# Towards orientation estimation using a loosely attached IMU with a recurrent neural network

S.J. van den Heuvel

Master of Science Thesis

Delft University of Technology

Delft Center for Systems and Control

# Towards orientation estimation using a loosely attached IMU with a recurrent neural network

Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

S.J. van den Heuvel

May 18, 2021

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology

# Abstract

Inertial measurement units (IMUs) are getting more and more incorporated into our lives due to their improving accuracy, lower cost and smaller sizes. Applications for inertial-based orientation estimation can already be found in the field of computer vision, aerospace engineering, robotics, navigation, biomechanics, health monitoring and sports. In order to enable non-intrusive, long-term tracking, this study addresses the problem of orientation estimation using a loosely attached IMU. To this end, a hybrid model is proposed combining classical orientation methods with machine learning (ML) techniques. Previous works have focused only on parts of this task, which is why to date, there are little to no solutions to this problem. In this study, a model is designed containing three steps that are shown to be beneficial for reducing the estimation error. With the first step, the model becomes robust to a misaligned mounting of the loosely attached IMU and it allows to identify which part of the error can be contributed to the average misalignment. For the second step, two algorithms are presented which make the model invariant to the absolute heading direction of the recorded data. In the third step, the data is prepared such that the size of the search space is reduced by a factor 2. Therefore, less data is required for learning the problem or, with an equal amount of data, a bigger part of the process can be discovered. Besides these steps, it is shown that the use of physics-based knowledge in the form of orientation estimates, as opposed to raw inertial data as input features for the ML method, is beneficial.

# Table of Contents

# List of Figures

# List of Tables

# Preface

To finish off my Systems and Control Master's degree program, I wanted to gain some more hands on experience in the field of ML. Soon, it became clear that little research was conducted in combining the quickly advancing field of ML with traditional ways of orientation estimation to overcome motion artifacts. My main research goal was born and I worked on the subject for about a year. In retrospect, this study has made my view on ML methods more informed. I am also surprised how big the field of orientation estimation actually is and what is possible with inertial-based orientation estimation. To give an example of my unawareness at the start of the project, I had never heard of the term unit quaternions. I really liked the fact that real sensors were involved in this study, it makes the work more practical and reveals issues that would occur in practice. In the end, this study has brought me several lessons about how I respond to setbacks and breakthroughs. Also, I am happy to have taken on this challenge. I have learned a lot about both the field of orientation estimation and the field of recurrent neural networks (RNNs), for which I am thankful.

I would like to thank my supervisor Dr. Manon Kok whose field of expertise is sensor fusion (SF) applied to orientation estimation. I must bring up the fact that she always has been positive and had bright comments on my work. It sometimes even meant that I could regain my motivation again which is crucial in my opinion. I could not have done this study in the same amount of time if it were not for her.

I also would like to thank my fellow students, Frida, Evan, Thomas, Bart, Daan, Henri, Janneke, Marijn, Niels and Jesse, with whom I met once every two weeks. It was nice to get to know you and hear about your research. Thank you for the curious and critical questions and views about my research. These helped me as well.

I am grateful for Cora, Robbert-Jan, Wiljo and Coen-Jan who have been proofreading my work and provided me with lots of feedback.

I want to thank my supporting parents and girlfriend. Thank you for believing in me. This pandemic period has been a tough time to graduate in and I am grateful that I could escape from the daily routine of studying when I was with you.

I want to thank my roommates for the occasional discussions we have had about the study and also the distraction from it during the evenings.

Delft, University of Technology S.J. van den Heuvel
May 18, 2021

# Glossary

## List of Acronyms

| | |
|---|---|
| **IMU** | inertial measurement unit |
| **ML** | machine learning |
| **RNN** | recurrent neural network |
| **SF** | sensor fusion |
| **EKF** | extended Kalman filter |
| **GRU** | gated recurrent unit |
| **LSTM** | long short-term memory |
| **FC** | fully connected |
| **3D** | three-dimensional |
| **MSE** | mean squared error |
| **RMSE** | root mean squared error |
| **MEMS** | microelectromechanical system |
| **RLG** | ring laser gyro |
| **FOG** | fiber optic gyro |
| **CPU** | central processing unit |
| **GPU** | graphics processing unit |

## List of Symbols

| | |
|---|---|
| $\alpha$ | Angle of rotation |
| $\beta$ | Dip angle |
| $\delta$ | Euclidean distance |
| $\delta$ | Sensor bias |
| $\omega$ | Angular velocity |
| $\phi$ | Roll angle |
| $\psi$ | Yaw angle |
| $\sigma$ | Sigmoid function |
| $\theta$ | Pitch angle |
| $E$ | The earth frame |
| $e$ | Noise |
| $f_s$ | Sampling frequency |
| $g$ | Gravitational acceleration |
| $I$ | The inertial frame |
| $K$ | Kalman gain |
| $l$ | Sequence length |
| $N$ | The navigation frame |
| $n$ | Extra absolute heading directions |
| $q$ | Unit quaternion |
| $R$ | Rotation matrix |
| $S$ | The sensor frame |
| $t$ | Notion of sequential data or time dependence |
| $T_s$ | Sampling period |
| $\exp$ | Exponential |
| $\mathrm{m}$ | Magnetic field |
| $\mathrm{N}$ | Data samples |

# Chapter 1

# Introduction

*This chapter starts by touching upon the background related to physics-based orientation estimation. Next, the relevance and motivation for the plan of approach of this study are explained. Then, the main research questions are presented, the contributions delivered by this study and subsequently an overview of the report's content is described.*

## 1-1 Background

With the advent of the digital era, multiple systems have been developed that enable the task of orientation estimation. This task can be an objective in itself, but it is also often used to compute the velocity and the location of an object [4, 5]. Among the many fields of application are the field of computer vision, aerospace engineering, robotics, biomechanics, health monitoring and sports [5, 6, 7, 8, 9].

As illustrated by the many fields of application, orientation estimation is applied for various purposes. In this study, it is assumed that orientation estimation will be applied to human motion. The systems to do orientation estimation for this can be divided into three subdomains: optical, mechanical and inertial, of which currently the optical and inertial systems are dominantly used [9, 10]. Optical systems sometimes use markers on the subject to track its orientation. An optical system often yields very high accuracy, however, it has its disadvantages. The tracking space is limited to what the optical systems can observe. Also, their performance suffers from obstructions and different illuminations [8, 9, 11]. When doing orientation estimation using inertial-based methods, the sensors are attached to the subject. They do not depend on an external setup which allows the subject to move in an infrastructure-free space. Inertial sensors are free from obstructions and different illuminations as they do not use any kind of vision related technique. Still, they have their own challenges as well. Data from an inertial measurement unit (IMU) is corrupted with noise and biases. Because the orientation estimation using IMUs is dependent on the integration of the imperfect angular velocity data, the orientation estimates suffer from drift. To correct for this drift, the most commonly occurring methods include measurements from an accelerometer and a magnetometer. The latter suffers greatly from magnetic disturbances. As this study focuses on orientation estimation of human motion it is convenient that the tracking space is not confined. Therefore, using IMUs is the most practical system to do orientation estimation with for this study.

## 1-2 Motivation

Because of the fact that inertial-based systems suffer from noise and biases, numerous ways of dealing with these deficiencies have already been developed. These methods and knowledge are referred to as 'classical methods' and 'physics-based knowledge' in the rest of the report. In this study, it is investigated which actions can be taken to improve the orientation estimation of an object to which an IMU is loosely attached. The notion of 'loosely' here can be broadly defined as not rigidly fixed or not tightly coupled, like a phone with an IMU in a pocket. The problem with sensors which are not rigidly attached to the object they are supposed to be tracking, is that they introduce so-called motion artifacts [9, 12, 13, 14]. Motion artifacts are estimation errors between the sensor and the object which is being tracked, caused by possibility of the sensor to move to a certain degree with respect to the object. They are present when a sensor is attached to human tissue by a rubber band and the object that should be tracked is the bone. These are called soft-tissue artifacts. Because it is unpractical to obtain ground truth orientation data of a human bone, it has been decided to look into another class of loosely attached connections that cause motion artifacts. This class comprises the errors that are introduced when someone attaches a sensor to garments or connects it via another non-solid material. Improving the performance of orientation estimation using sensors that are attached as such, would allow users to take sensors with them in a more comfortable way than strapping them down using a rubber band. For instance, it allows the use of the IMU in a mobile phone and opens up possibilities to non-intrusive, long-term tracking. The authors of [9] estimate the Euler angles of human joints by applying classical inertial methods to loosely attached IMUs, using a kinematic chain starting from the hip to the extremities of the body. They point out that the errors are the most severe at the upper body limbs, reaching up to a mean error of $\pm 60°$ and standard deviation of $120°$. While this is the case, there are little to no solutions found to this problem yet, motivating the goal of this study. In this study, it is aimed to identify the original error for several experimental setups containing loosely attached IMUs and to design steps that can be used to improve this error. These setups are of growing complexity and as a final experiment, a human wearing a loosely attached IMU is involved.

Over the past several decades, machine learning (ML) techniques have gained more and more attention. Along with the advancements in computational power of hardware and the availability of masses of data, ML techniques have already shown promising results in multiple fields of research [15, 16, 17]. In the field of orientation estimation, several works have demonstrated superior performance of ML techniques compared to classical methods [1, 5, 18, 19]. ML methods exploit big amounts of data, rather than using physics-based knowledge. By doing this, they are capable of finding complex patterns that are latent in the data. To date, most studies applying ML techniques either use them to supplement classical methods [20, 21, 22, 23] or replace classical methods all together [1, 5, 12, 13, 18, 19, 24]. The latter is referred to as end-to-end methods. To give an example, by employing such an end-to-end ML method, a reduction of 69% of the error is reported using multiple loosely attached IMUs on a human body [13]. The authors do, however, point out that their method is highly dependent on whether data was already seen by the network during training or not. In other words, an end-to-end method does not generalize very easily if it is not exposed to sufficient representative data.

