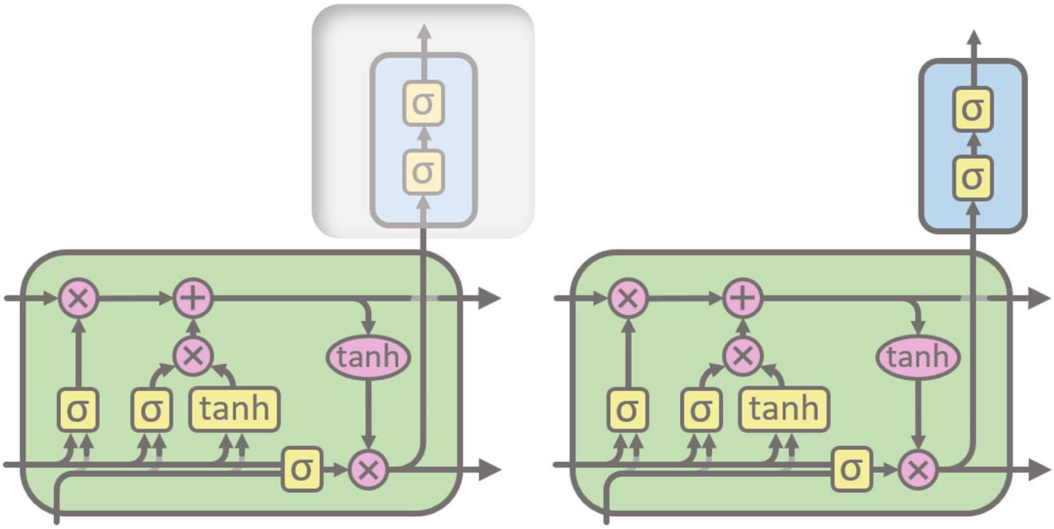

Learning a Better Attitude

A Recurrent Neural Filter for Orientation Estimation

Master's Thesis in Systems, Control and Mechatronics

AMANDA FRANSSON
OSCAR LUNDSTRÖM

DEPARTMENT OF MECHANICS
AND MARITIME SCIENCES

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020
www.chalmers.se

Master's thesis 2020:16

Learning a Better Attitude

A Recurrent Neural Filter for Orientation Estimation

AMANDA FRANSSON
OSCAR LUNDSTRÖM

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
Gothenburg, Sweden 2020

Learning a Better Attitude
A Recurrent Neural Filter for Orientation Estimation
AMANDA FRANSSON
OSCAR LUNDSTRÖM

Supervisor: Niclas Berglind, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Master's Thesis 2020:16
Department of Mechanics and Maritime Sciences
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Scematic illustration of a recurrent neural filter, as explained in Section 2.4.2.

Typeset in LaTeX
Gothenburg, Sweden 2020

Learning a Better Attitude
A Recurrent Neural Filter for Orientation Estimation
AMANDA FRANSSON
OSCARLUNDSTRÖM
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

In the current paradigm of sensor fusion orientation estimation from inertial measurements unit sensor data is done using techniques derived with Bayesian statistics. These derivations are based on assumptions about noise distributions and hand crafted equations describing the relevant system dynamics. Machine learning, and more specifically neural networks, may provide an alternate solution to the problem of orientation estimation where no assumptions or handcrafted relationships are present. This thesis aims to investigate whether a neural network-based filter can achieve a performance comparable to, or exceeding that of, the more conventional extended Kalman filter. Two network architectures based on *long short-term memory* layers are proposed, trained, evaluated and compared using data from the Oxford inertial odometry dataset. Of the two suggested model architectures the so-called *recurrent neural filter* is found to give a the better performance. The recurrent neural filter has a structure inspired by Bayesian filtering, with a prediction and an update step, allowing it to output a prediction in the event of missing data. Further, the evaluated models are trained to estimate orientation as well as a parameterized error covariance matrix. Our results show that the suggested recurrent neural filter outperforms the benchmark filter both in average root mean square error and in execution time. The result also indicates that the machine learning-based approach for sensor fusion problems may be an attractive alternative to hand crafted filters in the future.

Keywords: sensor-fusion, state estimation, absolute orientation estimation, recurrent neural filter, recurrent neural network, RNN, LSTM, IMU, MARG.

Acknowledgements

We would like to thank *CPAC Systems* for providing resources and giving us the opportunity to conduct our thesis with them. We would also like to thank Niclas Berglind, our supervisor, for supporting us and helping us get started with our work, and Peter Forsberg, our examiner for the many hours spent giving us feedback and input on our thesis, despite the unexpected circumstances caused by the COVID-19 pandemic.

Amanda Fransson & Oscar Lundström, Gothenburg, May 2020

Supervisor: Niclas Berglind, CPAC Systems AB
Examiner: Peter Forsberg, Department of Mechanics and Maritime Sciences

Contents

List of Figures

List of Tables

1
Introduction

In this chapter, the background of the thesis will be presented together with a summary of the current state of the field of machine learning based orientation estimation. The research questions and the contributions of the thesis are then presented, followed by an overview of the outline of the report.

1.1 Background

Sensor fusion is, simply put, the practice of combining measurements from different sources to infer some property with greater certainty than what would have been possible had the measurements been considered separately. Sometimes the property of interest is hidden or *latent* and can not be estimated unless several different measurements are considered together. This inference of latent states from measurements on related quantities has a significant number of application areas and is essential for a range of technologies such as robotics, image processing, navigation, and situation awareness [1],[2],[3]. A commonly used filter in the field of sensor fusion is the *Kalman filter* (KF) [2],[4], mainly due to its relative simplicity and the properties of optimality it provides. The KF is a recursive Bayesian estimator that is optimal[1] under the condition that i) the process and measurement noise is white[2], ii) the exact covariance of the noise is known and iii) that the model is linear and precisely known[3].

KFs are often tuned via trial and error until an adequate performance is achieved [7], and thus the optimal estimate is not necessarily produced. Furthermore, KFs are generally not sufficient in cases with significantly non-linear processes, for which they tend to diverge. Work has been done on filters adapted to deal with cases where the conditions i)-iii) above are not met. The *extended Kalman filter* (EKF) [8] and the *unscented Kalman filter* (UKF) are both examples of filters designed to deal with non-linear processes [4]. In applications where there is a plant[4]-model mismatch and/or the noise covariance is unknown, it is possible to make use of an *adaptive Kalman filter* (AKF) [9].

[1] Optimal in the sense that it minimizes the mean squared error. [5]

[2] White noise here refers to noise that is zero mean, uncorrelated over time, and of finite variance.

[3] The KF is still the optimal *linear filter* even though better non-linear filters may exist [5]. Also, although linear model abstractions are powerful tools, linear systems are very rare in the real world[6].

[4] Plant here refers to the model of the dynamics of a system.

Sensor fusion in general and KFs, in particular, are important techniques since they can convert noisy sensory data to useful and reliable information. Despite the proven capabilities of classical probabilistic sensor fusion techniques, such as the KF, *machine learning* (ML) methods have started to transform the field entirely [10]. The fundamental shift from manually describing the dynamics of the system (from which information is inferred from) to learning it from examples opens up the door to solving problems previously too complex to model [10]. Looking at the progress made with *neural networks* (NN) in the recent decades, in tasks such as classification, regression, and sequence-to-sequence modeling between domains, it is not far-fetched to imagine employing NNs in filtering applications. Some researchers have headed in this direction, striving to combine NNs with KFs or done filtering with NNs without involving KFs at all [11],[12].

A specific sensor fusion problem where such an attempt can be investigated is the objective of retrieving the absolute orientation estimation from an *inertial measurement unit* (IMU) and a magnetometer. This combination of sensors is sometimes referred to as a *Magnetic, Angular Rate, and Gravity* (MARG) sensor. An IMU does not provide enough information to infer absolute orientation. Instead, it has to be inferred from the measurements of acceleration, angular velocity and an additional corrector, such as magnetic field direction. Orientation estimation using IMU/MARG data is often done with EKFs, complementary filter, or more recent methods such as the *explicit complementary filter* (ECF) and *gradient descent based orientation filter* (GDOF) [13],[14],[15]. Often the linear acceleration measured by the accelerometer is assumed only to be influenced by gravity, which is a reasonable assumption under many conditions. However, in the instances where the assumption does not hold, it can lead to significant errors in orientation estimation unless dealt with properly. Designing filters that deal with the situation where this assumption does not hold is possible and has been done [16]. These filters often adapt the covariance of the measurement when the acceleration magnitude differs from the gravitational acceleration, or omit some measurements altogether. Common to both methods is that they add both computational and implementational complexity.

1.1.1 Related Work

There exists a variety of implementations that use NN to replace, or work in unison with, the KF and versions thereof. Such implementations vary both in architecture, application area, and overall approach. The applications found to be most relevant for this thesis are described below.

Neural Network Moderated Kalman Filter

In reference [12], where an NN in conjunction with a KF is proposed. The NN acts as a supervisor to monitor and improve the prediction accuracy of the KF algorithm. In an experiment this setup is used to predict actual temperature from noisy sensor data and the NN is used to estimate the extent of error in current readings and to update the process error covariance in the KF. Compared to a stand alone KF, where the process error covariance is static, the proposed implementation was able

to reduce the root mean squared error (RMSE) of the prediction noticeably.

Neural Network as Extended Kalman Filter Substitute

Another example is given in reference [17], where a sensorless approach to estimate the shaft speed of a dc-motor for closed-loop control using an NN is presented. The proposed NN algorithm performance was compared with the performance of an EKF and found to achieve a better performance.

Deep Kalman

In reference [18], a hybrid solution called the Deep Kalman Filter is presented. The key purpose of the filter is to learn the system model, and the application area of the study was to perform improved IMU positioning estimation by IMU error modeling. Further, the IMU error was modeled using *long short term memory* (LSTM) layers. This LSTM-based IMU error modeling is added to the filter alongside the standard filtering steps: prediction and update. The study showed promising results compared to the conventional filtering with an EKF.

OriNet

OriNet is a recently proposed network based on *recurrent neural networks* (RNN), that provides an orientation estimation from a single IMU sensor [19]. The network is trained with data from small flying robots[5], and is the first network to provide an end-to-end ML solution for orientation estimation. The architecture of OriNet consists of two parts. The first part combines four LSTM channels and fuses the learned feature representations of the channels in a *slow fusion gate* [19]. The second part consists of another LSTM and fully connected gates that output the difference between the quaternion of the last time-step and the present time-step [19]. Allegedly[6], OriNet improved the ability to estimate orientation with roughly 80% compared to reference [20] on the same dataset.

Sensor De-noising Network

The authors of reference [22] propose a method of de-noising IMU readings using a CNN. The achieved results are of such a quality that, using dead reckoning on the data de-noised by their model, they still manage to outperform OriNet on test data from the same dataset.

[5]The authors of reference [19] note that the data they use makes the assumption of stationary periods non-applicable.

[6]The performance of OriNet is reported in comparison with the quaternion-based complementary filter presented in [20] alongside the previously mentioned GDOF. However, their presented results of the GDOF is seemingly extremely poor, something that does not coincide with the performance of the GODF in other literature. Further, the GODF is not intended to estimate absolute orientation in 3D without access to magnetometer measurements [14],[21], but to the best of our knowledge, this is exactly what was done in the OriNet paper. As a consequence, the 80% improvement is questionable even though the method is still promising.

IONet

Yet an other example of an attempt to fuse sensor data with NN is IONet[7] [11] which estimates odometry from IMU readings. IONet is based on bi-directional LSTMs, and performs windowed smoothing[8] rather than filtering.

Recurrent Neural Filter

The *recurrent neural filter* (RNF) presented in [24] is an autoencoder/decoder architecture based on LSTMs. The RNF takes inspiration from conventional recurrent Bayesian estimators such as KFs where the update step and the prediction step are done separately. The encoder portion the architecture is comprised of LSTM layers responsible for the different Bayesian filter steps while the decoder consists of an feed forward NN. One of the significant novel contributions of this approach regards the memory of the LSTM layers. Instead of a particular layer using an individual separate hidden state, the hidden and cell states of the LSTM is shared horizontally with the other filter steps. To a large part this makes the structure similar to the distinct steps of the KF. The hidden state shared by the LSTM layers are in this analogy equivalent to the state estimate in the KF. A strong motivation for using the RNF structure is that it enables filtering when the system dynamics are unknown while still maneuvering missing data points.

Field Prospects

To date, and to our knowledge, no studies proposing end-to-end ML solutions for estimation of absolute orientation using IMU/MARG data has been published. The previously mentioned IONet [11], OriNet [19] and the sensor de-noising network [22] all tackle orientation estimation relative to an initial orientation. Reference [10] concludes that the first direction of future studies within the field of data fusion with ML methods should be to explore more application areas. Equally important, also according to [10], artificial NNs are particularly suitable and able to model various non-linear relationships that are difficult to express using hand crafted models.

1.2 Problem Statement and Aim

Although the KF, and its adaptations for nonlinear systems (EKF and UKF), are sufficient in many applications, they have some drawbacks. As a first accessible example, the tuning is typically performed by a weighting of the diagonal elements of the covariance matrices. This tuning is often done arbitrarily until a "good enough" result is achieved [7]. This is, of course, implementation-dependent, and if the performance satisfies all demands the filter will be useful. Nevertheless, this manner of tuning does not ensure that the best possible solution is utilized, which

[7] The authors behind IONet also published OxIOD [23], a datset with IMU readings and *ground truth* (GT) estimates from a vision system.

[8] Smoothing is the filtering of data with access to both past and future measurements. Windowed smoothing means that its smoothing over a fixed length window, and that the hidden state is not shared between windows.

in turn would be too time-consuming to actually find. More severe is the Markovian assumption[9] present in Bayesian filters, such as the KF [18]. Even though this assumption simplifies the system modeling noticeably and makes the KF efficient, it also makes it insensitive to system behavior with longer correlation times. On the whole, complicated error models with a correlation over long periods are neglected in the KF.

Further, a specific dynamic system can often operate in different regions of the dynamic state space, which can be hard to model in the same state estimator with classical models. In some applications, this is solved by dividing the dynamics into different *modes*, where each mode approximately describes different regions. A straightforward example of this is the trade-off between modeling a car with a constant velocity[10] or coordinated turn[11] model. Both of these models capture some aspects of the dynamics, but depending on whether the car is turning or traveling somewhat straight, their ability to describe the dynamics in the current mode will differ. There exist approaches for when a system exhibits this kind of multifaceted dynamics, that exploits the fact that describing the dynamics of the modes separately requires less effort than modeling every aspect of interest of the dynamics in one monolithic model. Examples of filters that do this are Interacting Multiple Models filters and Switching Kalman Filters [25]. However, the manual labor and the knowledge about the system dynamics required, and the computational burden is somewhat prohibitive when using these filter alternatives.

Finally, when modeling the state error distributions in KF, EKF and UKF one is limited to additive noise with multivariate white Gaussian distributions. The KF is also limited to modelling measurement and process noise as additive, although the EKF and UKF can, with minor modifications, be designed to deal with non-additive noise as well. Filter performance is generally adversely affected when the noise is not distributed in the way modeled, and outlier measurements[12] can have a large impact. An example of a method that explicitly deals with non-Gaussian distributions is the famed particle filter [26], which comes at the cost of computational expense.

