Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Eirik Berg Samuelsen

Attitude Estimation with IMUs using Machine Learning

Master's thesis in Mechanical Engineering
Supervisor: Olav Egeland
June 2022

NTNU
Norwegian University of
Science and Technology

Eirik Berg Samuelsen

Attitude Estimation with IMUs using Machine Learning

Master's thesis in Mechanical Engineering
Supervisor: Olav Egeland
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Norwegian University of
Science and Technology

Preface

This master's thesis is the final requirement for the degree of Master of Science in Mechanical Engineering at the Norwegian University of Science and Technology. The thesis was written during the spring of 2022, under the supervision of Professor Olav Egeland.

I would like to thank Olav Egeland for the supervision throughout this thesis. The freedom to explore the subject of this thesis has been appreciated. His feedback and support during the project has been very helpful.

I would also like to thank my family for all the support given during the course of this thesis. I would especially like to thank my girlfriend for all the emotional support given during times facing difficulties in the project.

Abstract

Low cost Micro Electro Mechanical System (MEMS) Inertial Measurement Units (IMU) measure angular velocities and accelerations using gyroscopes and accelerometers. They are used in a wide variety of applications for estimating orientation, also referred to as attitude. The signals from the gyroscopes and accelerometers are typically corrupted with noise and time-varying bias. This reduces the quality in the estimated orientation. Estimating the orientation from open-loop integration typically yields poor estimates that quickly drifts from the true value with time.

This thesis builds on an existing method of using a Convolutional Neural Network (CNN) to obtain reliable estimates for the orientation. This method is compared to an existing conventional orientation estimation filter to investigate how good the CNN-based method performs. A new method of data augmentation is used together with the CNN-based method in an attempt to improve the performance of the CNN. New CNN architectures replacing this existing CNN are also tested, to further improve the performance of the method.

The existing CNN-based approach was proven to be very effective for orientation estimation when compared to the conventional orientation estimation filter. The use of the new technique for data augmentation did not produce consistently better results, and was therefore considered to be ineffective in further improving the performance. The use of new CNN architectures combined with the existing method managed to perform better than the original CNN, improving the method.

Sammendrag

Rimelige *Micro Electro Mechanical System* (MEMS) *Inertial Measurement Units* (IMU) måler vinkelhastigheter og aksellerasjoner ved bruk av et gyroskop og et aksellerometer. De er brukt i mange forskjellige situasjoner til å estimere orienteringer. Signalene fra gyroskopet og aksellerometeret inneholder typisk mye støy og tids-varierende bias. Dette bidrar til å redusere kvaliteten til de estimerte orienteringene. Dersom orienteringene estimeres ved å direkte integrere opp vinkelhastighetene, får man typisk dårlige estimater som fort, med tiden, vil avvike fra de faktiske orienteringene.

Denne oppgaven bygger på en eksisterende metode der det brukes et konvolusjonelt nevralt nettverk til å estimere orienteringer. Denne metoden sammenlignes med et konvensjonelt filter for estimering av orientering for å undersøke hvor god ytelse metoden basert på nevrale nettverk har. En ny metode av *data augmentation* brukes sammen med den eksisterende metoden basert på et nevralt nettverk, i et forsøk på å forbedre ytelsen til det nevrale nettverket. Videre brukes nye arkitekturer av konvolusjonelle nevrale nettverk til å erstatte det nevrale nettverket i den eksisterende metoden i et forsøk på å forbedre ytelsen til metoden ytterligere.

Metoden basert på et konvolusjonelt nevralt nettverk ble påvist å ha en veldig god ytelse sammenlignet med det konvensjonelle filteret. Den nye metoden av *data augmentation* klarte ikke å konsekvent produsere bedre resultater. Det ble derfor konkludert med at dette ikke bidro til å forbedre den eksisterende metoden. Bruken av nye nevrale nettverksarkitekturer kombinert med den eksisterende metoden ga bedre resultater enn det opprinnelige nevrale nettverket. Dermed forbedret bruken av nye nettverksarkitekturer metoden.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

Micro Electro Mechanical Systems (MEMS) Inertial Measurement Units (IMU) are compact devices measuring angular velocities and proper accelerations. This is measured by the onboard 3-axis gyroscope sensors and 3-axis accelerometers. Due to the small size and low cost of MEMS IMUs, they are practical for use in a wide variety of applications [13]. Typical applications include estimating the orientation of an object or a robot. In order to reliably estimate the orientation of an object using an IMU, filtering of the angular velocities and accelerations is required. It is possible to estimate the orientation from open-loop integration of the angular velocities. This typically results in a poor estimate that quickly drifts from the true value with time. These poor estimates results from corruptions in the IMU-measurements. The measurements are typically corrupted by time varying-bias and noise in the signals, as well as calibration factors including non-orthogonal axes of the sensors, axis-misalignments and scale factors [23].

Existing methods of estimating the orientation from an IMU include the Madgwick gradient descent optimization filter [18], Kalman Filters such as the Multiplicative Extended Kalman Filter [16] and Complementary filters [19].

Recently, Brossard et al. [2] proposed a method of estimating the orientation from IMU measurements using deep learning. They used a Convolutional Neural Network (CNN) to denoise the angular velocities, and obtained accurate estimates of the orientation by open-loop integration of these angular velocities.

1.2 Problem description

The aim in this thesis was to compare the performance of the method developed in [2] with traditional methods of orientation estimation, in order to gain insight

into how well the new method performs. Additionally, methods for improving the performance of the works of [2] are explored. This includes investigating whether different architectures of the CNN perform better.

Chapter 2

Preliminaries

This chapter was first presented in a preliminary study on the subject [24], but this theory is essential for understanding this thesis as well. Therefore it is also included in this report.

2.1 Rotation matrices, the $SO(3)$-group

This section presents rotation matrices from the 3D-rotation group called the $SO(3)$-group for representing rotations. The material is based on [17] and [5].

2.1.1 Rotation matrices

Rotation matrices, $\boldsymbol{R} \in \mathbb{R}^{3\times3}$, are part of the special orthogonal group, $SO(3)$. They represent orientations or act as operators on reference frames or vectors. Used as an operator, $\boldsymbol{R}$ either rotates a vector or a reference frame, or changes the frame of reference in which a reference frame or vector is represented. A rotation matrix satisfies

$$\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I} \tag{2.1}$$

$$\det(\boldsymbol{R}) = 1 \tag{2.2}$$

From the definition it follows that

$$\boldsymbol{R}^{-1} = \boldsymbol{R}^T \tag{2.3}$$

2.1.2 The Lie algebra $\mathfrak{so}(3)$

The Lie algebra of the $SO(3)$-group is the set of all 3×3 skew-symmetric matrices and is denoted $\mathfrak{so}(3)$. The skew symmetric representation of a vector $\boldsymbol{u} = [u_1, u_2, u_3]^T \in \mathbb{R}^3$ is defined as

$$\boldsymbol{u}^\times = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \tag{2.4}$$

2.1.3 The exponential map in $SO(3)$

A rotation can be described as rotating around a unit vector $\boldsymbol{k} \in \mathbb{R}^3$ by an angle $\theta \in \mathbb{R}$. Thus the pair $(\theta, \boldsymbol{k})$ is often referred to as the angle-axis-parameters of a rotation. From the definition of the matrix exponential function we have that

$$\boldsymbol{R} = \exp(\theta \boldsymbol{k}^\times) = \boldsymbol{I} + \sin\theta \boldsymbol{k}^\times + (1 - \cos\theta)\boldsymbol{k}^\times \boldsymbol{k}^\times \tag{2.5}$$

which is called the Rodrigues equation. Here $\boldsymbol{k}^\times$ is the skew-symmetric representation of $\boldsymbol{k}$, defined in equation (2.4). Therefore $\boldsymbol{R}$ is the matrix exponential of the matrix $\boldsymbol{u}^\times = \theta \boldsymbol{k}^\times$. In the case where the rotation matrix $\boldsymbol{R}$ is computed directly from the logarithm $\boldsymbol{u}^\times$, it is generally not recommended to use equation (2.5) directly, because of the risk of dividing by zero when recovering the angle-axis-parameters from the logarithm. In this case, the recommended expression is

$$\boldsymbol{R} = \exp(\boldsymbol{u}^\times) = \boldsymbol{I} + \operatorname{sinc}(\|\boldsymbol{u}\|)\boldsymbol{u}^\times + \frac{1}{2}\operatorname{sinc}^2\left(\frac{\|\boldsymbol{u}\|}{2}\right)\boldsymbol{u}^\times \boldsymbol{u}^\times \tag{2.6}$$

where $\operatorname{sinc}(\cdot)$ is the unnormalized sinc function,

$$\operatorname{sinc}(x) \triangleq \begin{cases} \sin x / x, & x \neq 0 \\ 1, & x = 0 \end{cases} \tag{2.7}$$

2.1.4 Kinematic differential equation

The time derivative of the rotation matrix describes the rate of change of orientation with respect to time. It is defined as

$$\dot{\boldsymbol{R}} = \boldsymbol{R}\boldsymbol{\omega}^\times \in T_R SO(3) \tag{2.8}$$

where $T_R SO(3)$ is the tangent space of the $SO(3)$-group. Here, $\boldsymbol{R}$ describes the rotation from a spatial frame to the body frame, and $\boldsymbol{\omega}^\times$ is the skew symmetric representation of the angular velocity in the body frame. The corresponding integration scheme based on increments in angular velocity becomes

$$\boldsymbol{R}_n = \boldsymbol{R}_{n-1}\exp(\boldsymbol{\omega}_n dt) \tag{2.9}$$

Here, $n = 1, 2, 3, ...$ denotes the time step and dt is the time increment between measurements. It is assumed that the angular velocity $\boldsymbol{\omega}_n$ is constant during dt.

2.1.5 The logarithmic map in $SO(3)$

Because $\boldsymbol{R}$ is the exponential of $\boldsymbol{u}^\times = \theta \boldsymbol{k}^\times$, the matrix $\boldsymbol{u}^\times$ is the logarithm of $\boldsymbol{R}$. The logarithm of a rotation matrix is calculated using

$$\theta = \arccos \frac{\mathrm{tr}\boldsymbol{R} - 1}{2}, \quad 0 < \theta < \pi \tag{2.10}$$

and

$$\log(\boldsymbol{R}) = \begin{cases} 0.5\left(1 + \frac{1}{6}\theta^2\right)(\boldsymbol{R} - \boldsymbol{R}^T), & \theta < \delta \\ \frac{\theta}{2\sin\theta}(\boldsymbol{R} - \boldsymbol{R}^T), & \delta < \theta < \pi \end{cases} \tag{2.11}$$

Where δ is chosen such that the error is less than machine precision.

2.2 Quaternions

This section presents the use of quaternions for representing rotations in 3D-space, and is based on [4].

2.2.1 Definitions and main properties

A quaternion can be represented as a 4-dimensional vector defined as

$$\boldsymbol{q} \triangleq \begin{bmatrix} \alpha \\ \boldsymbol{\beta} \end{bmatrix} = \begin{bmatrix} q_s \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \in \mathbb{R}^4 \tag{2.12}$$

Where $\alpha \in \mathbb{R}$ is the scalar part and $\boldsymbol{\beta} \in \mathbb{R}^3$ is the vector part of the quaternion. The sum of two quaternions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ is defined as

$$\boldsymbol{q}_1 \pm \boldsymbol{q}_2 = \begin{bmatrix} \alpha_1 \\ \boldsymbol{\beta}_1 \end{bmatrix} \pm \begin{bmatrix} \alpha_2 \\ \boldsymbol{\beta}_2 \end{bmatrix} = \begin{bmatrix} \alpha_1 \pm \alpha_2 \\ \boldsymbol{\beta}_1 \pm \boldsymbol{\beta}_2 \end{bmatrix} \tag{2.13}$$

The quaternion product between quaternions $\boldsymbol{q}_1$ and $\boldsymbol{q}_2$ is given by

$$\boldsymbol{q}_1 \circ \boldsymbol{q}_2 = \begin{bmatrix} \alpha_1\alpha_2 - \boldsymbol{\beta}_1^T\boldsymbol{\beta}_2 \\ \alpha_1\boldsymbol{\beta}_2 + \alpha_2\boldsymbol{\beta}_1 + \boldsymbol{\beta}_1 \times \boldsymbol{\beta}_2 \end{bmatrix} \tag{2.14}$$

which can also be written as

$$\boldsymbol{q}_1 \circ \boldsymbol{q}_2 = \boldsymbol{Q}_L(\boldsymbol{q}_1)\boldsymbol{q}_2 = \boldsymbol{q}_1\boldsymbol{Q}_R(\boldsymbol{q}_2) \tag{2.15}$$

where

$$\boldsymbol{Q}_L(\boldsymbol{q}) = \begin{bmatrix} \alpha & -\boldsymbol{\beta}^T \\ \boldsymbol{\beta} & \alpha\boldsymbol{I} + \boldsymbol{\beta}^\times \end{bmatrix} \tag{2.16}$$

and

$$\boldsymbol{Q}_R(\boldsymbol{q}) = \begin{bmatrix} \alpha & -\boldsymbol{\beta}^T \\ \boldsymbol{\beta} & \alpha\boldsymbol{I} - \boldsymbol{\beta}^\times \end{bmatrix} \tag{2.17}$$

The quaternion conjugate is defined by

$$\overline{\boldsymbol{q}} = \begin{bmatrix} \alpha \\ -\boldsymbol{\beta} \end{bmatrix} \tag{2.18}$$

The norm of a quaternion follows from the inner product of two quaternions

$$\boldsymbol{q}_1 \cdot \boldsymbol{q}_2 = \boldsymbol{q}_1^T\boldsymbol{q}_2 \tag{2.19}$$

and is given by

$$\|\boldsymbol{q}\| = \sqrt{\boldsymbol{q} \cdot \boldsymbol{q}} = \sqrt{\boldsymbol{q} \circ \overline{\boldsymbol{q}}} = \sqrt{\boldsymbol{q}^T\boldsymbol{q}} \tag{2.20}$$

The inverse quaternion $\boldsymbol{q}^{-1}$ is defined as

$$\boldsymbol{q}^{-1} = \frac{\overline{\boldsymbol{q}}}{\|\boldsymbol{q}\|^2} \tag{2.21}$$

2.2.2 Unit quaternions for representing rotations

A unit quaternion is a quaternion of magnitude $\|\boldsymbol{q}\| = 1$, denoted $\boldsymbol{q} = [\eta \quad \boldsymbol{\epsilon}]^T$. The unit quaternion can be described by the Euler parameters as

$$\boldsymbol{q} = \begin{bmatrix} \eta \\ \boldsymbol{\epsilon} \end{bmatrix} = \begin{bmatrix} \cos\frac{\theta}{2} \\ \boldsymbol{k}\sin\frac{\theta}{2} \end{bmatrix} \tag{2.22}$$

where $\boldsymbol{k}$ is a unit vector. From this it can be seen that a unit quaternion can represent a rotation of an angle θ around an axis $\boldsymbol{k}$, similarly to a rotation matrix $\boldsymbol{R}$. The rotation matrix $\boldsymbol{R}$ corresponding to the unit quaternion is then

$$\boldsymbol{R} = \boldsymbol{I} + 2\eta\boldsymbol{\epsilon}^{\times} + 2\boldsymbol{\epsilon}^{\times}\boldsymbol{\epsilon}^{\times} \tag{2.23}$$

2.2.3 The exponential map for quaternions

Similarly as for the $SO(3)$-group, the exponential map for quaternions maps the angle-axis parameters $\phi = \theta/2,\ \boldsymbol{k}$ to a quaternion. The expression for this, defined for all $\boldsymbol{v} = \phi\boldsymbol{k}$ is

$$\exp(\boldsymbol{v}) = \cos||\boldsymbol{v}|| + \operatorname{sinc}(||\boldsymbol{v}||)\,\boldsymbol{v} \tag{2.24}$$

where $\operatorname{sinc}(\cdot)$ is defined in equation (2.7).