The authors of [15] endorse combining ML methods and physics-based methods, not only to improve the performance, but also to make ML methods more transparent or interpretable. In this study, that idea is adopted, as it is believed that using two sources of information can potentially increase the performance compared to using merely one source. These sources are physics-based knowledge represented by classical methods and knowledge that ML methods can deduce from data. Another argument to incorporate classical methods is that, by doing so, one is able to derive more compact representations of the information. Using orientation estimates instead of raw inertial data reduces the size of the input space. This means that less data would be required to obtain equal performance. This belief is supported by [17], where the authors use so-called ‘preintegrated’ input features rather than raw inertial data, and achieve better performance. Whilst several works focus on the problem of orientation estimation using ML techniques, the problem of reducing errors due to motion artifacts for a single loosely attached IMU remains largely unaddressed. Besides this, the possibility of using both sources of information only adds up to the reasons why it is worth exploring this area.

Finally, this study focusses on the orientation estimation using only one IMU. This is a big advantage because it is possible to extend the problem to situations where multiple IMUs are used [14]. This approach is in contrast to the approaches of [12, 13, 23, 25, 26], which all employ models that are already dependent on multiple IMUs. Naturally, using multiple IMUs can potentially yield an accumulative performance increase as pointed out by [13]. In addition to this, this approach allows users to use only one wearable device containing an IMU.

## 1-3 Research goals

The main goal of this study can be subdivided into three research goals that are central to this study. First, an algorithm to determine the baseline performance is needed in order to be able to evaluate any improvement on the performance. This way, it is possible to determine the original error between input and target data before applying any designed steps. Second, possible ways to improve this original error are explored. For this, it is assumed that adding physics-based knowledge to ML methods is beneficial for the orientation estimation using a loosely attached IMU. Lastly, a way of verifing this assumption is being sought. The research goals are as follows:

1. How can the original error between input and target data best be determined?
2. What steps can be taken to improve the orientation estimation using a loosely attached IMU by adding physics-based knowledge to ML methods?
3. Is this way of using physics-based knowledge actually beneficial for ML techniques in the context of this problem?

## 1-4 Problem decomposition and contributions

Before going into more details on the contributions, a global overview of the problem is discussed. All research conducted for this problem contributes to one main goal, illustrated by the title of this work: *Towards orientation estimation using a loosely attached IMU with a recurrent neural network*. For this, the difference between the orientation of the loosely attached IMU and the target orientation should be reduced. Therefore, possible sources of error are thought of beforehand. This subdivision and identification of the error is something that was not encountered in existing literature and is unique to this study.

One of the sources of error can be a differently oriented attachment of the input sensor. This error source is called the ‘average misalignment’ and has to do with the average relative orientation between the input and target orientation. Another source of error is the mutual movement between the input sensor and the object it is supposed to track. This is introduced by loosely attaching the input sensor. This source is referred to as the ‘process’. It is expected that a part of this process can be learned and another part that is too complex, unobservable or just undiscovered and thus will not be learned. A third source of error is introduced by the possibility that the subject can make identical movements in different directions resulting in different data. Therefore, it could be that a certain movement is represented in the training data, but that it is recorded facing a different direction. This movement then, is not recognized and orientation estimates differ greatly from the target orientation as the network was not trained for this data. This error source will be referred to as the ‘heading dependency’. Because the process between two sensors facing a different heading direction can not be estimated without addressing the heading dependency, the process and heading dependency are considered a joint part of the error. The sources of error are compactly visualized in Figure 1-1 where randomly distributed contributions to the error are used as an example.

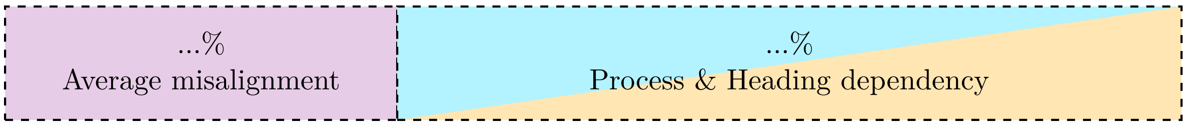


**Figure 1-1:** A visualization of a possible decomposition of the error in the data.

In order to lower the error between input and target orientation, in this study, three steps have been designed to preprocess the physics-based input features. By removing a misalignment between input IMU and object, the first step makes sure that this misalignment can not affect the baseline performance. This answers to the first research goal. In addition to that, this step ensures that the network is robust to such misalignments, which contributes to the second research goal. With the second step, the heading dependency is removed from the data. This also contributes to the second research goal. For this, two different algorithms are designed that fulfill this task. The third step, by which an attempt is made to reduce the search space of the input, also answers to the second research goal. There are a lot of movements a loosely attached IMU can make in many different situations. Since a trained network is only able to perform well if the training data represents the test data, it is advantageous if the process can be described by the least amount of data possible. This step processes the data in order to reduce the amount needed to achieve a certain degree of performance. This is done by removing ambiguity and discontinuities from the data. Besides these steps, this work also presents a short evaluation of the belief that using both physics-based methods and ML techniques is beneficial.

## 1-5 Organization and content

In this section an outline of the content of the report is given.

Chapter 2 gives a broad summary about orientation estimation using classical methods. The notion of coordinate frames is discussed, as well as different parameterizations of orientation. As a requirement and support for the use of the extended Kalman filter (EKF), the different type of sensors that are used are addressed together with their measurement models. Next, the dynamical model and fusion algorithm, the EKF, are presented. After this summary, the counterpart of this study is shortly addressed, namely, the recurrent neural network (RNN). Inspired by previous research, details on long short-term memory (LSTM) cells are explained.

Chapter 3 is focused on providing general information on the topic of using RNNs for orientation estimation and on problems that can occur with it. It is an informative chapter, which is used mainly to motivate the third preprocessing step.

Chapter 4 discusses the ideas to answer the research questions. It starts by discussing the framework of the hybrid model and the details on the EKF and RNN. Then this model is extended with three preprocessing steps. The motivation for these steps is given and the implementation of the preprocessing steps is explained. The first preprocessing step looks into a manner to improve and make the baseline performance more consistent, regardless of the mounting orientation of the input sensor. Also, this step contributes to a network that is robust to a misaligned attachment. For the second preprocessing step, various ways to cope with differently faced directions of recorded data are explored. This resulted in two powerful algorithms that either reduce the required amount of data or can increase the available amount of data artificially. After this, the third preprocessing step is discussed. This step focuses on properties of the orientation parameterization that pose problems for the RNN. It aims to remove ambiguity and discontinuities from the data. The last topic this chapter discusses is a validation of whether the use of physics-based methods is beneficial for the problem.

In Chapter 5, the hardware which is used to train the network, is introduced. To reveal the working and contribution of the steps, five different test setups have been designed. They are described in this chapter and the reasons for their designs are explained. There is one simulated setup and four experimental setups. The latter use real sensor data. After this, the datasets recorded to show the working of the designed preprocessing steps, are discussed.

The contributions are validated by applying them to data from the different setups. The results obtained from this are discussed in Chapter 6. A discussion is added to every result that is presented.

In Chapter 7 a conclusion is drawn based on the results. Added to this is a section that describes subjects that could be improved and areas that were unexplored and left to future work.

Chapter 2

# Related Work

*This chapter starts off with giving an idea of what research has already been conducted on the subject of orientation estimation with ML techniques and physics-based knowledge. Subsequently, the reader is given a more in-depth understanding of the subject of classical orientation estimation. The measurement models, the dynamical model and the fusion algorithm are discussed. After that, more on the RNN part of this study is explained. All information in this chapter is based on existing literature.*

## 2-1 Related work on orientation estimation using IMUs and RNNs

Supported by [15], it was argued in Section 1-2 that the use of physics-based input features would be beneficial for the task of orientation estimation using ML techniques and loosely attached IMUs. To further elaborate on this task, one IMU is loosely attached and another IMU is tightly attached to the object of which the orientation is desired. In order to derive the orientation of this object, called the target orientation, the inertial measurements of the tightly attached IMU are fused. A classical method that uses physics-based knowledge is employed to do this. For the input of the ML method, the inertial measurements of the loosely attached IMU are also fused and orientation estimates are computed. This way, physics-based knowledge is used to obtain the input features. These input orientation estimates are the most closely related to the target orientation estimates and, therefore, adequate physics-based input features. Using orientation estimates as inputs to ML techniques is supported by [17] and incorporates physics-based knowledge into a model. Another study that is known to use orientation estimates as input features is [23]. The authors of this work compare two ML techniques to estimate full-body poses using a reduced set (5) of IMUs.

Classical inertial-based orientation estimation relies on the integration of the angular velocity measurements and a correction by accelerometer and magnetometer measurements. Several commonly used fusion algorithms that fuse the data from the gyroscope, accelerometer and magnetometer have been developed and are listed below [27, 28, 29, 30]:

- A proportional integral control-based Mahony algorithm which is a nonlinear complementary filter.
- A gradient descent-based filter by Madgwick.
- An EKF and variants of it.

According to [26, 27], the EKF is the most well-known and widely adopted approach to deal with noisy measurements. It handles Gaussian noise only, by estimating a mean, being the orientation representation in this case, and a covariance matrix. Since the noise of IMU measurements are shown by [31] to be Gaussian distributed, this algorithm fits the problem well. In addition to the aforementioned studies, [4, 12, 20] are also known to use the EKF to do the orientation estimation. For these reasons, the EKF is used to do inertial-based orientation estimation in this study.

As mentioned in Section 1-2, the problems caused by a loosely attached IMU concern so-called motion artifacts. These are considered to be highly nonlinear, complex and unknown phenomena and because of this, it is not trivial to capture their behavior with hand-crafted models based on physics. Therefore, instead of addressing knowledge from physics, this study turns towards data-driven ML techniques. In the field of computer vision and language processing, ML methods have shown to outperform classical methods [32]. Also, in a number of recent studies in the field of orientation estimation they have shown superior performance with respect to classical physics-based methods [1, 5, 18, 19, 24]. The authors have replaced the classical orientation estimation methods by ML methods all together for different purposes and in different ways. For example, they use different network configurations, different methods or different inputs. The use of ML techniques to supplement classical orientation estimation methods is also becoming more common. The authors of [20] apply a ML technique to dynamically estimate the noise parameters of a Kalman filter. This way, they manage to enhance IMU-based dead-reckoning methods for wheeled vehicles. Another work that addresses the subject of wheeled vehicles is [21]. The authors employ ML methods in order to robustly detect situations where there is zero velocity or no lateral slip. Sometimes ML techniques are used for denoising gyroscope measurements like in [22]. Then, there are works that try to estimate full-body poses using multiple IMUs and ML techniques [12, 13]. The latter two aim to use loosely attached IMUs like in this study. The field of ML techniques has grown too large to fully cover and in order to narrow down the options, related research serves as a source of inspiration to choices in this study. The majority of orientation estimation related studies addressing ML techniques, use the RNN. This is a neural network designed specifically to detect patterns in time. The reason for this choice lies in the fact that the problem of orientation estimation is regarded to be of time dependent nature [16]. In this study the RNN will be employed as well.

