die einen Zustand über mehrere Sequenzen beim Training erhalten können. Zur Anwendung des TBPTT werden lange Sequenzen in mehrere kleine Teilsequenzen zerlegt (siehe Abbildung 6.8). So kann eine Sequenz mit einer Länge von 600 Zeitschritten z.B. in 3 Teilsequenzen mit jeweils 200 Zeitschritten zerlegt werden. Die 3 Teilsequenzen werden dann auf drei *Batches* aufgeteilt. Nach jedem Batch werden die Gewichte des Netzes durch die *Backpropagation* geupdatet. Um ein paralleles Trainieren zu ermöglichen, können pro Batch auch mehrere Sequenzen trainiert werden. Bei einer gewählten *Batch-Size* von 32, werden z.B. 32 Sequenzen parallel trainiert. Nach 3 *Batches* werden die Zustände der RNN-Layer zurückgesetzt und es beginnen die nächsten 32 Sequenzen. Natürlich können so auch noch längere Sequenzen trainiert werden, dann würden die Teilsequenzen auf zusätzliche *Batches* aufgeteilt werden.

6.8 Erstellung von Trainingsdaten

In den vorherigen Abschnitten wurden die notwendigen Teilschritte für die Vorverarbeitung der Daten im Einzelnen beschrieben. Nun soll die gesamte Datenpipeline mit

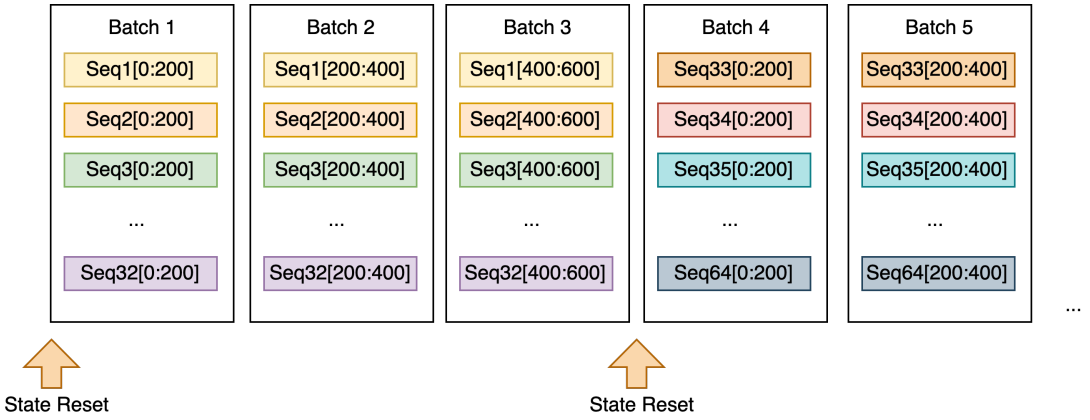


Abbildung 6.8: Truncated Backpropagation Through Time (TBPTT)

allen Teilschritten vorgestellt werden (siehe Abbildung 6.9). Die Schritte sind hierbei in folgender Reihenfolge:

- Die Daten werden auf eine einheitliche Frequenz resampelt (Details siehe Unterabschnitt 6.4.2).
- Die Koordinatensysteme für die unterschiedlichen Quelldaten werden angeglichen (Details siehe Unterabschnitt 6.4.1).
- Die Ausgangsgrößen werden aus der *Ground Truth* berechnet (Details siehe Abschnitt 4.4).
- Es werden OF- und ToF-Daten für alle externen Datensätze synthetisch berechnet (Details siehe Unterabschnitt 6.2.2).
- Es werden überlappende Sequenzen (engl. *Sliding Windows*) mit entsprechender Länge erstellt (siehe Abschnitt 6.7).
- Es kann optional ein Sensorrauschen für die Gyro- und Beschleunigungssensordaten erzeugt werden (Details siehe Abschnitt 6.6).
- Es kann optional eine zufällige Rotation der *Ground Truth* erzeugt werden, um die Sensordaten zu variieren (Details siehe Abschnitt 6.6).
- Die Reihenfolge der Sequenzen wird zufällig geändert.

- Die Daten werden pro Kanal standardisiert (siehe Abschnitt 6.5). Die Parameter werden für jeden Kanal separat abgespeichert und müssen bei der Nutzung des neuronalen Netzes angewandt werden.

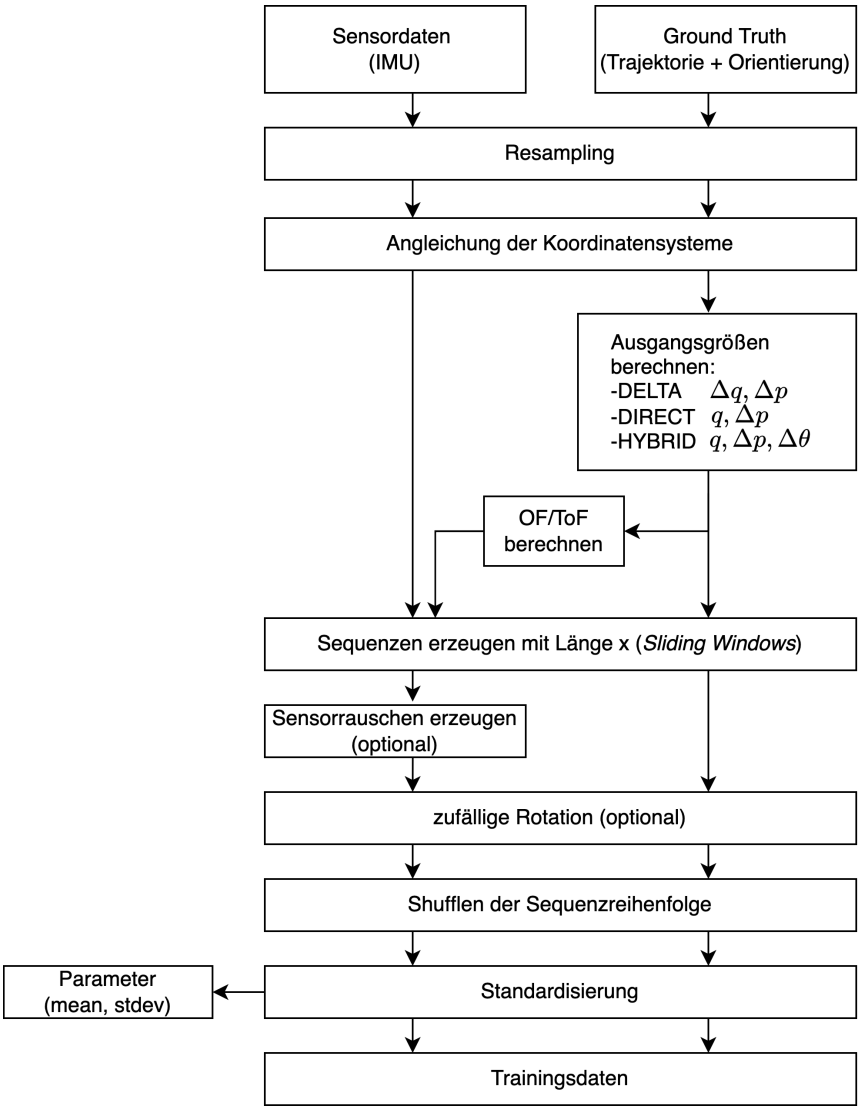


Abbildung 6.9: Datenpipeline (Erstellung von Trainingsdaten)

7 Versuche

In diesem Abschnitt sollen Versuche mit Variationen der Netzmodelle durchgeführt werden. Hierbei werden Hyperparameter und Eigenschaften des Netzes verändert, um deren Auswirkungen auf die *Pose*-Schätzungen zu erproben und das beste Netzmodell zu ermitteln. Nach einer Beschreibung der Versuche in Abschnitt 7.1, werden in Abschnitt 7.2 Kriterien zur Bewertung festgelegt, um die Ergebnisse der Versuche sodann gegenüberzustellen (Abschnitt 7.3).

7.1 Versuchsbeschreibung

Wenn möglich soll für jeden Versuch immer nur eine Änderung an einem Parameter (Hyperparameter oder Eigenschaft des Netzes) zwischen den Varianten vollzogen werden, um die Änderungen auf das Ergebnis besser nachvollziehen zu können. Parameter, die sich als sinnvoll gezeigt haben, sollen dann soweit möglich in den darauf folgenden Versuchen verwendet werden.

Alle neuronalen Netze werden für 100 Epochen mit dem Adam-Optimierer mit einer Lernrate von 0,0001 trainiert. Es wird eine *Batch Size* von 32 genutzt. Während des Trainingsvorgang wird immer das Modell mit dem besten *Validation Loss* gespeichert und für Schätzungen der *Pose* für alle Testdatensätze verwendet. Eine Übersicht der genutzten Testdatensätze ist in Tabelle 10.4 und Tabelle 10.3 dargestellt.

Insgesamt sollen sieben Versuche durchgeführt werden, um für die beiden Netzvarianten (TCN und GRU) die optimalen Parameter zu ermitteln. Die genaue Variation der Parameter bzw. Modelleigenschaft wird im jeweiligen Versuch detailliert beschrieben.

- **Versuch 1**: Repräsentation der Orientierung
- **Versuch 2**: Variation der Loss-Funktion

- **Versuch 3**: Variation der Neuronenanzahl
- **Versuch 4**: Variation der Sequenzlänge
- **Versuch 5**: Nutzung von Augmentation (Rauschen)
- **Versuch 6**: Nutzung von Augmentation (Rotationen)
- **Versuch 7**: Nutzung von OF- und ToF-Sensordaten

7.2 Bewertung der Performance

Nach dem durchgeführten Training soll die Qualität der Netzvarianten bewertet werden. Die berechnete Pose von den KNNs und den Referenz-Filtern wird mit der *Ground Truth* verglichen. Hierfür sollen in diesem Abschnitt Fehlermetriken zur Bewertung festgelegt werden. Diese Fehlermetriken ähneln den *Loss*-Funktionen in Abschnitt 4.6, sollen aber eine direkte Einschätzung der Performance zulassen.

7.2.1 Fehlermaß

Orientierungsfehler

Der Orientierungsfehler stellt den Fehler zwischen geschätzter Orientierung $\hat{q}$ und tatsächlicher Orientierung q dar. In dieser Arbeit soll der Orientierungsfehler in *Heading*- und *Attitude*-Fehler unterteilt werden, um die Performance davon unabhängig voneinander bewerten zu können. Es wird zunächst die Delta-Quaternion Δq_e zwischen Schätzung und *Ground Truth* berechnet [53]:

$$\Delta q_e = \hat{q} \otimes q^* \tag{7.1}$$

Der *Heading*-Fehler e_h berechnet sich Folgendermaßen [3][27]:

$$e_h = 2 \arctan \left| \frac{\Delta q_{e,z}}{\Delta q_{e,w}} \right| \tag{7.2}$$

Der *Attitude*-Fehler e_a berechnet sich Folgendermaßen [53][3][27]:

$$e_a = 2 \arccos \sqrt{\Delta q_{e,w}^2 + \Delta q_{e,z}^2} \tag{7.3}$$

Positionsfehler
Es wird der Fehler zwischen der tatsächlichen relativen Position p (*Ground Truth*) und der geschätzten relativen Position $\hat{p}$ als Betrag ermittelt:

$$e_p = \|p - \hat{p}\| \tag{7.4}$$

7.2.2 Root Mean Square Error (RMSE)

Für den Vergleich soll der RMSE für alle Datensätze berechnet werden (siehe Gleichung 7.5). Dies bedeutet konkret, dass *Attitude*-, *Heading*- und Position-Fehler für alle Zeitschritte aller Datensatzes quadratisch gemittelt werden. Der RMSE bezieht sich daher immer auf die komplette Zeitdauer aller Testdatensätze.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_i^2} \tag{7.5}$$

Für den *Heading*- und Position-Fehler ist systembedingt mit einer zeitlichen Zunahme des Fehlers auszugehen, da *Heading* und Position nicht absolut bestimmt werden können. Demnach dient der RMSE hier nur als Vergleichsmetrik zwischen den Filtern und neuronalen Netzen. Der Wert gibt keine Auskunft über das Zeitverhalten des Fehlers.