2.2.4 Kinematic differential equation

The kinematic differential equation for quaternions is defined as

$$\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{q} \circ \boldsymbol{\omega} \tag{2.25}$$

where $\boldsymbol{q}$ is the quaternion representing the rotation from the spatial frame to the body frame, and $\boldsymbol{\omega}$ is the angular velocity in the body frame. The corresponding integration scheme based on increments in angular velocity becomes

$$\boldsymbol{q}_n = \boldsymbol{q}_{n-1} \circ \exp(0.5\boldsymbol{\omega}_n dt) \tag{2.26}$$

which is similar to equation (2.9).

Chapter 3

Method

3.1 Overview

In this chapter the method this study is based on [2] is presented in detail.

3.2 Modeling of kinematics and IMUs

The Farrenkopf model [6] is a widely used measurement model for modeling gyroscope measurements from an IMU, and is described by

$$\boldsymbol{\omega}^{\text{IMU}}(t) = \boldsymbol{\omega}(t) + \boldsymbol{b}(t) + \boldsymbol{n}_1(t) \in \mathbb{R}^3 \tag{3.1}$$

where $\boldsymbol{\omega}^{\text{IMU}}$ is the angular velocity measured by the IMU, $\boldsymbol{\omega}$ is the true angular velocity, $\boldsymbol{b}$ is the bias and $\boldsymbol{n}_1$ is zero-mean white noise. The bias can be described by the Wiener process

$$\dot{\boldsymbol{b}} = \boldsymbol{n}_2 \tag{3.2}$$

where $\boldsymbol{n}_2$ is zero-mean white noise. This is a simple model, and is well suited for use in attitude estimation filters such as the Multiplicative Extended Kalman Filter [20]. The method proposed by Brossard et al. [2] includes a more complex measurement model than this, since they also account for calibration of the IMU. They also include accelerations measured by the IMU in the measurement model. The expression in discrete form becomes

$$\boldsymbol{u}_n^{\text{IMU}} = \begin{bmatrix} \boldsymbol{\omega}_n^{\text{IMU}} \\ \boldsymbol{a}_n^{\text{IMU}} \end{bmatrix} = \boldsymbol{C} \begin{bmatrix} \boldsymbol{\omega}_n \\ \boldsymbol{a}_n \end{bmatrix} + \boldsymbol{b}_n + \boldsymbol{n}_n \in \mathbb{R}^6 \tag{3.3}$$

where $\boldsymbol{b}_n$ is the bias and $\boldsymbol{n}_n$ is zero-mean white noise. $\boldsymbol{a}_n$ is the accelerations in the IMU body frame. $\boldsymbol{C}$ is the intrinsic calibration matrix accounting for scale-factors, g-sensitivity and axis-misalignments. $\boldsymbol{C}$ can be written as

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{S}_\omega \boldsymbol{M}_\omega & \boldsymbol{A} \\ \boldsymbol{0}_{3\times 3} & \boldsymbol{S}_a \boldsymbol{M}_a \end{bmatrix} \approx \boldsymbol{I}_6 \tag{3.4}$$

where $\boldsymbol{S}_\omega \approx \boldsymbol{I}_3$ and $\boldsymbol{S}_a \approx \boldsymbol{I}_3$ represents scale factors, $\boldsymbol{M}_\omega \approx \boldsymbol{I}_3$ and $\boldsymbol{M}_a \approx \boldsymbol{I}_3$ represents axis misalignments and $\boldsymbol{A} \approx \boldsymbol{0}_{3\times 3}$ represents g-sensitivity. Accelerations in the body frame excluding the effects of gravity can be written as

$$\boldsymbol{a}_n = \boldsymbol{R}_{n-1}^T((\boldsymbol{v}_n - \boldsymbol{v}_{n-1})/dt - \boldsymbol{g}) \in \mathbb{R}^3 \tag{3.5}$$

where $\boldsymbol{g} \in \mathbb{R}^3$ is gravity and $\boldsymbol{v}_n \in \mathbb{R}^3$ is the IMU-velocity in the spatial frame.

By propagating the angular velocity using equation (2.26) or (2.9), we can get an estimate of the orientation at each time step. Which of these equations is used, is determined by whether we want the orientations to be represented in quaternions or rotation matrices. In the method proposed by Brossard et al. [2] the rotation matrix representation is used. Note that propagating $\boldsymbol{\omega}_n^{\text{IMU}}$ directly through one of these equations generally leads to a poor estimate that drifts from the true value with time. This is a result of the noise and calibration factors present in the measurement, modeled by equation (3.3).

3.3 Training a Convolutional Neural Network for estimating IMU corrections

3.3.1 Overview

The main component of the method proposed by Brossard et al. [2] is the use of a Convolutional Neural Network (CNN) in order to denoise the measurement signals from an IMU. These noise-free measurements are then propagated through equation (2.9) to estimate the orientation of the IMU. An overview of the learning process is visualized in Figure 3.1. The CNN receives measurements of accelerations and angular rates from the IMU for each time step. The output from the CNN is then an estimate of the correction to the gyroscope signals needed in order to obtain noise free gyroscope measurements. This correction is then combined with the gyro-rate measurement from the IMU to obtain noise free gyro rates which are integrated open-loop to obtain an estimated orientation for each time step. The CNN is then trained using the loss calculated from a loss function based on the ground truth and estimated orientations. A static calibration matrix

is also optimized during training to negate the effects $\boldsymbol{C}$ has on the measurements in equation (3.3). The reason why the loss is calculated based on orientations and not gyro-rates is because accurate ground truth angular velocities are not achievable to obtain, whereas accurate ground truth orientations are [2].

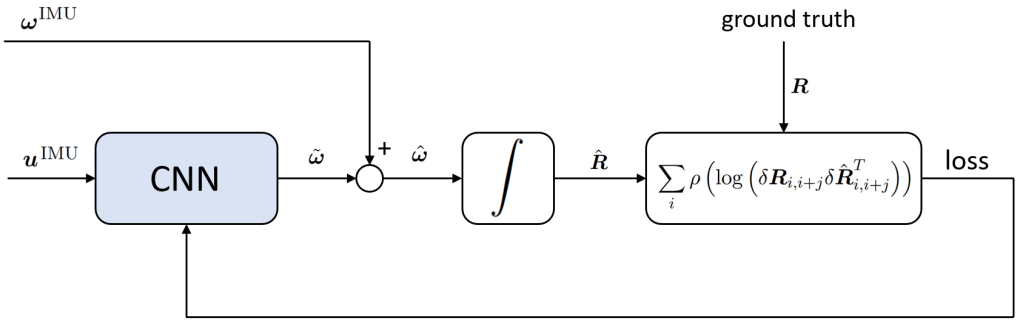


Figure 3.1: Training process for the CNN. Gyro-corrections are computed in the CNN using measurements of the angular velocities and accelerations from the IMU as inputs. The gyro-corrections are combined with the angular velocities from the IMU to produce angular velocities without noise. The angular velocities are then integrated to orientations in open-loop. These estimated orientations are compared to the ground truth orientation in order to calculate a value for the loss. The loss is then used to train the CNN. Figure initially presented in [24].

3.3.2 Modeling of noise free angular velocities from the IMU

The noise free angular velocities $\hat{\boldsymbol{\omega}}_n$ used to estimate the orientation are defined as

$$\hat{\boldsymbol{\omega}}_n = \hat{\boldsymbol{C}}_\omega \boldsymbol{\omega}_n^{\text{IMU}} + \tilde{\boldsymbol{\omega}}_n \tag{3.6}$$

which is based on equation (3.3). Here, $\hat{\boldsymbol{C}}_\omega = \hat{\boldsymbol{S}}_\omega \hat{\boldsymbol{M}}_\omega \in \mathbb{R}^{3\times 3}$ is the estimated static calibration matrix and $\tilde{\boldsymbol{\omega}}_n$ is the gyro-rate correction from the output of the CNN, which is modeled as

$$\tilde{\boldsymbol{\omega}}_n = \hat{\boldsymbol{c}}_n + \hat{\boldsymbol{b}} \tag{3.7}$$

where $\hat{\boldsymbol{c}}_n$ captures time-varying bias and noise, and $\hat{\boldsymbol{b}}$ captures static bias.

3.3.3 Convolutional Neural Networks for time series predictions

CNNs have been important in the field of computer vision for several years, ever since LeCun et al. proposed a CNN for classifying digits in 1998 [15]. More recently CNNs have been shown to be effective on time-series predictions as well, as demonstrated by [2] and [1]. One of the contributing factors for this success is the use of dilated convolutions.

The formal definition of a dilated convolution operation F on a 1-D time series $\boldsymbol{x} \in \mathbb{R}^n$ is given as [1]

$$F(s) = (\boldsymbol{x} *_d f)(s) = \sum_{i=0}^{k-1} f(i) \cdot \boldsymbol{x}_{s-d \cdot i} \tag{3.8}$$

Here $f : \{0, ..., k-1\} \to \mathbb{R}$ is the filter, k is the kernel size of the filter, d is the dilation factor and s is the element of the sequence the convolution is operating on. It is important that the convolutions in the CNN are causal. This means that each predicted output of the CNN is inferred only from past information of the inputs and not future information. In order to achieve this in the CNN architecture, zero-padding is used on the start of the sequence. Zero-padding is implemented by simply adding the amount of zeros needed to the start of the input vector. How many zeros that are needed, is computed from the kernel size and dilation of the current layer as

$$\text{padding} = d \cdot (k-1) \tag{3.9}$$

Figure 3.2 illustrates the use of padding and dilated convolutions and the effects it has on the dimensions of the output vector. As can be seen on the figure, the use of dilations increases the receptive field of the CNN. It is normal to increase the dilation exponentially for layers deeper in the network, which quite quickly gives a large receptive field and makes sure that each input to the CNN is included in the computation of the output. From the figure it is clear that the use of padding is essential to ensure that the CNN is causal

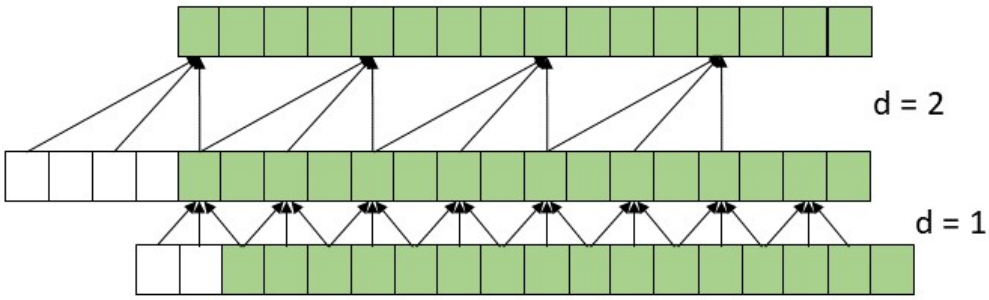


Figure 3.2: Dilated convolutions. The figure shows how the dilation gap in the convolutions produces a larger receptive field, allowing more values to contribute in calculating the current output. In the first layer, the receptive field is 3. in the second layer it has increased to 5. Zero-padding is added to the start of the sequence for each layer to ensure that the convolutions are causal, meaning that no information from future time steps are included in calculating the current time step.

3.3.4 Convolutional Neural Network for corrections of angular velocities

The learning method in [2] optimizes a static calibration matrix and trains a CNN to output gyro-corrections in real time. In relation to equation (3.6), $\hat{\boldsymbol{C}}_\omega$ is the optimized calibration matrix and $\tilde{\boldsymbol{\omega}}_n$ is the real-time gyro-corrections. The gyro-correction for each time-step n is calculated using the last N IMU-measurements. Measurements from time steps earlier than $n - N$ are not included in the calculation of $\tilde{\boldsymbol{\omega}}_n$. The magnitude of N is determined by the layer with the largest kernel size and dilation gap in the CNN,

$$N = \max(\text{kernel dimension} \times \text{dilation gap}) \tag{3.10}$$

and is called the receptive field of the CNN. The receptive field is thus a parameter which can be tuned for the best results possible. It was noted by [2] that keeping the receptive field relatively small mitigated the problem of overfitting to specific patterns in the training data, allowing the CNN to generalize better on unseen test data.