RNNs distinguish themselves from normal neural networks by a special building block. There are several variants of this block that can be used, each with different properties. They are listed below:

- The traditional RNN cell
- The gated recurrent unit (GRU)

- The LSTM cell

As stated in [1, 33], traditional RNN cells suffer from vanishing or exploding gradients during training when used to detect long-term dependencies. This will prevent the network from learning. As a result, the LSTM cell was designed which solves this problem [16, 34, 35]. The LSTM cell contains a memory unit in which information from the inputs can be stored. Access to the memory is controlled by an input and an output gate. In addition, it contains a forget gate which can discard the stored information. Later, another cell, the GRU was designed [36]. This cell contains a reset gate and an update gate. The reset gate controls which inputs are blocked and the update gate regulates which data is put through. Having fewer gates, this cell also contains fewer weights which need to be optimized. Although, this results in a better computationally efficient cell, it comes at a the expense of flexibility compared to the LSTM cell. Because, in advance, the required degree of flexibility of the model to learn the process is unknown, the LSTM cell is preferred over the GRU as it offers more flexibility. This choice is supported by numerous orientation estimation related works [1, 12, 14, 18, 19, 25]. Only [24] is known to use GRUs instead of LSTM cells for an orientation estimation related problem.

While only a few studies have been concerned with orientation estimation using loosely attached IMUs [12, 13], none of them have looked into the subject of subdividing and identifying different error sources. Knowing the sources of the error between input and output, allows the designing of models which are crafted for the specific type of error. Specifying the error sources like in Section 1-4 is new to the area of orientation estimation using loosely attached IMUs with a RNN.

Motivated by the presented related works, this study will use orientation estimates of the loosely attached IMU as input features to incorporated physics-based knowledge. The EKF will be used as the classical fusion algorithm to compute the input and target orientation estimates. Lastly, a LSTM-based RNN will be employed as the ML method to estimate the process between the input and target orientation estimates. The topic of orientation representation, classical inertial-based orientation estimation methods and the LSTM cell will be discussed more elaborately in the next sections.

## 2-2 Representing orientation

An orientation representation is a mathematical description that describes how a coordinate frame attached to an object or sensor is oriented with respect to another coordinate frame. If a frame of reference is denoted by $N$ and the object or sensor frame is denoted by $S$, then the orientation representation of the sensor with respect to the reference frame is denoted by $(\cdot)^{NS}$. There are multiple ways to represent this orientation. Each of these representations have their own specific characteristics. This section elaborates more on two orientation representations that are used in this study, namely, unit quaternions and Euler angles. A third way of representing orientation is by using orientation matrices. These serve as an intermediate step to convert a unit quaternion representation to an Euler angles representation. Details on this can be found in Appendix A.

Before going into more detail on these orientation representations, a small discussion on multiple coordinate frames commonly seen in the field of orientation estimation is described.

### 2-2-1 Coordinate frames

Throughout this study, the navigation frame ($N$) and the sensor frame ($S$) will be used most. Below is a summary of what these frames entail [31]:

- The navigation frame: The frame which the IMU navigates in. It is assumed for this study that the sensor does not travel long distances. Therefore, this frame is considered to be stationary with respect to the frame which is aligned with the earth ($E$).
- The sensor frame: The frame which is aligned with the IMU. The gyroscope expresses its measurements in this frame with respect to any stationary frame. The accelerometer and magnetometer express their measurements in this frame with respect to $N$.

### 2-2-2 Parameterizing orientation

In this section, the orientation parameterizations used in this study are discussed.

The unit quaternion is denoted by $q$ and represents 3D orientations using four components. The 2-norm of the components of a unit quaternion is always one,

$$||q||_2 = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1, \tag{2-1}$$

hence the notion 'unit' quaternion [37]. The orientation representation is non-singular but representation $q$ is equal to $-q$ so it is not perfectly unique.

The Euler angles parameterization describes orientation using rotations around three orthogonal axes. The Euler angles describe by how much an object is rotated around these axes. These rotations should be performed in a specific order because by rotating an object around one axis, the orientation of the other two axes changes [38]. There are multiple orders of rotation resulting in the fact that a set of Euler angles can represent different orientations when one does not know the order and axes of rotation. The parameterization also suffers from singularities, called gimbal lock. More on this subject will be discussed in Section 4-3-1. The advantage of this orientation parameterization is that it is intuitive for orientations defined in the 3D space. The three angles are usually called roll ($\phi$) for the angle of rotation around the x-axis, pitch ($\theta$) for the angle of rotation around the y-axis and yaw ($\psi$) for the angle of rotation around the z-axis. Those names and symbols will also be used throughout this report.

Unit quaternions will be the orientation parameterization of main interest. This is due to the fact that unit quaternions do not suffer from singularities. Also, with regard to orientation estimation using IMU data, they posses some nice properties, namely the associated differential equations can be analytically integrated. Last, concerning the ML part, it is beneficial that the components of the unit quaternions are bounded between -1 and 1 due to (2-1). This will be further explained in Section 4-6-1.

Sometimes, however, it has turned out to be necessary to perform some operations on the orientation representations using Euler angles. Therefore, the conversion from unit quaternions to Euler angles and back is required. To do this, the unit quaternions and Euler angles are expressed in terms of a rotation matrix ($R$) first. Details on these conversions can be found in Appendix A.

## 2-3 Inertial-based orientation estimation

The inertial data needed to derive orientation estimates for this study is obtained using a so-called microelectromechanical system (MEMS). It relies on the change in Coriolis force when the sensor is rotated. From this, the angular velocity is determined. Other examples of sensors that can be used to measure inertial data are a 3-axis gimbal, a ring laser gyro (RLG) and a fiber optic gyro (FOG). The former determines the angular velocity by measuring the rotation difference of one gimbal with respect to another. The latter two use the Sagnac effect. By inferring the phase shift between two oppositely directed laser beams, the angular velocity is determined [32]. MEMS are addressed in this study because they are widely used, cheap and small. Most likely because of these reasons, they can be found in a wide range of consumer products nowadays. A disadvantage that is brought along with these sensors is that their accuracy can suffer from noise and biases as already referred to in Section 1-1. The IMUs used for this study contain a magnetometer.

If one were to have a perfect initial orientation estimate and perfect angular velocity measurements, it is possible to determine perfect orientation estimates by integrating the angular velocity measurements. However, due to noise, the orientation estimates will drift over time and the error between the estimate and the ground truth will grow. To correct for this drift, accelerometer and magnetometer measurements can be used. By fusing the data of these three sensors, an optimal orientation estimate can be provided [19, 27]. The rest of this section describes the assumed measurement models, the model used to compute orientation estimates from these measurements and details on the EKF, the fusion algorithm.

### 2-3-1 Measurement models

In order to get an insight in possible measurement errors, this section gives a description of the models that are assumed to be underlying to the measurements.

As [31] points out, the IMU measurements are corrupted by a Gaussian distributed noise ($e$) which can be of non-zero mean, a bias ($\delta$).

The angular velocity of an IMU with respect to the inertial frame ($I$), which is stationary with the universe, expressed in $S$ is denoted by $\omega_{IS}^{S}$. Note that it is assumed here that the rotation of the earth is negligible and the sensor does not travel over long distances. Therefore, the rotation of $N$ with respect to $I$ can be assumed to be constant. Vectors expressed in $I$ can be transformed to $N$ by $R^{NE}R^{EI}$. To incorporate information about the sensor bias and measurement noise, the gyroscope measurements are modeled by

$$y_{\omega,t} = \omega_{IS,t}^{S} + \delta_{\omega,t}^{S} + e_{\omega,t}^{S} = \omega_{NS,t}^{S} + \delta_{\omega,t}^{S} + e_{\omega,t}^{S}, \tag{2-2}$$

where

$$e_{\omega,t}^{S} \sim \mathcal{N}(0, \Sigma_{\omega}).$$

In line with the assumption made in [30], it is assumed here that the noise over the three axes is uncorrelated. Therefore, the covariance matrix of the noise here is

$$\Sigma_{\omega} = \begin{bmatrix} \sigma_{\omega,x}^{2} & 0 & 0 \\ 0 & \sigma_{\omega,y}^{2} & 0 \\ 0 & 0 & \sigma_{\omega,z}^{2} \end{bmatrix}. \tag{2-3}$$

Measurements obtained from the accelerometer are expressed in $S$ with respect to $N$. The acceleration that will be measured is a composition of the gravitational acceleration ($g$) and the linear acceleration which the IMU experiences. The orientation of the gravitational force is known in $N$ and dominant under most circumstances. Therefore, the acceleration measurements contain information about the inclination of the sensor. In addition to that, because the gravitational force is dominant, it is common practice to assume a zero linear acceleration. The same holds for the earth's centrifugal acceleration and the Coriolis acceleration. Both are assumed to be neglectable with respect to the gravitational force. Like the gyroscope measurements, the accelerometer measurements are corrupted by Gaussian noise and a bias. The following model,

$$y_{a,t} = -R_t^{SN} g^N + \delta_{a,t}^S + e_{a,t}^S, \tag{2-4}$$

where

$$e_a^S \sim \mathcal{N}(0, \Sigma_a),$$

is used [31].