7.3 Versuchsdurchführung

7.3.1 Versuch 1: Repräsentation der Orientierung

Im ersten Versuch soll die bestmögliche Form der Repräsentation der Orientierung ermittelt werden. Die Ansätze für die Ausgangsgrößen wurden schon in Abschnitt 4.4 beschrieben. Es sollen drei verschiedene Ansätze verglichen werden:

- **DELTA**: Die Änderung der Orientierung wird pro Iterationsschritt als Quaternion geschätzt. Die Startorientierung muss bekannt sein.
- **DIRECT**: Die Orientierung wird direkt als Quaternion geschätzt.
- **HYBRID**: Die *Attitude* wird direkt geschätzt. Die *Heading* wird als Änderung geschätzt.

Um die Ergebnisse später vergleichbar zu machen, wird für die Varianten DELTA und HYBRID aus den ermittelten Änderungen iterativ eine absolute Orientierung nach der Prädiktion berechnet. Ergänzend wird für alle Varianten eine Trajektorie mit relativer Position gegenüber dem Startpunkt aus den Positionsänderungen hergeleitet. Die berechnete Trajektorie und Orientierung ermöglicht dann einen Vergleich zur *Ground Truth*.

Folgende weitere Parameter werden angenommen:

- Neuronenanzahl pro Layer: 100
- Sequenzlänge: 200
- Loss: Loss-Funktion 2 (siehe Unterabschnitt 7.3.2)
- Augmentation: kein

Tabelle 7.1: Ergebnis Versuch 1 - Repräsentation der Orientierung (Netzvariante 1 - GRU)

ID	**Ansatz**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU_100_200_DELTA	Delta	**-15,2**	92,72	59,73	11,99
GRU_100_200_DIRECT	Direct	-9,96	98,20	98,12	14,25
GRU_100_200_HYBRID	Hybrid	-15,09	**90,63**	**8,47**	**8,16**

Tabelle 7.2: Ergebnis Versuch 1 - Repräsentation der Orientierung (Netzvariante 2 - TCN)

ID	**Ansatz**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
TCN_100_200_DELTA	Delta	**-15,7**	99,47	74,62	6,17
TCN_100_200_DIRECT	Direct	-11,87	100,68	96,32	12,66
TCN_100_200_HYBRID	Hybrid	**-17,29**	**85,23**	**4,16**	**4,19**

Ergebnis:
Bei beiden Netzvarianten (GRU und TCN) zeigt der hybride Ansatz die geringsten Fehler für *Heading*, *Attitude* und *Position*. Insgesamt ist der *Heading*-Fehler mit über 90 ° jedoch bei allen Ansätzen recht hoch.

Bewertung
Die *Heading* lässt sich nur durch die verwendeten Sensoren (Gyroskop und Beschleunigungssensor) nicht absolut ermitteln. Mithilfe der Drehraten des Gyroskops lassen sich nur Änderungen der Orientierung ermitteln. Aufgrund der inhärenten Sensorfehler driftet die *Heading* jedoch. Dies spiegelt sich in den hohen Fehlern wieder. Die *Attitude* kann absolut ermittelt werden, da die Erdbeschleunigung als nahezu konstante Beschleunigung (bei gleicher Höhe über Grund) eine Neigungsermittlung ermöglicht. Demnach zeigt der hybride Ansatz hier die besten Ergebnisse und soll für alle weiteren Versuche verwendet werden.

7.3.2 Versuch 2: Variation der Loss-Funktion

In Abschnitt 4.6 wurde beschrieben, dass die *Loss*-Funktion für die *Attitude* aufgrund des arccos-Terms zu einem instabilen Training für Werte nahe [-1,1] führen kann. Es soll hier demnach eine alternative Loss-Funktion für die *Attitude*-Bestimmung erprobt werden, die nicht-trigonometrisch ist.

Die potentiellen *Loss*-Funktionen wurden bereits in Abschnitt 4.6 näher erläutert:

- **Loss-Funktion 1**: $e_a = 2 \arccos\left(\sqrt{\Delta q_w^2 + \Delta q_z^2}\right)$
- **Loss-Funktion 2**: $e_{a,safe} = 1 - \sqrt{\Delta q_w^2 + \Delta q_z^2}$

Für die Positionsänderung wird der e_{DPMAE} verwendet:

$$e_{DPMAE} = \|\Delta\hat{p} - \Delta p\|_1 \tag{7.6}$$

Für die *Heading* wird aufgrund des hybriden Ansatzes folgende Funktion genutzt:

$$e_\theta = |\Delta\hat{\theta} - \Delta\theta| \tag{7.7}$$

Folgende weitere Parameter werden angenommen:

- Neuronenanzahl pro Layer: 100
- Sequenzlänge: 200
- Repräsentation der Orientierung: HYBRID (siehe Unterabschnitt 7.3.1)
- Augmentation: kein

Tabelle 7.3: Ergebnis Versuch 2 - Variation der Loss-Funktion (Netzvariante 1 - GRU)

ID	**Loss-funktion**	**best val loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU_100_200_ACOS	1	-11,51	**85,16**	10,11	10,12
GRU_100_200_SAFE	2	**-15,09**	90,63	**8,47**	**8,16**

Ergebnis:
Bei der Netzvariante 1 (GRU) zeigt die *Loss*-Funktion 2 für *Attitude* und *Position* die

Tabelle 7.4: Ergebnis Versuch 2 - Variation der Loss-Funktion (Netzvariante 2 - TCN)

ID	**Loss-funktion**	**best val loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
TCN_100_200_ACOS	1	-12.89	94,97	**4,15**	4,31
TCN_100_200_SAFE	2	**-17.29**	**85,23**	4,16	**4,19**

geringsten Fehler. Lediglich der Fehler für *Heading* ist bei *Loss*-Funktion 1 geringer, aber dennoch sehr hoch.
Bei der Netzvariante 2 (TCN) zeigt die Nutzung der Loss-Funktion 2 den geringeren Fehler für *Heading* und *Position*, lediglich der Fehler für *Attitude* fällt geringfügig schlechter aus.

Bewertung
Die *Loss*-Funktion 2 und dessen Ableitung ist für Werte bei 1 numerisch stabil. Daher scheint das Netz bessere Ergebnisse für die *Loss*-Funktion 2 zu erreichen. Die *Loss*-Funktion 2 wird daher für alle weiteren Versuche für beide Netzvarianten genutzt.

7.3.3 Versuch 3: Variation der Neuronenanzahl

Im dritten Versuch soll die Auswirkung der Neuronenanzahl pro Layer auf die Performance des Netzes analysiert werden. Es wird eine Neuronenanzahl von 50, 100 und 200 für beide Netzvarianten getestet.

Folgende weitere Parameter werden angenommen:

- Sequenzlänge: 200
- Loss: Loss-Funktion 2 (siehe Unterabschnitt 7.3.2)
- Repräsentation der Orientierung: HYBRID (siehe Unterabschnitt 7.3.1)
- Augmentation: kein

Tabelle 7.5: Ergebnis Versuch 3 - Variation der Neuronenanzahl (Netzvariante 1 - GRU)

ID	Neuronen pro Layer	Validation Loss	RMSE Heading [°]	RMSE Attitude [°]	RMSE Position [m]
GRU_50_200	50	-15,07	**79,53**	8,52	13,03
GRU_100_200	100	-15,09	90,63	**8,47**	**8,16**
GRU_200_200	200	**-15,14**	85,07	9,46	11,76

Tabelle 7.6: Ergebnis Versuch 3 - Variation der Neuronenanzahl (Netzvariante 2 - TCN)

ID	Neuronen pro Layer	Validation Loss	RMSE Heading [°]	RMSE Attitude [°]	RMSE Position [m]
TCN_50_200	50	-16,94	**79,53**	8,52	13,03
TCN_100_200	100	**-17,29**	85,23	**4,16**	**4,19**
TCN_200_200	200	-16,95	89,48	4,24	4,24

Ergebnis

Für die Netzvariante 1 (GRU) zeigt sich, dass der geringste Fehler für die *Attitude* und Position bei 100 Neuronen pro Layer erreicht wird. Der Fehler für *Heading* ist bei allen Netzvarianten sehr hoch.

Für die Netzvariante 2 (TCN) zeigt sich, dass ebenso der kleinste Fehler für *Attitude* und Position bei 100 Neuronen pro Layer erreicht wird. Der Fehler für *Heading* ist auch hier

bei allen Netzvarianten sehr hoch.

Bewertung

Eine höhere Neuronenanzahl erhöht die Kapazität des Netzes, Zusammenhänge abzubilden. Es kann jedoch auch zu *Overfitting* führen. Eine Neuronenanzahl von 100 scheint hierbei ein guter Kompromiss zu sein. Für alle weiteren Versuche soll eine Neuronenanzahl von 100 genutzt werden.

7.3.4 Versuch 4: Variation der Sequenzlänge

Im vierten Versuch sollen die Auswirkungen der Sequenzlänge erprobt werden. (Hierbei muss berücksichtigt werden, dass sich die angegebene Sequenzlänge bei der GRU-Netzvariante nur auf den Trainingsvorgang bezieht. Während der Ausführung arbeitet das Netz im Modus *Stateful*, d.h. nach einem neuen Eingangssample (Sequenzlänge = 1) erfolgt eine neue Schätzung für alle Ausgangswerte. Bei der TCN-Netzvariante wird eine Sequenz mit der angegebenen Länge auch für die Ausführung genutzt. Demnach hat die Sequenzlänge bei der TCN-Variante direkte Auswirkungen auf die Ausführungszeiten. Bei der GRU-Variante ist die Ausführungszeit davon unabhängig.) Die Erstellung von Trainingsdaten (Sequenzen) wurde bereits in Abschnitt 6.8 näher beschrieben. Es soll eine Variation der Sequenzlänge von 400 bis 800 getestet werden.
Folgende weitere Parameter werden angenommen:

- Neuronenanzahl: 100
- Loss: Loss-Funktion 2 (siehe Unterabschnitt 7.3.2)
- Repräsentation der Orientierung: HYBRID (siehe Unterabschnitt 7.3.1)
- Augmentation: kein

Tabelle 7.7: Ergebnis Versuch 4 - Variation der Sequenzlänge (Netzvariante 1 - GRU)

ID	**Sequenzlänge**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU_100_100	100	-15,10	94,85	11,32	10,05
GRU_100_200	200	-15,09	**90,63**	8,47	**8,16**
GRU_100_400	400	**-15,14**	103,72	**8,29**	12,66
GRU_100_800	800	-15,01	94,70	9,4	8,87

Tabelle 7.8: Ergebnis Versuch 4 - Variation der Sequenzlänge (Netzvariante 2 - TCN)

ID	**Sequenzlänge**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
TCN_100_100	100	-16,72	**85,01**	4,66	3,72
TCN_100_200	200	**-17.29**	85,23	4,16	4,19
TCN_100_400	400	-16.79	85,29	**4,00**	**3,30**

Ergebnis:
Für die Netzvariante 1 (GRU) zeigt sich, dass eine Sequenzlänge von 400 zum geringsten Fehler für die *Attitude* führt. Der Fehler für *Heading* und Position ist jedoch bei einer Sequenzlänge von 200 am kleinsten.
Für die Netzvariante 2 (TCN) ist der Fehler für *Attitude* bei einer Sequenzlänge von 400 am kleinsten. Der Fehler für *Heading* ist jedoch bei einer Sequenzlänge von 100 am kleinsten.

Bewertung:
Die Sequenz der Sensordaten ermöglicht es dem neuronalen Netz, Zusammenhänge aus den Daten zu extrahieren. Bei einer zu kurzen Sequenzlänge können übergreifende Zusammenhänge schlecht erkannt werden. Bei einer zu langen Sequenz kann das *Exploding Gradient Problem* dazu führen, dass der Lernvorgang nicht konvergiert. Eine Sequenzlänge von 800 führt zu einem schlechteren Ergebnis. Eine Sequenzlänge von 200 scheint einen optimalen Kompromiss darzustellen und soll in allen weiteren Versuchen verwendet werden.

7.3.5 Versuch 5: Nutzung von Augmentation (Rauschen)

Im fünften Versuch soll getestet werden, ob die Nutzung von *Data Augmentation* die Ergebnisse verbessert. In Abschnitt 6.6 wurde *Data Augmentation* bereits näher erläutert. In diesem Versuch soll künstliches Rauschen und ein Bias auf die Sequenzen der Sensordaten addiert werden. Wie in Tabelle 7.9 dargestellt, soll hier zwischen zwei Stärken unterschieden werden.