The kinematic equation used for integrating angular velocities to orientations, equation (2.9), does not include accelerometer measurements in estimating the orientations. Regardless of this, the CNN uses both angular velocities and ac-

celerometer measurements as inputs in order to estimate the gyro-corrections. It was stated in [2] that the inclusion of accelerometer measurements as inputs to the CNN reduced the orientation errors by 50% compared to using only angular velocities. An intuition for why this is the case can be derived from equation (3.5). When assuming the variations in velocity to be small between time steps we get

$$\boldsymbol{a}_{n+1} - \boldsymbol{a}_n \approx -(\boldsymbol{R}_n - \boldsymbol{R}_{n-1})^T \boldsymbol{g} \approx -(\exp(-\boldsymbol{\omega}_n dt) - \boldsymbol{I}_3)\boldsymbol{R}_{n-1}^T \boldsymbol{g} \tag{3.11}$$

We see that the accelerometer measurements contains some information about the angular velocity in this case. The variations in velocity is not assumed to be small in this method. Regardless, this relation gives some intuition that the accelerometer measurements contains information that can be utilized by the CNN. The relation between the gyro-corrections $\tilde{\boldsymbol{\omega}}_n$ and the function learned by the CNN is

$$\tilde{\boldsymbol{\omega}}_n = f(\boldsymbol{u}_{n-N}^{\text{IMU}}, ..., \boldsymbol{u}_n^{\text{IMU}}) \tag{3.12}$$

where $\boldsymbol{u}_{n-N}^{\text{IMU}}, ..., \boldsymbol{u}_n^{\text{IMU}}$ are the angular velocities and acceleration measurements from the IMU.

3.3.5 description of the CNN

The general architecture of the CNN is visualized in Figure 3.3. Table 3.1 summarizes the details of how each layer of the CNN is implemented. The dimension of the kernel operating on each layer is 7. From the table we have that the largest dilation gap is 64. Using equation (3.10) we see that the receptive field N of the CNN is $7 \times 64 = 448$. This corresponds to 2.24 seconds of measurements from the IMU at a sample rate of 200 Hz.

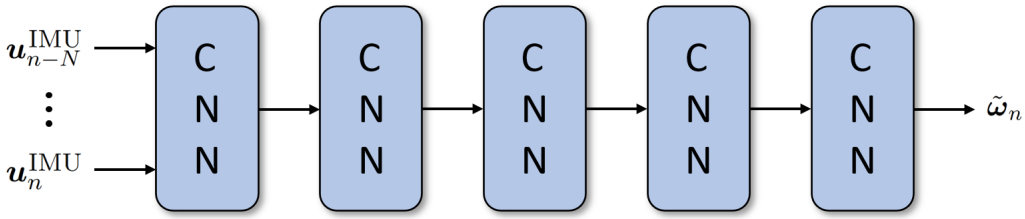


Figure 3.3: The CNN-architecture presented in [2]. Past measurements within the receptive field of 448 are included in calculating the current gyro-correction. Figure initially presented in [24].

The activation function used in the CNN is the non-linear Gaussian Error Linear Unit (GELU) [9], which is applied after each convolution operation. It is defined

as

$$\text{GELU}(x) = x\Phi(x) \tag{3.13}$$

Layer	Layer type	Number of filters	Dilation gap	Activation function
1	Conv1D	16	1	GELU
1	BatchNorm1D	-	-	-
2	Conv1D	32	4	GELU
2	BatchNorm1D	-	-	-
3	Conv1D	64	16	GELU
3	BatchNorm1D	-	-	-
4	Conv1D	128	64	GELU
4	BatchNorm1D	-	-	-
5	Conv1D	1	1	GELU
5	BatchNorm1D	-	-	-

Table 3.1: Visualisation of the layers of the CNN-architecture in [2]. A kernel dimension of 7 is used for all convolutional layers. Table initially presented in [24].

Here, $\Phi(x)$ is defined as the standard Gaussian cumulative distribution function. The difference between the GELU and the widely used Rectified Linear Unit (ReLU) activation functions is visualized in Figure 3.4. The main difference is that the GELU function is a smooth function where inputs both below and above 0 are weighted in the output, whereas the ReLU function is discontinuous and inputs below 0 are gated. Brossard et al. [2] observed that the CNN was more prone to overfitting to the training data when the ReLU function was used as opposed to when GELU was used. They assumed this was due to the discontinuity in the ReLU function, arguing that this caused the CNN to output too quick corrections for the gyro. They noted that corrections this quick does not make sense for physical signals.

A layer of Batch Normalization is used between the layers of the CNN, as displayed in Table 3.1. The function of a batch normalization layer is to normalize the output of the previous layer to a standard deviation of 1 and a zero mean. This is implemented to stabilize the training of the CNN [25].

Compared to other Visual-Inertial Odometry (VIO) methods, the number of trainable parameters in the CNN of this method is very low. VIO-methods often require more than 2 600 000 trainable parameters in total [2]. In contrast, the method of Brossard et al. requires 77 052 parameters to be trained. This means that the method requires far less computational resources both during training and for use

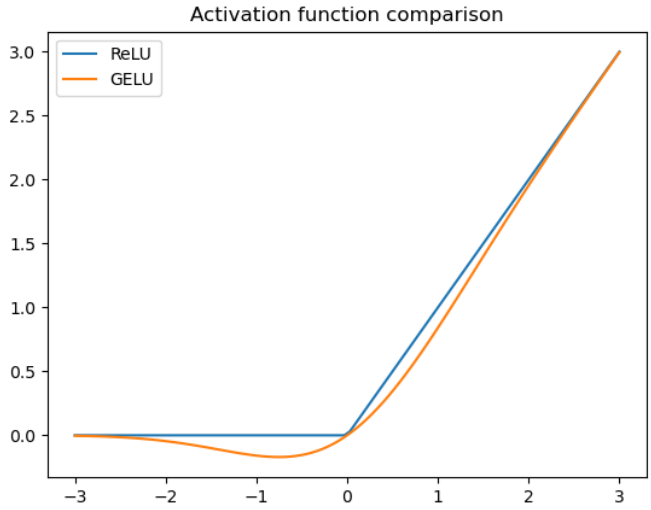


Figure 3.4: Comparison of GELU and ReLU activation functions. ReLU is discontinuous, and inputs less than 0 are set to 0. GELU is smooth and weighs values both above and below 0. Figure initially presented in [24]

in real time applications. This should give the method great value for the use in applications where computational resources are scarce.

The datasets used to train the CNN are relatively small, with 45 000 samples in the TUM VI [26] dataset and 90 000 samples in the EuRoC [3] dataset. The number of trainable parameters is therefore similar to the number of samples in the training data. It is therefore necessary to use techniques to avoid overfitting to the training data. For this reason, dropout and weight decay is utilized during training. Dropout works so that each channel of a layer in the network has a probability p of being set to zero during training. For this implementation the value of p is set to 0.1. This has the effect of limiting co-adaptions between the channels in the network [10], making each unit more independent of the activations other units in the network has. This should make each unit contribute more in making the correct predictions from the CNN.

3.3.6 Calculating loss

Calculating loss from angular velocities is not feasible, as mentioned in Subsection 3.3.1. The reason for this is that we need ground truth values corresponding to IMU measurements for all time steps in the time series. For the datasets used in this project, the IMUs have a sample rate of 200 Hz. The best accurate tracking

systems for angular velocities usually have a frequency of 20 – 120 Hz [2]. Due to this, the loss is instead based on the estimated orientations calculated using equation (2.9). To reduce the risk of overfitting to specific patterns in the training data, increments of rotation on the form

$$\delta \boldsymbol{R}_{i,i+j} = \boldsymbol{R}_i^T \boldsymbol{R}_{i+j} = \prod_{k=i}^{i+j-1} \exp\left(\boldsymbol{\omega}_k\right) \tag{3.14}$$

are used instead of calculating the loss from the orientation at every time step. The increments are down-sampled so that the IMU-frequency is reduced by a factor of j. Brossard et al. [2] stated that one advantage of using rotated increments on this form is that they are invariant to changes in the orientation and yaw-angle. This means that, for instance, left-multiplication of two rotation matrices with $\boldsymbol{R}$ does not change the value of the product of the two rotation matrices. The loss-function for these rotation increments is computed as

$$\mathcal{L}_j = \sum_i \rho\left(\log\left(\delta \boldsymbol{R}_{i,i+j} \delta \hat{\boldsymbol{R}}_{i,i+j}^T\right)\right) \tag{3.15}$$

where $\log(\cdot)$ is the logarithmic map defined in equation (2.11) and $\rho(\cdot)$ is the Huber loss function [12];

$$\rho(\epsilon) = \begin{cases} \frac{1}{2}\epsilon^2, & |\epsilon| \leq \delta, \\ \delta\left(|\epsilon| - \frac{1}{2}\delta\right), & \text{otherwise.} \end{cases} \tag{3.16}$$

where ϵ is the error between the ground truth and estimated value. This function is linear when the error is large and quadratic when the error is small. The Huber loss function is visualized in Figure 3.5, where it is plotted together with the Mean Square Error (MSE) and the Mean Absolute Error (MAE). Notice that the Huber function overlaps the MSE for small values and the MAE for large values. This is an appreciated property when dealing with datasets containing outliers in the ground truth data. If MSE was used instead, outliers would potentially have too much influence in the magnitude of the loss and negatively affect the training of the CNN.

For all experiments, a loss parameter $\delta = 0.005$ is chosen and the loss function used is

$$\mathcal{L} = \mathcal{L}_{16} + \mathcal{L}_{32} \tag{3.17}$$

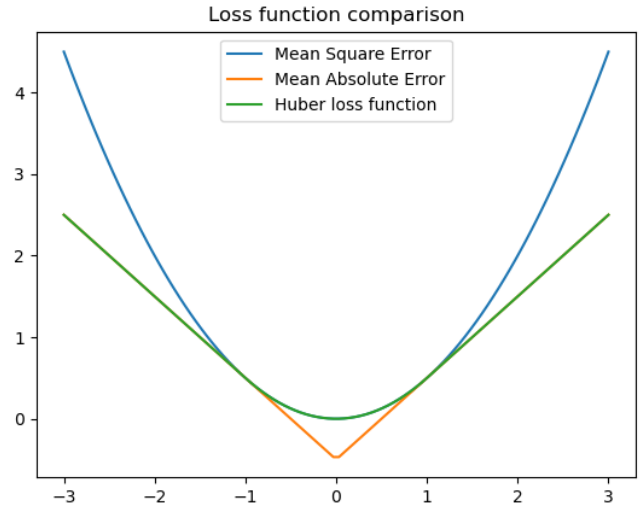


Figure 3.5: Comparison of the Huber loss function with the Mean Square Error (MSE) and Mean Absolute Error (MAE). For large values MAE and the Huber loss function coincides. For small values MSE and the Huber loss function coincides. This characteristic is beneficial on datasets with outliers in the ground truth values. Figure initially presented in [24].

Such that the increments used for computing the loss is reduced by a factor of 16 and 32.

3.3.7 Batch computations

Training a neural network demands great computational resources. When training using large datasets, the execution time is very important. In order to speed up the training process, parallelization is utilized so that several computations can be made in parallel at once. PyTorch is used to implement this in the form of batch operations on the GPU. For instance, when gyro-corrections are predicted by the CNN based on the IMU measurements, all gyro-corrections for each dataset is computed at once. Computations of kinematic equations and loss are also done in parallel to further decrease the execution time. The result is a significantly faster execution time compared to doing one computation at the time on the CPU.

In order to illustrate the effect of parallelization, a small experiment was conducted in the preliminary report preceding this thesis [24]. Equation (2.11) was used to compute the logarithm of the rotation matrix. It took 5 ms to compute the logarithm of one single rotation matrix. To compute one million rotation matrices

at once took 108 ms on an Nvidia GTX 950m GPU. This corresponded to 1.1×10^{-4} ms of time used per computation, which reduced the execution time with a factor of approximately 45 000. This details the significant performance increases gained from utilizing parallelization.

Another measure to improve the execution time is in the implementation of computing the rotated increments in equation (3.14). When computing this equation directly, this equation requires $j = 32$ computations if the loss function in equation (3.17) is used. Instead, Brossard et al. [2] viewed this equation as a *tree of matrix multiplications* [2], as shown in Figure 3.6. Here we can see that it takes only 2 computations to sub sample by a factor of 4. By this method, the necessary number of computations in equation (3.14) is reduced from $j = 32$ to $\log_2(j) = \log_2(32) = 5$.

The execution time is further improved by calculating the ground truth rotated increments only once, before training, and storing them for use during training. This is possible since the same increments for ground truth is used in the loss function every epoch.

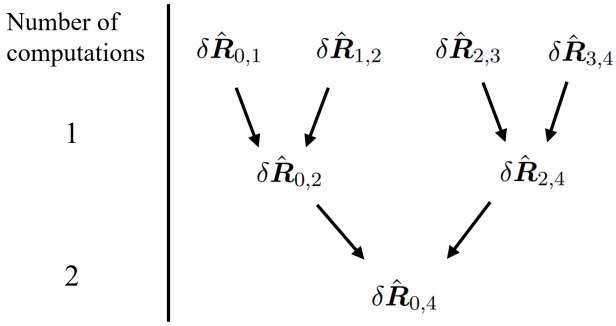


Figure 3.6: *Tree of matrix multiplications* [2]. The amount of computations necessary to down sample by a factor of j is reduced to $\log_2(j)$. In the figure, only 2 computations are necessary to down sample by a factor of 4. Figure initially presented in [24].

3.4 Evaluation metrics

Evaluating the performance of an orientation estimation algorithm is not trivial [28]. There are two main causes for this. The orientations (both estimated and ground truth) are collected for a large number of time steps. This implies that the collection of orientations we want to evaluate is inherently high dimensional. Furthermore, it is usually the case that the estimated orientation is expressed

in a different reference frame from the ground truth orientations. Zhang and Scaramuzza goes into detail on this topic in [28], and the evaluation methods used in [2] is highlighted there. Note that [2] used the openVINS toolbox [7] implemented in the Robot Operating System (ROS). ROS is a framework used for applications related to robotics. Since ROS was not used in this project, the openVINS toolbox was not used for evaluation either. Therefore the evaluation methods covered in this chapter was completely re-implemented in python for the use in this project.

Several different methods for evaluating a trajectory exists, and each method has its advantages and disadvantages. Two commonly used metrics are the absolute error and the relative error. These methods are detailed here. In order to achieve an informative evaluation it is recommended to use both the absolute error and the relative error in order to account for the advantages and disadvantages of each method.