Again, in line with the assumption made in [30], it is assumed that the noise over the three axes is uncorrelated. Therefore, the covariance matrix of the noise here is

$$\Sigma_a = \begin{bmatrix} \sigma_{a,x}^2 & 0 & 0 \\ 0 & \sigma_{a,y}^2 & 0 \\ 0 & 0 & \sigma_{a,z}^2 \end{bmatrix}. \tag{2-5}$$

The magnetometer part of an IMU is used to measure magnetic field, for example that of the earth. When assumed that the sensor does not travel long distances as mentioned before, it is possible to infer the inclination of the sensor, as the direction and magnitude of the earth's magnetic field (m) is known, location dependent and characterized by the so-called dip angle ($\beta$). This angle has a value of 67.1° for the city of Delft in the Netherlands. As this study was executed in Delft, this value is used. However, the earth's magnetic field strength is relatively small and the magnetometer measurements are subject to many other magnetic fields nearby caused by electromagnetic devices or magnetic material. The magnetometer will be used supplement the orientation estimation of the sensor with estimates of the heading direction with respect to the magnetic north only. The model that is used for the measurements is

$$y_{m,t} = R_t^{SN} m^N + e_{m,t}, \tag{2-6}$$

where

$$e_{m,t} \sim \mathcal{N}(0, \Sigma_m) \quad \text{and} \quad m^N = \begin{bmatrix} \cos(\beta) \\ 0 \\ \sin(\beta) \end{bmatrix}.$$

Again, it is assumed that the noise over the three axes is uncorrelated. Therefore, the covariance matrix of the noise here is [30]

$$\Sigma_m = \begin{bmatrix} \sigma_{m,x}^2 & 0 & 0 \\ 0 & \sigma_{m,y}^2 & 0 \\ 0 & 0 & \sigma_{m,z}^2 \end{bmatrix}. \tag{2-7}$$

### 2-3-2 The dynamical model

To do orientation estimation, a dynamical model is used. In this section, first, the state of the dynamical model is discussed. Then, the equations of the dynamical model are discussed. Two commonly used state vectors for estimating orientation are

$$\mathbb{X}_t = \begin{bmatrix} q_t^{NS} \\ \omega_{NS,\,t}^S \end{bmatrix} \quad \text{and} \quad \mathbb{X}_t = \begin{bmatrix} q_t^{NS} \end{bmatrix}. \tag{2-8}$$

They distinguish themselves by using the measurements of the gyroscope as part of the state or as direct inputs to the model.

For the state vector containing the gyroscope measurements, an extra motion model describing its behaviour is required. This can be advantageous as one can include knowledge about the motion of the sensor. In this study it is not clear what movement the sensor most likely will perform. Therefore, the best estimate of this motion model would be a random walk model

$$\omega_{NS,\,t+1}^S = \omega_{NS,\,t}^S + w_{\omega,\,t}, \tag{2-9}$$

where

$$w_{\omega,\,t} \sim \mathcal{N}(0, \Sigma_{w,\,\omega}).$$

A disadvantage is that the effect of newly obtained angular measurements will have a delayed effect of one sample period on the orientation estimate. The second state vector does not suffer from this delay and also the process noise has an intuitive meaning, namely, the measurement noise of the gyroscope [31]. For this reason the angular velocity measurements will be used as direct inputs to the state for this study. Using the second state vector, the model for the orientation estimation parameterized by unit quaternions is constructed.

The gyroscope measures the angular velocity of $S$, which it is attached to with respect to a stationary frame, $N$, and expresses these measurements in $S$. The gyroscope measurements are thereby related to the derivative of the orientation of $S$. To infer the actual orientation of $S$ with respect to $N$, the measurements should be integrated. Expressing this orientation derivative using unit quaternions, the differential equation

$$\frac{d}{dt} q_t^{NS} = q_t^{NS} \odot \frac{1}{2} \bar{\omega}_{NS,\,t}^S, \tag{2-10}$$

where $(\bar{\cdot})$ indicates that this value remains constant over the sample interval, is derived. The operator $\odot$ is the quaternion multiplication as described in Appendix B. For a derivation of (2-10), the reader is referred to [31, 39]. Integrating this differential equation yields,

$$q_{t+1}^{NS} = q_t^{NS} \odot \exp\left(\frac{T}{2} \omega_{NS,\,t}^S\right), \tag{2-11}$$

where

$$\exp\left(\frac{T}{2}\omega_{NS,\,t}^S\right) = \begin{bmatrix} \cos(||\frac{T}{2}\omega_{NS,\,t}^S||_2) \\ \frac{\frac{T}{2}\omega_{NS,\,t}^S}{||\frac{T}{2}\omega_{NS,\,t}^S||_2} \sin(||\frac{T}{2}\omega_{NS,\,t}^S||_2) \end{bmatrix}.$$

This assumption only approximates the reality when the sample frequency of the sensor is high enough. The orientation of $S$ with respect to $N$ is now described by $q_{t+1}^{NS}$.

### 2-3-3 The extended Kalman filter

The EKF is based on two main steps, named the ‘time update’ and the ‘measurement update’. Sometimes these steps are also referred to as the ‘process step’ and the ‘correction step’. In the first step, the angular velocity measurements are used, pretending that they and the dynamical model are perfect. In the second step, the accelerometer and magnetometer measurements are used to correct the errors due to incorrect measurements and model errors. In both steps, the covariance matrix is updated according to the (un)certainty of the measurement. In the second step, the Kalman gain ($K$) acts as a weight and is constructed using this covariance matrix. Lastly, it is possible that the estimate is not normalized after the second step. Therefore, a third step is included to normalize the orientation estimate again. An overview of the applied algorithm is shown in Algorithm 1.

**Algorithm 1:** Orientation estimation using an EKF with quaternion states

INPUTS: IMU data $\{y_{\omega,t}, y_{a,t}$ and $y_{m,t}\}_{t=1}^{N}$ and covariance matrices $\Sigma_\omega$, $\Sigma_a$ and $\Sigma_m$.
OUTPUTS: An estimate of the orientation, $\hat{q}_t^{NS}$, and its covariance $P_t$ for t = 1, ...N.

1. Calibrate gyroscope, accelerometer and magnetometer and initialize state.
2. **For** t = 2, ... N **do**

   (a) Time update

$$\hat{q}_{t+1}^{NS} = \hat{q}_t^{NS} \odot \exp\left(\tfrac{T}{2}\omega_{NS,\,t}^{S}\right)$$
$$P_{t+1} = F_t P_t F t^T + G_t Q G_t^T$$

where $Q = \Sigma_\omega$, $\quad F_t = \left(\exp\left(\tfrac{T}{2} y_{\omega,t}\right)\right)^R$ $\quad$ and $\quad G_t = -\tfrac{T}{2}\left(\hat{q}_t^{NS}\right)^L \frac{\partial \exp(e_{\omega,t})}{\partial e_{\omega,t}}$.

   (b) Measurement update

$$\hat{q}_{t+1}^{NS} = \hat{q}_t^{NS} + K_t(y_t - h_t(\hat{q}_t^{NS}))$$
$$P_{t+1} = P_t - K_t(H_t P_t H_t^T + R)K_t$$

where $K_t = P_t H_t^T (H_t P_t H_t^T + R)^{-1}$, $\quad y_t = \begin{bmatrix} y_{a,t} \\ y_{m,t} \end{bmatrix}$, $\quad h_t = \begin{bmatrix} -\hat{R}_t^{SN} g^N \\ \hat{R}_t^{SN} m^N \end{bmatrix}$

$$H_t = \begin{bmatrix} -\frac{\partial \hat{R}_t^{SN}}{\partial q_t^{NS}}\Big|_{q_t^{NS}=\hat{q}_{t-1}^{NS}} g^N \\ \frac{\partial \hat{R}_t^{SN}}{\partial q_t^{NS}}\Big|_{q_t^{NS}=\hat{q}_{t-1}^{NS}} m^N \end{bmatrix}, R = \begin{bmatrix} \Sigma_a & 0 \\ 0 & \Sigma_m \end{bmatrix}, g^N = \begin{bmatrix} 0 \\ 0 \\ -9.81 \end{bmatrix} \text{ and } m^N = \begin{bmatrix} \cos(\beta) \\ 0 \\ \sin(\beta) \end{bmatrix}.$$

   (c) Renormalize quaternion representation

$$\hat{q}_{normalized,t}^{NS} = \frac{\hat{q}_t^{NS}}{||\hat{q}_t^{NS}||_2}$$

## 2-3-4 The LSTM cell

This section provides more detailed information about the architecture of the LSTM cell. In Figure 2-1 its architecture is displayed. The three gates are highlighted, as well as the weights of the cell. The yellow boxes indicate activation functions, where $\sigma$ represents the sigmoid function. This function takes on values between 0 and 1 and is used for the three gates in order to pass information on ('1') or block information ('0').

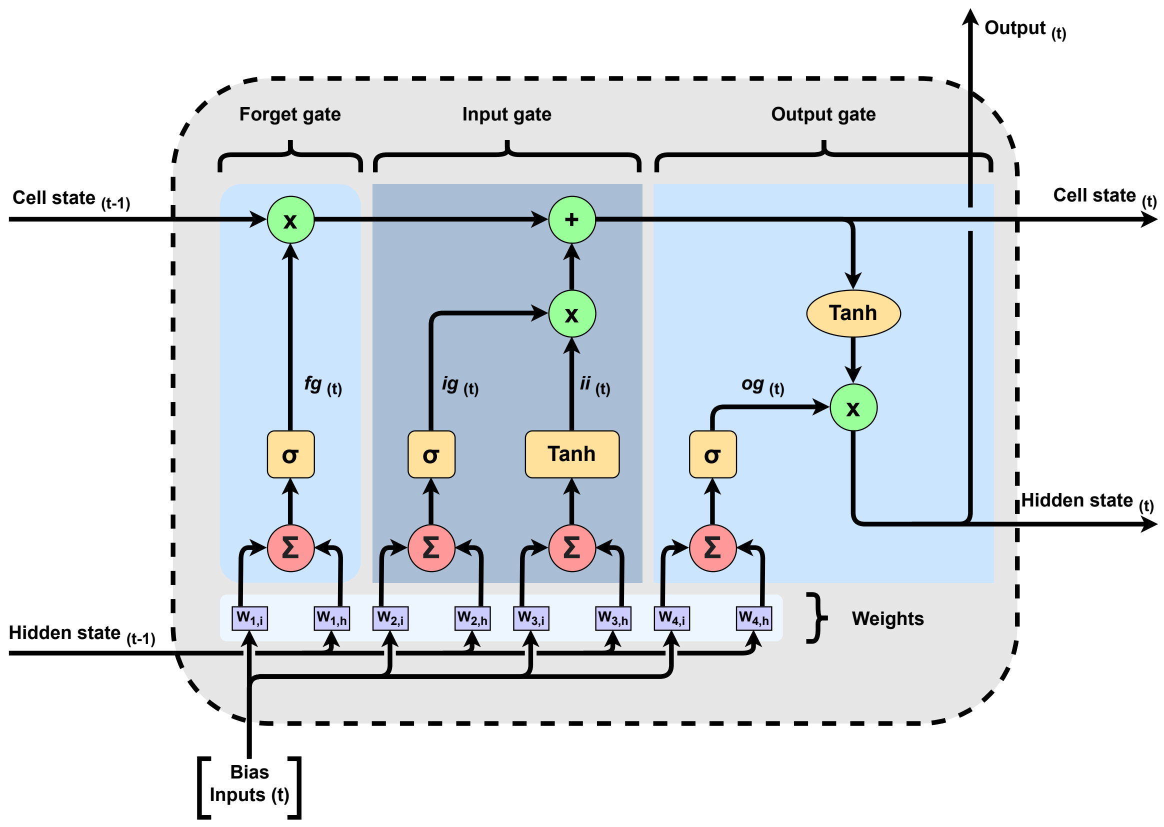


**Figure 2-1:** The architecture of a single LSTM cell inspired by [1, 2, 3].

For clarity, four signals have been declared in Figure 2-1:

- A forget gate signal, $fg_{(t)}$.
- An input gate signal, $ig_{(t)}$.
- A signal carrying the information of the input, $ii_{(t)}$.
- An output gate signal, $og_{(t)}$.