This thesis aims to benefit from the applicable traits of ML, particularly NNs, in order to estimate absolute orientation using IMU and MARG sensor data. In theory, an ML approach will be able to estimate altering system dynamics and noise distribution as well as be able to exploit system behavior correlated over time.

[9] Given information about the most recent state of a system with the Markov propery, information about any previous state will be uninformative [6].

[10] A constant velocity model assumes constant velocity. The acceleration's affect on velocity is modeled as noise.

[11] A coordinated turn model assumes constant velocity along a circle segment. Changes in radius and velocity are modeled as noise.

[12] Outliers are samples that deviate from the distribution of the rest of the population. They can be due to both natural variance in the population, or due to some error with the observation.

1.2.1 Research Questions

Is the application of machine learning approaches for sensor fusion purposes a proper alternative? Can indications of this, from related research, be supported?

How well does a NN based filter compare with a EKF in terms of quality (RMSE) and efficiency (computational expense) in estimating orientation from noisy IMU data?

Can a measure of the certainty of the produced estimates, similar to the covariance of the EKF estimations, be found for an NN based filter?

1.3 Delimitations

This thesis will be limited to the problem of absolute orientation estimation with IMU and MARG sensors. Hence, there will not be any attempt to estimate absolute position even if spatial translations are present, and no additional sensor data will be used. Further, the developed algorithms will be benchmarked against a traditional EKF and not any of the mentioned extensions that deal with issues present in the EKF. Further, in reference [10] a list of criteria, that a data fusion method ideally should fulfill, is established. The list is proposed to make a comprehensible evaluation of different data fusion algorithm more accessible. Based on the entries in mentioned compilation, this thesis will be limited to evaluate developed models in terms of quality, efficiency, robustness and tested on real world data. Thus, evaluation of properties within privacy, extensibility, and stability will be outside the scope of this thesis.

1.4 Contribution

The primary contribution of this study is to suggest an ML method that estimates the absolute orientation using IMU/MARG. Secondly we hope this thesis will contribute to extending the field of sensor fusion with ML and showcase the suitability of utilizing ML for data filtering.

1.5 Thesis Outline

This thesis constitutes of six chapters. The current chapter, the introduction, is the first out of the six. In the second chapter, relevant theory is presented. More specifically, theory on orientation representations, IMUs and ML is covered. Further, in the third chapter, the methods used in this thesis are motivated and accounted for. In chapter four the results are compiled. In the fifth and sixth chapters the presented results are first interpreted and discussed, and subsequently conclusions are drawn.

2
Theory

2.1 Representing Orientation

The mathematical description of orientation is based on attaching different frames of reference to the objects of interest. A simple example is two frames of reference, the body frame b, and the world frame i. The body frame is fixed to the object for which the orientation is to be modeled and the world frame is stationary. The conversion between the two frames is easiest represented with a rotation matrix R^{ib} relating the vectors $\boldsymbol{v}$ in the two frames

$$\boldsymbol{v}^i = R^{ib}\boldsymbol{v}^b. \tag{2.1}$$

In equation (2.1), $\boldsymbol{v}^i$ is a vector in the world frame, and $\boldsymbol{v}^b$ is the same vector only represented in the body frame. The rotation matrix R^{ib} applies the rotation in the Euclidean space from frame b to frame i.

Any rotation can be described using three consecutive elemental rotations. Thus, the rotation matrix R, parametrized as

$$R = R_x(\varphi)R_y(\vartheta)R_z(\psi), \tag{2.2}$$

can also describe any rotation in 3D space. R_x, R_y, and R_z are all elemental rotations about the axes x, y, and z with angles φ, ϑ, and ψ respectively. This parametrization is referred to as Euler angles or Tait-Bryan angels. The angles are in some application fields called roll, pitch, and yaw (φ, ϑ, and ψ). Further, the parametrization of a rotation matrix into single-axis rotations has two different possible interpretations depending on if an intrinsic or extrinsic rotation convention is used. As a result, the rotation matrix R can represent either the extrinsic rotation about fixed axes z, y, x in that specific order, or the intrinsic rotation about the axes of rotation x', y', z'. Furthermore, applying the rotation gives different results in the two cases. From here on, only extrinsic representations will be used.

The use of Euler angles is appealing, because they are somewhat visualizable and perhaps even the most intuitive representation for human comprehension. However, the Euler angles bring ambiguities in the form of *gimbal lock*, which occurs when no unique solution can be found for the Euler representation. Formatting a machine learning problem in such a way that input maps to outputs 1:1 makes modelling the output distribution as Gaussian viable. In problems where one input can map

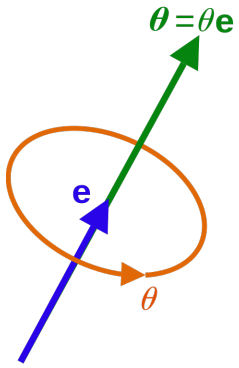

Figure 2.1: Illustration of the axis angle representaiton. From [29], CC0 1.0

to multiple correct outputs the usage of sum-of-squared-errors loss functions tend to result in models that predict the mean [27]. This motivates the employment of a quaternion representation which is free of the gimbal lock problem.

A quaternion $\boldsymbol{q}$ with basis $\{1, i, j, k\}$ is expressed as

$$\boldsymbol{q} = a + bi + cj + dk. \tag{2.3}$$

The real part of the quaternion, a, is often referred to as the scalar part, and the imaginary part, $bi+cj+dk$, is called the vector part of the quaternion. Quaternions that are used to represent orientation should be of unit length, i.e. $|\boldsymbol{q}| = 1$ should be satisfied. A quaternion can be seen as an encoding of an *axis-angle representation*. The quaternion can then be interpreted as

$$\boldsymbol{q} = \cos\left(\frac{\theta}{2}\right) + e_x \sin\left(\frac{\theta}{2}\right) i + e_y \sin\left(\frac{\theta}{2}\right) j + e_z \sin\left(\frac{\theta}{2}\right) k, \tag{2.4}$$

where $\mathbf{e} = \begin{bmatrix} e_x & e_y & e_z \end{bmatrix}^T$ is the unit vector that gives the direction of the axis around which the angle θ is applied. An illustration of the *axis-angle rotation* can be viewed in Figure 2.1 Note that, keeping the unit constraint on the quaternion also ensures that the axis $\mathbf{e}$ remains of unit length and in turn this will keep the represented rotation interpretable [28].

The unit constraint, on the orientation quaternions, is one of the few drawbacks with this representation. The constraint demands re-normalization after any operation that is not guaranteed to preserve the constraint, and it can also bring difficulties in optimization algorithms since the unity norm constraint is quadratic [28]. Furthermore, the quaternion representation is redundant in that that $\boldsymbol{q}$ and $-\boldsymbol{q}$ represent the same rotation. To explain, a rotation of magnitude θ around an axis $\boldsymbol{a}$ is equal to a rotation of $-\theta$ around the axis $-\boldsymbol{a}$ pointing in the opposite direction. To remove the redundancy of the representation it is possible to use canonized quaternions instead. In the canonized quaternions the scalar part of the quaternion vector is always positive.

There are a few ways in which a metric of the error between two quaternions can be expressed. In this thesis, we will have an estimation $\hat{\boldsymbol{q}}$ of the ground truth orientation quaternion $\boldsymbol{q}$. A metric of how close the estimation is to the ground truth is necessary for the evaluation of different model performances. As a first alternative to the error metric, the Euclidean distance between two 3D orientations δ_q can be used [30]

$$\delta_q = \min\left(\|\boldsymbol{q} - \hat{\boldsymbol{q}}\|, \|\boldsymbol{q} + \hat{\boldsymbol{q}}\|\right), \tag{2.5}$$

where $\|\,.\,\|$ is the 2-norm and $\delta_q \in \left[0, \sqrt{2}\right]$.

A second alternative to the error metric is the *quaternion orientation error* or *difference rotation quaternion* [31]

$$\boldsymbol{q}_e = \boldsymbol{q} \oplus \hat{\boldsymbol{q}}^{-1}. \tag{2.6}$$

Here, $\oplus$ indicates the *Hamilton product* [32] and $\hat{\boldsymbol{q}}^{-1}$ is the quaternion conjugation. Note that the difference error quaternion is still a unit quaternion and, as the name indicated, interpreted as the rotation difference between $\boldsymbol{q}$ and $\hat{\boldsymbol{q}}$.

As an extension of difference rotation quaternion, the angle of the difference rotation θ_{q_e} can also be calculated. According to equation (2.4) the scalar part of $\boldsymbol{q}_e$ will be equal to $\cos\left(\frac{\theta_{q_e}}{2}\right)$ and further, from extension of equation (2.6) the scalar part of $\boldsymbol{q}_e$ will also be equal to the dot product of $\boldsymbol{q}$ and $\hat{\boldsymbol{q}}$ which gives [30]

$$\theta_{q_e} = 2\arccos\left(\|\boldsymbol{q} \cdot \hat{\boldsymbol{q}}\|\right), \tag{2.7}$$

where $\theta_{q_e} \in [0, \pi]$.

In conclusion, we have the Euclidean distance δ_q, the difference rotation quaternion $\boldsymbol{q}_e$, and finally, the angle of the distance quaternion θ_{q_e} as possible error metrics. Neither alternative is directly comprehensible, and as a possibility, one can also calculate the error in Euler angles from the difference rotation quaternion. The error in Euler angle is at first hand meant to act as a visualization tool, and the error is easiest retrieved by simply transforming $\boldsymbol{q}_e$ to Euler angles. The transformation from quaternions to Euler angles is, for example, derived in [28].

2.2 The Inertia Measurement Unit

A device containing a three-axis gyroscope and a three-axis accelerometer is commonly referred to as an IMU. There are a wide range of IMU application areas such as robotics, navigation systems and virtual reality [33]. Much of the widespread use of IMUs is a result of the *microelectromechanical system* (MEMS) technology with which the IMUs have become both space efficien and cheap [9]. The gyroscope measures angular velocity while the accelerometer measures specific force i.e. acceleration relative to free-fall. Specific force is different from the more conventional linear acceleration. The specific force of a stationary object on the surface of earth will be constant $1g$ upwards while the linear acceleration in the same case will be zero.

When using IMUs in applications, to estimate position and orientation, one major concern is the presence of integration drift [14]. Since the measurements are derivatives of the desired observation the usage of dead reckoning, where the state is tracked by integrating the derivatives, is appealing. However, measurements are imperfect and the raw sensor readings will include both noise and biases. Integrating these measurements will lead to accumulation of errors over time. A signal with a constant error, integrated once, will lead to an estimation error that grows linearly. Integrating once more and the error will exhibit a quadratic growth over time [34].

In the objective of orientation estimation, assumptions about periods of zero linear accelerations are common for the purpose of utilizing the accelerometer data to act as a corrector for the roll and pitch angles and thus prevent the drift caused by integrating the gyro. Still, the drift issue remains for the yaw angle which entitles the use of additional sensors such as a magnetometer or *global navigation satellite system* (GNSS) in order to include a correctional term in the remaining angle. However, magnetometers are sensitive to disturbances from ferromagnetic objects and electrical devices in their surroundings [16] while GNSS perform poorly when the satellite signal is obstructed. Thus, neither is suitable in every application. In contrast, the measurements from the gyroscope and accelerometer are independent of any such environmental attributes [35].

Along side the drift issue, IMUs also suffers from complicated error sources that have both computational (numeric range and resolution), random, as well as systematic sources [18]. Generally, the collected IMU error sources are [36]:

- **Input Range**
  The input range is the maximum magnitudes the IMU can measure. This error source is more application dependent and need to be considered in the design process rather than in the fusion process, specifically if the application is of high-dynamic-range and risk of saturation may occur. Also to be considered is the amount of vibrations since substantial vibrations can lead to saturation if the device is at close to its dynamic-range border. Another subject of the input range is the bandwidth which can cause aliasing.

- **Bias Repeatabilty** [deg/hr], [m/s^2]
  Due to a number of different influences such as altering physical properties and initial conditions, the IMU *initial bias* will be different each time the device restarts. However, if the bias is very similar each restart (i.e. the repeatability is high), the bias can to some extent be compensated for by tuning he device. Bias repeatability is frequently also referred to as Run-to-Run or Turn-On to Turn-On Bias.

- **Bias Stability** [deg/hr/hr], [$m/s^2/hr$]
  During the IMU run-rime the *initial bias*, that is sometimes simplified as a constant, changes. Generally, this is caused by changing temperature or mechanical stress and it is common that manufactures adds temperature compensation to the IMU which increases the stability. This error source also goes by the name In-Run Bias.

- **Scale Factor** [ppm]
  The scale factor error describes the ratio between the sensed input and measured output of the sensor. The ideal ratio is, of course, one-to-one but when this is not the case, the output is proportional to the input with a scale factor. Additionally, the scale factor also has a non-linear part that can be described separately, but often is described with one collective value.

- **Sensor Misalignment** [mrad]
  The six *degrees of freedom* (DoF) IMU has three gyroscopes and three accelerometers set orthogonal to each other. The placement is however imperfect causing correlation between sensors. Sensor misalignment errors originates both from non-orthogonality between accelerometers or gyroscopes (within sensor sets) as well as between sensor sets. There is also the phenomenon of package misalignment where the attachment of IMU is misaligned with the object that it is attached to.

- **G Dependency** [deg/hr], [m/s^2]
  When a Coriolis effect based gyroscope is subjected to acceleration along its oscillation axis the acceleration will have an impact on the sensor readings. This effect is called G dependency, because gravity is often the main source of acceleration.

- **Latency**
  When IMU sensors are aided by cooperating sensors, latency can arise. In other words the latency error source emerge when IMU and the aiding sensor both senses same motion but their timing disagree. If this timing disagreement is substantial this error can become problematic.

2.3 Conventional IMU Data Fusion

There are many possible filter implementations that can be used to estimate the absolute orientation from IMU or MARG data. In this section two approachable ways will be covered.

2.3.1 The Complementary Filter

The complementary filter provides a straightforward solution to attain orientation estimates from IMU data. The accelerometer is a good indication of orientation under the assumption of stationary, or at least slow-moving conditions. Furthermore, even though the accelerometer output is noisy, it is accurate over long periods of time and does not suffer from accumulating drift when estimating orientation [16]. In contrast, the gyroscope provides a reliable measurement of the angular rate at each time step but integration will cause errors to accumulate over time. Hence, the accuracy properties of the accelerometer and the gyroscope complement each other to assure accuracy on long and short timescales, respectively [16].

The main idea in the complementary filter is to low-pass filter the angle estimates from the long-term reliable accelerometer readings while high-pass filtering the integrated estimates from the short-term reliable gyroscope readings. Despite the relatively simple implementation, the complementary filter can provide smooth and accurate estimates and demands less computation than for instance the EKF [37]. The structure of this filtering technique can be viewed, more in-depth, in Figure 2.2. Further, the complementary filter using only IMU can only estimate the roll and pitch but given MARG measurements the complementary filter can also estimate yaw, such an implementation can be found in [16].