Tabelle 7.9: Rauschen

	Acc Bias $[m/s^2]$	**Gyro Bias** $[rad/s]$	**Acc Rauschen** $[rad/s]$	**Gyro Rauschen** $[rad/s]$
NOISE1 (stark)	[-0,25;0,25]	[-0,1;0,1]	$\sigma = 0{,}5$	$\sigma = 0{,}05$
NOISE2 (schwach)	[-0,025;0,025]	[-0,01;0,01]	$\sigma = 0{,}05$	$\sigma = 0{,}005$

- Neuronenanzahl: 100
- Sequenzlänge: 200
- Loss: Loss-Funktion 2 (siehe Unterabschnitt 7.3.2)
- Repräsentation der Orientierung: HYBRID (siehe Unterabschnitt 7.3.1)
- Augmentation: kein

Tabelle 7.10: Ergebnis Versuch 5 - Rauschen (Netzvariante 1 - GRU)

ID	**Augment.**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU_100_200_NOISE1	stark	-15,08	91,53	8,79	9,09
GRU_100_200_NOISE2	schwach	**-15,29**	**85,85**	8,54	**4,80**
GRU_100_200	kein	-15,09	90,63	**8,47**	8,16

Ergebnis:
Für Netzvariante 1 (GRU) zeigt die Nutzung der zufälligen Addierung von Rauschen in der schwachen Variante eine Verbesserung bei *Heading* und *Position*. Der Fehler für die *Attitude* erhöht sich jedoch und ist in der Variante ohne Rauschen am kleinsten.
Für Netzvariante 2 (TCN) zeigt die Nutzung der zufälligen Addierung von Rauschen

Tabelle 7.11: Ergebnis Versuch 5 - Rauschen (Netzvariante 2 - TCN)

ID	**Augment.**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
TCN_100_200_NOISE1	stark	**-17,84**	93,34	**4,07**	6,2
TCN_100_200_NOISE2	schwach	-17,59	85,34	4,84	**4,11**
TCN_100_200	kein	-17,29	**85,23**	4,16	4,19

in der starken Variante eine Verbesserung bei der *Attitude*. Der Fehler für *Heading* und *Position* erhöht sich hier jedoch.

Bewertung:
Das Rauschen ermöglicht eine Variation der Daten pro Sequenz und soll dem Netz eine bessere Abstraktion ermöglichen. Ein zu starkes Rauschen kann das Signal jedoch soweit verfälschen, dass die Bestimmung der Ausgangsgrößen unmöglich wird. Für die weiteren Versuche sollen bei der Netzvariante 1 (GRU) ein schwaches Rauschen und bei Netzvariante 2 (TCN) kein Rauschen berücksichtigt werden.

7.3.6 Versuch 6: Nutzung von Augmentation (Rotationen)

Im sechsten Versuch soll getestet werden, ob die Nutzung von zufälligen Rotationen die Ergebnisse verbessert. In Abschnitt 6.6 wurde *Data Augmentation* bereits näher erläutert. Hierbei soll eine zufällige Quaternion erzeugt werden und die Sensordaten sowie die Orientierung der *Ground Truth* sollen um diese zufällige Quaternion rotiert werden.

Folgende weitere Parameter werden angenommen:

- Neuronenanzahl: 100
- Sequenzlänge: 200
- Loss: Loss-Funktion 2 (siehe Unterabschnitt 7.3.2)
- Repräsentation der Orientierung: HYBRID (siehe Unterabschnitt 7.3.1)
- Augmentation: kein

Tabelle 7.12: Ergebnis Versuch 6 - Rotationen (Netzvariante 1 - GRU)

ID	**Augment.**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU_100_200_ROT	rotation	**-15,24**	105,55	9,26	8,49
GRU_100_200	kein	-15,09	**90,63**	**8,47**	**8,16**

Tabelle 7.13: Ergebnis Versuch 6 - Rotationen (Netzvariante 2 - TCN)

ID	**Augment.**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
TCN_100_200_ROT	rotation	17.28	**68,06**	**4,00**	4,54
TCN_100_200	kein	**-17.29**	85,23	4,16	**4,19**

Ergebnis:
Für die Netzvariante 1 (GRU) sorgt die Nutzung der *Data Augmentation* für eine Verschlechterung des Ergebnisses.
Für die Netzvariante 2 (TCN) verringert die zufällige Rotation alle Fehler.

Bewertung:
Wie in Unterabschnitt 7.3.5 beschrieben, führt die zufällige Rotation zu einer Variation der Daten, die eine Verallgemeinerung für das Netz ermöglicht. Der Grund, warum eine zufällige Rotation bei Netzvariante 1 (GRU) zu einer Verschlechterung der Ergebnisse führt, konnte bis zum Ende der Arbeit nicht festgestellt werden. Für die Netzvariante 2 (TCN) soll für alle zukünftigen Versuche eine zufällige Rotation beim Training verwendet werden.

7.3.7 Versuch 7: Nutzung von OF- und ToF-Sensordaten

Im siebten Versuch soll getestet werden, ob die Nutzung der Daten des OF- und ToF-Sensors als zusätzliche Eingangsgrößen die Ergebnisse der *Pose*-Schätzungen verbessert. Die Berechnung der synthetischen Sensordaten wurden bereits in Abschnitt 6.2 beschrieben. Für die selbsterstellten Trainingsdaten wurden die Sensordaten wie in Abschnitt 6.3 beschrieben durch reale Messungen aufgezeichnet.

Folgende weitere Parameter werden angenommen:

- Neuronenanzahl: 100
- Sequenzlänge: 200
- Loss: Loss-Funktion 2 (siehe Unterabschnitt 7.3.2)
- Repräsentation der Orientierung: HYBRID (siehe Unterabschnitt 7.3.1)
- Augmentation: schwaches Rauschen (Netzvariante 1) und zufällige Rotationen (Netzvariante 2)

Tabelle 7.14: Ergebnis Versuch 7 - Nutzung von OF und ToF (Netzvariante 1 - GRU)

ID	**OF +TOF**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU_100_200_NOISE2	aus	-15,29	85,85	8,54	**4,80**
GRU_100_200_NOISE2_OF	aktiv	**-15,58**	**82,43**	**8,36**	10,70

Tabelle 7.15: Ergebnis Versuch 7 - Nutzung von OF und ToF (Netzvariante 2 - TCN)

ID	**OF +TOF**	**Validation Loss**	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
TCN_100_200_ROT	aus	**-17,28**	**68,06**	**4,00**	**4,54**
TCN_100_200_ROT_OF	aktiv	-17,84	73,34	4,03	5,32

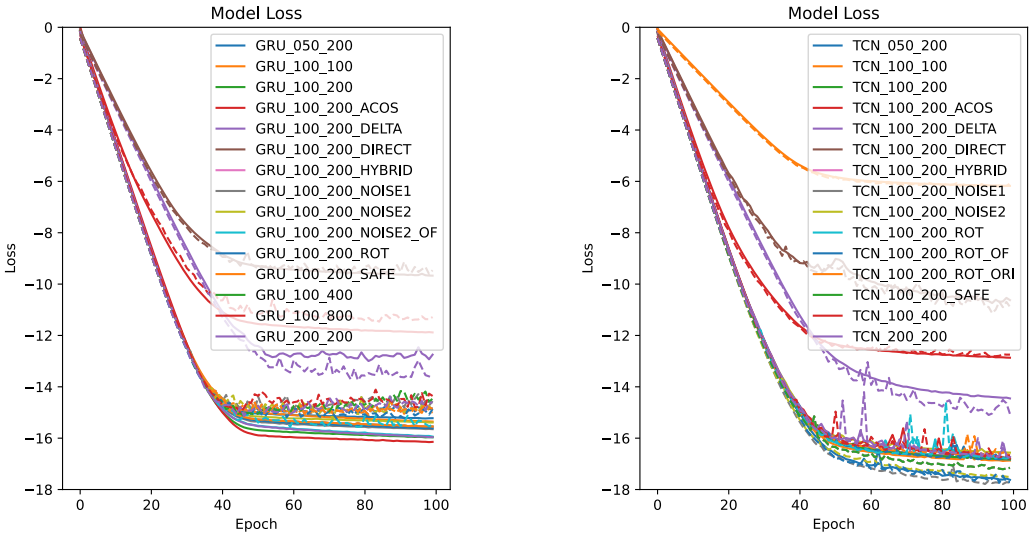
Ergebnis:
Bei der Netzvariante 1 (GRU) verkleinert die Nutzung von zusätzlichen Sensordaten den Fehler für *Heading* und *Attitude* geringfügig. Der Fehler für Position verschlechtert sich

jedoch stark.
Für die Netzvariante 2 (TCN) verschlechtert sich der Fehler für alle Ausgangsgrößen.

Bewertung:
Insgesamt führt die Nutzung von OF- und ToF-Sensordaten zu einer Verschlechterung der Ergebnisse. Es wäre zu erwarten gewesen, dass die zusätzlichen Stützinformationen die Positionsschätzung gegenüber der rein inertialen Navigation deutlich verbessern. Ein möglicher Erklärungsversuch hierfür wird in Unterabschnitt 8.3.1 gegeben.

7.4 Vergleich Loss

In Abbildung 7.1 sind der *Loss* und der *Validation Loss* für alle trainierten Netzvarianten gegenübergestellt. Insgesamt lässt sich erkennen, dass bei Netzvariante 1 (GRU) ein leichtes *Overfitting* stattfindet, da der *Validation Loss* zum Ende leicht ansteigt. Es wird ein *Dropout* für alle GRU-Layer von 25 % genutzt, um dem *Overfitting* generell entgegenzuwirken.



(a) Loss Netzvariante 1 (GRU)

(b) Loss Netzvariante 2 (TCN)

Abbildung 7.1: Training (gestrichelt: *Validation Loss*, durchgehend: *Loss*)

8 Vergleich

8.1 Referenz-Filter

Um die Performance der trainierten KNNs zu überprüfen, sollen einige Referenz-Filter eingesetzt werden. Es soll sowohl die Performance für die Bestimmung der Orientierung (*Attitude* und *Heading*) als auch für die Position zwischen den Netzen und den Referenzfiltern verglichen werden.
Für die Bestimmung der Orientierung sollen EKF, Mahony, Madgwick-Filter sowie eine einfache trigonometrische Bestimmung über die Daten des Beschleunigungssensors (hier als *Acc* bezeichnet) als Referenz dienen. Außerdem soll das neuronale Netz RIANN [53] als Referenz genutzt werden, welches jedoch nur die *Attitude* bestimmen kann. Ein Vergleich mit anderen neuronalen Netzmodellen aus der Referenzliteratur wird nicht durchgeführt, da diese auf bidirektionalen-RNNs basieren, sodass eine Schätzung erst nach einer Verzögerung zur Verfügung steht. Diese Netze können strukturbedingt besser performen, sind aber für einen Echtzeiteinsatz auf einer Drohne nicht sinnvoll. Für EKF, Mahony, Madgwick und Acc werden die Implementierungen aus der Python Bibliothek AHRS [40] verwendet.
Für die Bestimmung der Position soll ein eigener SINS-Algorithmus genutzt werden, der in Unterabschnitt 8.1.1 näher beschrieben wird. Alle Referenzfilter werden hierbei

Tabelle 8.1: alle Referenzfilter

Netzvariante/Referenzfilter	**Attitude**	**Heading**	**Position**
Acc	x		
Mahony	x	x	
Madgwick	x	x	
EKF	x	x	
INS (nur IMU)	x	x	x
INS (IMU+OF+TOF)	x	x	x
RIANN	x		

mit Standardparametern genutzt. Um einen fairen Vergleich zwischen den Netzmodellen und den Referenzmodellen zu ermöglichen, bekommen die Referenzfilter zu Beginn eine geschätzte Orientierung als Startgröße. Diese setzt sich aus der *Heading* und einer geschätzten *Attitude* zum Startzeitpunkt zusammen. Hierbei wird die *Attitude* mithilfe einer trigonometrischen Funktion aus den Daten des Beschleunigungsvektors errechnet. Die Start-*Heading* wird aus der *Ground Truth* bezogen. In Tabelle 8.1 ist eine Übersicht der Referenzfilter und der zu bestimmenden Größen zu sehen.