3.4.1 Trajectory alignment

Alignment of the estimated trajectory to the same reference frame as the ground truth trajectory is necessary for both the absolute error and the relative error. The trajectory is aligned using the orientation of the first time step and is trivial to compute. The rotational transformation $\boldsymbol{R}^{'}$ between the ground truth and estimated orientation can be described by

$$\boldsymbol{R}^{'} = \boldsymbol{R}\hat{\boldsymbol{R}}^{T} \tag{3.18}$$

where $\boldsymbol{R}$ is the ground truth orientation and $\hat{\boldsymbol{R}}$ is the estimated orientation for the first time step. The estimated orientations can now be aligned with the ground truth trajectory by rotating the orientation for each time step using

$$\hat{\boldsymbol{R}}_{n}^{'} = \boldsymbol{R}^{'}\hat{\boldsymbol{R}}_{n} \tag{3.19}$$

where $\hat{\boldsymbol{R}}_{n}^{'}$ is the aligned estimated orientation.

3.4.2 Absolute Orientation Error

For computing the Absolute Orientation Error (AOE), the estimated orientations are aligned using the initial orientation as described in Subsection 3.4.1. Then, the AOE is computed as

$$\text{AOE} = \sqrt{\sum_{n=1}^{M} \frac{1}{M} \left\| \log\left(\boldsymbol{R}_n^T \hat{\boldsymbol{R}}_n \right) \right\|_2^2} \tag{3.20}$$

which is the Root Mean Square Error (RMSE) of the orientations. Here, $\boldsymbol{R}_n$ is the ground truth orientation, $\hat{\boldsymbol{R}}_n$ is the estimated orientation and M is the total number of time steps. $\log(\cdot)$ is defined in equation (2.11).

One advantage of the AOE is that just one single number is computed in order to quantify the performance of an entire estimated trajectory. This makes the metric easy to use for comparing the quality of several trajectories. Secondly, the AOE is relatively easy to implement. The main disadvantage of the AOE is that the magnitude of the error is sensitive to when in the trajectory the error occurs. Errors occurring earlier in a sequence will have a much greater impact on the AOE than errors occurring later in the sequence. Zhang et al. [28] cited several researchers observing this problem.

3.4.3 Absolute Yaw Error

For orientation estimation algorithms such as VIO methods the estimate of the yaw-angle tends to drift with time [2]. It is therefore of interest to evaluate the performance of the method of [2] for estimating the yaw-angle as well as the 3D-orientation. For doing this, a slight variant of equation (3.20) is used. First, the orientation between the estimate and ground truth at time step n is calculated as

$$\tilde{\boldsymbol{R}}_n = \boldsymbol{R}_n^T \hat{\boldsymbol{R}}_n \tag{3.21}$$

Then, the yaw-angle θ is retrieved from $\tilde{\boldsymbol{R}}_n$ by utilizing the relation between the rotation matrix and the corresponding roll-pitch-yaw-angles (ϕ, θ, ψ),

$$\boldsymbol{R}_n = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & c_\phi c_\psi + s_\phi s_\theta s_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \tag{3.22}$$

Where the notation $c_\theta, s_\theta,$ ... is short for $\cos\theta, \sin\theta,$ ... . From this, we have that

$$\frac{r_{21}}{r_{11}} = \frac{c_\theta s_\psi}{c_\theta c_\psi} = \tan\psi \tag{3.23}$$

which means that the expression for retrieving the yaw-angle for each time step is given by

$$\psi_n = \tan^{-1} \frac{r_{21}}{r_{11}} \tag{3.24}$$

The Absolute Yaw Error (AYE) is now defined as the RMSE of the yaw-errors as

$$\text{AYE} = \sqrt{\sum_{n=1}^{M} \frac{1}{M} \psi_n^2} \tag{3.25}$$

As is the case for the AOE, the magnitude of the AYE is also sensitive to when an error occurs in the trajectory.

3.4.4 Relative Orientation Error

The computation of the Relative Orientation Error (ROE) metric is much more involved than the AOE and AYE. The ROE is calculated for each sub-trajectory of a given length of an entire sequence. Suppose that the sub-trajectory length is set to be N meters. Then, start with the orientation at time step n. Now, find the time step where the IMU has traveled N meters from the position at time step n, defined as time step $g(n)$, and use the orientation at this time step. Now, align the estimated orientations using the ground truth orientation at time step n and the method described in Subsection 3.4.1 For this sub-trajectory, calculate the orientation increments for ground truth $\delta \boldsymbol{R}$ and the estimate $\delta \hat{\boldsymbol{R}}$ as

$$\delta \boldsymbol{R}_{n,g(n)} = \boldsymbol{R}_n \boldsymbol{R}_{g(n)}^T \tag{3.26}$$

$$\delta \hat{\boldsymbol{R}}_{n,g(n)} = \hat{\boldsymbol{R}}_n \hat{\boldsymbol{R}}_{g(n)}^T \tag{3.27}$$

Now use these increments to calculate the ROE for the given sub-trajectory as

$$\text{ROE} = \left\| \log \left(\delta \boldsymbol{R}_{n,g(n)}^T \delta \hat{\boldsymbol{R}}_{n,g(n)} \right) \right\|_2 \tag{3.28}$$

Next, take time step $n+1$ and find the time step where the IMU has traveled N meters from this time step, $g(n+1)$. Align the orientations and calculate the orientation increments and the ROE for this sub-trajectory. Repeat this until the ROE has been calculated for all sub trajectories of this length.

The ROE is thus not a single number representing the performance of the algorithm, as opposed to the AOE, but rather a collection of errors from all the sub-trajectories. This collection of errors can now be used to compute statistics such as the mean, median and percentiles, and the results can be visualized using box-plots.

It is advantageous to calculate the ROE for sub-trajectories of several different lengths N to get more informative metrics. This can be helpful to assess the quality of the estimate both over shorter and longer distances. The error over shorter distances is related to local consistency while the error over longer distances is related to long-term accuracy [28].

The main advantage of the ROE compared to the AOE is that the ROE is not sensitive to the time an error occurs in the trajectory. The ROE also offers much more flexibility in terms of the choice of sub-trajectory lengths used as well as more informative statistics produced. The main disadvantage is that the ROE is more complicated to implement than the AOE.

Chapter 4

Method development

4.1 Overview

This chapter describes further development of the method presented in Chapter 3. To gain insight into how well the method of [2] and the newly developed methods perform, a conventional filter [18] is presented for comparison. Then, a technique of data augmentation based on virtual rotations of the IMUs in the datasets is detailed. Lastly, two new CNN architectures are presented, which are used to replace the CNN in the method of [2], one CNN based on ResNet [8] and another CNN based on DenseNet [11].

4.2 Comparison with existing conventional filters

Compared to Visual Inertial Odometry-based methods for attitude estimation, the method presented in [2] looks very promising. It is, however, of interest to compare the performance of the method in [2] to the performance of a well tuned conventional filter for attitude estimation. This is because both methods only use an IMU to estimate the attitude. For this comparison the Madgwick gradient descent optimization filter [18] is used. The open-source implementation of this filter from [21], written in python, is used for this comparison.

In order to evaluate the performance of the filter, the AOE and AYE is used for comparison with the CNN-based approach. Another metric is also included in order to get a more detailed overview of the error for this filter. This metric is the Absolute Inclination Error (AIE) defined as [14]

$$\text{AIE} = \sqrt{\sum_{n=1}^{M} \frac{1}{M} \left(2 \cos^{-1} \sqrt{q_s^2 + q_3^2} \right)} \tag{4.1}$$

which gives the error of the rotation excluding the error in the yaw-angle, and is calculated using the quaternion representation. This metric is included since the Madgwick filter is expected to preform better with regards to the AIE than the AOE or AYE when using only a gyroscope and accelerometer as inputs. The evaluation of how such a filter performs when compared to the method of [2] is meant to provide valuable insight into how well this method performs compared to more traditional methods.

4.3 Data augmentation

Data augmentation in machine learning is a technique used to make a dataset appear larger and more diverse than it originally is. It has for long been a popular technique in the field of computer vision. An important use case is when the original dataset is relatively small, often small enough that good results would be unobtainable without using regularization techniques. In computer visions, techniques for data augmentation include mirroring of images, tweaking of the color balance of the image and randomly rotating and cropping the image.

In [2] Gaussian noise was added to the gyro-rates and accelerations from the IMU as a technique of data augmentation. The added noise had a standard deviation of $0.01°/s$. In [27], Weber et al. compared an end-to-end Recurrent Neural Network (RNN) to conventional filters for estimating roll and pitch angles using gyro-rates and accelerations from an IMU. They reported a significant improvement in the performance of the RNN when applying a random virtual orientation to the IMU as a data augmentation technique. Inspired by these results, the data augmentation technique using virtual rotations is implemented while training the CNN of [2] to investigate if an equal improvement in performance can be observed here as well.

4.3.1 Data augmentation using virtual rotations

Virtually rotating the IMU simulates the case where an IMU is mounted imperfectly on a rigid-body. The technique is implemented by both rotating the angular velocities and accelerations from the IMU and the ground truth orientations by the same rotation $\boldsymbol{R}$. During training, this is implemented as follows: For each training epoch, a random roll, pitch and yaw angle (ϕ, θ, ψ) is generated for each sequence in the training dataset. These angles are generated to be within the same set range of values. Using these random angles, the corresponding rotation matrix is calculated using

$$\boldsymbol{R} = \begin{bmatrix} c_\theta c_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & c_\phi c_\psi + s_\phi s_\theta s_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix} \tag{4.2}$$

Now, the angular velocities, accelerations and ground truth orientations for all time steps in an entire sequence is rotated by this $\boldsymbol{R}$.

4.4 Use of different neural network architectures

It is of interest to replace the CNN in the method of [2] with other state of the art CNN architectures and observe if it is possible to get even better orientation estimates. Two different architectures have been tested to explore if this is possible. A CNN based on ResNet [8] and a CNN based on DenseNet [11].

4.4.1 Residual neural network

An important contribution in computer vision was the CNN-structure ResNet [8]. This CNN won the 1st place on several competitions, including the ImageNet competition in 2015. The use of a similar structure as ResNet has been used by Bai et al. [1] for the task of sequence modeling, with good results. Here a CNN was used to beat the performance of Recurrent Neural Networks on a large variety of different sequence modeling benchmarks. The good results obtained amongst others by these authors have inspired the use of a similar structure for the CNN used in this project. This type of CNN-architecture is used instead of the CNN presented in Subsection 3.3.5 for the relevant experiments.

In Figure 4.1 the structure of a Residual CNN is visualized. Here, each CNN-block represents the structure shown in Figure 4.2. Each block consists of two convolutional layers. The activation function used for each convolutional layer is the GELU activation function. Furthermore, batch normalization and dropout is applied to each layer. As shown in these two figures, the main component of a residual network is that the output of each block is added to the input of the block using skip connections that bypasses the convolutions and non linear functions. Because of this, the CNN does not need to learn the full transformation from the input to the output. Rather, the layers of the CNN now learns small modifications to the inputs of each layer instead. This mechanism speeds up the learning process and enables the use of very deep neural networks. This method allowed [8] to train a network with 152 layers and still achieve good results on computer vision related tasks.

In the implementation of the Residual CNN in this project, the channel dimensions

change within each block. This means that the output of each block has a different channel dimension from the input. Hence, the input and the output can no longer be added together. To mitigate this problem, a 1×1 convolutional layer is used to scale up the channel dimension of the skip connections before the addition while maintaining an identity mapping, as shown in Figure 4.2. The scaling up of the channel dimension happens in the first convolutional layer of each block. Each layer of each block then uses the same dilation gap and has the same channel dimension of each output.

In order to get an output from the CNN that has the correct dimensions for the gyro-corrections, $\tilde{\boldsymbol{\omega}}_n \in \mathbb{R}^3$, a 1×1 convolution layer is applied after the final CNN-block. This layer changes the dimension of the outputs to $\mathbb{R}^3$ and scales down the channel dimension to 1.

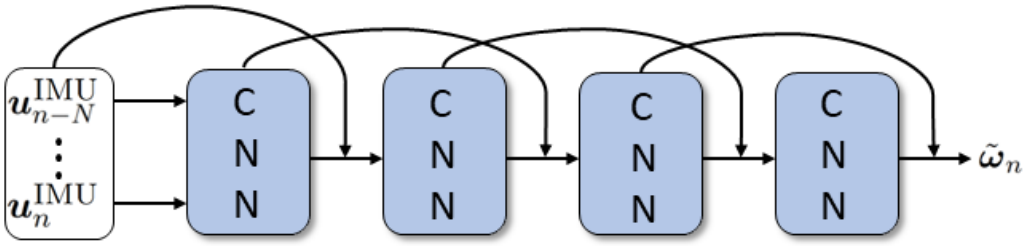


Figure 4.1: Residual network architecture. The input to each CNN-block is added to the output. Therefore each CNN-block learns modifications on identity mappings from the input as opposed to the entire transformation from input to output.

4.4.2 Dense neural network

DenseNet [11] is another important contribution to the field of computer vision. The general architecture of a Dense network is illustrated in Figure 4.3. The architecture is similar to the architecture of the Residual network, with the exception that every layer is connected to every subsequent layer in the network, using skip connections. As was the case for the Residual network, this also allows the use of very deep networks. This mechanism should improve the flow of information when compared to the Residual network [11]. Now each layer in the CNN receives information from every preceding layer in the network.