Their relations, together with the output signal are

$$\begin{aligned}
fg_{(t)} &= \sigma\left(\vec{W}_{1,i}\begin{bmatrix}\text{Bias}\\ \text{Inputs (t)}\end{bmatrix} + W_{1,h}\text{ Hidden state}_{(t-1)}\right),\\
ig_{(t)} &= \sigma\left(\vec{W}_{2,i}\begin{bmatrix}\text{Bias}\\ \text{Inputs (t)}\end{bmatrix} + W_{2,h}\text{ Hidden state}_{(t-1)}\right),\\
ii_{(t)} &= \tanh\left(\vec{W}_{3,i}\begin{bmatrix}\text{Bias}\\ \text{Inputs (t)}\end{bmatrix} + W_{3,h}\text{ Hidden state}_{(t-1)}\right),\\
og_{(t)} &= \sigma\left(\vec{W}_{4,i}\begin{bmatrix}\text{Bias}\\ \text{Inputs (t)}\end{bmatrix} + W_{4,h}\text{ Hidden state}_{(t-1)}\right)\text{ and}\\
\text{Output}_{(t)} &= og_{(t)}\tanh\big(fg_{(t)}\text{ Cell state}_{(t-1)} + ig_{(t)}\, ii_{(t)}\big). \qquad (2\text{-}12)
\end{aligned}$$

The amount of weights that needs to optimized for every cell is

$$\#_{Weights} = (\#_{Inputs} + \text{Bias}) \times 4 + 4 \times \text{Hidden state}. \qquad (2\text{-}13)$$

Chapter 3

# General subjects for orientation estimation using a RNN

*Before discussing any possibilities to improve orientation estimation related problems with the use of a RNN, a small study to general subjects and capabilities of RNNs is conducted and presented here. These subjects are the data format of the input, implications of using LSTM cells and requirements of input and output data. The last subject is subdivided into problems of ambiguous data and problems of discontinuities in sequential sequences.*

## 3-1 Dimensions input data

For those who are not familiar with the use of RNNs, this section explains a bit more about the naming, regarding parts of the input dataset. The input dataset for a normal neural network usually is a matrix containing the amount of data samples (N) and the amount of features. In this study, the data is sequential data which will be denoted by a subscript $t$. Using RNNs requires sequential sequences of limited length to be fed into them. This leads to the input dataset being a tensor, with the extra dimension being the length of a sequence ($l$). The total number of sequential sequences is still described by N. The shape of the input dataset can thus be described by [Amount of sequences (N) $\times$ Sequence length ($l$) $\times$ Features]. An image of this input tensor is displayed in Figure 3-1. As it is intended to use unit quaternions as input features, the features are already denoted by $q$.

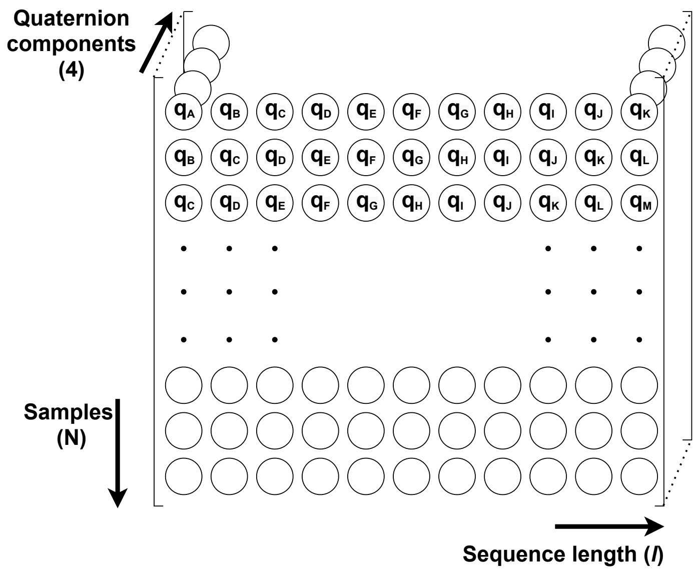

Quaternion components (4)
Samples (N)
Sequence length (l)
qA qB qC qD qE qF qG qH qI qJ qK
qB qC qD qE qF qG qH qI qJ qK qL
qC qD qE qF qG qH qI qJ qK qL qM

**Figure 3-1:** The dimensions of the input tensor.

## 3-2 Overfitting

LSTM layers are powerful in the context of storing information for an arbitrary amount of time steps and access stored information at any time step [33]. However, they have their drawbacks too. Besides that they are computationally less efficient than GRU and RNN layers, as explained in Section 2-1, another disadvantage of LSTM layers lies in the fact that they are prone to overfitting [19]. A network that is overfitting achieves very high performance on its training data but generalizes very badly on unseen data. To handle this problem, a first step is to be able to detect overfitting of the network. Next, it is necessary to know how to counteract overfitting. These two subjects will be discussed below.

The difference in performance between training data and validation data will be used as an indication of overfitting. A requirement to the validation data is that it has not been seen during the training phase. The learning curve of a network can give an indication of the problem of overfitting. For every epoch during training, it shows the performance of both the training data and the validation data. If the performance of the validation data decreases while the performance of the training data gets better, the network is overfitting. In Figure 3-2 an example of such a learning curve is given. Note that the data needed to generate this figure was self made-up.

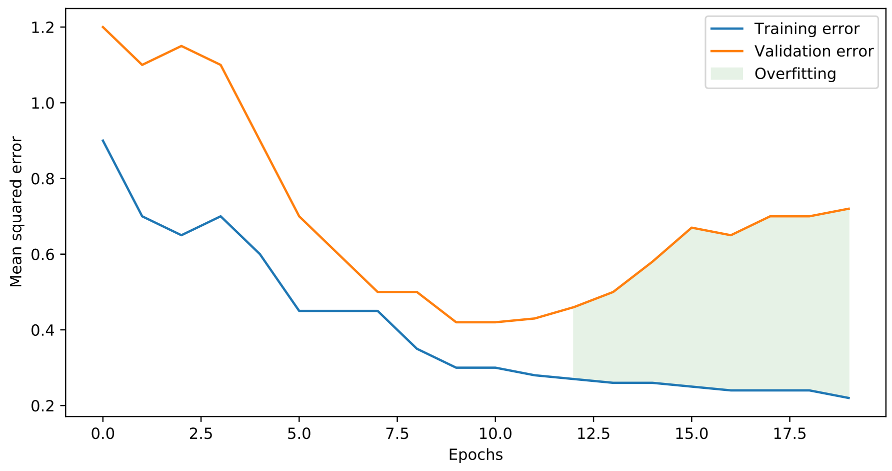


**Figure 3-2:** A plot showing an artificially generated learning curve that indicates overfitting.

If overfitting is detected there are several measures that can be taken to prevent this. Among those are [40, 41]:

1. Applying dropout. For RNNs, two types of dropout methods are available [41]: A dropout method which sets random outputs of cells to 0 and a dropout method which sets random input weights to 0. The latter goes by the name 'DropConnect'. The goal of this technique is that every cell learns independent from the others. Both types are considered very effective but have their disadvantages too. The first method of dropout is suspected to reduce the capability of detecting long-term dependencies [40]. DropConnect solves this, however, it is not compatible yet with the optimized cuDNN implementation. This optimized cuDNN implementation accelerates the training process by addressing an available graphics processing unit (GPU) alongside the central processing unit (CPU) [42]. Not being able to use this will cause an increase in the time it takes to train the network. As training the network already takes quite some time, this is highly undesirable.
2. Increasing the amount of representative data.
3. Applying batch normalization layers which normalize their input data.
4. Applying weight regularization by penalizing the 2-norm of weights of the network.
5. Early stopping of the training process.
6. Decreasing the amount of parameters in the model by decreasing the network's size. This makes the model less flexible and thus also less prone to overfitting.

In this study the learning curve of a trained network without regularization methods will be examined first. Then, only if overfitting was clearly indicated by the learning curve, regularization methods will be applied. First, the first dropout method like in [19] will be applied. The amount of required regularization will be determined empirically. If the network will be still overfitting then, it will be tried to increase the amount of data.

## 3-3 Ambiguous data

One of the things to take into account when using neural networks is that the presented data should have a unique input-output relation. This means that only one output corresponds to a given input. To illustrate this requirement, a test is performed on a simple RNN. The loss function that will be used to train the network is the mean squared error (MSE). For this test, an input and target output is designed and the neural network should learn the relation between those. The input is linear data ranging from 0 to 1. To form the output, this input is multiplied by 100 and mapped by a cosine function to the domain $-\pi$ to $\pi$, meaning that multiple inputs correspond to the same output hence the input-output relation is not unique. In Figure 3-3 a plot is shown of the input and output data and the squared error between the target and predicted output data. In this case, the MSE is minimal if the network just forces its output to zero, resulting in a network that has not learned anything. This phenomenon occurs for periodic data, however, it could just as easily happen to other datasets.