8.1.1 Eigener INS-Filter

Es wurde ein eigener SINS-Algorithmus entwickelt, der auf dem in Unterabschnitt 2.4.1 vorgestellten Verfahren basiert. Für die Orientierung wird jedoch ein Madgwick-Filter und für die Positionsbestimmung ein Kalman-Filter genutzt. Hierbei wurden zwei Varianten erstellt:

- **INS (nur IMU)**: Es werden nur die Daten der IMU genutzt. Die Orientierung wird über ein Madgwick-Filter berechnet. Ein Kalman-Filter schätzt die relative Position unter Berücksichtigung der Beschleunigungen und der aktuellen Orientierung.
- **INS (IMU+OF+TOF)**: Der Aufbau ist ähnlich zur inertialen Version. Das Kalman-Filter nutzt jedoch mit hoher Gewichtung die Daten des OF- und ToF-Sensors für die Berechnung der Positionsänderung und mit geringer Gewichtung die Daten des Beschleunigungssensors.

8.2 Vergleich mit Referenz-Filtern (Inertiale Navigation)

In diesem Abschnitt wird die Performance der trainierten Netzen, die eine rein inertiale Navigation durchführen, untereinander und mit den Referenzfiltern verglichen. Im Folgenden werden die beiden besten Netzvarianten für die inertiale Navigation aus Kapitel 7 verwendet (GRU_100_200_NOISE2 und TCN_100_200_ROT).

In Tabelle 8.2 sind für die beiden Netzvarianten aus den Versuchen (Kapitel 7) die gemittelten Fehler für alle Testdatensätze gegenübergestellt. Tabelle 8.3 stellt die Referenz-Filter und deren Ergebnisse für die gleichen Testdatensätze gegenüber. In Abbildung 8.1

sind die Ergebnisse außerdem in einem Boxplot[1] dargestellt. Die Ausreißer wurden zur Übersichtlichkeit hier ausgeblendet.

Tabelle 8.2: RMSE für GRU und TCN Netz (inertiale Navigation)

Netzvariante	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU (nur IMU)	85,85	8,54	4,80
TCN (nur IMU)	**68,06**	**4,00**	**4,54**

Tabelle 8.3: RMSE für die Referenzfilter (inertiale Navigation)

Referenzfilter	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
INS (nur IMU)	42,63	6,17	**8576,16**
Mahony	39,33	3,52	-
Madgwick	42,62	6,15	-
EKF	**35,09**	11,35	-
RIANN	-	**3,21**	-
Acc	-	9,47	-

Vergleich der neuronalen Netzvarianten untereinander

Zwischen den beiden Netzvarianten performt das TCN bei allen Vergleichsgrößen besser. Der *Attitude*-Fehler ist mit 4,00 ° weniger als halb so groß wie beim GRU-Netz mit 8,54 °. Der Positionsfehler ist mit 4,8 m (GRU) und 4,54 m (TCN) ähnlich groß. Der *Heading*-Fehler ist bei beiden Netzvarianten recht hoch, jedoch beim TCN-Netz knapp 17 ° geringer als beim GRU-Netz.

Vergleich der Referenzfilter

Zwischen den Referenz-Filtern performt das RIANN-Filter mit einem Fehler von 3,21 ° für die *Attitude* Bestimmung am besten. Das Mahony-Filter zeigt ähnlich gute Ergebnisse zu dem RIANN-Filter mit 3,52 °. Das Madgwick-Filter, das auch im INS-Algorithmus verwendet wird, hat einen Fehler von 6,15 ° bei der *Attitude*. Das EKF ist bei der *Attitude* mit 11,35 ° am schlechtesten. Bei der *Heading* haben alle Filter recht hohe Fehler. Das EKF mit 35,09 ° performt jedoch bei der Schätzung der *Heading* am besten.

[1] Bei einem Boxplot (auch Box-Whisker-Plot genannt) wird eine Box dargestellt, in der sich die mittleren 50% der Daten befinden. Der Strich innerhalb der Box kennzeichnet den Median. Die Antennen (Whisker) gehen zum min. und max. Wert bzw. bis zum max. 1,5-fachen der Höhe der Box (Interquartilsabstand). Ausreißer außerhalb des 1,5-fachen Interquartilsabstands werden gewöhnlich separat dargestellt.

Vergleich der neuronalen Netzvarianten gegenüber den Referenzfiltern
Beim Vergleich der Referenzfilter zu den beiden Netzvarianten zeigt sich, dass das TCN-Netz bezüglich der *Attitude* zwar besser ist als das Madgwick- und EKF-Filter, jedoch schlechter als das RIANN-Netz und das Mahony-Filter. Im Boxplot lässt sich jedoch erkennen, dass der obere Whisker des RIANN-Netzes deutlich oberhalb des TCN-Netzes liegt, hieraus lässt sich schlussfolgern, dass die Varianz des Fehlers beim RIANN-Netz größer ist. Das TCN-Netz arbeitet hier auf alle Datensätze betrachtet zuverlässiger.
Bezüglich der Position sind sowohl TCN- als auch GRU-Netz um mehrere Größenordnungen besser als das INS mit einem Fehler von 8576 m. Im Boxplot sind der Positionsfehler von TCN- und GRU-Netz hierbei verschwindend gering gegenüber dem INS-Algorithmus.
Der *Heading*-Fehler ist sowohl für Referenzfilter als auch für die beiden Netzvarianten sehr hoch. Im Boxplot ist jedoch ersichtlich, dass der Median der Referenzfilter hier deutlich tiefer liegt als beim GRU- und TCN-Netz.

In Abbildung 8.2 sind die Trajektorien für jeweils einen Testdatensatz für GRU- und TCN-Netz, sowie den INS-Algorithmus dargestellt. In der Abbildung lässt sich erkennen, dass die Trajektorie des INS-Algorithmus sofort aus dem Bild verschwindet. Es werden sehr starke Positionsfehler erreicht, so liegt der RMSE bei 8576 m (siehe Tabelle 8.2). Die Trajektorien der Netzvarianten GRU und TCN sind jedoch gegenüber dem INS-Algorithmus deutlich näher an der *Ground Truth*. Für den OxIOD-Datensatz (siehe Abbildung 8.2a) zeigt sich, dass die geschätzte Trajektorie, genau wie die *Ground Truth*, eine ovale Form abbildet, die sich nach oben bewegt und mit der Höhe zunehmend eine Verdrehung gegenüber der *Ground Truth* annimmt. In den Datensätzen EuRoC MAV und ARKit ist die grundlegende Bewegungsform ähnlich zur *Ground Truth*, jedoch sind die Trajektorien skaliert bzw. um die Hochachse gedreht. Die Trajektorien des TCN-Modells sind hierbei sichtbar näher an der *Ground Truth*. Bei dem BROAD-Datensatz verschwindet die geschätzte Trajektorie sehr weit von der *Ground Truth*, eine ähnliche Bewegungsform kann optisch nicht ermittelt werden.

In Abbildung 8.3 ist der RMSE für *Heading*, *Attitude* und *Position* zeitlich für die ersten 30 Sekunden für den Datensatz MH-02-easy-mav0-imu0 exemplarisch abgebildet. In der ersten Grafik ist der Betrag der Drehraten und Beschleunigungen als Referenz zeitlich aufgetragen. Hier lassen sich Beschleunigungen und Rotationen der Drohne ablesen.
Es zeigt sich, dass der *Heading*-Fehler des TCN unterhalb der Referenzfilter (Mahony, Madgwick und EKF) bleibt. Der *Heading*-Fehler für das GRU-Netz ist dagegen jedoch über die komplette Zeitdauer oberhalb des Referenzfilters.
Der *Attitude*-Fehler des GRU-Ansatzes zeigt zu Beginn einen Sprung an die 40 °, der

jedoch schnell korrigiert wird und ist dann deutlich oberhalb der anderen Fehlerverläufe der Referenzfilter. Stärkere Drehraten und Beschleunigungen der IMU lassen sich als Sprünge im Fehlerverlauf erkennen. Das TCN-Modell zeigt über die ersten 30 Sekunden einen konstant kleinen Fehler, der größtenteils unterhalb der Referenzfilter liegt. Der Positionsfehler des INS-Algorithmus nimmt über die ersten 30 Sekunden mit einem exponentiellen Verlauf zu, wogegen die Fehlerverläufe für TCN- und GRU-Netz deutlich darunter bleiben.

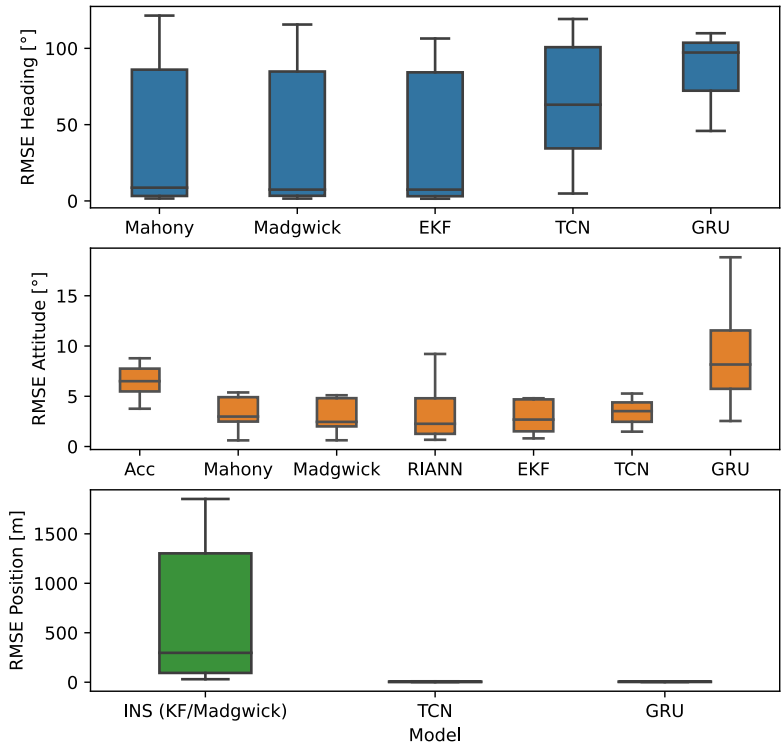
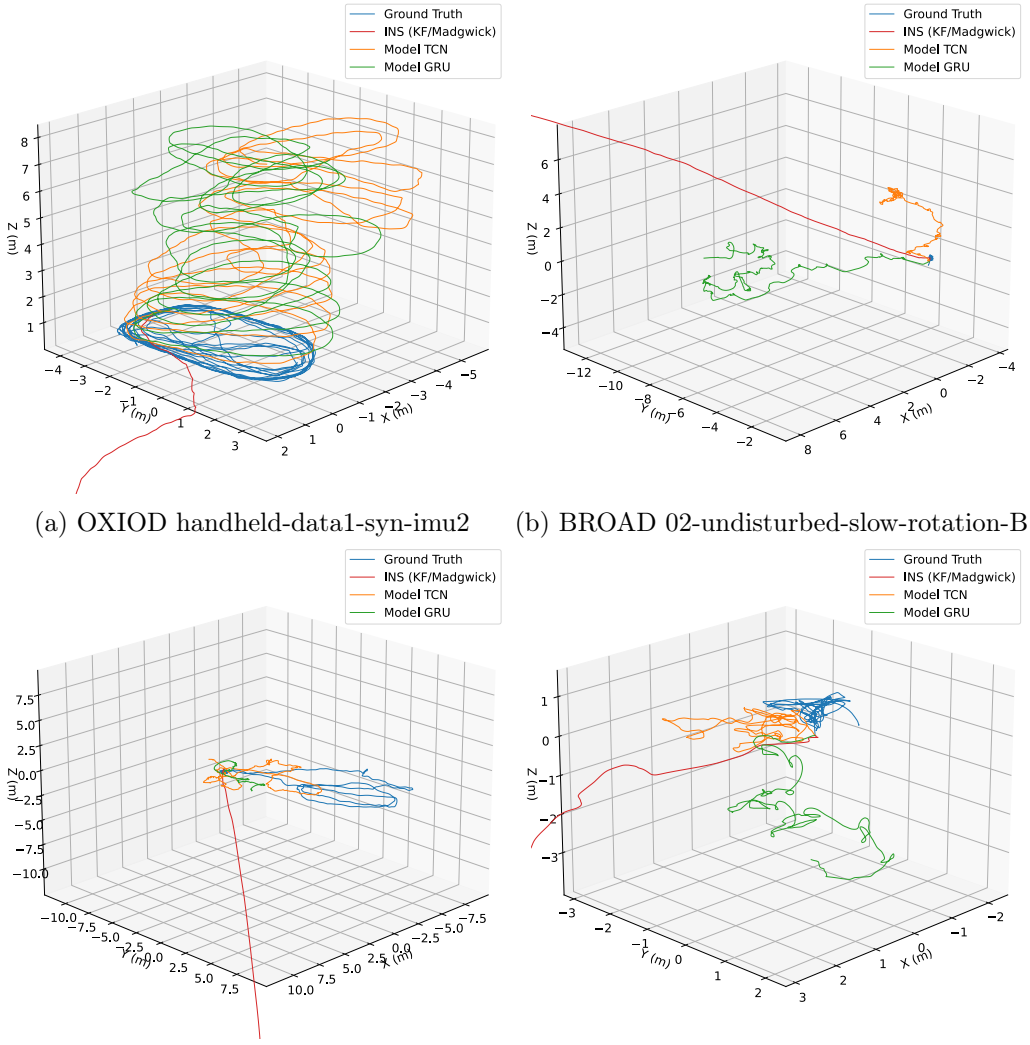