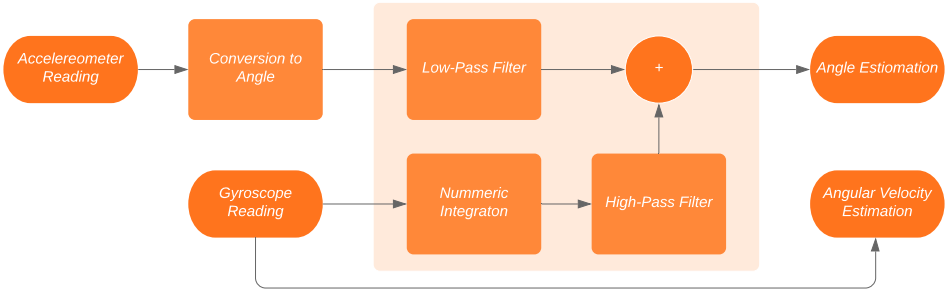

Figure 2.2: Block diagram of a complementary filter estimation angle and angular velocity.

2.3.2 The Extended Kalman Filter

As mentioned in the introduction, the EKF is a commonly used filter in the field of sensor fusion, with the aim to take nonlinear effects into account [16]. The EKF is a two-step algorithm, constituted of the prediction and update steps.

The non-linearity of the EKF lies in the motion model $f(.)$ and measurement model $h(.)$ which can be non-linear as long as they are differentiable. Here, in equation (2.8) and (2.9), F_k and H_k are defined as the Jacobians of $f(.)$ and $g(.)$ with respect to the state $\boldsymbol{x}$. The Jacobiance are evaluated at the current state estimation, and in the case of F_k also at the current control signal.

$$F_k = \left.\frac{\partial f}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k} \tag{2.8}$$

$$H_k = \left.\frac{\partial h}{\partial \boldsymbol{x}}\right|_{\hat{\boldsymbol{x}}_{k|k-1}} \tag{2.9}$$

Prediction step

$$\text{State prediction:} \quad \hat{\boldsymbol{x}}_{k|k-1} = f(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k). \tag{2.10}$$

$$\text{Covariance prediction:} \quad P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q. \tag{2.11}$$

In the prediction step, the predicted state vector $\hat{\boldsymbol{x}}_{k|k-1}$ is computed via the propagation of the best state estimate from the previous time step $\hat{\boldsymbol{x}}_{k-1|k-1}$ and the control vector of the current time step $\boldsymbol{u}_k$ through the motion model $f(.)$ as given in equation (2.10) [38]. Also in the prediction, the covariance prediction matrix $P_{k|k-1}$ is calculated as in equation (2.11) where Q denotes the process noise covariance matrix [38].

Update step

$$\text{Innovation:} \quad \boldsymbol{v}_k = \boldsymbol{y}_k - h(\hat{\boldsymbol{x}}_{k|k-1}) \tag{2.12}$$

$$\text{Innovation covariance:} \quad S_k = H_k P_{k|k-1} H_k^T + R \tag{2.13}$$

$$\text{Kalman gain:} \quad K_k = P_{k|k-1} H_k \boldsymbol{S}_k^{-1} \tag{2.14}$$

$$\text{State update:} \quad \hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + K_k \boldsymbol{v}_k \tag{2.15}$$

$$\text{Covariance update:} \quad P_{k|k} = P_{k|k-1} - K_k H_k P_{k|k-1} \tag{2.16}$$

The goal of the update step is to calculate the best state estimation and covariance given new observations. First, the innovation $\boldsymbol{v}_k$ is attained as in equation (2.12) and can be understood as the difference between the new given observation $\boldsymbol{y}_k$ and the expectation of the observation given the state prediction $\hat{\boldsymbol{x}}_{k|k-1}$. Second, the innovation covariance S_k is calculated as in (2.13) where R is measurement noise covariance matrix [38]. Third, the Kalman gain K_k is achieved according to (2.14). The Kalman gain is then the influence that decides the relative impact of the innovation compared to state prediction, which can be viewed in equation (2.15) where the new updated state is estimated $\hat{\boldsymbol{x}}_{k|k}$. Finally, covariance is also updated as in equation (2.16).

Up until now, the general steps of the EKF have been described, and in order to adapt the EKF to produce orientation estimates, a number of design choices are required. Firstly, the state vector can be designed with different orientation conventions, such as Euler angles or unit quaternions. Moreover, the state can be composed of either only the actual orientation or, the orientation and the rotation rate or, augmented to also allow estimation of sensor biases. Then, depending of the design of the state vector, motion and measurement models have to be developed for chosen state representation. Furthermore, the covariance matrices Q and R introduce tuning parameters. The measurement noise covariance Q can usually be set based on the uncertainties of the sensors. The process noise covariance R is more tricky however, and should reflect the certainty of the chosen motion model. The ratio between the trace of the two matrices also affects the performance. The ratio $tr(Q)/tr(R)$ is sometimes referred to as *signal to noise ratio* (SNR), and has large implications on the convergence rate [5]. The specific design choices made in this thesis are stated in the methods chapter, Section 3.6, but as can be understood there exist a variety of possible implementations. A few examples of implementations which adopt unit quaternions as the orientation encoding can be found in [13],[16],[39].

2.4 Machine Learning and Neural Networks

NNs have recently become the most prominent approach in ML due to their ability to model complex relationships in data. An NN consists of a layered structure of operations that maps input features to some output. One of the simplest NN layers is the linear feed-forward layer. The linear feed-forward layer performs a linear mapping $\mathbb{R}^n \to \mathbb{R}^m$ of a feature vector, $x \in \mathbb{R}^n$. It does this by taking the matrix-vector product of a weight matrix $W \in \mathbb{R}^{m \times n}$ and x and by adding a bias term $b \in \mathbb{R}^m$

$$z(x, W, b) = Wx + b. \tag{2.17}$$

However, layering many of these linear feed-forward layers is of no interest, as they can be reduced to a single layer operation. Hence, to model more complex relationships, a key feture of NNs are their non-linearities. In the context of NNs, non-linearities are referred to as *activation functions*[1], since the general feature of the non-linearity is (typically) to suppress small input and let through large input, i.e. be inactive or activate. One such function is the *sigmoid function*,

$$\sigma(z) = \frac{e^z}{e^z + 1} \tag{2.18}$$

which will be close to zero for $z < 0$ and 1 for $z > 0$. The output of an activation function is referred to as an *activation*,

$$a = \sigma(z(x, W, b)). \tag{2.19}$$

Alternating linear feed-forward layers and activation functions gives rise to models that, if the different weight matrices and bias vectors are carefully selected, can model almost any relationship. NNs consisting of only linear feed-forward layers and activation functions are sometimes referred to as *multi layer perceptrons* (MLPs).

The most common way of selecting weights and biases is to randomly initialize them, and subsequently tune them using *stochastic gradient decent* (SGD) based optimization techniques in utilizing gradients computed with back propagation [40]. This is called *training* and is often done using examples of input with known desired output, which is referred to as supervised learning. A specific SGD based optimization technique which has been widely adopted since its release in 2014 is the ADAM optimizer, which key feature is *adaptive moment estimation*[41].

A *loss function* is a measure of the error between model output $f(x, \mathcal{W})$ and desired output y, $\mathcal{L}(f(x, \mathcal{W}), y)$. It is this function that is minimized with respect to the model parameters $\mathcal{W}$, (i.e. W and b) during training. Minimizing $\mathcal{L}$ with respect to $\mathcal{W}$ with SGD-based methods requires the computation of $\frac{\partial \mathcal{L}}{\partial \mathcal{W}}$, which is done with back propagation. The model weights are updated by subtracting the gradient, scaled by some factor. The scaling factor applied to the gradient is referred to as

[1]The term activation function stems from the fact that artificial NNs are inspired by biological NNs. Biological neurons fire or *activate* when they are excited sufficiently enough to overcome inhibitory input. Likewise, an artificial neuron modeled as $\sigma(z(x, W, b))$ will fire if weighted input excites the artificial neuron sufficiently.

the *learning rate*. In regression models, some simple examples of loss functions are *mean square error* and *mean absolute error*.

Apart from the mentioned sigmoid activation function

$$\tanh(z) = \frac{2}{1 + e^{-2z}} - 1, \tag{2.20}$$

and the *Rectified Linear Unit*

$$\text{ReLu}(z) = \max(0, z) \tag{2.21}$$

will be used in the following sections.

2.4.1 Recurrent Neural Networks and Long Short-Term Memory

While MLP networks are static networks and unaffected by the input of the previous time step, RNNs are able to process time series information [42]. RNNs achieve this by using a *hidden state* h which stores information about the previous inputs x to the network, enabling the detection of temporal correlations between events separated by time [43]. This ability motivates why RNNs are useful in forecasting applications using sequence data. In short, RNNs adds memory to the system and in the most simple structure of an RNN, the activation a at time t is available to the same layer at the next time step, illustrated in Figure 2.3.

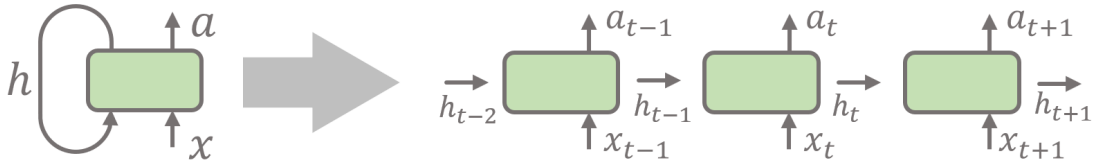

Figure 2.3: Illustration of an RNN unfolded trough time.

As mentioned, NNs are often trained using SGD-based optimization methods. Due to the hidden state, the gradient methods needs to be adjusted and one well-established way to train RNNs is to use *Back Propagation Through Time* (BPTT) [44]. The problem with BPTT, for RNNs, is that gradient forms a product of Jacobian matrices for the different time steps [43]. Hence, if the elements in the Jacobians are less than one this will cause the norm of the gradient to decrease exponentially towards zero. Unfortunately, the opposite can also occur and the norm of the gradient will then grow exponentially. These issues makes it impractical to train RNNs and are referred to as the *vanishing* and *exploding gradients problem* respectively [43].

As a solution to the problem with vanishing as well as exploding gradients, the LSTM unit was proposed in 1997 [45]. In the original paper the LSTM unit consisted of two multiplicative gate units, namely the input gate and the output gate. Shortly after that, in 1999 [46] an additional forget gate was proposed and, today the architecture where each LSTM unit contains all three gates is the most common. The verison with a forget gate is shown in Figure 2.4.

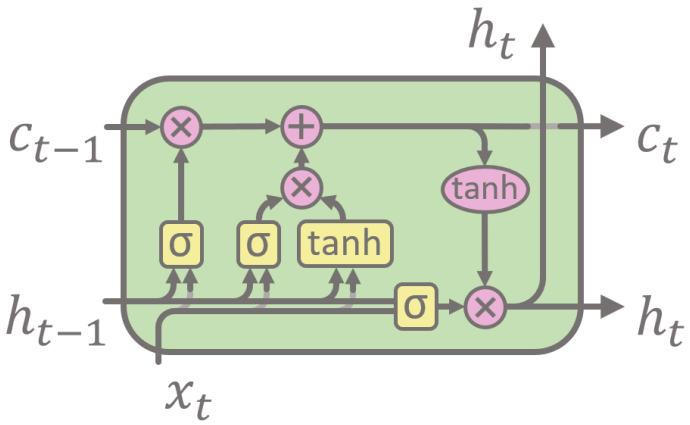

Figure 2.4: A illustration of the LSTM layer. Yellow boxes indicate linear layers with activation functions and pink ellipses indicate element-wise operations.

At each time step the LSTM unit is feed information from three sources: i) the input vector x_t is given, ii) the hidden state vector of the last time step h_{t-1} which is also the output of the last time step, and iii) the cell state from last time step c_{t-1}. In the original paper [45], the cell state is referred to as the constant error carrousel which runs back and creates a connection to the past. The functions most important of the cell state is firstly, that it can store information over arbitrarily long periods of time which gives the LSTM its memory capabilities. Secondly, by construction, the gradient of the cell state has an additive property, instead of multiplicative as in the case with the original RNN, which prevents the gradient from vanishing during backpropagation [44].

As mentioned, the LSTM unit includes the three multiplicative gates whose purposes are to preserve and regulate the cell state. Firstly, the activation from the forget gate f_t enables the possibility for the network to learn what information to forget and what to remember. The activation is given by [46]

$$f_t = \sigma\left(W_f \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_f\right), \tag{2.22}$$

where W_f is the learned weight matrix and b_f the learned bias of the forget gate. The σ-function assures that the activation f_t will be bounded between 0 and 1. Hence, when f_t is element wise multiplied with the cell state the value of f_t determines how much of each element in c_{t-1} will be remembered. One can say that the forget gate's activation can reset the cell state [44].

Secondly, the input gate activation vector i_t determines from which elements, of the current input x_t, information should be added to the cell state. The input gate activation vector is given by

$$i_t = \sigma\left(W_i \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_i\right), \tag{2.23}$$

which is similar to the calculation of f_t only with different weight matrix and bias vector. The input gate activation is then multiplied with the input activation vector $\tilde{c}_t$ which contains information from the current input.

$$\tilde{c}_t = \tanh\left(W_c \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_c\right) \tag{2.24}$$

In short, the input gate and $\tilde{c}_t$ enables writing to the memory of the LSTM and altogether, equations (2.22), (2.23) and (2.24) determine what the updated cell state will be [45],[46]

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t. \tag{2.25}$$

Here, "$\cdot$" imply element wise multiplication.

Lastly, the activation of the output gate o_t decides what information, from the current cell state, to output and consequently enables reading from the cell state.

$$o_t = \sigma\left(W_o \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} + b_o\right) \tag{2.26}$$

Moreover, the new hidden state and the output of the current time step is given by

$$h_t = o_t \cdot \tanh(c_t). \tag{2.27}$$

All in all, the multiplicative gates and the cell states constitutes the majority of what is required to understand the basics of the LSTM unit [45],[46].

2.4.2 The Recurrent Neural Filter

The RNF, presented in reference [24], is an autoencoder/decoder architecture based on LSTMs and inspired by Bayesian filtering. Instead of replacing the conventional Bayesian filtering framework with a single "block" network, the key idea behind the RNF is to benefit from the structure of the Bayesian filtering steps. The motivation behind this is that it enables filtering when the system dynamics are unknown while still maneuvering missing data points. The encoder portion of the architecture is comprised of LSTM layers responsible for the different Bayesian filter steps. These LSTMs share a single cell state and hidden state[2], h, which they modify sequentially. The architecture of the RNF is illustrated in Figure 2.5. The decoder consists of an MLP, and is responsible for converting the network state representation to the desired estimate.

[2]Here h refers to both the cell state and the hidden state.