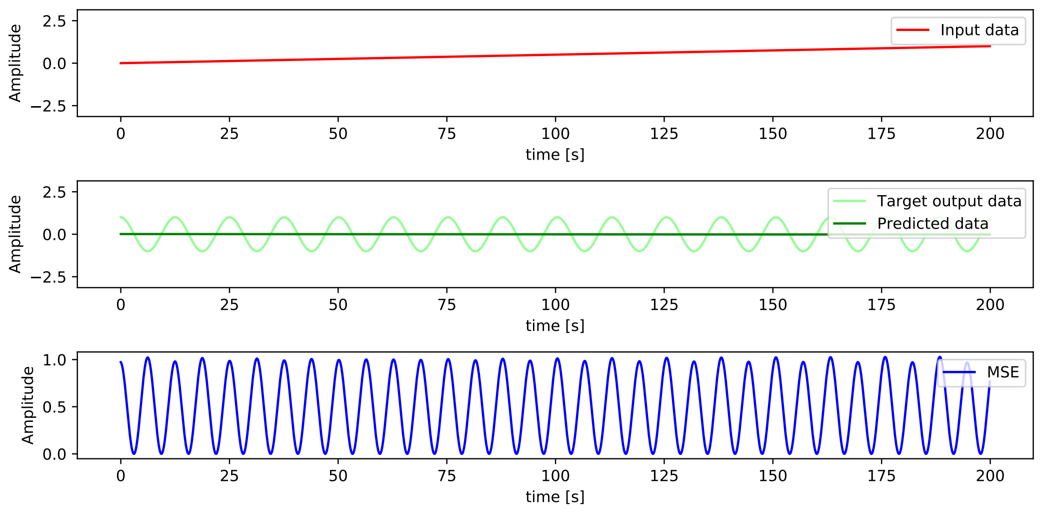


**Figure 3-3:** Data and performance of the RNN on ambiguous data.

As mentioned, unit quaternions will be used as in- and output data for this study. These unit quaternions offer multiple advantages over other orientation representations, however, they are not entirely unique. As discussed in Section 2-2-2, every unit quaternion $q$ represents the same orientation as $-q$. For this cause, special care has to be taken using the unit quaternions as in- and output data.

## 3-4 Discontinuous sequences

If a trained network is examined carefully, one realizes that in the end it is a huge mathematical expression composed of continuous functions that depend on the input. This implies that the output can only be discontinuous if the input is discontinuous. Hence, a certain level of continuity must be imposed on sequential data. To reveal this, a simple RNN is employed which is supposed to learn a linear input-output relation. The used loss function for training is the MSE again. The input and output data's range is between -1 and 1. When exceeding this range, the data is mapped to the other end of the range, meaning that a value of 1.2 becomes -0.8. There is an offset between the input and output that the network should learn. The first test, shown in Figure 3-4, addresses the situation where there is no discontinuity in the data. The second test, in Figure 3-5, shows the performance of the network when a discontinuity exists in the input data sequences. For the first test, the network will most likely learn a bias to map the input to the output. In contrast, the second requires a 'discontinuous' jump to be learned, which is a different operation than simply adding a constant to the input. The network will try to use its so-called flexibility to minimize the total error during the learning phase, however, the result will always be an approximation. This becomes clear in this example, as the error is much higher for the second test. For this reason it is important to take measures to prevent such situations and to use a model that has the flexibility that meets the input-output data relation.

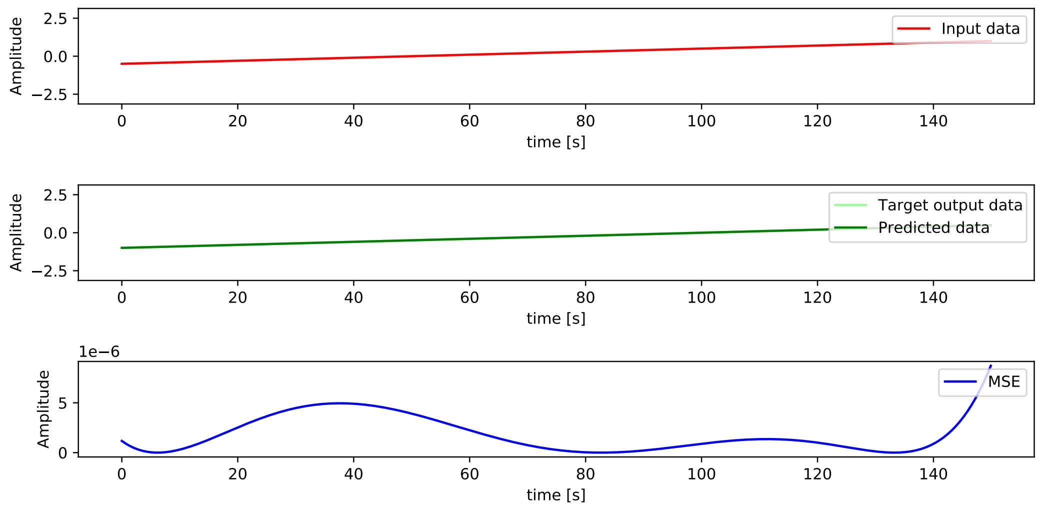


**Figure 3-4:** Data and performance of the RNN on continuous data sequences.

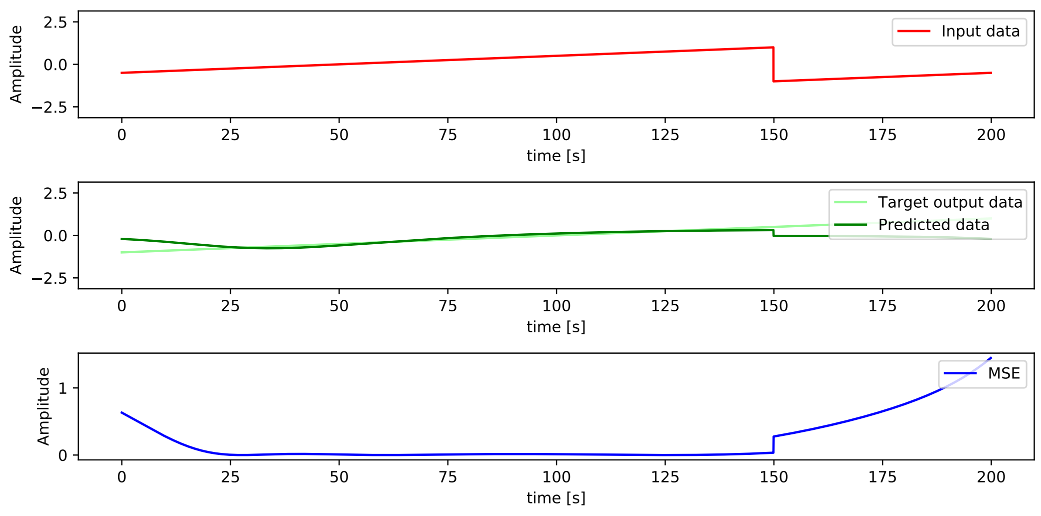


**Figure 3-5:** Data and performance of the RNN on discontinuous data sequences.

# Chapter 4

# Methods

*This chapter starts with describing the framework of the model and details on the EKF and the RNN which are part of this model. Next, the reader is given an overview of the complete proposed model. To do this, three preprocessing steps are introduced that are ought to constitute beneficial steps in the estimation of orientation using a loosely attached IMU with a RNN and physics-based knowledge. After that, a more in-depth explanation and the implementation of the preprocessing steps that this study has led to, are given. The first step which is described removes the average misalignment between the data of two sensors and makes the network more robust. Then, the second step is discussed. This step deals with irrelevant information in data about the absolute heading and how either the dataset is extended or how the search space is reduced. The third step that is explained, removes ambiguity and discontinuities from the data. This contributes to a reduced need for data samples. The last section of this chapter gives attention to the decision of using physics-based methods. It presents a way of verifying whether this decision was beneficial in the context of this problem.*

## 4-1 The framework of the model

This section shortly discusses the main options for combining the EKF and the RNN. The input of the model is the inertial data that is coming from the loosely attached IMU denoted by $y^{S_{loose}}_{(\cdot),t}$. If this data is fed to the EKF, orientation estimates of the loosely attached IMU become available and are denoted by $\hat{q}^{NS_{loose}}_{EKF,t}$. From this sequential array of unit quaternions, a matrix of unit quaternions is formed as described in Section 3-1, containing sequential sequences of unit quaternions denoted by $\hat{q}^{NS_{loose}}_{EKF,t,l}$. In this study, this operation is applied to form the input tensor for the RNN and will not be explicitly mentioned but will be indicated by the additional subscript $l$. To denote IMU data of the tightly attached IMU, the notation $y^{S_{tight}}_{(\cdot),t}$ is used. The target output of the model is the orientation of the tightly attached IMU denoted by $q^{NS_{tight}}_{Target,t}$. The two parts will be implemented in a cascade fashion where the output of the first part will be the input to the second part. This hybrid model, as it is called, can be constructed in two basic ways displayed in Figure 4-1.