Abbildung 8.1: Boxplot (nur inertiale Navigation)



(a) OXIOD handheld-data1-syn-imu2

(b) BROAD 02-undisturbed-slow-rotation-B

(c) EuRoC MH-02-easy-mav0-imu0

(d) ARKit timeseries-8

Abbildung 8.2: Trajektorien (nur inertiale Navigation)

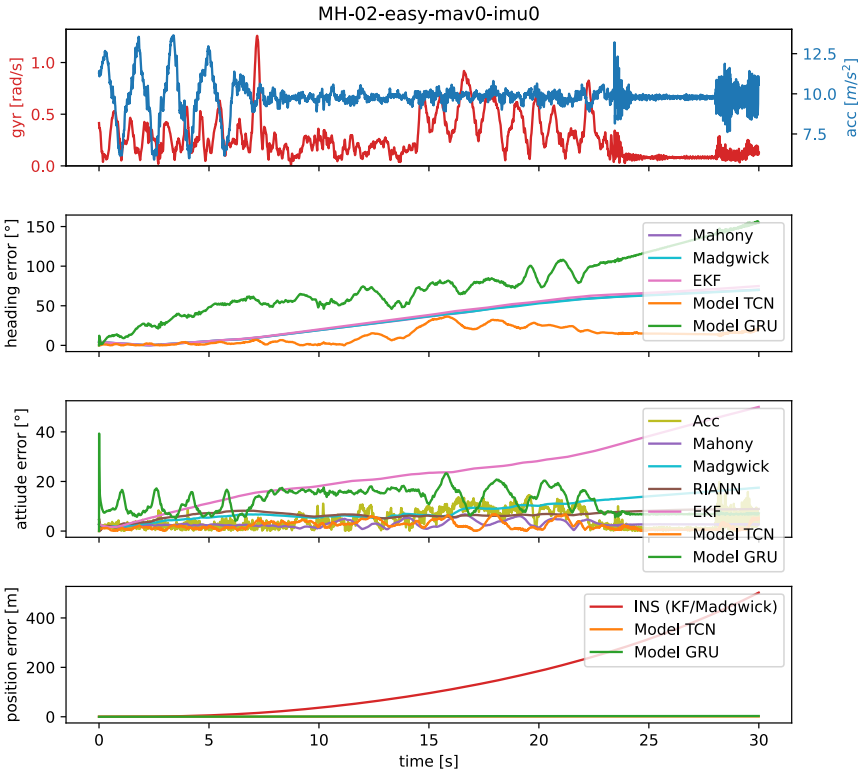


Abbildung 8.3: Verlauf der Fehler für die ersten 30 Sekunden (nur inertiale Navigation)

8.2.1 Bewertung

Die konventionellen Filter performen bei der Orientierungsbestimmung (*Attitude* und *Heading*) deutlich besser als das GRU-Netz. Das TCN-Netz kommt bei dem *Attitude*-Fehler in eine ähnliche Größenordnung wie die Referenzfilter und performt teilweise sogar besser. Beim *Heading*-Fehler sind die Referenzfilter jedoch deutlich näher an der *Ground Truth*. Bei der Position performen die beiden neuronalen Netzvarianten um mehrere Größenordnungen besser.
Generell ist die Navigation eines INS-Systems nicht langzeitstabil. Da kleine Driftfehler zu starken Positionsfehlern führen können, war das schlechte Ergebnis des INS-Filters zu erwarten.
Die neuronalen Netzmodelle schaffen hier jedoch eine Positionsschätzung mit verhältnismäßig kleinem Fehler. Der mittlere Fehler des TCN-Netzes liegt gerade mal bei 4,54 m. Der mittlere Fehler des GRU-Netzes liegt bei 4,8 m. Jedoch müssen die Werte auch kritisch hinterfragt werden. Während des Trainingsvorgangs werden die neuronalen Netze zu kleinen *Loss*-Fehlern optimiert. Die Netzmodelle können hierbei fälschlicherweise auch lernen, möglichst kleine Positionsanpassungen zu machen, was indirekt zu einem kleinen Positionsfehler führt, wenn die eigentliche *Ground Truth*-Trajektorie nur eine geringe Ausdehnung hat. So erscheint das Ergebnis deutlich besser, als es eigentlich ist.

Weiterhin muss angemerkt werden, dass für die hohen *Heading*-Fehler das fehlende Magnetometer verantwortlich ist. Eine langzeitstabile Schätzung ist ohne die Messung des Erdmagnetfeldes mit den verwendeten Sensordaten nicht möglich. Die fehlerhafte Schätzung der *Heading* hat auch direkten Einfluss auf die Positionsbestimmung. Die Positionsänderung wird vom neuronalen Netz, sowie dem INS im körpereigenen Koordinatensystem ermittelt. Unter Annahme einer falschen *Heading* wird die Positionsänderung in eine falsche Richtung angenommen.

Die TCN-Variante performt wie oben beschrieben deutlich besser als die GRU-Variante. Grund hierfür können die TCN-Layer sein, diese ermöglichen durch *Dilation* über eine Sequenz hinweg eine Abstraktion über mehrere Zeitschritte, sodass z.B. Muster in den Signalverläufen besser erkannt werden können. Das GRU-Netz hat lediglich seine internen Zustände und bekommt bei jeder Iteration ein neues Sensor-Sample. Hierdurch können vermutlich Zusammenhänge und Muster über mehrere Zeitschritte hinweg schlechter abstrahiert werden. Es muss jedoch angemerkt werden, dass das RIANN-Filter deutlich besser bei der *Attitude* Bestimmung performt als die eigene GRU-Netzvariante, obwohl

der Aufbau sehr ähnlich ist. Die genauen Gründe, warum die GRU-Netzvariante schlechter performt als das RIANN-Filter in der Referenzliteratur, konnten bis zum Ende der Arbeit nicht ergründet werden.

8.3 Vergleich mit Referenzfiltern (Unterstützung durch Optical Flow und Time-of-Flight)

In diesem Abschnitt werden die Filter gegenübergestellt, die ergänzend Daten von OF- und ToF-Sensor nutzen. Dies sind die beiden besten Netzvarianten aus Kapitel 7 (GRU_-100_200_NOISE2_OF und TCN_100_200_ROT_OF).

In Tabelle 8.4 werden die Ergebnisse der besten Modelle der beiden Netzvarianten dargestellt. Tabelle 8.5 zeigt die Ergebnisse des INS Algorithmus. In Abbildung 8.4 sind die Ergebnisse außerdem in einem Boxplot gegenübergestellt.

Tabelle 8.4: RMSE für GRU und TCN-Netz (IMU + OF + ToF)

Netzvariante	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
GRU (IMU+OF+TOF)	82,43	8,36	10,70
TCN (IMU+OF+TOF)	**73,34**	**4,03**	**5,32**

Tabelle 8.5: RMSE für die Referenzfilter (IMU + OF + ToF)

Referenzfilter	**RMSE Heading [°]**	**RMSE Attitude [°]**	**RMSE Position [m]**
INS (IMU+OF+TOF)	**42,64**	**6,16**	**2,3**

Vergleich der neuronalen Netzvarianten untereinander

Es zeigt sich, dass auch mit Unterstützung der OF- und ToF-Daten das TCN-Netz weiterhin die besseren Ergebnisse unter den beiden Netzmodellen liefert. Der Positionsfehler beim GRU-Netz hat sich anders als zu erwarten von 4,8 m auf 10,7 m verschlechtert. Der Positionsfehler vom TCN-Netz hat sich ebenso von 4,54 m auf 5,32 m verschlechtert. In Abbildung 8.5 lässt sich erkennen, dass sich die geschätzten Trajektorien der beiden Netze von den Trajektorien der rein inertialen Navigation unterscheiden. Die Ausdehnung der geschätzten Trajektorien ist bei allen Datensätzen ähnlich oder schlechter geworden gegenüber der rein inertialen Navigation. Der *Attitude*-Fehler hat sich beim TCN-Netz gegenüber der rein inertialen Navigation etwas verschlechtert und beim GRU-Netz leicht verbessert. Die *Heading*-Fehler sind bei beiden Netzvarianten weiterhin sehr hoch.

Vergleich der Referenzfilter

In Abbildung 8.5 lässt sich erkennen, dass die Trajektorie des INS-Filters nun im Gegen-

satz zu der Navigation ohne OF und ToF-Daten über die komplette Zeitdauer deutlich dichter an der *Ground Truth* ist. Im OxIOD und ARKit-Datensatz bleibt die geschätzte Trajektorie auf der gleichen Höhe und im gleichen Bereich wie die *Ground Truth*. Im BROAD-Datensatz entfernt sich die Trajektorie jedoch deutlich vom Ausdehnungsbereich der *Ground Truth*. Der Positionsfehler des INS Algorithmus hat sich nun von 8576 m auf 2,3 m verbessert.
Die Fehler für *Attitude* und *Heading* wurden schon im vorherigen Abschnitt erläutert, diese haben sich gegenüber der reinen inertialen Navigation kaum geändert, da die zusätzlichen Sensordaten aus OF- und ToF-Sensor nicht in die Orientierungsschätzung einfließen, sondern lediglich zur Positionsschätzung beitragen können.

Vergleich der neuronalen Netzvarianten gegenüber den Referenzfiltern
Insgesamt ist der mittlere Positionsfehler des INS-Algorithmus mit 2,3 m nun weniger als halb so groß wie beim TCN-Netz. Die Positionsschätzung des INS-Algorithmus hat sich demnach gegenüber der rein inertialen Navigation stark verbessert.