Similarly to the Residual network, each CNN-block in Figure 4.3 contain the same layers as the layers shown in Figure 4.2. The channel dimensions and dilation gap are also the same for both convolutional layers in each block. It is also necessary in the Dense network to use a 1×1 convolution for each skip-connection in the

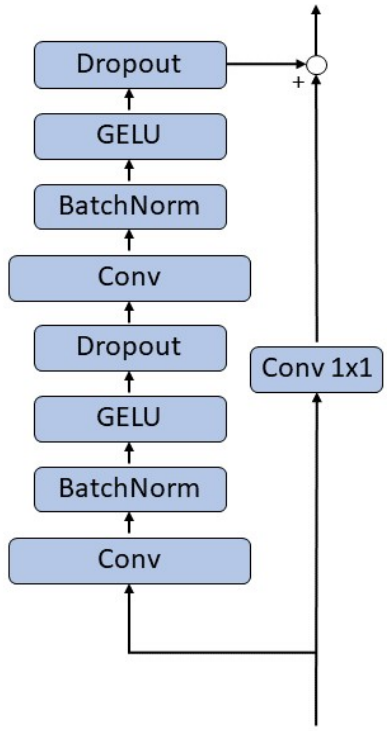


Figure 4.2: Residual block. Each block contains two convolutional layers. The skip-connections from the input contain a 1×1 convolutional layer to ensure matching channel dimensions when the skip-connections are added to the outputs of the convolutional layers.

network in order to obtain the correct channel dimension. As for the Residual network, a final 1×1 convolutional layer is applied to ensure the correct dimensions of the output of the network.

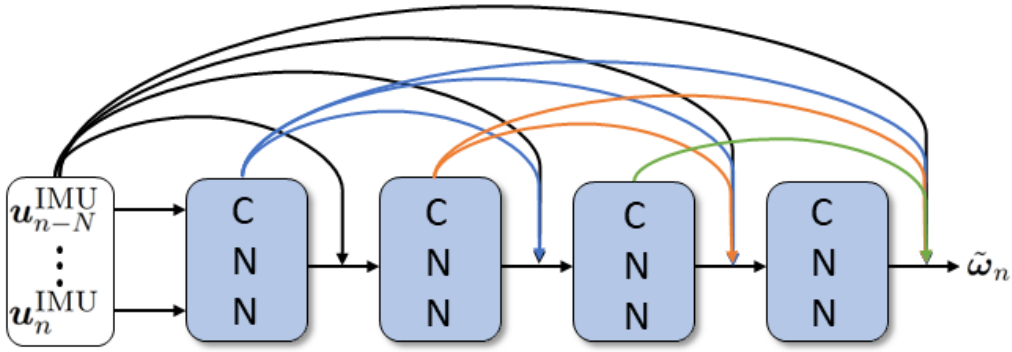


Figure 4.3: Dense network architecture. Every layer in the network is connected to every subsequent layer. Each layer in the network thus receives information from every preceding layer.

Chapter 5

Experimental trials, results and discussion

5.1 Overview

This chapter starts with detailing the reproducibility of results when training neural networks using PyTorch. Then a brief description of the datasets used in the experiments follows. Experimentation done with regards to the new methods presented in Chapter 4 is thereafter detailed: Firstly, experimentation from using the Madgwick filter is presented and compared to the method of [2]. Then an analysis of the effects of data augmentation on the datasets follows. Lastly, the use of new CNN architectures replacing the CNN in [2] is studied.

5.2 Reproducible results in deep learning

During early testing and validation of the method of Brossard et al. [2] it was observed that the results varied significantly for each time the CNN was trained, even when all parameters were kept exactly equal. For instance, considering the results of the AOE obtained on the TUM VI dataset, the results varied between 1.56° to 4.93°. The same variation was observed for the ROE. It was furthermore impossible to reproduce the exact results obtained in [2].

The explanation of this variation is found in the PyTorch documentations regarding reproducibility [22]. Reproducible results are not guaranteed across platforms or PyTorch releases. Reproducible results are not guaranteed between GPU or CPU executions either, as was the observation made here. It is, however, possible to limit the nondeterministic behaviour in order to obtain reproducible results between executions on the same device. First, the seed value for generating random numbers is held constant between executions. Furthermore it is specified that

PyTorch uses only deterministic algorithms. These measures have been tested to ensure reproducible results between executions on the same GPU. Using different GPUs between executions of the same program did not produce reproducible results. Therefore all experiments has been done on the same GPU, an Nvidia Tesla V100. For the specific seed values used in these experiments, the CNN model developed by Brossard et al. achieves slightly worse results here than those obtained in [2]. This detail is not considered to be important. It is much more important that the results are consistent between executions. This ensures that differences in the performance observed when modifying the method of [2] is caused only by the modifications made, and not by the random behaviour of training the CNN. The PyTorch version used for all experiments is PyTorch 1.10.2, and the operating system used is CentOs Linux 8.2.2004.

5.3 Dataset descriptions

The datasets used for all experiments are the EuRoC dataset [3] and the TUM VI dataset [26]. These are the same datasets that were used by Brossard et al. [2]. These datasets were used to keep conditions of the experiments as similar as possible to the original work and limit potential sources of error.

5.3.1 TUM VI dataset

The TUM VI dataset [26] has been captured using a hand-held device containing a sensor-suite of stereo cameras and an IMU. The IMU is a Bosch BMI160 recording 3-axis accelerometer readings and gyroscope-rates at a sample rate of 200 Hz. For acquisition of ground truth orientations and positions several infrared reflective markers has been placed on the device. An Optitrack motion capture system tracks these reflective markers in order to capture ground truth poses at a rate of 120 Hz. The motion capture system consists of 16 Flex13 infrared cameras.

The sequences in the dataset has been captured in a diverse set of sceneries, including indoors and outdoors. Of these sequences, only the sequences labeled *room* are the ones containing ground truth poses for the entire sequence. Hence, these sequences are the ones used in this project, since it is vital to have ground truth available for the method of machine learning used in this project. This constitutes 6 sequences, each lasting between 2–3 minutes. The CNN is trained using the first 50 seconds of the sequences *room 1*, *room 3* and *room 5*. The remainder of each sequence is used for validation. Sequences *room 2*, *room 4* and *room 6* are left as the test dataset. It is noted that the IMU is properly calibrated.

5.3.2 EuRoC dataset

The EuRoC dataset [3] has been captured using an AscTec Firefly hex-rotor Micro Aerial Vehicle (MAV) containing a sensor-setup. The sensor-suite consists of a stereo camera arrangement and an ADIS16448 IMU capturing accelerometer values and gyroscope-rates at 200 Hz. Two systems for capturing ground truth poses has been used. A Leica Nova MS50 laser tracker was used to capture the position of a prism mounted on top of the MAV at a rate of 20 Hz for some sequences. For other sequences a Vicon motion capture system was used to capture pose measurements from reflective markers mounted on the MAV at a rate of 100 Hz.

The dataset consists of 11 sequences lasting between 2–3 minutes each. For this dataset all sequences contain ground truth positions and orientations, and therefore all sequences of the dataset have been used in this project. Five sequences were captured in a Machine Hall, labeled MH 01, ..., MH 05. The remaining six sequences were captured in a smaller room and labeled V1 01, V1 02, V1 03, V2 01, V2 02 and V2 03. Each sequence also has a label of easy, medium or difficult, symbolising the difficulty of each sequence. Of these sequences, the first 50 seconds of sequences MH 01, MH 03, MH 05, V1 02, V2 01 and V2 03 were used for training the CNN. The remainder of these sequences were used for validation. The remaining 5 sequences were used for testing the CNN. In this dataset, the IMU is not properly calibrated, as is reflected in the results when compared to the results from the TUM VI dataset.

5.3.3 Similarities and differences between EuRoC and TUM VI

To fully understand and compare the results between the performance of the CNN on the EuRoC and TUM VI datasets, it is important to understand how the datasets are similar, and how they differ. The most notable difference between the datasets is that the IMU in the TUM VI dataset has been properly calibrated by the authors of [26] while the IMU in EuRoC has not been calibrated. For the TUM VI dataset, this calibration includes correction in static bias as well as compensation for scale factors and axis misalignment. It was noted by Brossard et al. [2] that calibrating the IMU leads to a much better performance (this was said in the context of using a training method for calibrating the IMU, but the same holds true for conventional methods of calibration as well). How much effect calibration has on the results is reflected in the orientation error obtained simply by open loop integration on each dataset, where EuRoC has an average AOE of 120.9° whereas TUM VI has an average AOE of 6.26°.

Other differences between the datasets includes differences in the device the IMUs are mounted on. For the TUM VI dataset, the IMU is mounted on a hand-held

device. For the EuRoC dataset, it is mounted on a rotor-driven MAV. Hence, the IMU-measurements on the EuRoC dataset should contain more noise than the TUM VI dataset, coming from the rotors. The rotors should produce colored noise [2] which in theory would be harder to estimate. The datasets are, however, similar in the movement patterns captured. Both datasets contain mostly translational movement patterns.

5.4 Comments on evaluation metrics

Some measures have been taken in order to ease the comparison of the performance between several executions. If nothing else is stated, this is how the metrics are reported: When the AOE and AYE is reported, this is the average AOE/AYE over all sequences in the dataset. The ROE is calculated for sub-trajectory distances of 7, 21 and 35 meters. The ROE is then scaled for each value by distance, in order to get one collection of errors in degrees per meter. This collection of errors is then used to produce box plots and to compute the mean ROE for use in tables.

5.5 Madgwicks gradient descent optimization filter

5.5.1 Experimentation details

In order to get a better understanding of the results from the method of [2], it is beneficial to compare it to well established conventional methods of attitude estimation. As mentioned in Section 4.2, the conventional filter used for this comparison is the Madgwick gradient descent optimization filter [18]. This filter only has one tuning parameter, β. In order to find the optimal tuning, the filter is applied to all sequences using a tuning parameter β in the range 0.01 to 0.3 in steps of 0.01 for the EuRoC dataset and 0.0001 to 0.01 in steps of 0.000165 for the TUM VI dataset. Then the average AOE, AYE and AIE from all sequences are plotted against each value of β in order to find the optimal tuning for each error metric. The range of tuning parameters suited for use was initially found with some trial and error. Once an acceptable parameter was found, fine tuning the filters was done using the ranges specified above.

5.5.2 Results and discussion

The plots of the errors for different tuning parameters β for EuRoC and TUM VI can be seen in Figures 5.1 and 5.2, respectively. These figures visualize how tuning of the filter was performed on both datasets. In Table 5.1 the AOE, AYE and AIE is compared for the results of open loop integration, the method

of [2] and the Madgwick filter. Note that the metrics provided in the table for the Madgwick filter are the values corresponding to the optimal tuning for each respective metric. In Figures 5.1 and 5.2 it can be seen that the optimal tuning parameter β for the AOE, AYE and AIE, respectively, does not share the same value. It is evident that the Madgwick filter fails to estimate the 3D-orientation when considering the AOE for both datasets. On the EuRoC dataset a slight decrease in the error is observed compared to open loop integration. On the TUM VI dataset the Madgwick filter achieves the exact same error as open loop integration. The reason why the the error decreases for the EuRoC dataset and not for the TUM VI dataset is most likely related to the fact that the IMU in TUM VI is calibrated while the IMU in EuRoC is not. Most likely the filter is able to attenuate some of the noise present in the EuRoC dataset, which leads to better results for the AOE compared to open loop integration.

The source of the poor estimates for the 3D-orientation is found in the results for the AYE. Surprisingly, open loop integration actually yielded better performance than the Madgwick filter on both datasets. Figures 5.1 and 5.2 show that the filter does not manage to provide good estimates of the yaw-angle, regardless of the value of β. The reason for these poor results stem from the fact that we use an IMU with only a gyroscope and an accelerometer. Without another sensor such as a magnetometer, the only direction vector we have for corrections in the estimate is the direction of gravity measured by the accelerometer. Therefore the filter has no observations that can be used to correct for the yaw angle, which leads to poor results with regards to the AYE, and, ultimately, also the AOE.

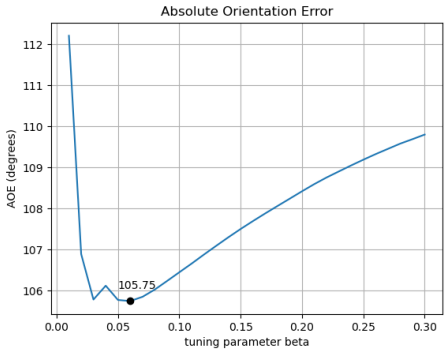
Contrary to the results obtained from open loop integration and the Madgwick filter, the CNN of [2] achieves very good results for the AOE and AYE, and the error is reduced substantially on both datasets. The CNN has been trained by minimizing the loss for the 3D-orientation. It is evident that good estimates of the 3D-orientation rely on good estimates of the yaw-angle as well, and that minimizing the loss for the orientation generally leads to good estimates of the yaw-angle. As discussed, the Madgwick filter does not manage to estimate the yaw angle since it is not able to utilize information from the gyroscopes or accelerometers to infer this angle. The results for the CNN proves that the gyroscopes and accelerometers actually contain some information of the yaw-angle. This demonstrates one of the advantages of using a CNN. The CNN is able to find relations in the IMU measurements that are not modeled by the kinematic equations. Some intuition for how the accelerometer measurements relate to the angular velocities was provided in equation (3.11), but this relation was derived by assuming negligible velocity variations between time steps, which is an invalid assumption on these datasets. The results for the CNN in Table 5.1 indicate that a similar relation exists also when velocity variations are present between time steps.

The results for the AIE in Table 5.1, which captures the errors in the roll and pitch angles, are more favourable for the Madgwick filter. Very similar performances between the CNN and the Madgwick filter is achieved on the TUM VI dataset, and slightly better results are achieved for the CNN on EuRoC. As expected, the Madgwick filter does a really good job of estimating the roll and pitch angles when compared to open loop integration. The CNN, however, manages to provide similar or better results for the AIE even though the CNN was trained to estimate the 3D-orientation. A surprising observation is the AIE for open loop integration for the TUM VI dataset, which achieves an error of 1.02 degrees. This indicates that very good estimates of the roll and pitch angles are possible to obtain simply by open loop integration from a well calibrated IMU.

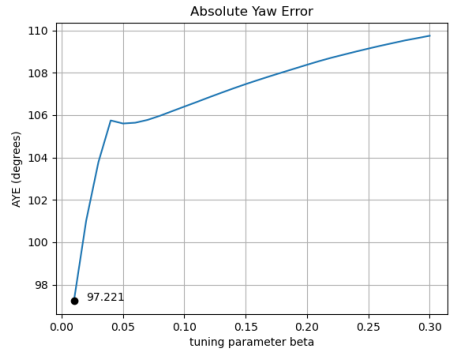
In light of the results obtained from the Madgwick gradient descent filter, it becomes evident just how successful the method of [2] is at estimating the 3D-orientation. Since the method of [2] is based on open-loop integration of denoised gyro-rates, no corrections in the estimated orientation is possible to apply in real time. The performance is therefore completely dependent on how good the gyro-corrections from the CNN are so that drift in the estimated orientations from the ground truth orientations is avoided. Despite this, the method of [2] manages to get excellent results on the TUM VI and EuRoC datasets.