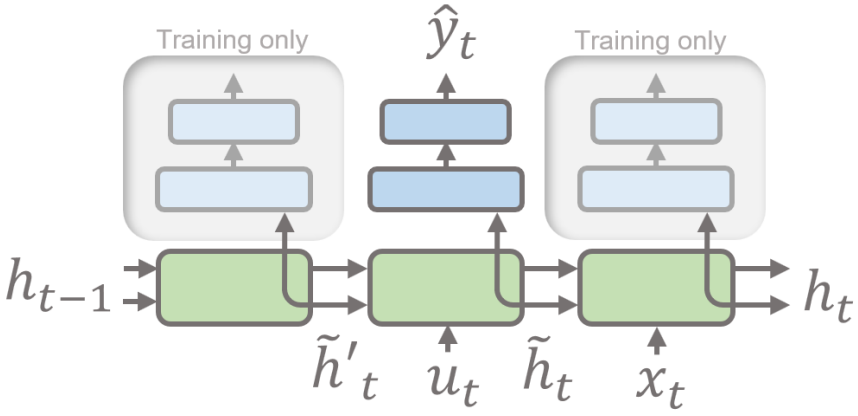

Figure 2.5: The Recurrent Neural Filter as presented in [24].

The RNF in [24] has three LSTM layers; propagation, input dynamics and error correction. The propagation step and the input dynamics step are together the equivalent of propagation step in a Bayesian filter. The propagation LSTM is solely intended for propagation and does not take any input beyond the shared hidden state. The input dynamics LSTM takes the control signal or input signal, u, as input and is intended to adjust the prediction made by the propagation LSTM. The last of the three horizontal LSTMs, the error corrector, takes the measurements, x, and adjusts the prediction made to take the measurements into account. This is analogous to the update step in the Bayesian framework.

To enforce a consistent representation of the state estimation in the hidden state, h, two measures are taken. i) The three LSTMs are trained with the same decoder and ii) *skip training* is employed. In skip training the elements of $\tilde{h}_t$ and h_t are randomly overwritten with values from previous states, $\tilde{h}'_t$ and $\tilde{h}_t$ respectively. The probability of this skip to occur is a tune-able hyperparameter called *skip-rate*.

2.4.3 Estimating Covariance

Knowing the error covariance of the estimates produced by the filter networks is useful for fusing the estimated orientation with other estimates, and also for evaluating performance. In reference [47], they explore modeling of uncertainty of state estimations from neural networks. They model their k-dimensional state estimation as a multivariate Gaussian,

$$p(\mathbf{y}|x) = \frac{1}{\sqrt{(2\pi)^k|\Sigma(x)|}} e^{-\frac{1}{2}(\mathbf{y}-f(\mathbf{x}))^T\Sigma(x)(\mathbf{y}-f(x))}, \tag{2.28}$$

in the same way the state estimate in a Kalman filter is modeled. Where $\mathbf{y}$ is the ground truth. The networks are trained to output the mean, $\mu \in \mathbb{R}^k$, and a parameterized covaraince matrix $\Sigma(s, r)$ where $s \in \mathbb{R}^k$ is the vector parameterization of the diagonal elements and $r \in \mathbb{R}^{k(k-1)/2}$ is a vector parameterization of the off-diagonal elements, see equation (2.30). The mean and covariance can be expressed

as a function of the network input x,

$$\begin{aligned} \mu &= f(x) \\ \Sigma(s, r) &= \Sigma(x). \end{aligned} \tag{2.29}$$

The parameterization of the covariance matrix,

$$\Sigma_{ij}(s, r) = \begin{cases} i = j, & \sigma_i^2 = e^{s_i} \\ j < i, & \rho_{ij}\sigma_i\sigma_j = \tanh(r_{ij})\sqrt{e^{s_i}e^{s_j}} \end{cases} \tag{2.30}$$

describes the diagonal and the upper triangular portion of the matrix with i and j being row and column indices both ranging from 1 to k. The covariance matrix is symmetric, and lower triangular portion follows from the upper.

Minimizing the negative log likelihood of the parameterized multivariate Gaussian distribution (see eq (2.28)) is what allows for simultaneously training the model to estimate the mean as well as the error covariance. Hence

$$\mathcal{L}(x, \mathbf{y}) = \frac{1}{2}(\mathbf{y} - f(x))^T\Sigma(x)(\mathbf{y} - f(x)) + \frac{1}{2}\ln|\Sigma(x)| \tag{2.31}$$

is used as loss function.

The authors of reference [47] also suggest two minor alterations in order to improve numerical stability when using equation (2.31) as loss during training. The first suggestion is multiplying r with a constant $(1 - \epsilon)$, with $\epsilon \approx 0.001$. The second suggestion is to multiply the input to tanh with a small constant, $\alpha \approx 0.05$, to avoid saturating the activation function.

2.4.4 Regularization

To prevent *overfitting*, of model parameters on noise in the training data, regularization techniques are employed. There are many different ways to achieve regularizing effects. Some of the common approaches are the following:

Early Stopping

When improvements are seen in the training loss but not in the validation loss, it is a key indication of poor generalization to data in the validation set. Early stopping is the practice of stopping the iterative training when no improvements are seen in some chosen metric or in the calculated loss on the validation set [48].

Mini Batches

Mini batches is the term commonly used to refer to the batching of training data. The key idea is to consider several samples in a single update step during training. This is done by computing the gradient of the loss function, evaluated over all the samples in the mini batch, with respect to the model parameters. The impact of noise in the data will be smaller, the gradient will be a better approximation of the gradient computed using the entire dataset, and more general solutions are achieved. Batch processing also improve the efficiency of operations during training [49].

Weight Decay

Weight decay is the introduction of a penalty of weights into the loss function. The most common way of doing this is to use L1 or L2 regularization, where the L1 or L2 norm of the weights is added to the loss function, scaled by some constant. The reasoning behind doing this is that solutions with smaller weights tend to generalize better and be more robust to noise [50].

Batch Normalization

Batch normalization is the practice of normalizing activations of intermediate layers in a neural network based on metrics of a batch. This is done to deal with what is referred to as *covariance shift*. Covariance shift is when the changes in weights of a layer, during training, causes the distribution in its activations to shift. Subsequently, the inputs to the next layer represents new features, which its weights are not tuned for [51].

2.4.4.1 Drop-out

One approach that almost always improves model performance is to train several similar models and average their predictions. However, this is an expensive approach. Drop-out is a method inspired by this averaging of many different models, and is done by dropping out a subset of the model, resulting in a "thinned" network. By sampling a different subset network during each training step, and only updating the weights of this subset, a regularizing effect is achieved [52].

2.4.5 Feature Scaling

The numerical range of the input features affect the rate of convergence of NNs during training. If the features are ill-conditioned[3] SGD based optimization will be slow to converge. This can be attributed both to oscillating network weights and the differing magnitude of the update steps for the first layer and the consecutive layers. To improve the convergence rate, the input features and the output labels are often re-scaled [53]. Two common ways to do feature scaling are *min-max normalization*, equation (2.34), and *standardization*, equation (2.33) [54].

$$\text{std}(x) = \sqrt{\frac{1}{N}\Sigma_{i=1}^{N}\left(x_i - \text{mean}(x)\right)^2} \tag{2.32}$$

$$z = \frac{x - \text{mean}(x)}{\text{std}(x)} \tag{2.33}$$

$$z = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.34}$$

[3]Features with magnitude that differs vastly results in the gradient of the loss function with respect to the model parameters to vary in magnitude as well. As a consequence, a learning rate appropriate for one model parameter will be much too large for another, causing either unstable or slow training. Features with large magnitude can also have a disproportionate influence on predictions.

The mean and standard deviation or the min and max are computed from the test data. All data fed to the network is then scaled in the same way, both during training and inference. Min-max normalization of features is an attempt to ensure that the scaled features are constrained to $[0, 1]$. Likewise, standardisation is an attempt to ensure that the distribution of the scaled features has zero mean and variance one.

2.4.6 Domain Randomization

When training ML models on synthetic data there is often a *reality gap*, where the simulated data differs from data captured in the real world. A simple way to deal with some of these differences is to use *domain randomization* (DR). In DR relevant parameters in the simulation are randomized to reflect the uncertainty of their real world counterpart [55].

3
Methods

This chapter will cover the methods used to answer the questions posed in Section 1.2.1. First, the tools used throughout the process are described, followed by a description of the problem set-up, and the data sets used. Then, the model architectures, as well as the loss function and metrics used when training and evaluating the models, are described. Subsequently, the benchmarks used for performance evaluation of the model architectures are presented, and the considerations done when choosing them are accounted for. Finally, the experiments performed and the hyperparameters used during training are reported on.

3.1 Machine Learning Framework

TensorFlow is a machine learning interface that supports both training and inference phases. The interface is developed open source by Google and suitable for most machine learning application areas. The flexibility of TensorFlow lies in that it allows for experimenting with new models and researching with sufficiently high performance while still robust enough to allow for the deployment of machine learning models [56]. TensorFlow is categorized as a low-level *deep learning* framework while its *application programming interface* Keras can be considered a high-level framework. Keras is most commonly used with TensorFlow as backend, although it supports a few other low-level frameworks [57].

3.2 Separation into 2D and 3D Problem

The main focus of this thesis is to estimate absolute orientation utilizing ML approaches. Related literature, [11] and [19], employs supervised learning, which is an accessible approach, and therefore also adopted in this thesis. Here, a separation of the problem into two parts will be established.

To investigate the general suitability of utilizing an ML approach for sensor fusion purposes a simplified 2-dimensional problem, with fewer DoFs, is set up. Here, the sensor device will be allowed to rotate around one single axis, and therefore only one angle will be estimated.

In order to avoid the discontinuity arising where angles transition between $+\pi$ and $-\pi$, the angles fed to and from the network are represented using $\left[\sin\theta, \cos\theta\right]$. This representation is also establishing normalization of the inputs and outputs to the

NNs. Further, the representation can easily be converted back into angles using $\theta = \arctan 2(\sin\theta, \cos\theta)$.

As a second task, the problem of retrieving an absolute orientation estimation in 3D shall be investigated. As explained in Section 2.1, a quaternion representation of the absolute orientation can be beneficial for the 3D case. Further, the quaternions used will be canonized such that the scalar part of the orientation quaternion is always positive. The reason for adopting canonized quaternions is that it simplifies the problem for the network such that the network does not have to learn two representations for the same orientation, requiring more complex models of the output distributions [27]. In short, the 3D networks will be trained to estimate the canonized orientation quaternion representation with supervised learning.

3.3 Datasets

Two different data sets are used for separate problems of estimation, a 2D orientation, and an absolute orientation in 3D, respectively. Namely, the sufficient number of correctors, in 2D and 3D, have influenced the choice of datasets.

3.3.1 Synthetic 2D Data

Gnss-ins-sim [58] is a simulation tool for generating 3D IMU, MARG, and GNSS sensor data. It is also capable of generating 2D data, which is how it is used in the following experiments. The tool consists of a python module that enables specific settings to mimic the true behavior of IMU sensor noise (bias, random walk, and bias stability). The module enables the attainment of the ground truth both in quaternions and Euler angles. In 2D, it is sufficient to represent the orientation with a single axis rotation, and therefore the quaternion representation will not be necessary.

A training set containing 60 000 sequences, each consisting of 50 samples sampled at 100 Hz, is generated with DR of the noise parameters (see Table 3.1). The sequence length had to be limited mainly to reduce computation per update step, but also to allow for a large variety of sequences relative to the dataset size. Similarly, a test set containing 500 sequences, each consisting of 1 000 samples sampled at the same frequency, is generated in the same noise domain. In these simulations, the initial orientation is sampled from $\mathcal{U}(-\pi, \pi)$, and the initial angular velocity was set to zero. The data is generated without linear acceleration. During training, 5% of the training set is used for validation.

[1] These are typical values, based on [59]

[2] This is the correlation coefficient used by gnss-ins-sim to model the bias as a first order Gauss-Markov process [60].

Table 3.1: The distributions used for domain randomization of the noise parameters. These distributions are sampled for each generated 2D sequence. $\mathbf{1}_2$ is a vector of length two containing all ones.

Error source	**Unit**	**Typ.**[1]	**Distribution**
gyro bias	$[deg/hr]$	1440	$\mathcal{U}(\pm[400, 2000])$
gyro angle random walk	$[\frac{deg}{\sqrt{hr}}]$	0.15	$\mathcal{U}(0.1, 0.2)$
gyro bias instability	$[deg/hr]$	2	$\mathcal{U}(1, 3)$
gyro bias instability corr.[2]	-	-	100
acc. bias	$[m/s^2]$	$1.37 \cdot 10^{-2}$	$\mathcal{U}(\mathbf{1}_2 10^{-2}, \mathbf{1}_2 2 \cdot 10^{-2})$
acc. vel. random walk	$[\frac{m/s}{\sqrt{hr}}]$	0.012	$\mathcal{U}(\pm\mathbf{1}_2[0.008, 0.022])$
acc. bias instability	$[m/s^2]$	$3.6 \cdot 10^{-5}$	$\mathcal{U}(\mathbf{1}_2 2 \cdot 10^{-5}, \mathbf{1}_2 5 \cdot 10^{-5})$
acc. bias instability corr.[2]	-	-	$\mathbf{1}_2 200$

3.3.2 3D MARG Dataset

For training the models in the 3D case, the OxIOD dataset [23] is used. In contrast to the 2D dataset, OxIOD is not synthetic. It consists of 158 sequences totaling 43 km in traveled distance. Each sequence contains MARG sensor readings as well as a reference orientation from a computer vision system. The dataset is provided with a suggested training/testing split of the data[3], which is used in the following 3D experiments.

The authors of the OxIOD dataset compares it with similar datasets and presents a higher accuracy ground truth than other datasets. However, they only compare translational accuracy [23]. This reported accuracy is 0.5 mm at 100 Hz [4].

OxIOD was chosen for its larger size compared to similar datasets, while still providing an absolute orientation. Further, the dataset includes different types of motions making it somewhat diverse. The data sequences were recorded using a smartphone attached to different parts of a person[5]. There are, for example, sequences with the phone in a walking person's pocket, hand, handbag, and on a trolley. There are also variations where the person is running, walking, and walking slowly. An example of a sequence, with a person carrying the phone in a handbag whilst walking, can be seen in Figure 3.1. reference.

[3]The suggested train/test split for OxIOD is to use specific sequences for training and testing, covering each of the different types of motion.

[4]For reference, the datasets with the second-highest accuracy are TUM IV (1 mm, 200 Hz) [61] and the dataset that OriNet is trained on, EuRoC MAV (1 mm, 200 Hz) [62].

[5]To the best of our knowledge, no PhD student was severely harmed during the data collection.