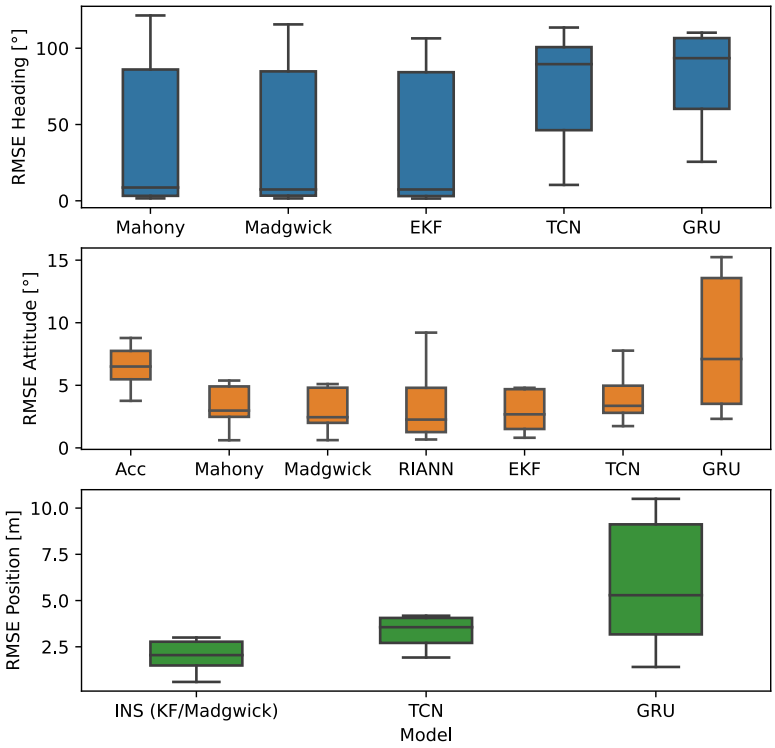
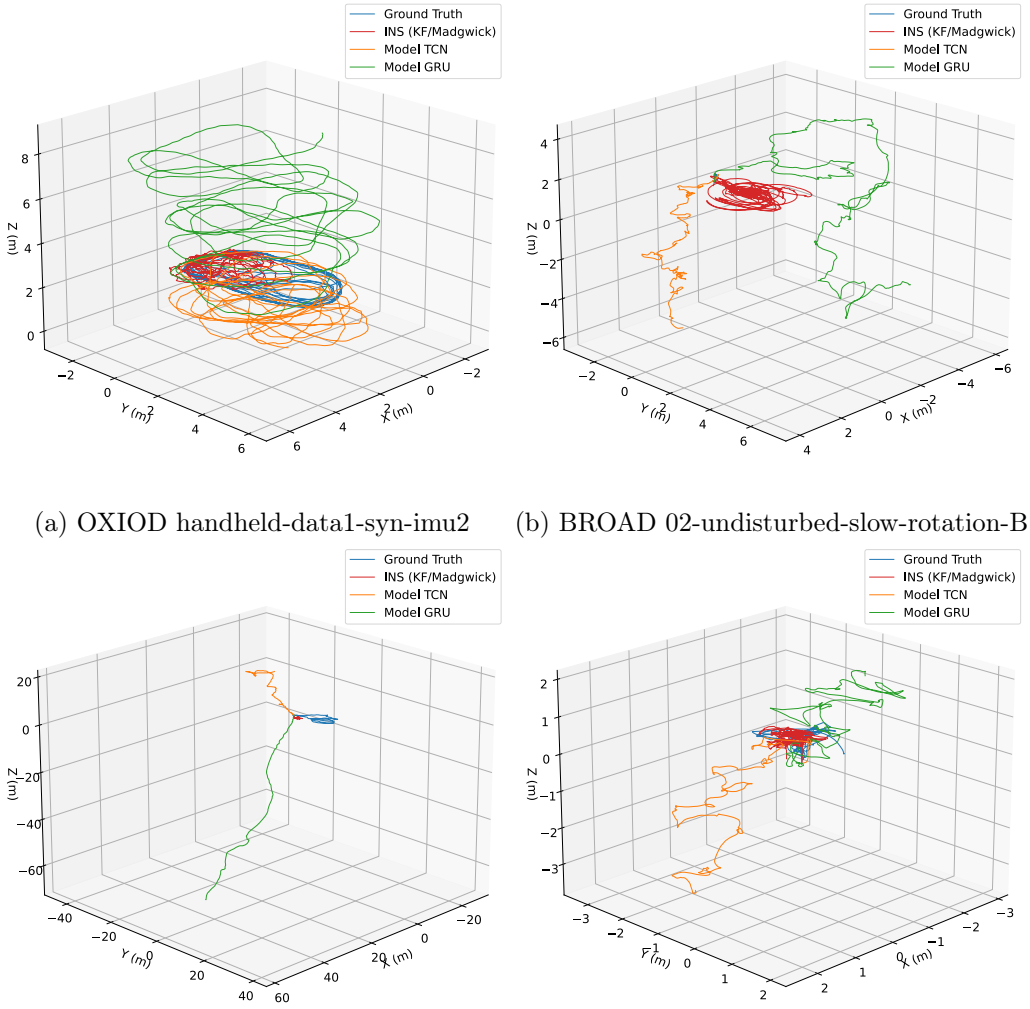


Abbildung 8.4: Boxplot (mit OF und ToF-Sensordaten)



(a) OXIOD handheld-data1-syn-imu2

(b) BROAD 02-undisturbed-slow-rotation-B

(c) EuRoC MH-02-easy-mav0-imu0

(d) ARKit timeseries-8

Abbildung 8.5: Trajektorien (mit OF und ToF-Sensordaten)

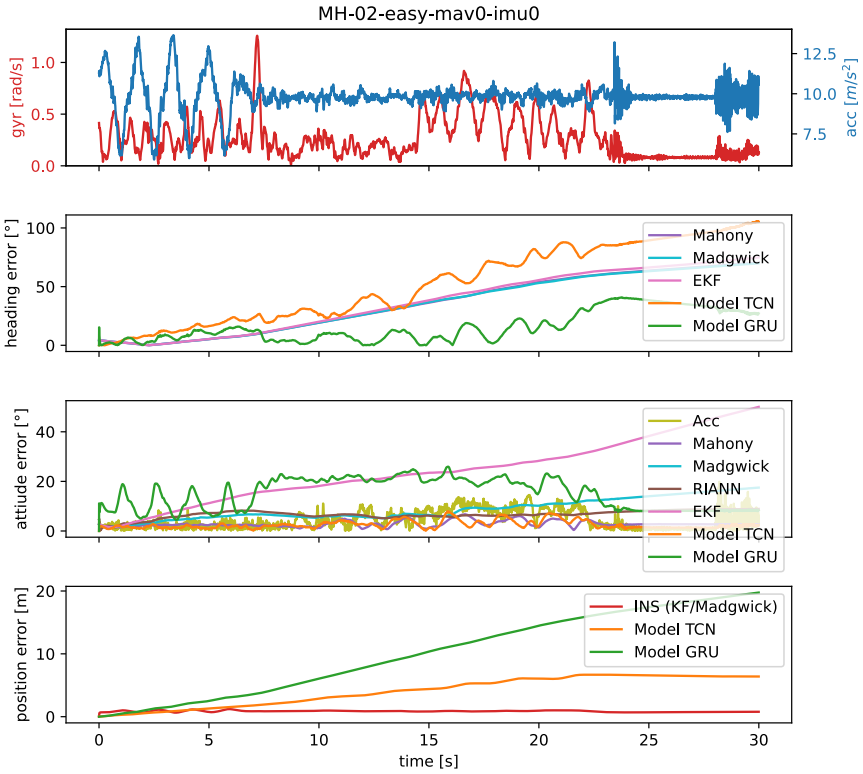


Abbildung 8.6: Verlauf der Fehler für die ersten 30 Sekunden (mit OF und ToF-Sensordaten)

8.3.1 Bewertung

Insgesamt performt der INS-Algorithmus für die Positionsbestimmung deutlich besser als die beiden Netzmodelle. Generell können die Bewegungsinformationen aus dem OF- und ToF-Sensor die inertiale Navigation stark stützen, da sie Bewegungsinformationen enthalten, die im Gegensatz zu den Daten des Beschleunigungssensor nur einfach integriert werden müssen.

Die neuronalen Netzmodelle scheinen jedoch nicht von den zusätzlichen Informationen des OF- und ToF-Sensors zu profitieren. Die Positionsschätzungen der beiden Netzmodelle haben sich gegenüber der rein inertialen Navigation sogar verschlechtert. Es wird vermutet, dass hierfür Rotationen mit einer Neigung größer als 90 ° verantwortlich sind, bei denen sich die Drohne auf den Kopf neigt (z.B. Loopings). Die verwendeten Quelldaten stammen größtenteils nicht von Drohnen direkt, sondern es wurde lediglich die Pose von zufällig rotierten IMUs erfasst. Daher unterscheiden sich die Bewegungsformen von typischen Drohnenflügen. Auch die verwendete *Data Augmentation*, die eine zufällige Rotation einer Sequenz vorsieht, kann zu einer umgedrehten Lage der Drohne führen. Kritisch wird dies, wenn für solche Kurven synthetische Daten berechnet werden. Der ToF-Sensor kommt bei großer Neigung schnell in den gesättigten Bereich von 2 m. Der optische Fluss kann aufgrund der falschen Abstandsinformationen in keinen sinnvollen Kontext gesetzt werden, hierdurch kann das neuronale Netze keine sinnvollen Zusammenhänge abstrahieren. Dies führt vermutlich zu der Verschlechterung der Ergebnisse bei den synthetischen OF und ToF-Sensordaten.
Gleichzeitig wird vermutet, dass das Ergebnis des INS-Filter durch die synthetischen Sensordaten für OF- und ToF-Sensor positiv verfälscht wird. Da im INS-Algorithmus die exakten mathematischen Zusammenhänge genutzt werden, die auch zur Erstellung der synthetischen Daten verwendet wurden, performt das INS-Modell hier deutlich besser als dies in der Realität möglich wäre.

9 Zusammenfassung

In dieser Arbeit wurde untersucht, ob neuronale Netze auf einer Miniaturdrohne zur Bestimmung von Position(-sänderung) und Orientierung(-sänderung) genutzt werden können. Hierfür wurden die vielversprechendsten Ansätze aus der Literatur herausgearbeitet und eigene Netzmodelle entwickelt. Es wurden Trainingsdatensätze aus offenen Quellen hinsichtlich der Eignung bewertet und ausgewählt. Sensordaten, die nicht in den Datensätzen vorhanden sind, wurden synthetisch nachberechnet. Des Weiteren wurden Sensordaten mit der Miniaturdrohne selber aufgezeichnet und zeitlich an die *Ground Truth* angeglichen. Die Netzmodelle wurden in unterschiedlichen Varianten trainiert und die Hyperparameter variiert. Die besten Varianten wurden ausgewählt und mit den Referenz-Filtern gegenübergestellt. Für den Vergleich der Positionsschätzung mit einem konventionellen Filter wurde ein eigener INS-Algorithmus basierend auf einem Kalman- und Madgwick-Filter entwickelt, der ergänzend zur inertialen Navigation auch Daten von OF und ToF-Sensor verwenden kann.

9.1 Diskussion

Die Ergebnisse aus Kapitel 8 zeigen, dass neuronale Netze zur *Pose*-Schätzung generell verwendet werden können. Die Performance für *Attitude*, *Heading* und Position wird im Folgenden differenziert betrachtet.

Das eigene Netzmodell (TCN-Variante) kann die *Attitude* ähnlich gut bestimmen wie die meisten der verglichenen konventionellen Filter und performt teilweise sogar besser. Lediglich der Mahony-Filter performt von den konventionellen Filtern noch besser als das TCN-Netz. Der effizientere GRU-Ansatz kommt jedoch nicht an die Performance vom TCN-Netz heran. Leider konnten die Ergebnisse für die *Attitude*-Schätzung des RIANN-Netzes aus der Referenzliteratur nicht vollumfänglich reproduziert werden.

Bei der Schätzung der *Heading* existiert genau wie bei den konventionellen Filtern die Problematik, dass ohne Magnetometer-Daten keine langzeitstabile Schätzung möglich ist. Hier ergeben sich keine Vorteile durch die Nutzung der neuronalen Netze. Im Gegenteil ist die *Heading*-Schätzung sogar deutlich schlechter. Der TCN-Ansatz performt hier etwas besser als der GRU-Ansatz.

Bei der Positionsschätzung rein mit inertialen Daten performen die beiden neuronalen Netzansätze besser. Hierbei sind GRU- und TCN-Netz auf einem ähnlichen Niveau, d.h. die TCN-Variante hat hier kaum Vorteile. Obwohl die Positionsfehler deutlich kleiner sind als beim INS, weicht die *Position* über die Zeit von der *Ground Truth* ab. Die Problematik der inhärenten Messfehler und der doppelten Integration lässt sich auch mit dem Einsatz von ML nicht eliminieren. Jedoch scheinen Muster in den Sensordaten, die vom neuronalen Netzen genutzt werden, einen Teil der inhärenten Messfehler der IMU zu kompensieren und die Positionsschätzung gegenüber einer konventionellen Filterung zu verbessern. Auch wenn eine vollständige Navigation mit rein inertialen Sensordaten für einen langen Zeitraum nicht möglich scheint, kann für einen Ausfall von Stützsystemen, wie GNSS oder bei eingeschränkten Sichtbereich zum Himmel, eine Überbrückung für einen kleinen Zeitraum stattfinden.

Die Ergebnisse der *Pose*-Schätzung der neuronalen Netze konnten durch den ergänzenden Einsatz von OF und ToF-Sensoren nicht verbessert werden. Der INS-Algorithmus profitiert jedoch stark von der Nutzung der zusätzlichen Stützinformationen. In Unterabschnitt 8.3.1 wurde hierfür ein möglicher Erklärungsansatz gegeben, der das Verhalten durch den Einsatz von synthetischen Daten begründet, die gerade bei großen Neigungswinkeln oder Loopings der Drohne zu Fehlern führen können.