	AOE (deg)			AYE (deg)			AIE (deg)		
	open loop	CNN [2]	Madgwick	open loop	CNN [2]	Madgwick	open loop	CNN [2]	Madgwick
EuRoC	120.9	2.88	105.8	91.2	1.80	97.2	82.1	2.64	3.27
TUM VI	6.26	1.56	6.26	5.91	1.38	6.22	1.02	0.56	0.55

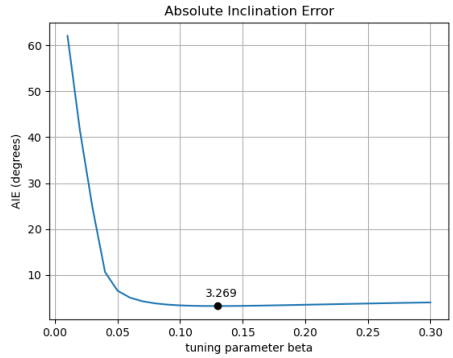
Table 5.1: Results comparing the Madgwick filter with the CNN from [2] and open loop integration of angular velocities. The Madgwick filter does not perform well with regards to the AOE and AYE. Good performance is achieved for the filter with regards to the AIE. The CNN achieves excellent results for all metrics.



(a) AOE EuRoC Madgwick filter

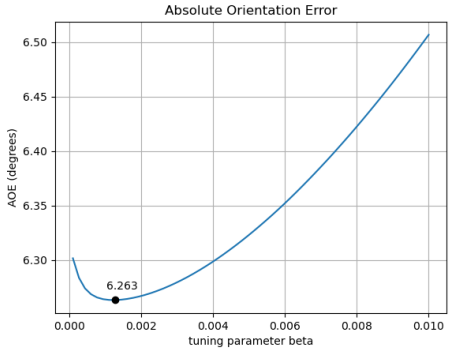


(b) AYE EuRoC Madgwick filter

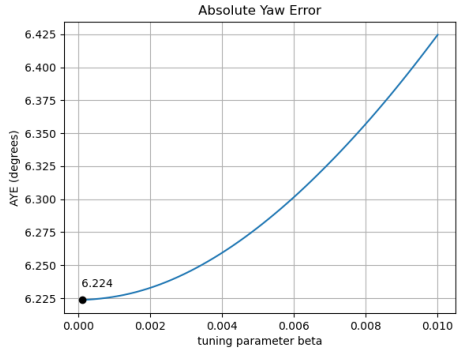


(c) AIE EuRoC Madgwick filter

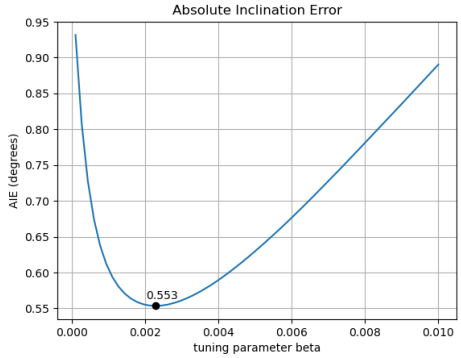
Figure 5.1: AOE, AYE and AIE on the EuRoC dataset using the Madgwick filter. The optimal tuning of the filter obtained an AOE, AYE and AIE of 105.75, 98.22 and 3.27 degrees respectively. Hence the filter only obtained good results for the AIE, representing the errors in the roll and pitch angles. The filter failed to estimate the yaw-angle, and as a result, the full 3D-orientation.



(a) AOE TUM Madgwick



(b) AYE TUM Madgwick



(c) AIE TUM Madgwick

Figure 5.2: AOE, AYE and AIE on the TUM VI dataset using the Madgwick filter. The optimal tuning of the filter obtained an AOE, AYE and AIE of 6.26, 6.22 and 0.55 degrees respectively. As for the EuRoC dataset, the filter only obtained good results for the AIE, representing the errors in the roll and pitch angles. The filter failed to estimate the yaw-angle, and as a result, the full 3D-orientation.

5.6 The effects of data augmentation

5.6.1 Experimentation details

The experimentation of the effects of data augmentation on the method in [2] was performed by analysing two different types of data augmentation. The effects virtual orientations had on the performance was the main focus of the experiments. It is, however, possible to apply too many techniques of data augmentation at once. Applying too many of these techniques at once can degrade the performance of the CNN. For this reason, every experiment performed with respect to virtual rotations, was done both with and without the method implemented in [2] of adding Gaussian noise to the IMU measurements as a data augmentation technique.

As was noted in Section 4.3, the orientation $\boldsymbol{R}$ used to rotate the IMU and ground truth data was calculated from roll, pitch and yaw angles generated from within a specified range of values. During early testing it was observed that how large this range of values was set to be, had a significant impact on the performance of the CNN. For this reason, a wide variety of ranges for these rotation angles were tested. The results are summarized in box-plots for each of these rotation ranges in Figures 5.3 and 5.4. In these plots, a virtual rotation range of for instance 5 deg means that $\boldsymbol{R}$ was constructed from roll, pitch and yaw angles with values between -5 and 5 degrees for each epoch during training. In the results presented from these experiments, the results from the original method of [2] is referred to as baseline.

5.6.2 Effects of data augmentation on EuRoC

Figure 5.3 shows the effects of different magnitudes of virtual rotation augmentation on the ROE, both with and without added noise to the IMU-measurements. For a more detailed analysis, Table 5.2 is included, which contains the AOE, AYE and mean ROE for each execution. In this comparison, the method proposed by Brossard et al. [2] is termed baseline. From the figure we can see that the best performance was achieved using the baseline method with noise to the IMU-input, which is the exact method proposed in [2]. Using the baseline method without added noise produced slightly worse results. When applying a virtual rotation to the IMU, two distinct patterns emerge. Firstly, for both executions with and without added noise to the IMU, the ROE increases significantly as the range of the virtual orientations increase. This pattern is confirmed in Table 5.2 for both the AOE, AYE and mean ROE. Secondly, for most of the executions, better performance is achieved by using virtual orientations only, as opposed to both adding noise and virtual orientations at the same time. When using a small range of virtual orientations only, the performance is similar to or slightly worse than

the results from baseline.

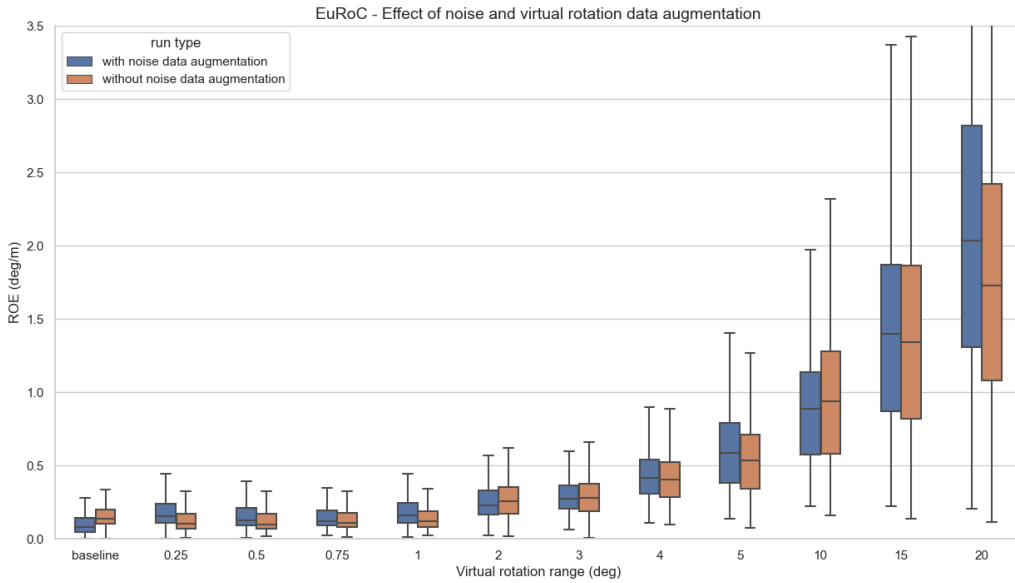


Figure 5.3: ROE on EuRoC using virtual rotation data augmentation. The best results for the ROE was obtained from the baseline configuration with Gaussian noise on IMU measurements as data augmentation. Increasing the range of rotations when constructing the virtual rotation led to gradually worse results.

	AOE/AYE (deg)		mean ROE (deg/m)	
	With noise	Without noise	With noise	Without noise
open loop	-	120,92/91,2	-	5,25
baseline	**2,88/1,8**	4,75/3,13	**0,13**	0,18
±0.25°	6,24/4,17	3,98/2,37	0,2	0,15
±0.5°	5,63/3,76	4,44/2,48	0,18	0,16
±0.75°	5,71/2,95	5,17/2,62	0,18	0,17
±1°	6,7/3,93	5,15/2,48	0,21	0,17
±2°	10,88/5,14	11,93/6,35	0,29	0,32
±3°	11,77/4,86	11,91/5,67	0,33	0,33
±4°	18,11/7,19	17,13/7,28	0,48	0,46
±5°	25,32/14,06	21,44/10,14	0,65	0,59
±10°	34,13/17,44	36,71/22,46	0,95	1,04
±15°	54,51/40,56	51,41/38,33	1,51	1,47
±20°	76,12/70,85	66,07/60,74	2,24	1,91

Table 5.2: Results on EuRoC using virtual rotation data augmentation. The best results for the AOE, AYE and mean ROE was obtained from the baseline configuration with Gaussian noise on IMU measurements as data augmentation. All metrics became gradually worse when increasing the range of rotations used for constructing the virtual rotations.

5.6.3 Effects of data augmentation on TUM VI

Figure 5.4 shows the results from using virtual rotations as data augmentation on the TUM VI dataset. Table 5.3 summarizes the AOE, AYE and mean ROE for all executions. An interesting observation is that the baseline CNN performed best without the use of noise as data augmentation. The best results from these experiments, however, was to use a very slight amount of virtual rotations with a range of $\pm 0.25°$. A very similar performance was achieved when using this virtual rotation both with and without noise on IMU measurements at the same time. The execution with added noise performed best with respect to the AOE, AYE and mean ROE. When looking at the box plot in Figure 5.4 for a rotation range of $\pm 0.25°$, it is harder to argue which of these executions performed best. The execution with Gaussian noise has a lower median value. The execution without Gaussian noise, however, has a lower maximum value and a lower value for the upper quartile. Generally on the TUM VI dataset the range of rotations specified while constructing the virtual rotation has much less impact than on the EuRoC dataset. Similar results are obtained over a wide variety of rotation ranges, both with and without Gaussian noise added to the IMU measurements.

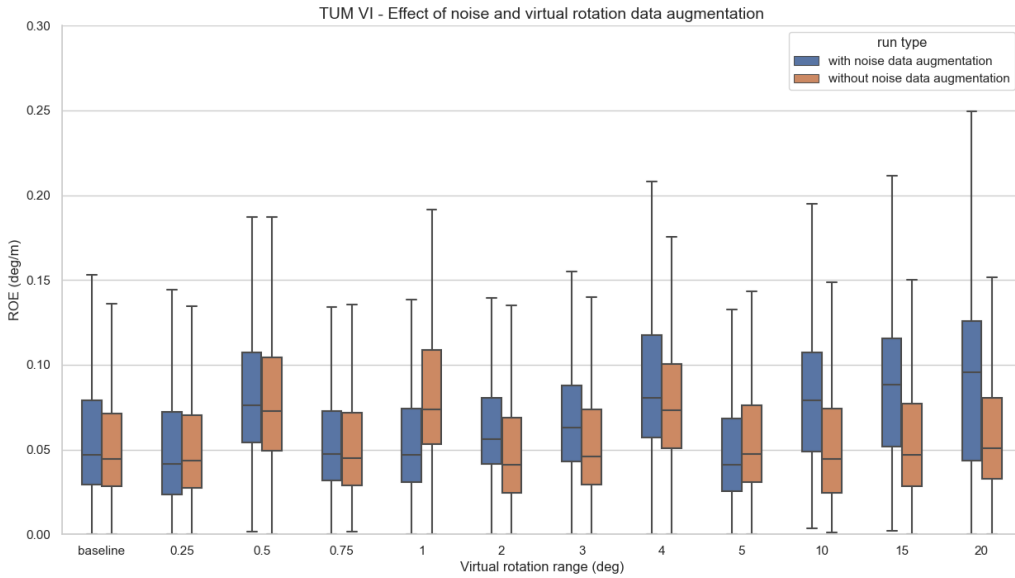


Figure 5.4: ROE on TUM VI using virtual rotation data augmentation. The baseline configuration performed best without using Gaussian noise on the IMU measurements as data augmentation. The best performance was obtained using a slight virtual rotation range of $\pm 0.25°$. Using varying ranges of virtual rotations did not consistently perform better than the baseline configuration.

	AOE/AYE (deg)		mean ROE (deg/m)	
	With noise	Without noise	With noise	Without noise
open loop	-	6,26/5,92	-	0,145
baseline	1,56/1,38	1,34/1,12	0,062	0,059
±0.25°	**0,9/0,68**	1,31/1,08	**0,057**	0,058
±0.5°	3,2/2,92	2,94/2,71	0,087	0,084
±0.75°	1,56/1,36	1,42/1,21	0,062	0,059
±1°	1,48/1,26	3,28/3,05	0,061	0,087
±2°	2,02/1,76	1,15/0,94	0,07	0,056
±3°	2,29/2,08	1,41/1,2	0,073	0,06
±4°	3,47/3,18	3,18/2,97	0,092	0,083
±5°	0,96/0,78	1,49/1,25	**0,057**	0,062
±10°	3,25/3,02	1,01/0,67	0,086	0,058
±15°	3,82/3,49	1,51/1,26	0,092	0,061
±20°	4,07/3,75	1,97/1,72	0,096	0,065

Table 5.3: Results on TUM VI using virtual rotation data augmentation. The best results for the AOE, AYE and mean ROE were obtained using both Gaussian noise on the IMU measurements and a slight virtual rotation with a range of ±0.25°. Using different ranges of virtual rotation did, however, not consistently improve the performance of the CNN.