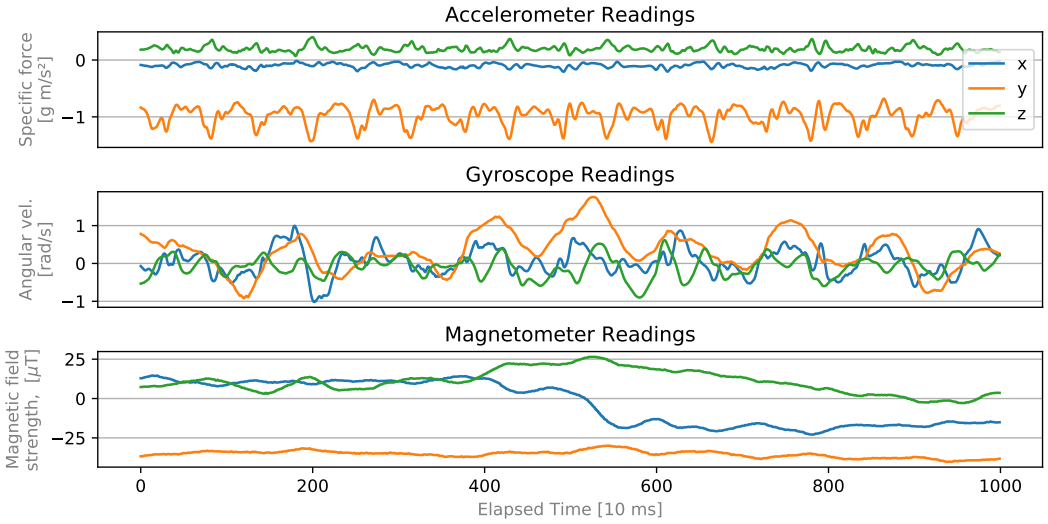

Figure 3.1: A 10 s data sample from the test set (OxIOD), showing readings from the accelerometer, gyroscope, and magnetometer in the sensors body frame.

All of the data in OxIOD is sampled at 100 Hz. We processed the data by dividing it into shorter sequences: a training set containing 51 784 sequences, each consisting of 50 samples and a test set containing 342 sequences, each consisting of 1 000 samples. Of the training data, 5% is used for validation.

3.4 Model Architectures

Two main types of model structures are develop for the purpose of estimation orientation. Both of architectures follow an encoder-decoder structure with a recurrent encoder consisting of LSTM units. The encoder outputs both the estimate as well a parameterized covariance matrix from Section 2.4.3.

3.4.1 The Recurrent Neural Network

Given the time-series nature of the IMU/MARG data and the dynamics of the orientation state, some sort of RNN is a natural choice, since RNNs do not require a fixed look-back window. More specifically, LSTMs were considered, in favor of simpler RNNs. This, due to their robustness towards vanishing gradients, as described in Section 2.4.1. Also, despite the success of attention-based models in fields such as natural language processing and image captioning, the fact that they rely on a discretization of the output makes them unsuitable when dealing with continuous outputs [24].

The proposed RNN models, designed and evaluated in this thesis, have a simple encoder-decoder structure. The encoder consists of a single LSTM layer, while the decoder consists of an MLP with the first layer having double the size of the hidden state. The second layer of the MLP has $2k + \frac{k(k-1)}{2}$ units, with k being the dimension of the estimate ($k = 2$ in the 2D case and $k = 4$ in the 3D case).

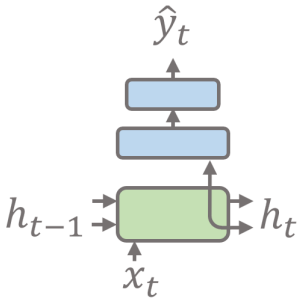

Figure 3.2: Illustration of the RNN architecture evaluated. The green box represents an LSTM layer and the blue boxes represents fully connected layers.

3.4.2 The Recurrent Neural Filter

A variation of the RNF presented in Section 2.4.2 is evaluated, in which only two of the three separate LSTM encoders are used. Since there is no natural control signal in the case of orientation estimation from IMU/MARG data, the input dynamics layer is omitted and no control signal is used. This results in a slightly different RNF than the one proposed in reference [24] where the part of the RNF that models input dynamics has been removed. The resulting structure is illustrated in Figure 3.3. The two steps then constitute a prediction and an update step, where all measurements are given to the update part. The RNF model uses the same type of decoder as the previously described RNN. During training the decoder decodes the hidden state of both the LSTMs comprising the propagation step, and the LSTM comprising the error correction step. [24]

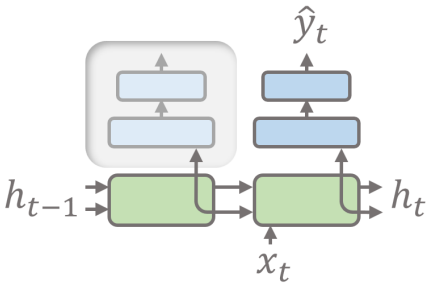

Figure 3.3: Illustration of the RNF architecture evaluated. The green boxes represents an LSTM layer and the blue boxes represents fully connected layers.

3.5 Loss Function and Error Metrics

The *negative log likelihood* loss function and the *root mean square of Euclidean distance* metrics used have a direct relation to the performance of the ML model. While the loss function has a direct connection to the training of the network where it is minimized, the metric is detached from the training and can be chosen more freely.

3.5.1 Loss Function

Both 2D and 3D experiments have been performed with the negative log likelihood function (equation (2.31)) which was described in Section 2.4.3. The benefit of estimating the error covariance of the estimates produced is first that it is useful for fusing the estimate with other information, and also for evaluating the performance of the estimates.

3.5.2 Error Metrics

Two types of error metrics are used to measure and compare performance of the trained models over the entire test set. Given a time series error vector δ the *root mean square error* RMSE can be calculated as

$$\text{RMSE}(\delta) = \sqrt{\frac{1}{n}(\delta_1^2 + ... + \delta_n^2)} \tag{3.1}$$

where δ_i is the scalar error at time $i \in [1, n]$. Averaging over the RMSE gives, our first error metric, the *average root mean square error* (aRMSE). Similarly we attain our second error metric from averaging the quantity

$$\text{RMSE}_{20}(\delta) = \sqrt{\frac{1}{n-20}(\delta_{21}^2 + ... + \delta_n^2)} \tag{3.2}$$

Including the errors prior to the estimation has converged makes it difficult to compare performance on sequences of different length. The second metric is referred to as the aRMSE_{20} and is utilized to reduce the impact of sequence length on the metric, as well as to provide a metric in which the EKF is not favoured due to being provided the ground truth as prior in the first time step. 20 time steps was found to be sufficient to achieve robustness towards the sequence length.

In the 2D case, the network only estimates one angle. The error δ,

$$\delta = \theta - \hat{\theta} \tag{3.3}$$

in this case, consists of the difference between the ground truth θ and the estimation $\hat{\theta}$ in degrees. This error δ is then used to calculate the aRMSE and the aRMSE_{20} as described in equations (3.1) and (3.2).

From Section 2.1 we have the Euclidean distance δ_q, the difference error quaternion $\boldsymbol{q}_e$ and the angle of the error quaternion θ_{q_e} as possible error metrics in 3D rotations represented by quaternions. However, since the difference error quaternion is not a scalar, it is inconvenient to use it as a metric. Furthermore, the angle of the error quaternion brings a more complex calculation compared to the Euclidean distance. Therefore, the Euclidean distance δ_q is chosen as the 3D-error metric, which we calculate the aRMSE and the aRMSE_{20} as in equations (3.1) and (3.2).

3.6 Benchmarks

Two EKFs, one for 2D orientation estimation and one for 3D orientation estimation, are used as a benchmark for model performance. Section 2.3.2 introduced the general theory behind the EKF, here the specific design choices of each of the benchmark EKFs will be presented.

2D EKF

The EKF benchmark used in the 2D case utilize a state vector $\boldsymbol{x}$ consisting of the angle θ and its derivative ω,

$$\boldsymbol{x} = \begin{bmatrix} \theta \\ \omega \end{bmatrix}. \tag{3.4}$$

The chosen motion model, $f(\boldsymbol{x})$, assumes constant angular velocity and models the angle by integrating the velocity over time. Further, the motion model constrains θ by wrapping it to the range $[-\pi,\ \pi)$, in order to avoid the the problem of $\theta = 2\pi + \theta$. The wrapping function $\mathrm{w}(.)_{[-\pi,\,\pi)}$ is defined as

$$\mathrm{w}_{[-\pi,\,\pi)}(x) = \mod(x + \pi,\ 2\pi) - \pi. \tag{3.5}$$

The motion model and the Jacobian of the motion model, $F(\boldsymbol{x})$, are then given by

$$f(\boldsymbol{x}) = \begin{bmatrix} \mathrm{w}_{[-\pi\ \pi)}(\theta + \Delta t\, \omega) \\ \omega \end{bmatrix}, \qquad F(\boldsymbol{x}) = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{3.6}$$

.

The update step of the 2D EKF is rather straightforward. The chosen measurement vector $\boldsymbol{y}$ consists of the gyroscope reading ω_g, and the sin(.) and cos(.) of the angle retrieved from the normalized accelerometer reading θ_a,

$$\boldsymbol{y} = \begin{bmatrix} \omega_g \\ \sin(\theta_a) \\ \cos(\theta_a) \end{bmatrix}. \tag{3.7}$$

The measurement model $h(\boldsymbol{x})$ and its Jacobian $H(\boldsymbol{x})$ are defined as

$$h(\boldsymbol{x}) = \begin{bmatrix} \omega \\ \sin(\theta) \\ \cos(\theta) \end{bmatrix}, \qquad H(\boldsymbol{x}) = \begin{bmatrix} 1 & 0 \\ 0 & \cos(\theta) \\ 0 & -\sin(\theta) \end{bmatrix}. \tag{3.8}$$

Given the measurement and motion model, the 2D EKF then follows the steps as explained in Section 2.3.2.

3D EKF

In the 3D case, an EKF, estimating the absolute orientation in quaternions, is used to benchmark the developed ML models. The chosen state vector $\boldsymbol{x}$ is

$$\boldsymbol{x}_k = \boldsymbol{q}_k, \tag{3.9}$$

where $\boldsymbol{q}$ is the unit quaternion encoding the true orientation at the discrete time-step k. Further, since the derivative of $\boldsymbol{q}_k$ is excluded from the state, the angular rate $\boldsymbol{\omega}_k$ is instead included in the control signal[6] $\boldsymbol{u}_k$. Note, that the control signal is expressed in the body frame and constitutes a three-axis rotation in Euclidean space. Other design alternatives are for example to use an augmented state vector to estimate the bias of the gyro and the accelerometer. It is also possible to include the quaternion rates as a part of the state vector. Both of these alternatives could potentially improve the estimation accuracy of the EKF, but also increases the complexity of the implementation.

The continuous-time derivative of the orientation quaternion, $\dot{q}$, with respect $\boldsymbol{\omega}$ (see reference [31]) is

$$\dot{\boldsymbol{q}} = \frac{1}{2} \underbrace{\begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{bmatrix}}_{=\Omega(\boldsymbol{\omega})} \boldsymbol{q}, \tag{3.10}$$

which can also be re-written as

$$\dot{\boldsymbol{q}} = \frac{1}{2} \underbrace{\begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix}}_{=\bar{\Omega}(\boldsymbol{q})} \boldsymbol{\omega}. \tag{3.11}$$

Given the derivative, the discretized motion model f can be chosen as

$$f\left(\hat{\boldsymbol{x}}_{k|k-1}, \boldsymbol{u}_k, \boldsymbol{v}_k\right) = \left(I + \frac{\Delta t}{2}\Omega\left(\boldsymbol{u}_k\right)\right)\hat{\boldsymbol{x}}_{k-1|k-1} + \frac{\Delta t}{2}\bar{\Omega}\left(\hat{\boldsymbol{x}}_{k-1|k-1}\right)\boldsymbol{v}_k, \tag{3.12}$$

where Δt is the sampling time, and $\boldsymbol{v}_k$ is the process noise. The process noise is modeled as an additive noise contribution to the angular rate $\boldsymbol{\omega}$ in the body frame, i.e.,

$$\boldsymbol{u}_k = \boldsymbol{\omega}_k + \boldsymbol{v}_k. \tag{3.13}$$

Thereby, the gyroscope measurements can be utilized as the control vector and the gyroscope measurement noise will a part of the process noise.

[6]The signal from the gyro is relatively accurate, and rapidly changing. Including it in the state vector and subsequently estimating it will generally not result in large improvements in its estimation. However, estimating it results in a larger state vector and subsequently larger system matrices and more computation.

In order to simplify the expressions in equation (3.12), we introduce

$$F_k = \left(I + \frac{\Delta t}{2} \Omega \left(\boldsymbol{u}_k \right) \right), \qquad G_k = \frac{\Delta t}{2} \bar{\Omega} \left(\hat{\boldsymbol{x}}_{k-1|k-1} \right). \tag{3.14}$$

By assuming $\boldsymbol{v}_k \sim \mathcal{N}(0, Q_k)$, the prediction step of the EKF, analogous to equations (2.10)-(2.11), is given as

$$\hat{\boldsymbol{x}}_{k|k-1} = F_k \hat{\boldsymbol{x}}_{k-1|k-1}, \tag{3.15}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + G_k Q_k G_k^T. \tag{3.16}$$

Note that the $\boldsymbol{v}_k$ is given in the body frame and translated to the world frame via G_k. Thus, G_k is also applied to the process covariance Q_k in equation (3.16).

The update step of the EKF is divided into two separate steps which updates the measurement models of the accelerometer and the magnetometer, respectively. This separation is introduced mainly to introduce adaptivity to the algorithm, since the EKF could potentially be run without the magnetometer data. Furthermore, the separation makes it possible to turn off either of the measurement models, which increases the transparency of the verification and debugging process.

The accelerometer measurements $\boldsymbol{y}_{a,k}$ can be modeled as

$$\boldsymbol{y}_{a,k} = \left(R(\boldsymbol{q}_k) \right)^T \left(\boldsymbol{g}_0 + \boldsymbol{f}_{a,k} \right) + \boldsymbol{e}_{a,k}, \tag{3.17}$$

where $R(\boldsymbol{q}_k)$ is the function that transforms the current orientation $\boldsymbol{q}_k$ to its corresponding rotation matrix. Further, $\boldsymbol{g}_0$ is the nominal gravitation vector, and $\boldsymbol{f}_{a,k}$ is the linear acceleration. Both $\boldsymbol{g}_0$ and $\boldsymbol{f}_{a,k}$ are expressed in the world frame and added together, they form the specific force. Thus, the transposed rotation matrix, $R(\boldsymbol{q}_k)^T$, converts the specific force to the body frame. Lastly, $\boldsymbol{e}_{a,k}$ denotes the accelerometer measurement error, also in the body frame.