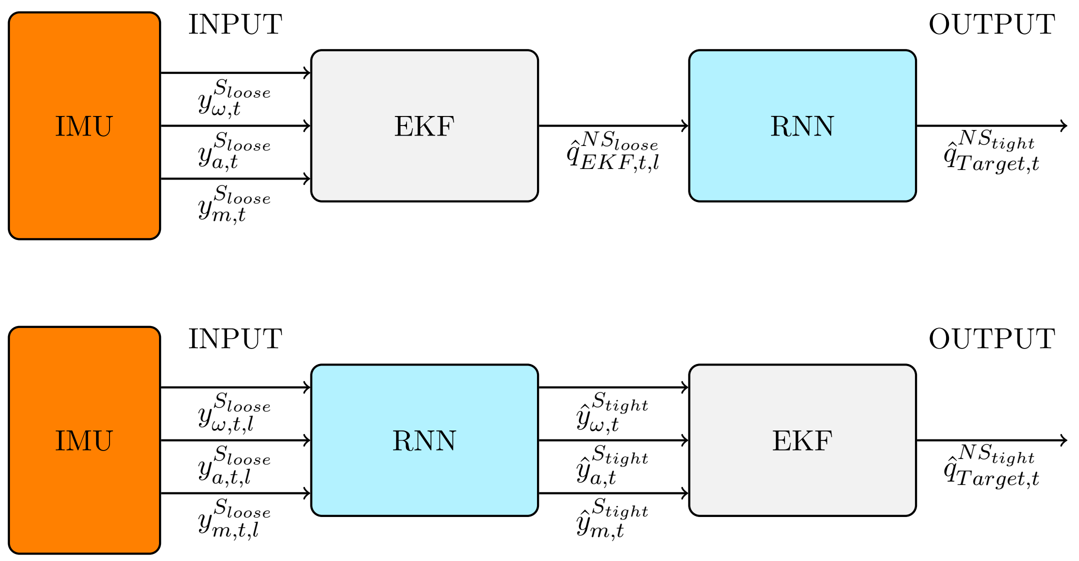

INPUT
OUTPUT
IMU
$y^{S_{loose}}_{\omega,t}$
$y^{S_{loose}}_{a,t}$
$y^{S_{loose}}_{m,t}$
EKF
$\hat{q}^{NS_{loose}}_{EKF,t,l}$
RNN
$\hat{q}^{NS_{tight}}_{Target,t}$
INPUT
OUTPUT
IMU
$y^{S_{loose}}_{\omega,t,l}$
$y^{S_{loose}}_{a,t,l}$
$y^{S_{loose}}_{m,t,l}$
RNN
$\hat{y}^{S_{tight}}_{\omega,t}$
$\hat{y}^{S_{tight}}_{a,t}$
$\hat{y}^{S_{tight}}_{m,t}$
EKF
$\hat{q}^{NS_{tight}}_{Target,t}$

**Figure 4-1:** Basic structure of the hybrid models.

Because of the properties that the EKF possesses with regard to dealing with Gaussian noise, it is a logical choice to stick to the first model. This choice is supported by the fact that unit quaternions represent the information about orientation much more compact than inertial data. Therefore, the search space of the RNN in the second model is much bigger and expected is that more data would be required to fully learn the problem. Lastly, as explained in Section 1-2 and supported by the authors of [17], it is aimed to use physics-based knowledge. The first model allows to exploit the physics-based knowledge using the EKF better because signals from the IMU could later still be supplemented to the input features of the RNN anyway. For these reasons, the first model is implemented in this study.

### 4-1-1 The extended Kalman filter

For the implementation of the EKF, Algorithm 1 is adopted. As mentioned in Section 2-3-1, the inertial measurements can contain a bias. For the gyroscope, this bias is slowly time varying as [43] mentions and can form a serious problem if not taken into account. In order to account for these time varying biases, a random walk model like

$$\delta^S_{\omega,t+1} = \delta^S_{\omega,t} + e^S_{\delta_\omega,t}, \tag{4-1}$$

where

$$e^S_{\delta_\omega,t} \sim \mathcal{N}(0, \Sigma_{\delta_\omega}),$$

can be assumed as explained in [31]. However, a simpler method for measurements that are of short duration is to properly calibrate the gyroscope before the recording and assume it to be constant during the recording [44]. For coping with biases in gyroscope measurements, the latter method will be applied in this study.

To cope with possible biases mentioned in the accelerometer measurement model (2-4), the average norm of measurements from a stationary sensor is verified. If it equals the gravity vector, it is assumed that the factory calibration is adequate for this study [44]. This turned out to be the case.

Also mentioned in Section 2-3-1, the magnetometer measurements are subject to other magnetic fields nearby. As a result, disturbed magnetic environments easily corrupt the estimated heading direction of the sensor. Therefore, a simple fault detection algorithm is employed as a measure against this. Before the recording, the magnetometer is calibrated such that the norm of the measured field is scaled to 1. If then the norm of recorded measurement data exceeds a certain lower or upper bound, the measurement is rejected. These bounds are empirically determined and set to 0.9 and 1.1.

### 4-1-2 The recurrent neural network

There are several parameters that have been decided on regarding the configuration of the RNN displayed in Figure 4-1. The configuration is among others defined by the amount of layers in the network, the amount of cells in every layer, the activation function of the cells and the sequence length ($l$) of the input. These are multiple variables offering many possible configurations. Optimizing them for the problem of this study requires a lot of time. As optimizing this is not the goal of the study and time is limited, it is decided to save time on this part. Therefore, to determine suitable parameters for the configuration, common sense and information is drawn from successful related studies.

With regard to the number of layers in the network, a distinction is made between LSTM layers for detecting time dependencies and fully connected (FC) layers. The latter are implemented to shape the input of the network to a usable input for the LSTM layers and to shape the output of the LSTM layers to the output of the network. The studies [1, 17, 18, 19, 32, 45] all use two layers to detect time dependencies. Part of the explanation of why usually not more layers are used is that it does not improve the performance and the training becomes more computationally demanding. This amount of LSTM layers is adopted, as well as two FC layers before and after the LSTM layers.

The six layers are numbered in order to address the amount of cells used in the layers. The number of cells in the first, second and fifth layer is often around a few dozen. The last layer contains as many cells as outputs of the network, four in this case. The number of cells in the first layer is preferably more than the number of inputs or otherwise information may be lost. This assumes that all inputs contain useful information. To prevent loosing information in the other layers as well, the number of cells should be higher than or equal to the number of variables by which the target information can be described. As there is generally no insight in this information it is better to use too many than too few cells. If cells are redundant, the neural network is capable of learning this too by setting its input weights to 0. The amount of cells is set to 40 for the first, second and fifth layer. For the number of cells in the third and fourth layer, the LSTM layers, inspiration is drawn from [17, 18, 19, 32, 45]. They use LSTM layers with an amount of cells ranging from 96 up to 258. As for the FC layers, it is true that it is better to use too many than too few cells and by that line of thought, each LSTM layer will contain 200 cells for this study.

It is chosen to use the ‘tanh’ function as activation function for the cells. Due to their nonlinearity they allow the network to learn nonlinear processes. It is a function that is centered around zero and which is smooth. Therefore, its derivative, which is used during

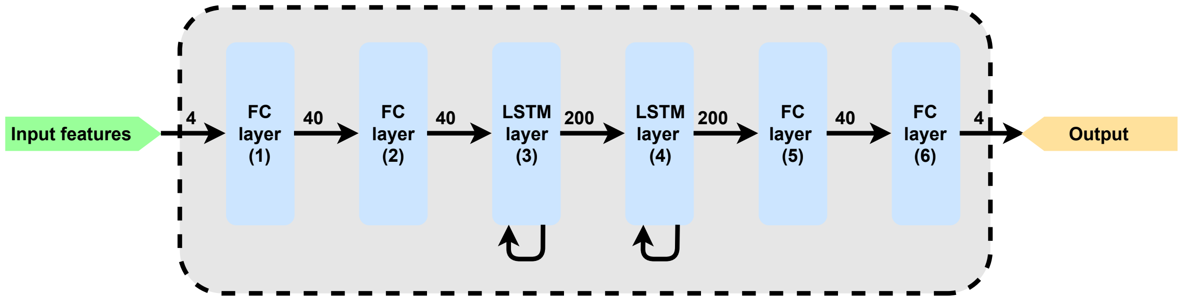
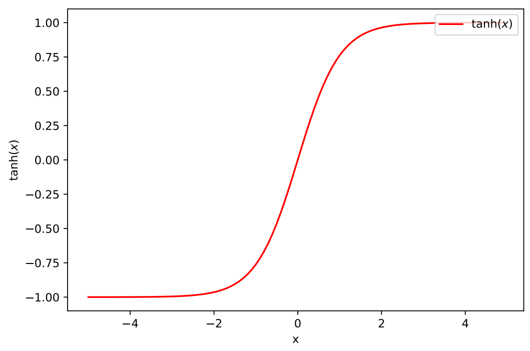
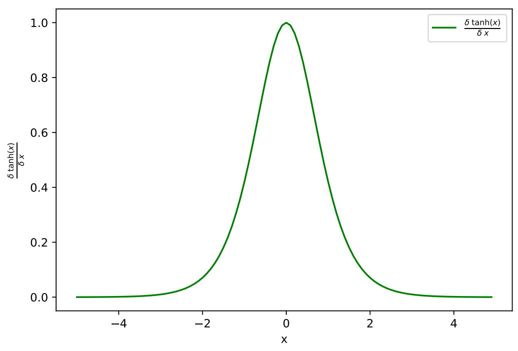


**Figure 4-2:** The network's architecture.

training, is continuous. An advantage of using the tanh function is that it is compatible with the 'cuDNN implementation' [42]. Figure 4-3a and Figure 4-3b show plots of the function and its derivative. To obtain the data for these plot, Python was used.



**(a)** The tanh function.



**(b)** The derivative of the tanh function.

**Figure 4-3:** The activation function for cells in the neural network.

The amount of samples in sequential sequences that the networks uses to learn and do estimation is another variable. It is sometimes also called the window size. In this study it is denoted by $l$ and set to 11 samples. This value is inspired by [32], which points out that a sequence length of that value performs well for orientation inference. This sequence length value is also used by the authors of [18], who try to predict orientation estimates as well.

## 4-2 The proposed model

Before explaining the exact working of the three steps of preprocessing the data, the reasons why and the way they fit into the hybrid model are presented. This way, the reader is provided with an overview of the whole model in advance.

It is very probable that during usage of the model, the loosely attached IMU is mounted with a different orientation than when the network was trained. Figure 4-4 shows an example of a phone that contains an IMU. In this example the phone is used only to illustrate the problem and any other device with an IMU in it or just an IMU itself could be used.