Im Abschnitt 4.5 konnte nachgewiesen werden, dass das GRU-Netz mit 100 Neuronen auf der Miniaturdrohne mit einer Ausführungszeit von 3 ms genutzt werden könnte. Die Netzvariante TCN, die bessere Ergebnisse zeigt, ist jedoch aufgrund des hohen Rechenbedarfs nicht für die Miniaturdrohne geeignet. Aufgrund der hohen Schätzfehler bei den Versuchen wurde von einem Feldversuch auf der Miniaturdrohne abgesehen.

Insgesamt lässt sich zusammenfassen, dass die konventionellen Filter für die Schätzung der Orientierung weiterhin eine Daseinsberechtigung haben. Besonders der Mahony-Filter arbeitet hier über alle Testdatensätze zuverlässiger als die neuronalen Netze und die Fehler haben eine geringere Streuung als bei den eigenen Netzmodellen und dem RIANN-Filter. Bei der Positionsschätzung konnten die neuronalen Netze ihre Stärke gegenüber den konventionellen Filtern jedoch deutlich zeigen. Es wird geschlussfolgert, dass der

Einsatz neuronaler Netze gerade für die Positionsschätzung (vorallem bei rein inertialer Navigation) eine sinnvolle Alternative zu den konventionellen Filtern darstellt.

9.2 Ausblick

Bei zukünftigen Ansätzen sollte die Verwendung von Magnetometer-Daten untersucht werden, um die Schätzung der *Heading* zu stabilisieren. Da die Schätzung der *Heading*, genau wie die Orientierung für die Positionsschätzung notwendig ist, wird davon ausgegangen, dass durch eine optimierte *Heading*-Schätzung auch die Positionsschätzung deutlich verbessert werden kann.
Ergänzend wäre ein Ansatz denkbar, bei dem ein neuronales Netz trainiert wird, um magnetische Störgrößen zu erkennen und von dem eigentlichen Erdmagnetfeld zu unterscheiden. Hierdurch könnte die *Heading*-Schätzung gerade im Indoorbereich stabilisiert werden.
Bei weiteren Arbeiten sollten vorzugsweise reale Sensordaten von Drohnen-Flügen verwendet werden, sodass die Bewegungsmuster mehr den Flügen von Drohnen entsprechen. Alternativ könnte auch eine Simulationssoftware wie Gazebo verwendet werden. Hierdurch können Sensordaten wie der optische Fluss deutlich realitätsnäher bestimmt werden, als dies mit den mathematischen Modellen in dieser Arbeit durchgeführt wurde.
Aufgrund der Stärken der neuronalen Netze bei der Positonsschätzung und der Zuverlässigkeit der konventionellen Filter bei der Orientierungsschätzung wäre eine kombinierter Ansatz von einem konventionellen Filter für die Orientierungsschätzung und einem neuronalen Netz zur Positionsschätzung denkbar. Hierdurch können die Vorteile aus beiden Welten miteinander kombiniert werden.
Des Weiteren sollte auch die stetige Entwicklung der Rechenleistung im Auge behalten werden. In dieser Arbeit wurde eine Miniaturdrohne mit geringer Rechenleistung verwendet, daher ist der Einsatz des TCN-Netzes nicht möglich. Neuere Generationen von Mikrocontrollern werden jedoch leistungsfähiger und können möglicherweise bald die notwendige Rechenleistung bereitstellen. Für größere Drohnen mit höherer Nutzlast wäre eine Nutzung schon jetzt denkbar.

10 Anhang

10.1 Netzmodelle

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru (GRU)                   (1, 200, 100)             32400

 gru_1 (GRU)                 (1, 200, 100)             60600

 dense (Dense)               (1, 200, 8)               808

=================================================================
Total params: 93808 (366.44 KB)
Trainable params: 93808 (366.44 KB)
Non-trainable params: 0 (0.00 Byte)
```

Abbildung 10.1: Netzvariante 1: GRU (Zusammenfassung)

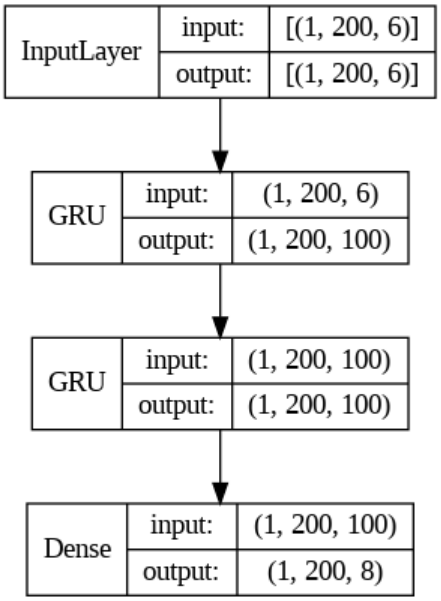


Abbildung 10.2: Netzvariante 1: GRU (Graph)

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 190, 128)          8576

 conv1d_1 (Conv1D)           (None, 180, 128)          180352

 max_pooling1d (MaxPooling1  (None, 60, 128)           0
 D)

 lstm (LSTM)                 (None, 60, 128)           131584

 lstm_1 (LSTM)               (None, 128)               131584

 dense (Dense)               (None, 8)                 1032

=================================================================
Total params: 453128 (1.73 MB)
Trainable params: 453128 (1.73 MB)
Non-trainable params: 0 (0.00 Byte)
```

Abbildung 10.3: Netzvariante 2: TCN-GRU (Zusammenfassung)

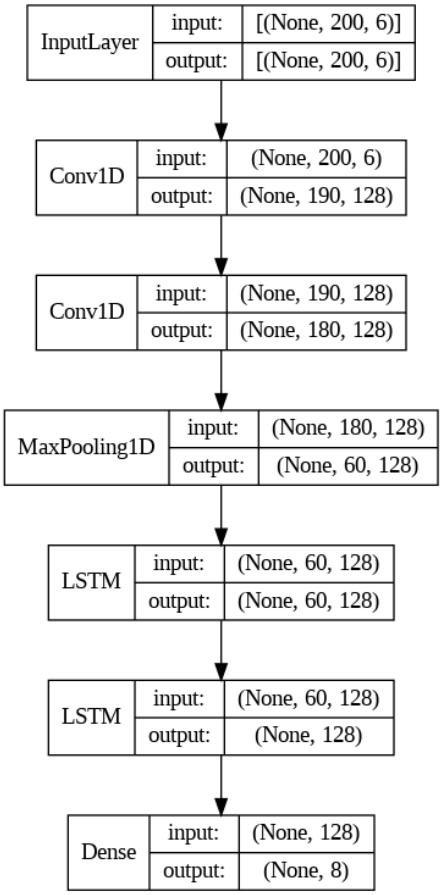


Abbildung 10.4: Netzvariante 2: TCN-GRU (Graph)

10.2 Trainingsdaten

Tabelle 10.1: Eigene Trainingsdatensätze

Quelle	**Name**	**Länge [h]**
ARKit	timeseries-1	00:01:04
ARKit	timeseries-2	00:01:08
ARKit	timeseries-3	00:01:19
ARKit	timeseries-5	00:01:06
ARKit	timeseries-6	00:01:09
ARKit	timeseries-7	00:01:16
ARKit	timeseries-9	00:01:05
ARKit	timeseries-10	00:01:04
ARKit	timeseries-12	00:01:21
ARKit	timeseries-13	00:01:20
ARKit	timeseries-14	00:01:30
ARKit	timeseries-15	00:01:22
ARKit	timeseries-16	00:01:20
ARKit	timeseries-19	00:01:16
ARKit	timeseries-20	00:01:15
ARKit	timeseries-21	00:01:25
ARKit	timeseries-22	00:01:20
ARKit	timeseries-23	00:01:19
ARKit	timeseries-25	00:01:11
ARKit	timeseries-26	00:01:18
ARKit	timeseries-27	00:01:08
ARKit	timeseries-28	00:01:10
ARKit	timeseries-29	00:01:20
ARKit	timeseries-30	00:01:23
ARKit	timeseries-31	00:01:24
ARKit	timeseries-32	00:01:26
ARKit	timeseries-33	00:01:19

Tabelle 10.2: Externe Trainingsdatensätze

Quelle	**Name**	**Länge [h]**
OXIOD	handheld-data1-syn-imu1	00:06:01
OXIOD	handheld-data1-syn-imu3	00:02:53
OXIOD	handheld-data1-syn-imu4	00:03:21
OXIOD	handheld-data1-syn-imu5	00:05:06
OXIOD	handheld-data1-syn-imu6	00:05:10
OXIOD	handheld-data1-syn-imu7	00:02:05
OXIOD	handheld-data2-syn-imu1	00:05:11
OXIOD	handheld-data2-syn-imu2	00:04:56
OXIOD	handheld-data2-syn-imu3	00:04:45
OXIOD	handheld-data3-syn-imu2	00:06:04
OXIOD	handheld-data3-syn-imu3	00:09:53
OXIOD	handheld-data3-syn-imu4	00:08:42
OXIOD	handheld-data3-syn-imu5	00:06:08
OXIOD	handheld-data4-syn-imu2	00:05:07
OXIOD	handheld-data4-syn-imu4	00:07:03
OXIOD	handheld-data4-syn-imu5	00:05:35
OXIOD	handheld-data5-syn-imu4	00:05:50
EuRoC	MH-01-easy-mav0-imu0	00:03:01
EuRoC	MH-03-medium-mav0-imu0	00:02:11
EuRoC	MH-04-difficult-mav0-imu0	00:01:38
EuRoC	MH-05-difficult-mav0-imu0	00:01:51
EuRoC	V1-02-medium-mav0-imu0	00:01:23
EuRoC	V1-03-difficult-mav0-imu0	00:01:44
EuRoC	V2-01-easy-mav0-imu0	00:01:52
EuRoC	V2-03-difficult-mav0-imu0	00:01:54
BROAD	01-undisturbed-slow-rotation-A	00:00:38
BROAD	03-undisturbed-slow-rotation-C	00:02:00
BROAD	06-undisturbed-fast-rotation-A	00:00:45
BROAD	10-undisturbed-slow-translation-A	00:01:50
BROAD	11-undisturbed-slow-translation-B	00:02:01
BROAD	12-undisturbed-slow-translation-C	00:02:08
BROAD	16-undisturbed-fast-translation-B	00:01:52
BROAD	20-undisturbed-slow-combined-360s	00:05:35

Tabelle 10.3: Eigene Testdatensätze

Quelle	**Name**	**Länge [h]**
ARKit	timeseries-4	00:01:12
ARKit	timeseries-8	00:01:07
ARKit	timeseries-11	00:01:15
ARKit	timeseries-17	00:01:18
ARKit	timeseries-18	00:01:29
ARKit	timeseries-24	00:01:10

Tabelle 10.4: Externe Testdatensätze

Quelle	**Name**	**Länge [h]**
OXIOD	handheld-data1-syn-imu2	00:03:39
OXIOD	handheld-data3-syn-imu1	00:04:52
OXIOD	handheld-data4-syn-imu1	00:05:02
OXIOD	handheld-data4-syn-imu3	00:09:50
OXIOD	handheld-data5-syn-imu1	00:04:55
EuRoC	MH-02-easy-mav0-imu0	00:02:29
EuRoC	V1-01-easy-mav0-imu0	00:02:23
EuRoC	V2-02-medium-mav0-imu0	00:01:55
BROAD	02-undisturbed-slow-rotation-B	00:01:52
BROAD	07-undisturbed-fast-rotation-B	00:01:57
BROAD	15-undisturbed-fast-translation-A	00:00:43

10.3 Inhalt der CD

- **01_Quellcode**
  - **myDrone_H743**
    Flugsteuerung für die Miniaturdrohne (STM32CubeIDE Projekt)
  - **Training_keras.ipynb**
    Training und Validierung der neuronalen Netze (Jupyter Notebook)
- **02_Modelle und Ergebnisse**
  - **Modelle und Ergebnisse.zip**
    alle trainierten Netzmodelle mit Trajektorien und ermittelten Fehlern (keras)
- **03_Laufzeitermittlung**
  - **Laufzeitermittlung.zip**
    Modelle zur Laufzeitermittlung auf der Drohne (keras)
- **Masterarbeit_Grotkasten.pdf**

Literaturverzeichnis

[1] AbdulMajuid, Ahmed ; Mohamady, Osama ; Draz, Mohannad ; El-bayoumi, Gamal: *GPS-Denied Navigation Using Low-Cost Inertial Sensors and Recurrent Neural Networks.* 2021