Equation (3.17) can be approximated using Taylor expansion, and with an assumption of zero linear acceleration, $\boldsymbol{f}_{a,k} = 0$, the accelerometer measurement model can be written as

$$\boldsymbol{y}_{a,k} \approx \left(R\left(\hat{\boldsymbol{x}}_{k|k-1} \right) \right)^T \boldsymbol{g}_0 + \left(\frac{dR}{d\boldsymbol{x}} \left(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_{k|k-1} \right) \right)^T \boldsymbol{g}_0 + \boldsymbol{e}_{a,k}. \tag{3.18}$$

Then let,

$$H_k = \left(\frac{dR}{d\boldsymbol{x}} \left(\hat{\boldsymbol{x}}_{k|k-1} \right) \right)^T \boldsymbol{g}_0, \tag{3.19}$$

and the accelerometer measurement model component of the EKF update, similar

to equations (2.12)-(2.16) in Section 2.3.2, becomes

$$\boldsymbol{v}_k = \boldsymbol{y}_{a,k} - \left(R\left(\hat{\boldsymbol{x}}_{k|k-1}\right)\right)^T \boldsymbol{g}_0 \tag{3.20}$$

$$S_k = H_k P_{k|k-1} H_k^T + R_{a,k} \tag{3.21}$$

$$K_k = P_{k|k-1} H_k S_k^\dagger \tag{3.22}$$

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + K_k \boldsymbol{v}_k \tag{3.23}$$

$$P_{k|k} = P_{k|k-1} - K_k S_k K_k^T \tag{3.24}$$

Similar to how the measurement model of the accelerometer has been derived, the magnetometer part of the update step can be attained. The magnetometer measurement model is developed from

$$\boldsymbol{y}_{m,k} = \left(R\left(\boldsymbol{q}_k\right)\right)^T \left(\boldsymbol{m}_0 + \boldsymbol{f}_{m,k}\right) + \boldsymbol{e}_{m,k}, \tag{3.25}$$

where $\boldsymbol{m}_0$ is the earths magnetic field strength in the world frame, $\boldsymbol{f}_{m,k}$ constitute other magnetic fields present, and $\boldsymbol{e}_{m,k}$ is the measurement error.

The tuning used in this thesis, that was found to give the best performance on the test sequences, was

$$Q_k = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad R_{a,k} = \begin{bmatrix} 150 & 0 & 0 \\ 0 & 150 & 0 \\ 0 & 0 & 150 \end{bmatrix}, \quad R_{m,k} = \begin{bmatrix} 200 & 0 & 0 \\ 0 & 200 & 0 \\ 0 & 0 & 200 \end{bmatrix}. \tag{3.26}$$

As a reminder Q_k is the process noise covariance, $R_{a,k}$ is the accelerometer measurement covariance, and $R_{m,k}$ is the magnetometer measurement covariance. Further, the initial state of the EKF was chosen close to the ground truth for each test sequence. This was done in order to assure that the EKF would converge in all sequences.

3.7 Experiments

A quantitative approach is taken in the regard of training the developed models. Here, different variations of two main types of architectures, the RNN and the RNF from Section 3.4, are trained and evaluated in both 2D and 3D. The approach is quantitative in that several models are trained with the same hyperparameter settings but different, randomly initialized, weights. The performance of these multiple instantiations is then evaluated in terms of best and average performance, i.e. the aRMSE and aRMSE_{20} metrics. Different settings are then compared to each other to indicate which parameter setting works better. This systematic parameter testing is commonly referred to as *grid search* [63], and the parameters evaluated in the experiments are the hidden state size, and the drop-out rate. Other parameters like the batch size, early stopping, learning rate, and the skip-rate are varied and chosen by engineering judgement.

Other relevant information for the experiments are i) that the Adam optimizer is chosen, ii) gradient norm scaling is applied with the maximum L2 norm set to 0.5, iii) the training data is shuffled after each epoch to ensure unbiased subsequent updates, and iv) the training validation split is resampled for each trained model.

3.7.1 The 2D Case Experiments

In the 2D case three different RNN types, with different recurrent state sizes, are trained with three different drop-out rates, resulting in nine different version of RNNs. For each RNN version, 20 instances are trained with randomly initialized weights, to verify consistent results. Similarly, three different RNF types are trained, also with varying recurrent state size, but not with drop-out due to problems with divergence. These models (the RNNs and the RNFs) are all trained with a learning rate of 10^{-3}. Training is done for 60 epochs, or until the loss function does not improve for three consecutive epochs (early stopping). The synthetic gyroscope and accelerometer measurements are all standardized using mean and standard deviation computed from the entire training set.

3.7.2 The 3D Case Experiments

The models designed to estimate orientation in 3D are trained to predict the canonized quaternion. For reasons explained in Section 2.4.5, we know that feature scaling can be beneficial for the convergence rate of the developed RNN and RNF. Therefore, in this thesis the accelerometer, magnetometer and gyro data are feature scaled using standardization. The ground truth labels, however, are already scaled due to the constraint of the quaternions being unit length.

The 3D models are trained on sequences 50 time steps long, corresponding to half a second worth of sensor readings. In total the models are trained on 51 784 sequences, whereof 5% are used for validation. The test set consists of 342 sequences with a length of 1 000 time steps. Further, models are trained with a learning rate of 10^{-2} for the first five epochs followed by 10^{-5} during the rest of training. Training is done for 60 epochs, or until the loss function does not improve for three consecutive epochs (early stopping).

4
Results

In this chapter the results of the performed experiments are presented. Firstly, the results of the proposed 2D RNN and RNF models are displayed and compared. Secondly, the results of the 3D models are presented and compared to the benchmark EKF. Finally, results from an ablation study, and a missing data test with the 3D RNF is reported.

4.1 Results of the 2D Models

Table 4.1 provides a comparison of the performance between the different RNN versions, trained for the 2D problem, in terms of the aRMSE and the aRMSE_{20}. The errors reported from the 2D models are in degrees and twenty instances of each version was trained. Additionally, the best result achieved for each version is also reported on in Table 4.1 in the columns min(aRMSE) and min(aRMSE_{20}).

The results in Table 4.1 show that two parameter settings, version 1 and 8, are delivering the best performance. This result is somewhat counter-intuitive since the best parameter settings are farthest apart in the table.

Table 4.1: Recorded test results from the trained 2D RNN model versions.

RNN Version	Hidden State Size	Drop Out	max aRMSE	aRMSE	min aRMSE	max aRMSE_{20}	aRMSE_{20}	min aRMSE_{20}
1.	8	0.0	3.16	0.5613	0.2297	1.573	**0.2803**	**0.1226**
2.	8	0.2	2.5	0.5464	0.1711	1.591	0.3108	0.134
3.	8	0.4	6.326	0.6758	0.1946	6.381	0.5176	0.1153
4.	16	0.0	2.538	0.4743	0.2564	2.52	0.3695	0.1742
5.	16	0.2	0.6635	0.3889	0.1982	0.5844	0.3274	0.1584
6.	16	0.4	1.692	0.459	0.1992	1.69	0.3841	0.1674
7.	32	0.0	1.663	0.4024	0.1909	1.628	0.3697	0.1832
8.	32	0.2	1.317	**0.358**	**0.1655**	1.307	0.3396	0.1653
9.	32	0.4	1.341	0.4319	0.212	1.326	0.3825	0.1871

Figure 4.1 provides a test sequence of the generated 2D data, estimation from a trained RNN model, and estimates from the benchmark EKF. Notably, the proposed RNN manages fairly well to estimate the ground truth of the test sequence.

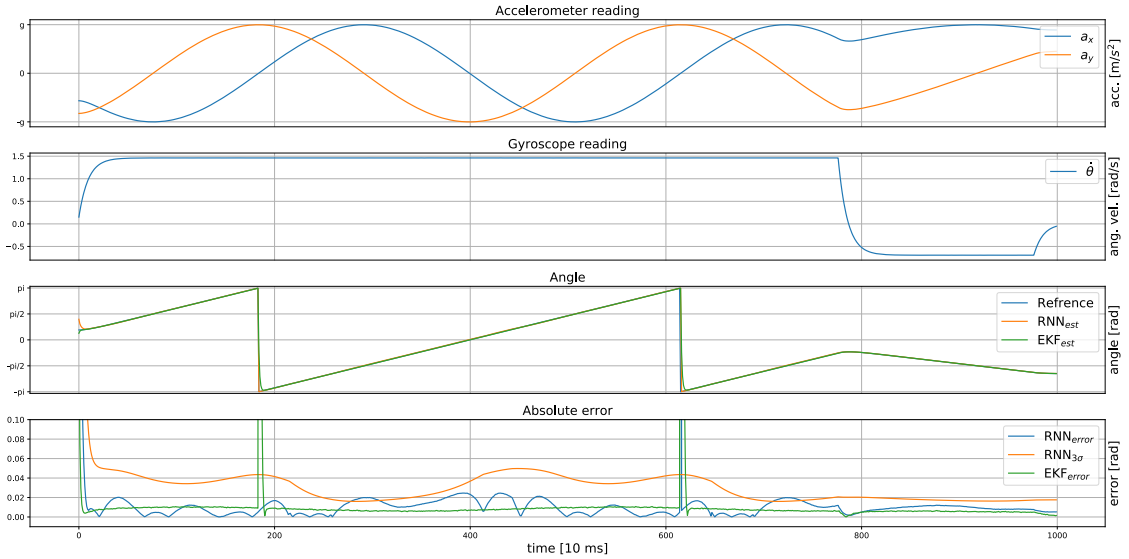

Figure 4.1: Plot of a 2D test sequence with orientation estimates from a trained RNN and the benchmark EKF.

Similarly to Table 4.1, Table 4.2 also shows a comparison between the performance, in terms of aRMSE and aRMSE$_{20}$, but for the proposed 2D RNF models. Also here, 20 instances of each version have been tested, and the best instance of each version is indicated in terms of min(aRMSE) and min(aRMSE$_{20}$). Note that the RNNs where the dropout was different from zero failed to converge for the 2D RNN experiments, and thus, are not reported in the table. Remembering that the motivation behind the 2D tests is to investigate the suitability of applying ML to sensor fusion problems. Finding an indication of which parameter setting works better is thus not crucial in the 2D case.

Table 4.2: Recorded test results from the trained 2D RNF model versions.

RNF Version	Hidden State Size	Drop Out	max aRMSE	aRMSE	min aRMSE	max aRMSE$_{20}$	aRMSE$_{20}$	min aRMSE$_{20}$
1.	8	0.0	2.772	0.6884	0.2575	2.638	0.4963	0.1593
2.	16	0.0	0.5101	0.3052	0.1597	0.4786	**0.2777**	**0.1453**
3.	32	0.0	0.4433	**0.2799**	**0.1499**	0.445	0.2798	0.1504

In Table 4.3 a comparison between the orientation estimation errors generated by the benchmark EKF, the best instances of the proposed 2D RNNs and RNFs can be reviewed.

Table 4.3: Performance comparison of the benchmark EKF and the best instances of the proposed RNN and RNF for the 2D problem.

Model	aRMSE	$aRMSE_{20}$
RNN	0.1655	0.1226
RNF	0.1499	0.1453
EKF	2.187	0.0884

4.2 Results of the 3D Experiments

In Table 4.4 and 4.5 the aRMSE and $aRMSE_{20}$ for the trained instances of the proposed RNN and RNF models are presented. For the RNN architecture the best parameter setting was found for version 9, and the RNF architecture version 6 indicates best results. Note that, for the RNF with smaller hidden state sizes, 8 and 16, was not trained with drop out since it was concluded that larger hidden state sizes was needed.

Table 4.4: Recorded test results from the trained 3D RNN model versions.

RNN Version	State Size	Drop Out	max aRMSE	aRMSE	min aRMSE	max $aRMSE_{20}$	$aRMSE_{20}$	min $aRMSE_{20}$
1.	8	0.0	0.2708	0.1694	0.1442	0.2706	0.1689	0.1439
2.	8	0.2	0.1796	0.1613	0.1489	0.1795	0.1610	0.1486
3.	8	0.4	0.2108	0.1668	0.1422	0.2108	0.1664	0.1411
4.	16	0.0	0.3139	0.1586	0.1326	0.3139	0.1582	0.1321
5.	16	0.2	0.1695	0.1468	0.1330	0.1694	0.1464	0.1325
6.	16	0.4	0.1669	0.1410	0.1271	0.1667	0.1405	0.1265
7.	32	0.0	0.2129	0.1394	0.1170	0.2121	0.1390	0.1164
8.	32	0.2	0.1619	0.1354	0.1189	0.1618	0.1350	0.1183
9.	32	0.4	0.1613	**0.1328**	**0.1148**	0.1609	**0.1323**	**0.1141**

Table 4.5: Recorded test results from the trained 3D RNF model versions.

RNF Version	State Size	Drop Out	max aRMSE	aRMSE	min aRMSE	max $aRMSE_{20}$	$aRMSE_{20}$	min $aRMSE_{20}$
1.	8	0.0	0.3218	0.1615	0.1304	0.3222	0.1613	0.1302
2.	16	0.0	0.297	0.1305	0.1099	0.2974	0.1304	0.1097
3.	32	0.0	0.119	0.106	0.0949	0.1188	0.1059	0.0947
4.	32	0.1055	0.1152	0.1067	0.096	0.1151	0.1065	0.0959
5.	32	0.2254	0.1332	0.1175	0.1065	0.1331	0.1174	0.1064
6.	64	0.0	0.1056	**0.09569**	**0.082**	0.1055	**0.09553**	**0.0817**
7.	64	0.1055	0.1035	0.09704	0.0871	0.1034	0.09691	0.0869
8.	64	0.2254	0.108	0.09955	0.0921	0.1079	0.09941	0.092

In Figure 4.2 a plot of a ground truth orientation quaternion and estimations of the ground truth can be viewed. Alongside the orientation quaternion, the estimation errors in Euler angels can also be viewed. The estimates are produced by the best performing instance of the RNF model version 6 and the benchmark EKF. Also visible in the plot, is the estimated error covariance of the RNF estimates and the calculated error covariance of the EKF estimates. The test sequence chosen for Figure 4.2 was selected with the intention to reflect the average performance of the best RNF version, and the plot neither shows the best or the worst performance.

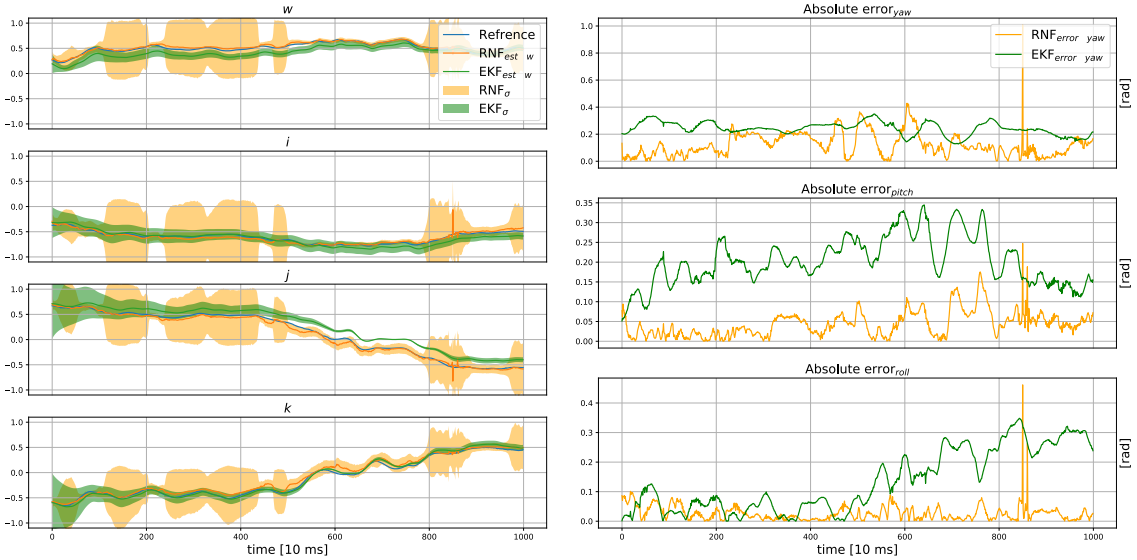

Figure 4.2: The estimated sequence of quaternions (left) and the estimation error in Euler angles (right) from the RNF and the EKF in 3D.