5.6.4 Discussion

Using virtual rotations as a means of data augmentation was not found to consistently improve the performance of the CNN in [2] for either of the two datasets. On the EuRoC dataset the best results were achieved by using the exact method in [2]. On the TUM VI dataset, better results were obtained by using a slight virtual rotation with angles within ±0.25° as data augmentation for each epoch. Using other rotation ranges did, however, not consistently improve the results. On the EuRoC dataset, using a too large rotation range resulted in substantially worse performance.

It was not expected that the specified range in values for the roll, pitch and yaw angles would affect the performance as much as it did in these experiments, and that using a large range for the angles would degrade the performance as much as it did for the EuRoC dataset. Most likely this happened because the datasets now appeared to be too diverse. This could have made the CNN put more emphasis on patterns in the datasets that were only introduced as a result of this data augmentation technique. Hence, there is a chance that the CNN generalized on too diverse patterns in the training data, and that the training data no longer gave a good enough representation of how the IMU measurements used as inputs were supposed to look like, leading to the CNN performing worse on the unseen

test data. The reason why data augmentation using virtual rotations performed worse on the EuRoC dataset than on the TUM VI dataset is also most likely related to this problem. Since the IMU measurements in the EuRoC dataset are uncalibrated, and since colored noise is present from the rotors of the MAV, this dataset potentially contains more diverse characteristics than the TUM VI dataset without the use of data augmentation.

In conclusion, it was not possible to obtain better results on the EuRoC dataset using virtual rotation data augmentation. On the TUM VI dataset, better results were obtained using some of the combinations of Gaussian noise and virtual rotations as data augmentations. Better results were, however, only achieved for some of the specified ranges of values for the roll, pitch and yaw, and not consistently over several different rotation ranges. It is therefore concluded that the method of using virtual rotations as data augmentation is not very effective, and hence it is not relied on for any of the further experiments.

5.7 Different CNN architectures

5.7.1 Experimentation details

When testing the Dense and Residual CNN architectures from Section 4.4, many different configurations were tested in order to find an optimal architecture. Initially, different configurations using only one single convolutional layer per block were tested, as opposed to two convolutional layers visualized in Figure 4.2. It was not possible to produce better results than the CNN of Brossard et al. [2] using this type of architecture, regardless of how deep the Neural Network was. For this reason, the structure in Figure 4.2 was adopted for each CNN-block instead, with two convolutional layers per block. This is the same structure as the one used by Bai et al. [1] which produced good results in their study.

The testing of different CNN architectures was performed by testing deeper and deeper networks by adding one CNN-block (of the structure visualized in Figure 4.2) at a time. For each configuration, different dilation gaps and kernel dimensions of each layer were used in order to test each network with a varying size of receptive fields. The receptive fields tested varied between 224 and 28 672 samples, corresponding to between 1.12 seconds to 2.4 minutes of measurements, so that a wide range of different receptive fields were tested. The smallest CNN-configuration tested was using a CNN consisting of 3 blocks, and the largest CNN consisted of 7 blocks. Due to constraints on the availability of computer resources, it was not possible to test deeper networks than this. The same procedure for testing different CNN-architectures with different receptive fields was done for both configurations of the Dense network and Residual network.

It was not possible to find a CNN-configuration that preformed better than the CNN of [2], in this section referred to as baseline, for both the EuRoC and TUM VI datasets simultaneously. It was, however, possible to find two different architectures that preformed better than baseline for each of the two datasets. An architecture based on the Dense CNN preformed best on the EuRoC dataset. The structure is summarized in Table 5.4. An architecture based on the Residual CNN preformed best on the TUM VI dataset, and is summarized in Table 5.5. Both networks consist of 6 blocks. The receptive fields of the Dense and Residual networks are 1701 and 7168 samples respectively. This corresponds to 8.5 s and 35.8 s of past IMU measurements respectively. Note that the channel dimension for both networks is the same for blocks 5 and 6. When the channel dimension of the last block was increased to 512, following the pattern of doubling the dimension per block, the networks required too much time to train. Therefore, the channel dimension of block 6 was limited to 256 for both architectures.

CNN block	1	2	3	4	5	6	final layer
kernel dimension	7	7	7	7	7	7	1
dilation gap	1	3	9	27	81	243	1
channel dimension	16	32	64	128	256	256	1

Table 5.4: Dense network architecture details. The receptive field of this network is $N = \max(\text{kerneldimension} \times \text{dilationgap}) = 7 \times 243 = 1701$, corresponding to 8.5 s of past IMU measurements.

CNN block	1	2	3	4	5	6	final layer
kernel dimension	7	7	7	7	7	7	1
dilation gap	1	4	16	64	256	1024	1
channel dimension	16	32	64	128	256	256	1

Table 5.5: Residual network architecture details. The receptive field of this network is $N = \max(\text{kerneldimension} \times \text{dilationgap}) = 7 \times 1024 = 7168$, corresponding to 35.8 s of past IMU measurements.

5.7.2 Comments on the error metrics

The results of the ROE for the baseline network, Dense network and Residual network is visualized in Figures 5.5a and 5.5b respectively. Note that the ROE in the box plots is not scaled per meter in the plots. Rather, the ROE is plotted for the distances of 7, 21 and 35 meter traveled by the IMU, which means that the ROE is given in degrees rather than deg/m. This is done in order to provide a detailed comparison of the results achieved by the different CNN architectures, specifically how they compare across different sequence lengths. In Table 5.6 the

AOE and AYE as well as the mean ROE in deg/m is displayed for each architecture over all sequences in each of the datasets. The results for each network on all sequences is included to provide an even more detailed comparison than just comparing the mean values.

Note that the results for the AOE and AYE reported in [2] are better than the results achieved in these experiments for the method developed in [2], here referred to as baseline, as well as the results for the new CNN architectures. For the baseline architecture Brossard et al. [2] achieved an AOE/AYE of 2.10°/0.96° and 1.28°/0.82° for the EuRoC and TUM VI datasets respectively. When the same architecture was evaluated in this project, an AOE/AYE of 2.88°/1.80° and 1.56°/1.38° was achieved for the same datasets. The reason for this discrepancy is explained in Section 5.2. The method regarding reproducibility described in this section was used for these experiments. This ensures that the results obtained from these experiments are comparable, and that any improvements in the results are caused solely from the new methods implemented. For this reason, the results reported in Table 5.6 are not directly comparable to the results reported in [2].

5.7.3 Results and discussion

Performance on the EuRoC dataset

When looking at Figure 5.5a, it is clear that the Residual Network performs worse than baseline for all distances. When comparing the Dense network to the baseline network, however, some interesting observations can be made. Over a distance of 7 meters, both networks have a nearly identical ROE. Over distances of 21 and 35 meters the Dense network clearly beats baseline. This implies that both architectures have a similar local consistency over small distances. Meanwhile, the Dense Network has a greater long term accuracy, meaning that it performs better over longer sequences.

The results for the EuRoC dataset in Table 5.6 details the performance for each sequence in the dataset. This table shows that the Residual network performed worse on all metrics for all sequences, as was the case for the box plots of the ROE for this network as well. When comparing the performance of the Dense network to the baseline network the table shows that the Dense network does not perform better than baseline for all sequences. Baseline still performs better on sequences MH 02 easy and V1 03 difficult, and scores a better AOE on the sequence MH 04 difficult. Regardless of this, the Dense network manages to get a better performance on average, with an AOE/AYE of 2.87/1.70 compared to 2.88/1.80 for baseline.

In light of the results presented, only a slight increase in the performance was

observed with regards to the mean AOE, AYE and ROE when using the Dense network. The most significant performance gains of the Dense network was seen for the ROEs over 35 meters traveled, where the mean ROE decreased with 0.47 degrees compared to baseline. This indicates that the Dense network has a greater long-term accuracy while maintaining a good local consistency. This increase in long-term accuracy is most likely related to the large receptive field of the network. With the receptive field of the Dense network covering 35.8 s of past IMU measurements, the receptive field covers almost all measurements over a distance of 35 m for the velocities the IMU has in both datasets. This means that the Dense network can combine more information from past IMU measurements to infer the current gyro-correction.

Performance on the TUM VI dataset

From Figure 5.5b it is clear that the Dense network performs worse than baseline for the ROE over all distances. The Residual network, however, manages to perform better than baseline for the ROE over all distances. This is in agreement with the results for the TUM VI dataset presented in Table 5.6, where the Residual network has a lower error for all metrics on all sequences in the dataset, except for a slightly worse result for the AOE on the *room 4* sequence. The Residual network thus both has a better short term consistency and a better long term accuracy than baseline on this dataset.

		AOE/AYE (deg)			mean ROE (deg/m)		
dataset	sequence	baseline	Residual Net	Dense Net	baseline	Residual Net	Dense Net
EuRoC [3]	MH 02 easy	**2.91/1.53**	4.20/2.35	3.50/2.71	**0.0845**	0.1259	0.0983
	MH 04 difficult	**1.27**/0.88	3.71/2.60	1.34/**0.75**	0.0432	0.1063	**0.0353**
	V1 01 easy	3.98/2.64	5.07/2.95	**3.64/2.28**	0.2893	0.2942	**0.2710**
	V1 03 difficult	**1.96/1.29**	3.19/1.97	2.27/1.63	**0.0702**	0.0986	0.0768
	V2 02 medium	4.28/2.37	6.93/3.03	**3.60/1.16**	0.1439	0.2011	**0.1254**
	mean	2.88/1.80	4.62/2.58	**2.87/1.70**	0.1307	0.1685	**0.1266**
TUM VI [26]	room 2	1.96/1.83	**1.57/1.38**	2.61/2.49	0.0573	**0.0555**	0.0681
	room 4	**0.95**/0.68	0.98/**0.57**	1.37/0.93	0.0592	**0.0581**	0.0844
	room 6	1.78/1.63	**1.59/1.31**	2.13/1.99	0.0723	**0.0638**	0.0768
	mean	1.56/1.38	**1.38/1.09**	2.04/1.80	0.0624	**0.0588**	0.0754

Table 5.6: Results using different CNN architectures. The Residual network performed better than baseline on all sequences on the TUM VI dataset for all metrics except the AOE on *room 4*. The Dense network performed better than baseline on the EuRoC dataset on three of the five sequences used, and obtained a better result for the AOE, AYE and mean ROE than baseline on average.

Discussion of the results for both datasets

The performance gains achieved by the Residual network on the TUM VI dataset are greater than those achieved by the Dense network on the EuRoC dataset. During experimentation it was observed that it was much easier to find a better performing CNN architecture for the TUM VI dataset than for the EuRoC dataset, and several of the tested configurations performed better than baseline on TUM VI. For the EuRoC dataset, the Dense network configuration presented here was the only configuration found to perform better than baseline. As was mentioned in Section 5.3, the most notable difference between the datasets is that the TUM VI dataset is calibrated while the EuRoC is not. This could in turn effect the performance of the networks and make them behave differently for each dataset. It seems logical that it should be easier for a CNN to perform better on the TUM VI dataset since it is calibrated. One more reason that it was easier to find a better architecture for the TUM VI dataset could be related to the fact that Brossard et al. [2] tuned their network to perform well on both datasets at the same time. There is a possibility that their network was tuned for the best possible performance on the EuRoC while maintaining a good performance on the TUM VI dataset. If this was the case, it would be harder to achieve performance gains on the EuRoC dataset.

Surprisingly, the receptive field of both of the architectures tested in these experiments is substantially greater than baseline. As mentioned in Chapter 3, baseline has a receptive field of 448. The receptive fields of the Dense and Residual networks are 1701 and 7168 samples, corresponding to 8.5 s and 35.8 s of data respectively. Especially interesting is the size of the receptive field on the Residual Network, which is 16 times greater than the receptive field of the baseline network. This indicates that as the CNN models become increasingly large and complex, they manage to combine more and more information from the past IMU measurements in estimating each gyro-correction. Based on the results presented in the previous sections, this seems to be a beneficial attribute contributing in the CNN-models performing better on the datasets.

A surprising observation made during testing of the different CNN architectures was that no configuration of the Dense network tested managed to perform better than the baseline network on the TUM VI dataset. Similarly no configuration of the Residual network performed better than baseline on the EuRoC dataset. This suggests that there are some different traits in each of the CNN architecture types that makes each of the CNNs better suited to their respective datasets. The main difference between the two architectures is: For the Dense network each layer receives the information for every preceding layer in the network, including the IMU measurements used as inputs to the CNN. For the Residual network an identity mapping is passed forward from layer to layer, and each CNN-block learns small

modifications to the these mappings. Since the IMU in EuRoC is uncalibrated, and contains colored noise from the rotors of the MAV it is a possibility that each layer in the Dense CNN has an advantage of gaining the knowledge of the input as well as the knowledge of each preceding layer in the network, since the original IMU signal is more corrupted. Likewise, for the TUM VI dataset, there is a possibility that since the IMU signals are calibrated, only slight modifications to the gyro-rates are needed, which are captured well by the small modifications to the identity mapping present in each layer of the Residual network.

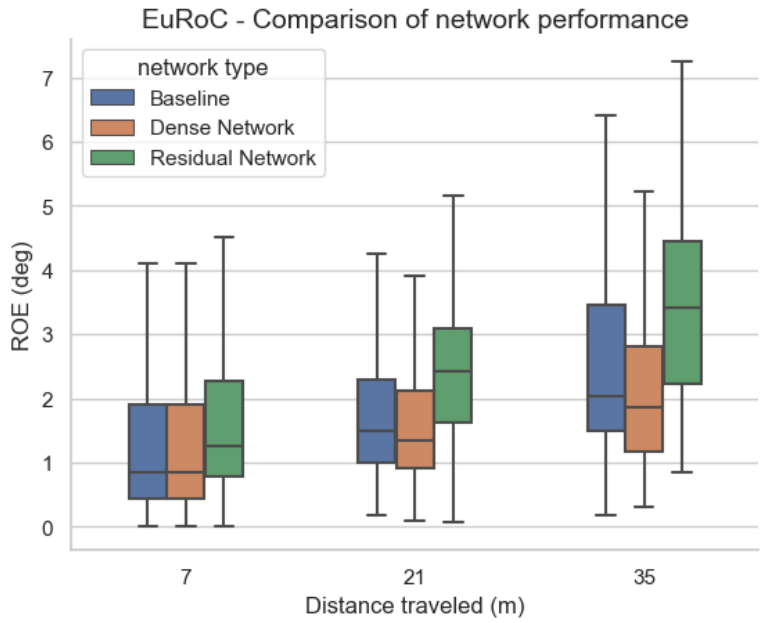
One of the primary observations done from testing the different architectures is that in order to get the best possible results, an attitude estimation method based on CNNs should be individually tuned to the specific dataset or IMU it is intended to be used on. This is similar to tuning conventional attitude estimation filters such as the Madgwick filter [18]. When regarding the results for the AIE from Section 5.5, a different tuning parameter β is required to produce the best possible results for each of the datasets, which is analogous to what is observed for the CNNs.