[2] Apple: *ARKit 6.* https://developer.apple.com/augmented-reality/arkit/

[3] Asgharpoor Golroudbari, Arman ; Sabour, Mohammad: End-to-End Deep Learning Framework for Real-Time Inertial Attitude Estimation using 6DoF IMU. (2023), 02

[4] Brossard, Martin ; Barrau, Axel ; Bonnabel, Silvère: RINS-W: Robust Inertial Navigation System on Wheels, 11 2019, S. 2068–2075

[5] Burri, Michael ; Nikolic, Janosch ; Gohl, Pascal ; Schneider, Thomas ; Rehder, Joern ; Omari, Sammy ; Achtelik, Markus ; Siegwart, Roland: The EuRoC micro aerial vehicle datasets. In: *The International Journal of Robotics Research* 35 (2016), 01

[6] Caruso, Marco ; Sabatini, Angelo M. ; Knaflitz, Marco ; Gazzoni, Marco ; Croce, Ugo D. ; Cereatti, Andrea: Orientation Estimation Through Magneto-Inertial Sensor Fusion: A Heuristic Approach for Suboptimal Parameters Tuning. In: *IEEE Sensors Journal* 21 (2021), Nr. 3, S. 3408–3419

[7] Chen, Changhao ; Lu, Xiaoxuan ; Markham, Andrew ; Trigoni, Niki: *IONet: Learning to Cure the Curse of Drift in Inertial Odometry.* 2018

[8] Chen, Changhao ; Lu, Xiaoxuan ; Wahlström, Johan ; Markham, Andrew ; Trigoni, Niki: Deep Neural Network Based Inertial Odometry Using Low-Cost Inertial Measurement Units. In: *IEEE Transactions on Mobile Computing* PP (2019), 12, S. 1–1

[9] Chen, Changhao ; Zhao, Peijun ; Lu, Chris X. ; Wang, Wei ; Markham, Andrew ; Trigoni, Niki: *OxIOD: The Dataset for Deep Inertial Odometry.* 2018

[10] Chiang, Kai-Wei ; Chang, Hsiu-Wen ; Li, Chia-Yuan ; Huang, Yun-Wen: An Artificial Neural Network Embedded Position and Orientation Determination Algorithm for Low Cost MEMS INS/GPS Integrated Sensors. In: *Sensors (Basel, Switzerland)* 9 (2009), 04, S. 2586–610

[11] Cirillo, Pasquale ; Cirillo, Andrea ; De Maria, G. ; Natale, Ciro ; Pirozzi, Salvatore: *A comparison of multisensor attitude estimation algorithms*. S. 529–539, 09 2016. – ISBN 9781498745710

[12] Cortés, Santiago ; Solin, Arno ; Kannala, Juho: Deep Learning Based Speed Estimation for Constraining Strapdown Inertial Navigation on Smartphones. (2018), 08

[13] Developer, Nvidia: *Recurrent Neural Network*. https://developer.nvidia.com/discover/recurrent-neural-network

[14] Geiger, Andreas ; Lenz, P ; Stiller, Christoph ; Urtasun, Raquel: Vision meets robotics: the KITTI dataset. In: *The International Journal of Robotics Research* 32 (2013), 09, S. 1231–1237

[15] Geron, Aurelien: *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd. O'Reilly Media, Inc., 2022. – ISBN 1492032646

[16] github.com: *INAVFlight*. https://github.com/iNavFlight

[17] Grotkasten, Robin: *Aufbau einer Miniaturdrohne*. HAW Hamburg

[18] Gugger, Sylvain ; Howard, Jeremy: *Deep learning for coders with fastai and PyTorch*. Sebastopol, CA : O'Reilly Media, Juli 2020

[19] Herath, Sachini ; Yan, Hang ; Furukawa, Yasutaka: RoNIN: Robust Neural Inertial Navigation in the Wild: Benchmark, Evaluations, New Methods, 05 2020, S. 3146–3152

[20] Horn, Berthold ; Schunck, Brian: Determining Optical Flow. In: *Artificial Intelligence* 17 (1981), 08, S. 185–203

[21] Invensense: *ICM-20602 Datasheet*. https://invensense.tdk.com/wp-content/uploads/2016/10/DS-000176-ICM-20602-v1.0.pdf

[22] Invensense: *MPU6000 Datasheet*. https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

[23] Jiménez, Antonio ; Seco, Fernando ; Prieto, C. ; Guevara, Jaime: A comparison of Pedestrian Dead-Reckoning algorithms using a low-cost MEMS IMU, 09 2009, S. 37 – 42

[24] Joseph, Manu: *Modern Time Series Forecasting with Python.* Birmingham, England : Packt Publishing, Juni 2022

[25] jpsml: *6-DOF-Inertial-Odometry.* https://github.com/jpsml/6-DOF-Inertial-Odometry

[26] Kang, Wonho ; Han, Youngnam: SmartPDR: Smartphone-Based Pedestrian Dead Reckoning for Indoor Localization. In: *IEEE Sensors Journal* 15 (2014), 01, S. 1–1

[27] Laidig, Daniel ; Caruso, Marco ; Cereatti, Andrea ; Seel, Thomas: BROAD—A Benchmark for Robust Inertial Orientation Estimation. In: *Data* 6 (2021), 06, S. 72

[28] Landwirtschaft, Bundesinformationszentrum: *Wofür braucht man Drohnen in der Landwirtschaft?* https://www.landwirtschaft.de/landwirtschaft-verstehen/wie-funktioniert-landwirtschaft-heute/wofuer-braucht-man-drohnen-in-der-landwirtschaft

[29] Lee, Soo-Hwan ; Kim, Won-Yeol ; Seo, Dong-Hoan: Automatic self-reconstruction model for radio map in Wi-Fi fingerprinting. In: *Expert Systems with Applications* 192 (2022), S. 116455. – URL https://www.sciencedirect.com/science/article/pii/S0957417421017395. – ISSN 0957-4174

[30] Liu, Xiaoqin ; Li, Xiang ; Shi, Qi ; Xu, Chuanpei ; Tang, Yanmei: UAV attitude estimation based on MARG and optical flow sensors using gated recurrent unit. In: *International Journal of Distributed Sensor Networks* 17 (2021), 04, S. 155014772110098

[31] Lucas, Bruce ; Kanade, Takeo: An Iterative Image Registration Technique with an Application to Stereo Vision (IJCAI), 04 1981

[32] Ludwig, Simone A. ; Burnham, Kaleb D.: Comparison of Euler Estimate using Extended Kalman Filter, Madgwick and Mahony on Quadcopter Flight Data. In: *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2018, S. 1236–1241

[33] Madgwick, S.O.H. ; Vaidyanathan, R. ; Harrison, A.J.L.: An Efficient Orientation Filter for Inertial Measurement Units (IMUs) and Magnetic Angular Rate and Gravity (MARG) Sensor Arrays / Department of Mechanical Engineering. URL http://www.scribd.com/doc/29754518/A-Efficient-Orientation-Filter-for-IMUs-and-MARG-Sensor-Arrays, April 2010. – Forschungsbericht

[34] Mahony, Robert ; Hamel, T. ; Pflimlin, Jean-Michel: Nonlinear Complementary Filters on the Special Orthogonal Group. In: *Automatic Control, IEEE Transactions on* 53 (2008), 07, S. 1203 – 1218

[35] Marins, J.L. ; Yun, Xiaoping ; Bachmann, E.R. ; McGhee, R.B. ; Zyda, M.J.: An extended Kalman filter for quaternion-based orientation estimation using MARG sensors. In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)* Bd. 4, 2001, S. 2003–2011 vol.4

[36] MDR: *Drohne liefert Medikamente: Testflüge in Wohngebiet in Dessau erfolgreich.* https://www.mdr.de/nachrichten/sachsen-anhalt/dessau/dessau-rosslau/drohne-lieferung-medikamente-arznei-apotheke-100.html

[37] Monte Lima, João P. Silva do ; Uchiyama, Hideaki ; Taniguchi, Rin-ichiro: End-to-End Learning Framework for IMU-Based 6-DOF Odometry. In: *Sensors* 19 (2019), 08, S. 3777

[38] NXP: *Tilt Sensing Using a Three-Axis Accelerometer.* https://www.nxp.com/docs/en/application-note/AN3461.pdf

[39] PyojinKim: *ARKit-Data-Logger.* https://github.com/PyojinKim/ARKit-Data-Logger. 2021

[40] pypi: *AHRS 0.3.1.* https://pypi.org/project/AHRS/

[41] Qin, Tong ; Li, Peiliang ; Shen, Shaojie: VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. In: *IEEE Transactions on Robotics* PP (2017), 08

[42] Qureshi, Umair ; Umair, Zuneera ; Hancke, Gerhard: Indoor Localization using Wireless Fidelity (WiFi) and Bluetooth Low Energy (BLE) signals, 06 2019, S. 2232–2237

[43] Rahni, Ashrani Aizzuddin A. ; Yahya, Iskandar: Obtaining translation from a 6-DOF MEMS IMU — an overview. In: *2007 Asia-Pacific Conference on Applied Electromagnetics*, 2007, S. 1–5

[44] Rambach, Jason ; Tewari, Aditya ; Pagani, Alain ; Stricker, Didier: Learning to Fuse: A Deep Learning Approach to Visual-Inertial Camera Pose Estimation, 09 2016

[45] Schubert, David ; Goll, Thore ; Demmel, Nikolaus ; Usenko, Vladyslav ; Stückler, Jörg ; Cremers, Daniel: The TUM VI Benchmark for Evaluating Visual-Inertial Odometry, 10 2018, S. 1680–1687

[46] Seo, Hong-Il ; Bae, Ju-Won ; Kim, Won-Yeol ; Seo, Dong-Hoan: DO IONet: 9-Axis IMU-Based 6-DOF Odometry Framework Using Neural Network for Direct Orientation Estimation. In: *IEEE Access* 11 (2023), S. 55380–55388

[47] Seong, J. H. ; Seo, D. H.: Selective Unsupervised Learning-Based Wi-Fi Fingerprint System Using Autoencoder and GAN. In: *IEEE Internet of Things Journal* 7 (2020), Nr. 3, S. 1898–1909

[48] Siciliano, Bruno ; Khatib, Oussama: *Springer Handbook of Robotics*. Berlin, Heidelberg : Springer-Verlag, 2008. – ISBN 978-3-540-23957-4

[49] ST: *STM32H742xI/G STM32H743xI/G Datasheet*. https://www.st.com/resource/en/datasheet/stm32h743vi.pdf

[50] Szczęsna, Agnieszka ; Skurowski, Przemysław ; Pruszowski, Przemyslaw ; Pęszor, Damian ; Paszkuta, Marcin ; Wojciechowski, Konrad: Reference Data Set for Accuracy Evaluation of Orientation Estimation Algorithms for Inertial Motion Capture Systems, 09 2016, S. 509–520. – ISBN 978-3-319-46417-6

[51] Wagstaff, Brandon ; Kelly, Jonathan: LSTM-Based Zero-Velocity Detection for Robust Inertial Navigation, 09 2018, S. 1–8

[52] Wang, Sen ; Wen, Hongkai ; Clark, Ronald ; Trigoni, Niki: Keyframe based large-scale indoor localisation using geomagnetic field and motion pattern, 10 2016, S. 1910–1917

[53] Weber, Daniel ; Guehmann, C. ; Seel, Thomas: Neural Networks Versus Conventional Filters for Inertial-Sensor-based Attitude Estimation. (2020), 05

[54] Weber, Daniel ; Guehmann, C. ; Seel, Thomas: RIANN - A Robust Neural Network Outperforms Attitude Estimation Filters. (2021), 04

[55] Wendel, Jan: *Integrierte Navigationssysteme*. München : Oldenbourg Wissenschaftsverlag, 2011. – URL https://doi.org/10.1524/9783486705720. – ISBN 9783486705720

[56] Xiang, Li ; Peng, Zhang ; Qi, Shi ; Yanmei, Tang: Three-dimensional attitude determination using vector observations and optical flow sensors. In: *Journal of Physics: Conference Series* 2005 (2021), 08, S. 012098

[57] Yan, Hang ; Shan, Qi ; Furukawa, Yasutaka: RIDI: Robust IMU Double Integration. (2017), 12

[58] Zeinali, Behnam ; Zandizari, Hadi ; Chang, J.: IMUNet: Efficient Regression Architecture for IMU Navigation and Positioning. (2022), 07

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original