4.2.1 Comparison and Benchmarks

The best results of the 3D experiments are compared with the benchmark EKF in Table 4.6. A closer inspection of the table shows that the proposed RNF performs better than both the EKF and the RNN for the chosen metrics. Also visible in the table is the number of trainable weights for the NN models, and the average one-step computation time for all three models. The average time to compute one step with the 3D EKF is 0.33668 ms on the used hardware[1]. The same figure for the RNN and RNF is 0.07017 ms and 0.1929 ms respectively.

[1]All of the models and filters reported on are run on a Lenovo ThinkPad T490s with a Intel® Core™ i7-8565U CPU @ 1.80GHz × 8. The TensorFlow models are run on the CPU.

Table 4.6: Comparison of best results in 3D along with the models execution time.

Model	RMSE	$RMSE_{20}$	Trainable Weights	Execution Time
RNN	0.1148	0.1141	8 398	0.07017
RNF	0.08200	0.08170	45 282	0.1929
EKF	0.1017	0.1021	-	0.3368

4.2.2 Ablation Study

Removing parts of a network architecture to evaluate the impact it has on performance is referred to as an *ablation study*[2] [64]. The two presented model architecture types are very similar. The proposed RNN architecture is essentially the same as the proposed RNF architecture, trained without separate LSTMs for propagation and prediction. Any differences in performance of the two model types can be attributed to this LSTM layer responsible for propagation. Comparing the performance of 3D RNN version 7 and 3D RNF version 3, two models with no dropout and identical state size, the RNF has a 23% lower aRMSE and $aRMSE_{20}$ with an increase from 8 398 to 13 774 (roughly a 65% increase) trainable weights.

4.2.3 Missing Data

In Section 2.4.2 the RNF model's potential capability to handle missing data points was introduced. Figure 4.3 and Figure 4.4 shows the same sequence, estimated with the same model, as in Figure 4.2. The only difference is that sensor data was only given to the proposed RNF every other time step in Figure 4.3, and no sensor data was given to the RNF in Figure 4.4 after 400 time steps.

[2]The term *ablation study* comes from experimental biology and refers to the removal of brain tissue, with the intent to study the impact its removal has, and infer its function.

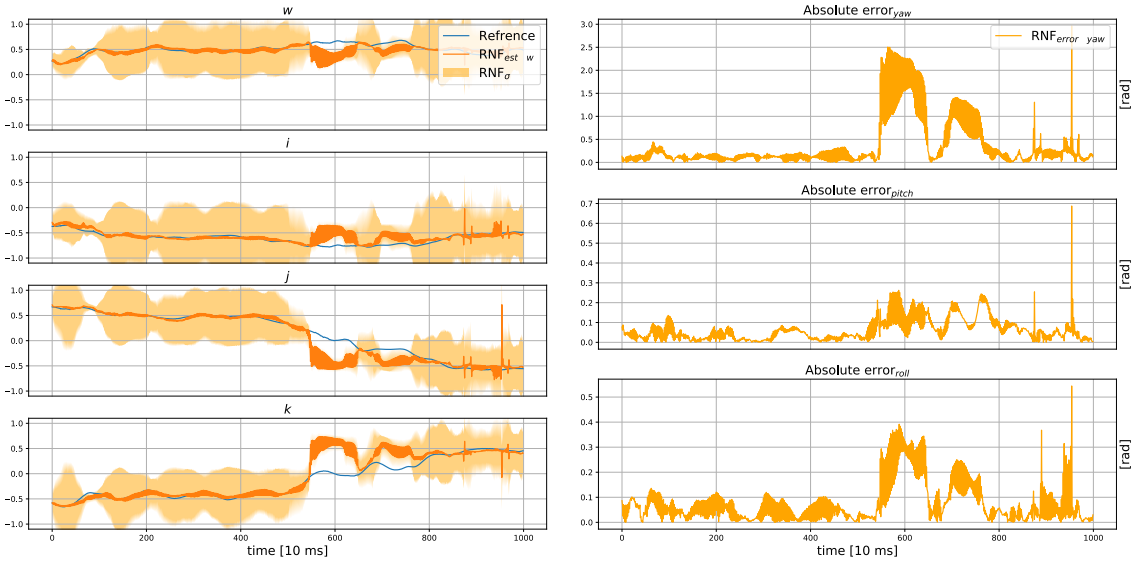

Figure 4.3: Plot showing an orientation sequence and estimation (left), and the estimation error (right) produced by the trained RNF when input data was missing every other time step.

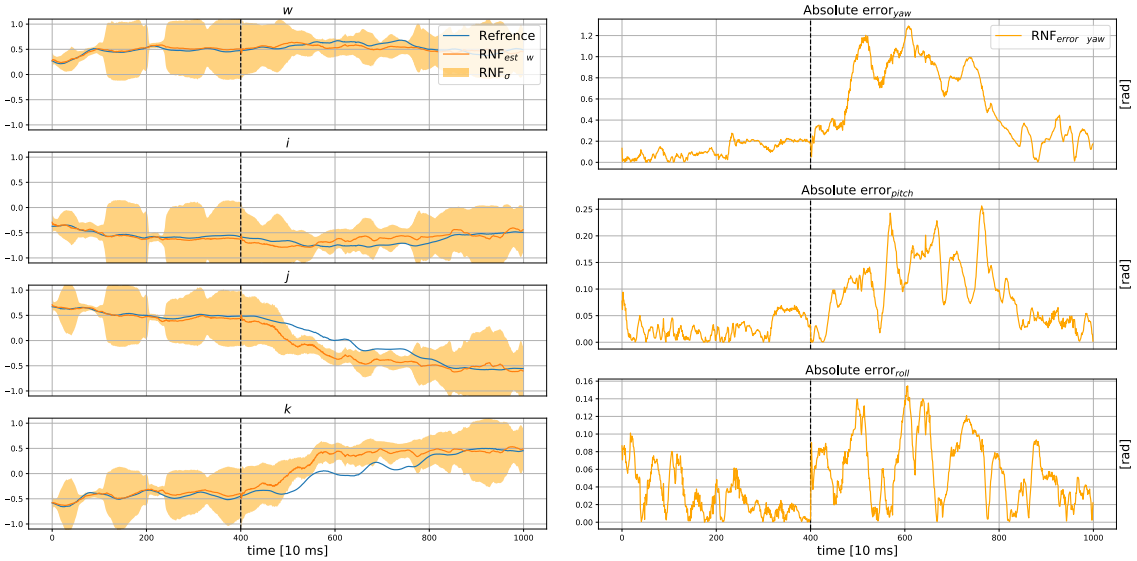

Figure 4.4: Plot showing an orientation sequence and estimation (left), and the estimation error (right) produced by the trained RNF when input data was missing after 400 time steps.

5
Discussion

The aim of this study was to investigate the viability of using ML-based models to estimate absolute orientation given MARG data. Two models with different attributes have been developed, namely an RNN based and an RNF based design. In this chapter, the results reported in chapter 4 will be discussed, followed by some insights into the future potential of similar ML-based orientation estimation approaches.

5.1 Performance and Uncertainty of the Developed ML Models

An initial objective of the project was to verify that the problem is at all assessable by an ML approach. To answer this, simpler experiments in 2D were executed where the current study found that the 2D models were able to out-performed the EKF on synthetic sequences without linear acceleration in the aRMSE metric. However, in the $aRMSE_{20}$ metric, the EKF still performs better than the ML models (see Table 4.3). The shown convergence of the presented models in 2D indicates the feasibility of an ML approach that estimates orientation.

The presented ML models have been shown to perform similarly to the EKF benchmarks with an aRMSE of comparable magnitude. More specifically, for the 3D results, this thesis has reported that the best instance of the RNF model performs better than the EKF in both the aRMSE and the $aRMSE_{20}$ metric (see Tabel 4.6). This result indicates that the proposed RNF model is a better choice of architecture compared to the RNN-base one. Another feature that argues in favor of the RNF is that the RNF has the potential to handle occurrences of missing data-points, see Figures 4.3 and 4.4. In real-time applications, this property may be highly valuable.

Another finding was that the execution time was approximately eight times shorter for the proposed RNF compared to the EKF, and the proposed RNN was even faster. A shorter execution time implies a less computationally expensive filter, which speaks in favor of the ML models. Intuitively, it is easy to believe the NNs to be heavy to run due to the number of weights and non-linearities found in the activation functions. This is not the case here, and a contributing factor is that the proposed networks are not very deep. A note regarding the comparison of the proposed models and the benchmark EKF is that it is highly dependent on the specific EKF implementation used. Still, a reasonable execution time is necessary

to, at all, consider the developed models as feasible.

The third research question regarded the possibility of quantifying the certainty of the measurement. As a solution, the error covariance estimation was included in the loss function presented in Section 2.4.3. The plot in Figure 4.2 shows the result of the implemented covariance estimation. Furthermore, the covariance estimation itself also appears to be of reasonable magnitude in relation to the estimation and its error. At this stage, the correctness of the covariance estimation has not been evaluated enough. Still, the inclusion of error covariance estimation is a highly desirable trait in many applications and should be explored further in future work.

The relatively good aRMSE_{20} of 2D RNN version 1 might be an indication of the small state size forcing the network to learn a much simpler, less rich, representation of the system dynamics.

5.1.1 Research Methodology

A number of factors affects the findings of this study. As mentioned in Section 3.6, the benchmark EKF is given a prior close to ground truth to ensure convergence on all test sequences. The proposed NN models however, are not given any good prior, their hidden state is initialized with all zeros at the start of a sequence. Potentially, the presented ML models can learn to perform well both with accurate priors as well as when the priors are poor. Training the models with short sequences, as described in Section 3.7, has potentially optimized them to converge at a rate that is beneficial given the length of the training sequences. Converging fast will have a greater impact on decreasing the loss on shorter sequences due to the relative contribution of the errors at each time step. If an NN does not have enough parameters to model the relationships that are required to make a good estimate when its prior is poor, as well as when its prior is good, there has to be a trade-off. An illustration of the potential trade-off between training on long versus short sequences can be found in Figure 5.1. Since the models are evaluated on test sequences 20 times longer than the training sequences, if this trade-off is at play, different length of training sequences might alter performance[1]. A way to steer this trade-off in the direction that is desired (fast convergence/accurate estimates over long time) could be weighting the loss. By weighting the loss of the first time steps higher a fast convergence could be encouraged. Alternatively, a lower weight of the loss at the first time steps could encourage more accurate estimates after convergence. Nevertheless, it can not be excluded that that the performance of the developed model could improve with longer training sequences.

[1]If minimizing the metric on the test data is the main concern, choosing training sequences of the same length as the test sequences will most likely yield the best result. However, doing so is both computationally expensive and the resulting model might still not perform better in real time applications with near infinite length sequences.

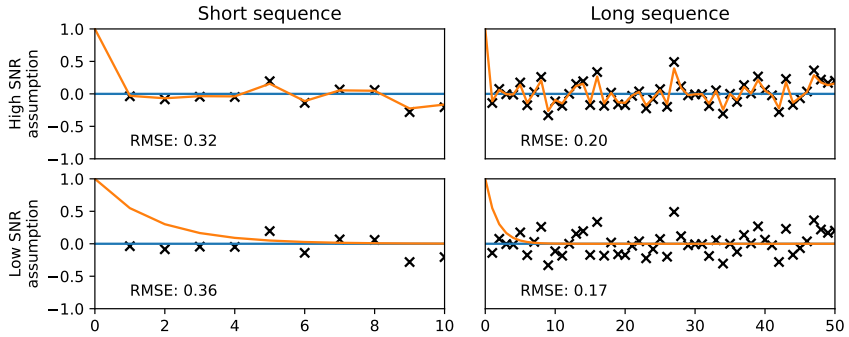

Figure 5.1: An illustrative example of two univariate state estimators, with different SNR assumptions applied on two sequences of different length. Ground truth is represented by the blue line, the estimate by the orange, and the measurements by crosses.

Another aspect that may influence the way the results are perceived is the benchmark EKFs. It can be noted that the EKFs were not developed with the intention to make them "perfect", but rather to achieve satisfactory orientation estimates. Further, the EKFs are given a quite unfair advantage in their priors, as previously mentioned. As a solution the aRMSE_{20} is used since it gives an indication of performance post convergence. The resulting implementation is deemed sufficient as a benchmark. Even so, the benchmark does not give the best possible performance, and this needs to be considered when interpreting the results.

Also important is the data used when training the models, and some aspects regarding the dataset has to be addressed. First, the OxIOD dataset accuracy is reported in mm, which is the translational uncertainty. No accuracy for the orientation ground truth is provided, which is the relevant measure in this application. Having a good ground truth is important, and may set an upper bound on performance for ML based estimation techniques. Second, the OxIOD datset is likely the best dataset with both IMU data and orientation ground truth, publicly available, to this date and to our knowledge. However, without knowing the accuracy of the ground truth it is hard to access the performance of the presented models in absolute terms. The models have been conditioned to include any potential bias occurring in the ground truth in their estimates.

The reference frame of the ground truth orientation in OxIOD is constant, with a vertical z-axis, but the orientation of the other axes in relation to the magnetic/-geographic north is not known. The unknown rotation offset around the z-axis is not considered problematic when verifying the viability of the model architectures. In a hypothetical dataset, more adapted for absolute orientation estimation, the reference frame is known. However, given the current state of the models, the offset does make it challenging to employ the model in real world applications.

Figure 5.2: The dynamics covered in the source domain (OxIOD) might not cover the target domain sufficiently.

5.2 Relation and Contribution to the Field

To our knowledge, no machine learning model for absolute orientation estimation using MARG data with an inherent capability to deal with missing data has been reported on in literature. Further, no end-to-end solution has provided error covariance estimation of the orientation estimates. We argue that these two attributes adds a significant contribution to the field.

Further, both the 2D and 3D results of this thesis imply that the ML approach is an appropriate alternative for data fusion problems. This confirms results of related references [11] and [19].

5.3 Outlook

The RNF model provided a promising results with the given prerequisites. A study of which parameter settings that works was performed, but it was in no sense exhaustive. As a result the final RNF model reached in this study is most likely not perfected to its full potential. In addition, other model architectures may have equivalent or better estimation abilities. Further research could usefully explore both other possible architectures, and a further development of the RNF.