Adding to the discussion of the use of data augmentation from Section 5.6, the CNN architectures developed in these experiments were trained both with and without these data augmentation techniques. It was observed that the use of virtual orientations had little effect on the performance of the networks. The use of added Gaussian noise to the IMU measurements during training made a substantial difference, however. For all the results provided in Table 5.6 Gaussian noise was added to the IMU measurements for data augmentation. When training the Residual network from this table without this data augmentation technique on the TUM VI dataset, an AOE/AYE of $9.50°/9.06°$ was achieved. By simply integrating the gyro-rates from the IMU, an AOE/AYE of $6.26°/5.91°$ was achieved. This means that the CNN actually contributed to make the orientation estimates worse in this case. In contrast, using the baseline network without the use of noise as data augmentation actually produced better results than with the use of noise, as was demonstrated in Table 5.3. The use of this data augmentation was therefore much more beneficial when training the deeper CNN-models presented in this section. This illustrates the importance of regularization techniques such as data augmentation when training large CNN-models on small datasets.

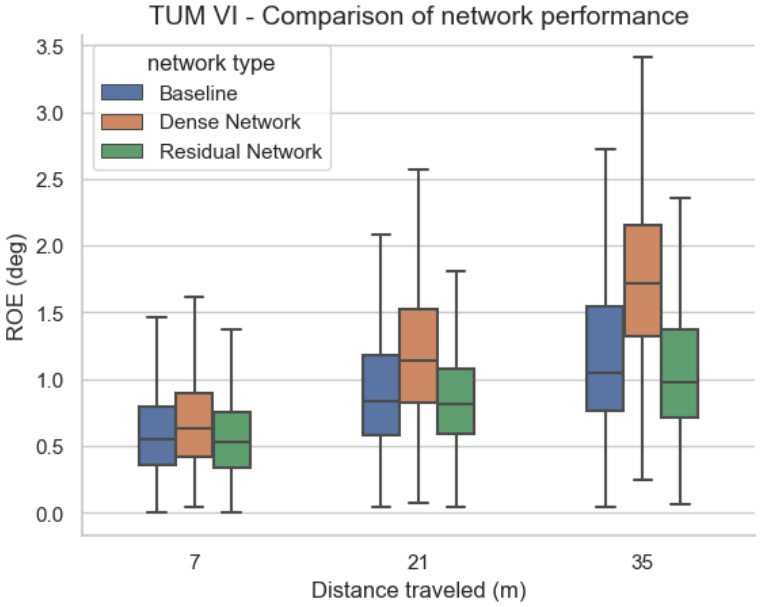
In the preliminary study [24] the method of Brossard et al. [2] was tested on the BROAD dataset [14] without success. One of the reasons for this could be related to the architecture of the CNN in [2]. The BROAD dataset is a much more diverse dataset than the datasets used in these experiments. For this reason, there is a possibility that success on the BROAD dataset requires the use of a more complex CNN model that is able to model the more diverse patterns of this dataset. Both the Residual and Dense networks developed in this thesis were

tested on the BROAD dataset. But it was still not possible to produce acceptable results. Therefore producing good results on the BROAD dataset remains an unsolved issue.

Despite the fact that it was not possible to produce good results on the BROAD dataset, it was possible to improve the method of [2] by using a different CNN architecture and tuning the network to each individual dataset. Slight improvements were made on the EuRoC dataset and more substantial improvements were obtained for the TUM VI dataset. These improvements comes at a cost of increasing the complexity of the CNN model, making the new models require greater computational resources. While the CNN-model in [2] took about 3 minutes to train on a V100 GPU, the CNN-models presented in this chapter took about 15–20 minutes to train. Therefore, the new developed CNN models should be considered on systems where computational resources are in abundance, whereas the CNN model of [2] should be preferred on systems where computational resources are scarce. Furthermore, a CNN method individually tuned for the specific dataset or IMU in use is preferred.



(a) ROE of different CNN architectures on EuRoC. The Dense network performs nearly identical to baseline on sub-sequences with length of 7 meters. For longer sequences, the Dense network beats baseline. The residual network obtains worse results over all metrics.



(b) ROE of different CNN architectures on TUM VI. The Residual network performs better than baseline for sub-sequences of all lengths. The Dense network performs worse than baseline over all lengths.

Chapter 6

Conclusion

In this thesis, the performance of the method proposed in [2] using a CNN to estimate the orientation has been compared to the performance of the Madgwick orientation filter [18] on the TUM VI and EuRoC datasets. Compared to this filter, the method of [2] is extremely successful. The Madgwick filter fails to estimate the 3D-orientation since it failed to estimate the yaw-angle from an IMU containing only a gyroscope and accelerometer. In contrast, the method of [2] managed to achieve good estimates both for the yaw angle and the 3D-orientation.

A new method of data augmentation was used together with the method of [2] to try to increase the performance of the CNN. A thorough analysis was performed with regards to virtually rotating the IMU to create a more diverse dataset. It was not possible to produce better results from this method on the EuRoC dataset. On the TUM VI dataset better results were achieved by applying very slight virtual rotations to the IMU. The results on the TUM VI dataset were not consistent for different amounts of virtual rotations, and the results were therefore not satisfying. In conclusion, using virtual rotations for data augmentation did not improve the performance of the method from [2].

Replacing the CNN in [2] with new CNN architectures increased the performance of the method. Using a Dense neural network yielded slight improvements on the EuRoC dataset. Using a Residual neural network yielded good improvements on the TUM VI dataset. The combination of a larger receptive field and a more complex CNN-model individually tuned for each dataset proved to be successful.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling". In: *arXiv preprint arXiv:1803.01271* (2018).

[2] M. Brossard, S. Bonnabel, and A. Barrau. "Denoising imu gyroscopes with deep learning for open-loop attitude estimation". In: *IEEE Robotics and Automation Letters* 5.3 (2020), pp. 4796–4803.

[3] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* 35.10 (2016), pp. 1157–1163.

[4] O. Egeland. *Quaternions, attitude estimation and SLAM*. NTNU, 2021.

[5] O. Egeland. *Robot Vision*. NTNU, 2021.

[6] R. L. Farrenkopf. "Analytic steady-state accuracy solutions for two common spacecraft attitude estimators". In: *Journal of Guidance and Control* 1.4 (1978), pp. 282–284.

[7] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang. "Openvins: A research platform for visual-inertial estimation". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4666–4672.

[8] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[9] D. Hendrycks and K. Gimpel. "Gaussian error linear units (gelus)". In: *arXiv preprint arXiv:1606.08415* (2016).

[10] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors". In: *arXiv preprint arXiv:1207.0580* (2012).

[11] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely connected convolutional networks". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 4700–4708.

[12] P. J. Huber. "Robust estimation of a location parameter". In: *Breakthroughs in statistics*. Springer, 1992, pp. 492–518.

[13] M. Kok, J. D. Hol, and T. B. Schön. "Using inertial sensors for position and orientation estimation". In: *arXiv preprint arXiv:1704.06053* (2017).

[14] D. Laidig, M. Caruso, A. Cereatti, and T. Seel. "BROAD—A Benchmark for Robust Inertial Orientation Estimation". In: *Data* 6.7 (2021), p. 72.

[15] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[16] E. J. Lefferts, F. L. Markley, and M. D. Shuster. "Kalman filtering for spacecraft attitude estimation". In: *Journal of Guidance, Control, and Dynamics* 5.5 (1982), pp. 417–429.

[17] K. M. Lynch and F. C. Park. *Modern robotics*. Cambridge University Press, 2017.

[18] S. O. H. Madgwick. "An efficient orientation filter for inertial and inertial/magnetic sensor arrays". In: *Report x-io and University of Bristol (UK)* 25 (2010), pp. 113–118.

[19] R. Mahony, T. Hamel, and J.-M. Pflimlin. "Nonlinear complementary filters on the special orthogonal group". In: *IEEE Transactions on automatic control* 53.5 (2008), pp. 1203–1218.

[20] F. L. Markley. "Attitude error representations for Kalman filtering". In: *Journal of guidance, control, and dynamics* 26.2 (2003), pp. 311–317.

[21] G. M. Mayitzin. *AHRS: Attitude and Heading Reference Systems*. 2022. URL: https://github.com/Mayitzin/ahrs (visited on 05/15/2022).

[22] *Reproducibility - PyTorch*. 2022. URL: https://pytorch.org/docs/stable/notes/randomness.html (visited on 05/16/2022).

[23] J. Rohac, M. Sipos, and J. Simanek. "Calibration of low-cost triaxial inertial sensors". In: *IEEE Instrumentation & Measurement Magazine* 18.6 (2015), pp. 32–38.

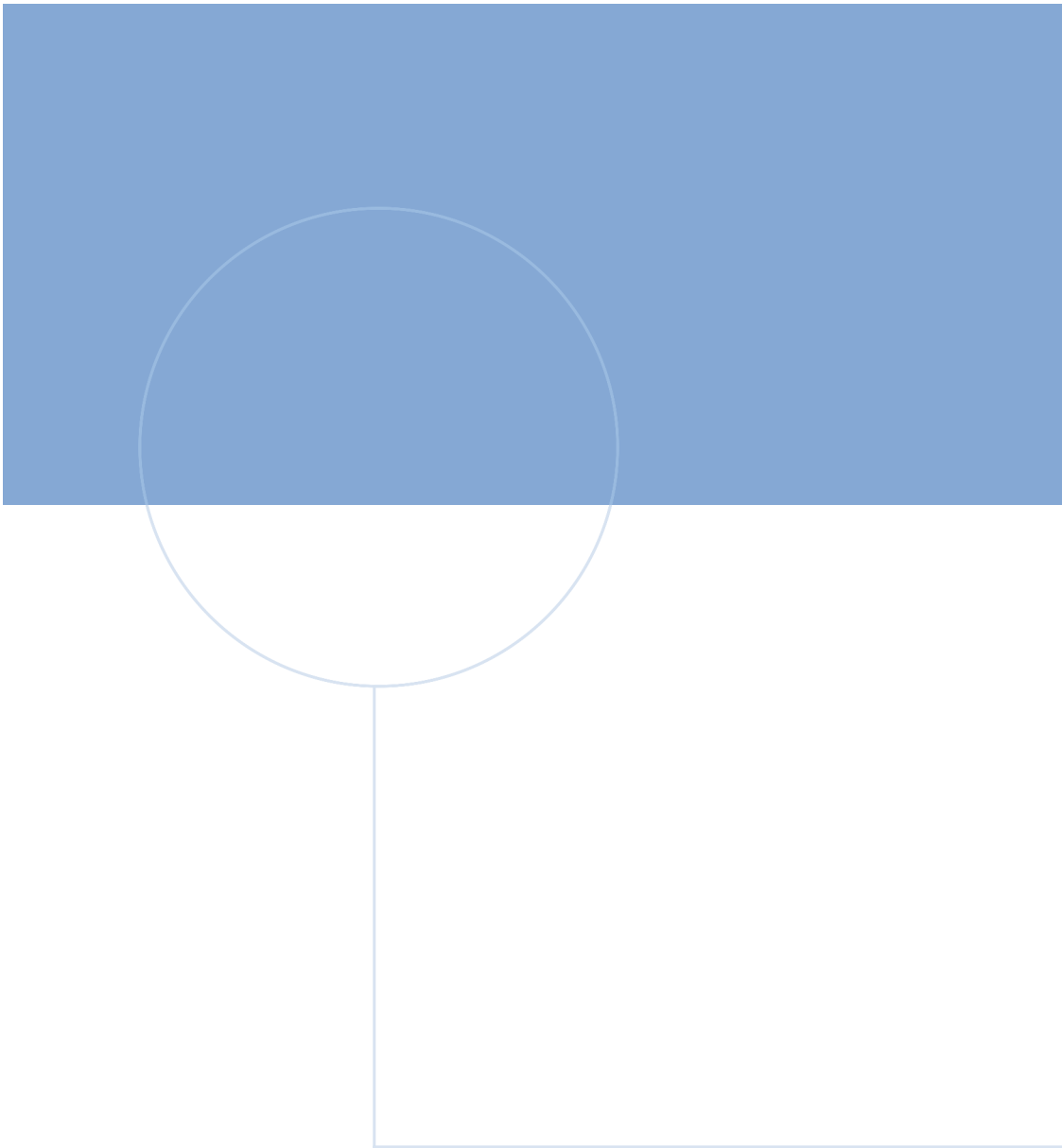
[24] E. Samuelsen. *Estimation of Noise in IMUs with Deep Learning*. NTNU, 2021.

[25] S. Santurkar, D. Tsipras, A. Ilyas, and A. Mądry. "How does batch normalization help optimization?" In: *Proceedings of the 32nd international conference on neural information processing systems*. 2018, pp. 2488–2498.

[26] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stückler, and D. Cremers. "The TUM VI benchmark for evaluating visual-inertial odometry". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 1680–1687.

[27] D. Weber, C. Gühmann, and T. Seel. “Neural networks versus conventional filters for Inertial-Sensor-based attitude estimation”. In: *2020 IEEE 23rd International Conference on Information Fusion (FUSION)*. IEEE. 2020, pp. 1–8.

[28] Z. Zhang and D. Scaramuzza. “A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry”. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7244–7251.



NTNU

Norwegian University of
Science and Technology