One of the motivations behind the utilization of ML, for data fusion purposes, is that the generated model would be more general. That is, more general in the sense of resistance towards alternations in the process dynamics and noise distributions. This has not been explored enough in this thesis, and is in itself something we

recommend investigating. Moreover, it is possible to question whether an approach, like this one, is at all general. To clarify, the composition of the training data is what decides which scenarios that are reasonable to expect the ML model to handle. The OxIOD dataset covers different motions of a phone which constitute a specific domain. At this stage, it can not be definitely concluded what this domain covers, but certainly a model trained on this dataset cannot be expected to estimate orientations generated by motions too far outside this said domain. That said, the question whether the ML model can truly be viewed as a more general alternative is doubtful as long as the data is a limitation. If not, the alternative would be to move over towards the field of unsupervised leaning where completely different challenges awaits.

Exploiting high order system dynamics is, as discussed Section 1.1, generally something that in conventional filtering requires complex hand crafted filters. Extending the suggested models to explicitly identify modes that are characterised by different dynamics could potentially further improve the estimates. For example, in the OxIOD dataset there are different types of motions: running, walking et.c. Training the model to not only estimate orientation, but also to classify the type of motion might yield improved results.

6
Conclusion

This thesis has explored the possible benefits of utilizing ML approaches for sensor fusion purposes. To enable this, two ML based filters, the RNN and the RNF, were developed. Out of the two, the RNF has proven to have a better estimation capability than both the RNN and, more importantly, the benchmark EKF. In conclusion, the RNF is the NN based filter suggested in this report.

Further, two beneficial traits of the suggested RNF has been accounted for. Firstly, a possibility to maneuver missing sensor data has been indicated. Secondly, a method that enables the estimation of the error covariance of the output of the RNF. Together, the two functionalities provides a way to include traditional traits of the classical Bayesian estimation techniques to ML based estimators in the field. Further, we conclude that named functionalities will be of great importance when deploying ML based state estimators in real world applications.

The estimation capabilities of the NN based filters accounted for in this research confirms, what previous research has indicated, that the application of ML to sensor fusion problems is indeed possible. Nevertheless, this study has identified the risk that any NN based filter, trained with supervised learning, will be limited by the accuracy and scope of the domain of the training data. Without sufficient data NN based filters will not be useful in any greater extent.

Bibliography

[1] Bahador Khaleghi et al. "Multisensor Data Fusion: A Review of the State-of-the-art". In: *Information Fusion - INFFUS* 14 (2013). DOI: 10.1016/j.inffus.2011.08.001.

[2] Shiuh Ku Weng, Chung Ming Kuo, and Shu Kang Tu. "Video object tracking using adaptive Kalman filter". In: *Journal of Visual Communication and Image Representation* 17.6 (Dec. 2006), pp. 1190–1208. ISSN: 10473203. DOI: 10.1016/j.jvcir.2006.03.004.

[3] I. P REFACE. "Sensor data fusion : state of the art survey". In: 2010.

[4] E. A. Wan and R. Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. 2000, pp. 153–158.

[5] Satish Rao and David Tse. *CS 70 Discrete Mathematics and Probability Theory Lecture 19 Inference Example 3: The Kalman Filter*. Tech. rep. 2009, p. 1.

[6] William S Levine. *The Control Handbook (three volume set)*. CRC press, 2018.

[7] P. Matisko and V. Havlena. "Optimality tests and adaptive Kalman filter". In: *IFAC Proceedings Volumes (IFAC-PapersOnline)*. Vol. 16. PART 1. 2012, pp. 1523–1528. ISBN: 9783902823069. DOI: 10.3182/20120711-3-BE-2027.00011.

[8] Z. Xue and H. Schwartz. "A comparison of several nonlinear filters for mobile robot pose estimation". In: *2013 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2013*. 2013, pp. 1087–1094. ISBN: 9781467355582. DOI: 10.1109/ICMA.2013.6618066.

[9] S. D. Brown and S. C. Rutan. "Adaptive Kalman Filtering". In: *Journal of Research of the National Bureau of Standards* 90.6 (Nov. 1985), p. 403. DOI: 10.6028/jres.090.032.

[10] T. Meng et al. "A survey on machine learning for data fusion". In: *Information Fusion* 57 (May 2020), pp. 115–129. ISSN: 15662535. DOI: 10.1016/j.inffus.2019.12.001.

[11] C. Chen et al. "IONet: Learning to Cure the Curse of Drift in Inertial Odometry". In: *arXiv e-prints*, arXiv:1802.02209 (Jan. 2018), arXiv:1802.02209. arXiv: 1802.02209 [cs.RO].

[12] I. Ullah, M. Fayaz, and D. Kim. "Improving Accuracy of the Kalman Filter Algorithm in Dynamic Conditions Using ANN-Based Learning Module". In: *Symmetry* 11.1 (Jan. 2019), p. 94. ISSN: 2073-8994. DOI: 10.3390/sym11010094. URL: http://www.mdpi.com/2073-8994/11/1/94.

[13] A. M. Sabatini. "Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation". In: *Sensors* 11.10 (Oct. 2011), pp. 9182–9206. ISSN: 14248220. DOI: 10.3390/s111009182.

[14] Z. Z. He F. Alam and H. J. Jia. "A Comparative Analysis of Orientation Estimation Filters using MEMS based IMU." In: *Proceedings of the International Conference on Research in Science, Engineering and Technology*. Mar. 2014. DOI: 10.15242/IIE.E0314552.

[15] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. "Estimation of IMU and MARG orientation using a gradient descent algorithm". In: *2011 IEEE International Conference on Rehabilitation Robotics*. 2011, pp. 1–7.

[16] M. Kok, J. D. Hol, and T. B. Schön. "Using Inertial Sensors for Position and Orientation Estimation". In: *Foundations and Trends in Signal Processing* 11.2 (2017), pp. 1–153. DOI: 10.1561/2000000094. arXiv: 1704.06053v2. URL: http://dx.doi.org/10.1561/2000000094.

[17] Z. Aydogmus and O. Aydogmus. "A comparison of artificial neural network and extended Kalman filter based sensorless speed estimation". In: *Measurement: Journal of the International Measurement Confederation* 63 (2015), pp. 152–158. ISSN: 02632241. DOI: 10.1016/j.measurement.2014.12.010.

[18] S. Hosseinyalamdary. "Deep Kalman Filter: Simultaneous Multi-Sensor Integration and Modelling; A GNSS/IMU Case Study". In: *Sensors* 18.5 (Apr. 2018), p. 1316. ISSN: 1424-8220. DOI: 10.3390/s18051316. URL: http://www.mdpi.com/1424-8220/18/5/1316.

[19] M. A. Esfahani et al. "OriNet: Robust 3-D Orientation Estimation With a Single Particular IMU". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 399–406.

[20] R. Valenti, I. Dryanovski, and J. Xiao. "Keeping a Good Attitude: A Quaternion-Based Orientation Filter for IMUs and MARGs". In: *Sensors* 15.8 (Aug. 2015), pp. 19302–19330. ISSN: 1424-8220. DOI: 10.3390/s150819302. URL: http://www.mdpi.com/1424-8220/15/8/19302.

[21] S. O. H. Madgwick. "An efficient orientation filter for inertial and inertial / magnetic sensor arrays". In: 2010.

[22] M. Brossard, S. Bonnabel, and A. Barrau. *Denoising IMU Gyroscopes with Deep Learning for Open-Loop Attitude Estimation*. 2020. arXiv: 2002.10718 [cs.RO].

[23] C. Chen et al. "OxIOD: The Dataset for Deep Inertial Odometry". In: *arXiv e-prints*, arXiv:1809.07491 (Sept. 2018), arXiv:1809.07491. arXiv: 1809.07491 [cs.RO].

[24] B. Lim, S. Zohren, and S. Roberts. *Recurrent Neural Filters: Learning Independent Bayesian Filtering Steps for Time Series Prediction*. 2019. arXiv: 1901.08096 [stat.ML].

[25] Xianghui Yuan, Feng Lian, and Chongzhao Han. "Models and Algorithms for Tracking Target with Coordinated Turn Motion". In: (2014). DOI: 10.1155/2014/649276. URL: http://dx.doi.org/10.1155/2014/649276.

[26] M Sanjeev Arulampalam et al. *A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking*. Tech. rep. 2. 2002.

[27] Christopher M Bishop. *Mixture Density Networks*. Tech. rep. 1994. URL: http://www.ncrg.aston.ac.uk/.

[28] J. Diebel. "Representing attitude: Euler angles, unit quaternions, and rotation vectors". In: *Matrix* 58.15-16 (2006), pp. 1–35.

[29] *File:Angle axis vector.svg - Wikipedia*. Wikipedia, Oct. 2015. URL: https://en.wikipedia.org/wiki/File:Angle_axis_vector.svg.

[30] Du Huynh. "Metrics for 3D Rotations: Comparison and Analysis". In: *Journal of Mathematical Imaging and Vision* 35 (2009), pp. 155–164. DOI: 10.1007/s10851-009-0161-2.

[31] Ahmad Bani Younes and Daniele Mortari. "Derivation of All Attitude Error Governing Equations for Attitude Filtering and Control". In: *Sensors* 19.21 (Oct. 2019), p. 4682. ISSN: 1424-8220. DOI: 10.3390/s19214682. URL: https://www.mdpi.com/1424-8220/19/21/4682.

[32] "Quaternion Algebra". In: *Geometric Algebra for Computer Graphics*. London: Springer London, 2008, pp. 39–48. ISBN: 978-1-84628-997-2. DOI: 10.1007/978-1-84628-997-2_5. URL: https://doi.org/10.1007/978-1-84628-997-2%7B%5C_%7D5.

[33] N. Ahmad et al. "Reviews on Various Inertial Measurement Unit (IMU) Sensor Applications". In: (). DOI: 10.12720/ijsps.1.2.256-262.

[34] L. Nomdedeu et al. "Sensing capabilities for mobile robotics". In: *Using Robots in Hazardous Environments*. Elsevier, 2011, pp. 125–146. DOI: 10.1533/9780857090201.2.125.

[35] P. Figueiredo e Silva et al. "Challenges and Solutions in Received Signal Strength-Based Seamless Positioning". In: *Geographical and Fingerprinting Data to Create Systems for Indoor Positioning and Indoor/Outdoor Navigation*. Elsevier, 2019, pp. 249–285. DOI: 10.1016/b978-0-12-813189-3.00013-7.

[36] *IMU Errors and Their Effects*. Calgary, Canada: NovAtel, 2014. URL: https://www.novatel.com/assets/Documents/Bulletins/APN064.pdf (visited on 04/24/2020).

[37] P. Gui, L. Tang, and S. Mukhopadhyay. "MEMS based IMU for tilting measurement: Comparison of complementary and kalman filter based data fusion". In: *2015 IEEE 10th Conference on Industrial Electronics and Applications (ICIEA)*. June 2015, pp. 2004–2009. DOI: 10.1109/ICIEA.2015.7334442.

[38] Bashar Alsadik. "Chapter 10 - Kalman Filter". In: *Adjustment Models in 3D Geomatics and Computational Geophysics*. Ed. by Bashar Alsadik. Vol. 4. Computational Geophysics. Elsevier, 2019, pp. 299–326. ISBN: 978-0-12-817588-0. DOI: https://doi.org/10.1016/B978-0-12-817588-0.00010-6. URL: http://www.sciencedirect.com/science/article/pii/B9780128175880000106.

[39] J. L. Marins et al. "An extended Kalman filter for quaternion-based orientation estimation using MARG sensors". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No.01CH37180)*. Vol. 4. 2001, 2003–2011 vol.4.

[40] Aston Zhang et al. *Dive into Deep Learning*. https://d2l.ai. 2020.

[41] Diederik P Kingma and Jimmy Lei Ba. *ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION*. Tech. rep. arXiv: 1412.6980v9.

[42] K.-L. Du and M. N. S. Swamy. *Neural Networks and Statistical Learning*. Springer London, 2019. DOI: 10.1007/978-1-4471-7452-3.

[43] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "On the difficulty of training recurrent neural networks". In: *International conference on machine learning*. 2013, pp. 1310–1318.

[44] Alex Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Vol. 385. 2012. DOI: 10.1007/978-3-642-24797-2.

[45] S. Hochreiter and J. Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[46] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to forget: Continual prediction with LSTM". In: (1999).

[47] R. L. Russell and C. Reale. "Multivariate Uncertainty in Deep Learning". In: (Oct. 2019). arXiv: 1910.14215. URL: http://arxiv.org/abs/1910.14215.

[48] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. "On early stopping in gradient descent learning". In: *Constr. Approx* (2007), pp. 289–315.

[49] Mu Li et al. "Efficient Mini-batch Training for Stochastic Optimization". In: (2014). DOI: 10.1145/2623330.2623612. URL: http://dx.doi.org/10.1145/2623330.2623612.

[50] Anders Krogh · and John A Hertz. *A Simple Weight Decay Can Improve Generalization*. Tech. rep.

[51] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167. URL: http://arxiv.org/abs/1502.03167.

[52] Nitish Srivastava et al. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Tech. rep. 2014, pp. 1929–1958.

[53] Warren S. Sarle. "Ill-Conditioning in Neural Networks". In: (Sept. 1999). URL: ftp://ftp.sas.com/pub/neural/illcond/illcond.html.

[54] S Gopal, Krishna Patro, and Kishore Kumar Sahu. *Normalization: A Preprocessing Stage*. Tech. rep. URL: www.kiplinger.com,.

[55] J. Tobin et al. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World". In: *arXiv e-prints*, arXiv:1703.06907 (Mar. 2017), arXiv:1703.06907. arXiv: 1703.06907 [cs.RO].

[56] M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[57] J. Moolayil. "An Introduction to Deep Learning and Keras". In: *Learn Keras for Deep Neural Networks*. Berkeley, CA: Apress, 2019, pp. 1–16. ISBN: 978-1-4842-4240-7. DOI: 10.1007/978-1-4842-4240-7_1. URL: https://doi.org/10.1007/978-1-4842-4240-7%7B%5C_%7D1.

[58] Aceinna. *gnss-ins-sim*. https://github.com/Aceinna/gnss-ins-sim, *commit:* c3a42d8247b5196e2eda1f6380b204c76. Mar. 2020.

[59] Analog Devices. *Precision MEMS IMU Module*. Tech. rep. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/adis16465.pdf.

[60] Dong Xiaoguang. *gnss-ins-sim-doc*. https://github.com/dxg-aceinna/gnss-ins-sim-doc, *commit:* 5cb2d6059dbdb1a12871d4c3adf8511e2e0ceece. Mar. 2020.

[61] D. Schubert et al. "The TUM VI Benchmark for Evaluating Visual-Inertial Odometry". In: *International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018. URL: https://arxiv.org/abs/1804.06120.

[62] Michael Burri et al. "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* (2016). DOI: 10.1177/0278364915620033. eprint: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.full.pdf+html. URL: http://ijr.sagepub.com/content/early/2016/01/21/0278364915620033.abstract.

[63] Petro Liashchynskyi and Pavlo Liashchynskyi. *Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS*. Tech. rep. arXiv: 1912.06059v1.

[64] Richard Meyes et al. *Ablation Studies in Artificial Neural Networks*. Tech. rep. arXiv: 1901.08644v2. URL: https://github.com/RichardMeyes/AblationStudies.