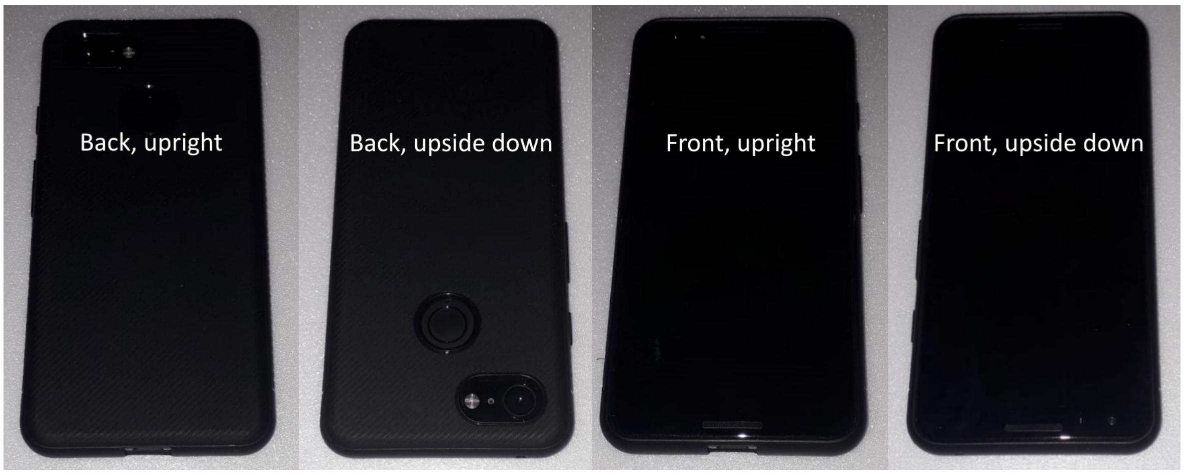


**Figure 4-4:** Four different orientations a phone can take given a fixed space, like a pocket.

This phone can take already four different orientations without even using a differently shaped space. If the space where the IMU is attached to is not fixed, the sensor could be mounted in any orientation. If the network is presented with differently oriented data than it was trained for, its estimation performance will deteriorate. It is also highly undesirable that the network has to be trained every time it is used. Therefore, the aim is to make the network robust to such misaligned attachments. This is the first step of preprocessing the data. It entails the removal of the average misalignment between the orientation of the loosely attached IMU and the orientation of the target or tightly attached IMU, from the orientation estimates of the loosely attached IMU, $\hat{q}^{NS_{loose}}_{EKF,t,l}$. This average misalignment is denoted by $q^{S_{tight}S_{loose}}_{Average\ misalignment}$ and is constant. The resulting orientation estimates are denoted by $\hat{q}^{NS_{loose}}_{Aligned,t,l}$. This step answers to the first research question, as it allows to determine the error before removing the average misalignment and afterwards. This way, the error due to a misaligned mounting of the IMU can be isolated. In addition, training the network with data of which the average misalignment has been removed, results in a model that is robust to such misaligned mountings.

The second step of preprocessing the data is inspired by [19]. The authors of this paper try to estimate the attitude of an IMU using its corresponding inertial data and ML techniques. One of the things they do to improve the performance is to add data to the dataset which makes it invariant to the recorded attitude. They do this by rotating the recorded data by random rotations and augment their dataset with this new data. Something similar can be applied to the problem of this study, however, care should be taken. The movement and thus the orientation of the loosely attached IMU with respect to the tightly attached IMU can be affected by gravity. For example, the movement of falling downwards is different to a movement of 'falling' upwards which is actually impossible in itself. So, making the data invariant to all orientations, like they do in [19], would remove relevant information. One can imagine though, that in the case of the problem of this study, performing the same movement when facing north, east, south or west should not result in different data describing that movement, irrespective of what kind of movement that might be. The direction that a sensor is facing is referred to as the 'absolute heading'. Inspired by this idea, two different methods are designed to achieve absolute heading independence. One which is closely related to the method that is applied in [19] and a second one which was thought of during the study. They are listed below:

1. Remove dependency of data on absolute heading by augmenting the dataset.
2. Remove dependency of data on absolute heading by removing absolute heading information.

The heading dependency gets removed from the aligned orientation estimates $\hat{q}^{NS_{loose}}_{Aligned,t,l}$ and the resulting orientation estimates are denoted by $\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l}$. This step also answers to the second research question. By making the orientation estimates independent from the heading direction, it does not matter in which heading direction training data or test data is recorded.

The third step removes possible ambiguity and discontinuities from the data. This is in response to the problems with data that is fed to a RNN, demonstrated in Section 3-3 and Section 3-4. The resulting orientation estimates are referred to as $\hat{q}^{NS_{loose}}_{Unique,t,l}$. This step is designed to contribute to answering the second research question as it lowers the amount of data needed to learn the process or improves the performance.

The resulting hybrid model is displayed in Figure 4-5. Note that for removing the heading dependency, two methods are designed. The first method directly estimates the heading direction and in that case the orientation estimates of the RNN are the estimates of the target orientation $\hat{q}^{NS_{tight}}_{Target,t}$. The second requires to restore the heading direction of the orientation estimates of the RNN, this method is indicated by the dashed arrows. The orientation estimates of the tightly attached IMU made by the RNN for this method are denoted by $\hat{q}^{NS_{tight}}_{\psi_{invariant},t}$. The three steps of preprocessing are indicated in the model.

## 4-3 Removing the average misalignment

In this section the details of the first preprocessing step are explained. This step will address the part of the error that is described by the 'average misalignment' in Chapter 1. It serves two purposes.

1. It can be used to identify the partition of the error between input and output data caused only by the differently oriented attachment. The remaining part of the error is likely caused by the process and the heading dependency of the data.
2. It is used to make the model robust to misaligned attachments.

### 4-3-1 The baseline performance

To be able to compare the performance of the different preprocessing steps and the RNN, there is a need for a reference or a baseline performance of the data. This baseline performance should therefore not be determined using any ML method. In the paragraph that follows, the most basic algorithm is described. Next, a more sophisticated algorithm to set a baseline performance is presented. The latter applies the first preprocessing step and removes the average misalignment from the data.

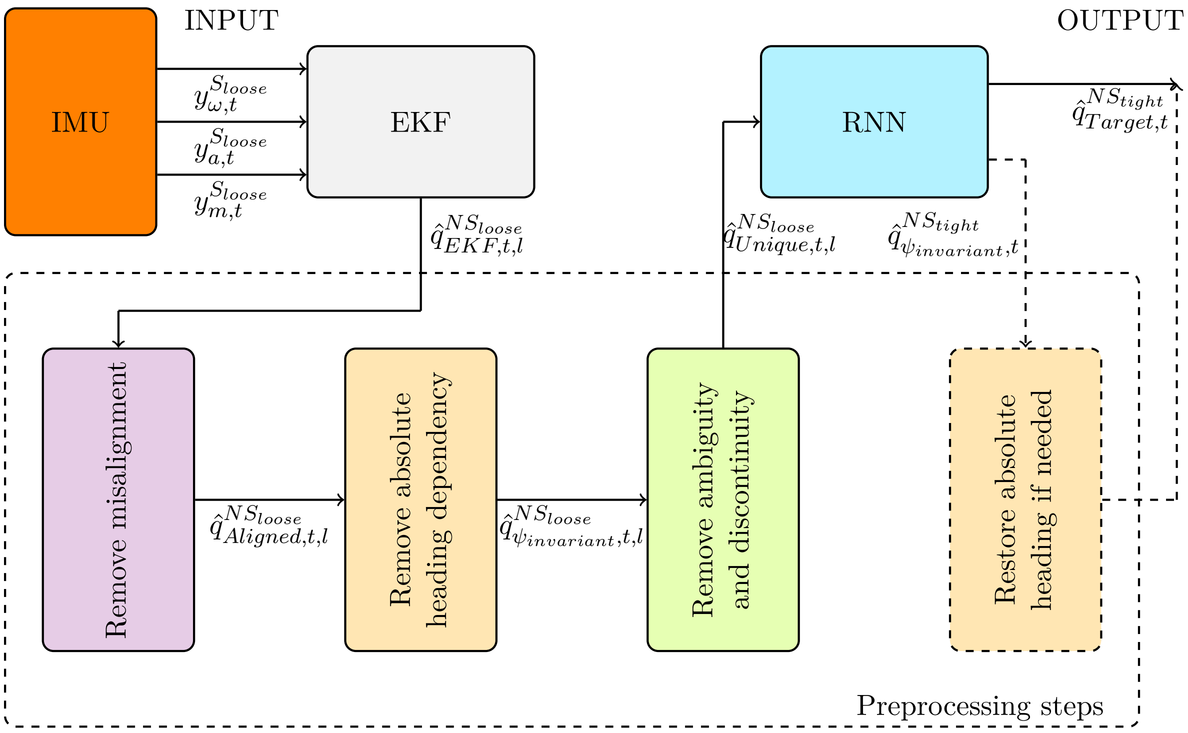

INPUT
OUTPUT
IMU
EKF
RNN
$y_{\omega,t}^{S_{loose}}$
$y_{a,t}^{S_{loose}}$
$y_{m,t}^{S_{loose}}$
$\hat{q}_{Target,t}^{NS_{tight}}$
$\hat{q}_{EKF,t,l}^{NS_{loose}}$
$\hat{q}_{Unique,t,l}^{NS_{loose}}$
$\hat{q}_{\psi_{invariant},t}^{NS_{tight}}$
Remove misalignment
$\hat{q}_{Aligned,t,l}^{NS_{loose}}$
Remove absolute heading dependency
$\hat{q}_{\psi_{invariant},t,l}^{NS_{loose}}$
Remove ambiguity and discontinuity
Restore absolute heading if needed
Preprocessing steps

**Figure 4-5:** Basic structure of the hybrid model with preprocessing steps included.

The simplest way to obtain a baseline performance is constructed by taking the last input samples of all sequences and compare these to the target samples. Describing the input by $\hat{q}_{EKF,t,l}^{NS_{loose}}$ and the output by $\hat{q}_{Target,t}^{NS_{tight}}$, the last samples of the input sequences, $\hat{q}_{EKF,t,11}^{NS_{loose}}$, are compared to the target samples, $\hat{q}_{Target,t}^{NS_{tight}}$. The last subscript of the input data, denotes the sample in the sequence. Details on how two orientation representations are compared will be discussed in Section 6-1.

The performance resulting from this will most likely not form the best baseline performance. In addition, its performance is highly dependent on how big the average misalignment between input and target sensor is. Due to the uncertainty as to what the error of this simple way represents, not much information about the error caused by the loosely attachedness can be obtained. Because the interest of this study also lies in the mutual movement between the two sensors, a more advanced algorithm is designed to set a baseline performance.

If the average misalignment is removed from the data, the baseline performance would in most cases be much better. It would also give a much better indication of what percentage of the initial error is caused by the loosely attachedness. To this end, a more advanced way of determining a baseline performance is described here. The average orientation difference between input and target data is computed and removed from the input data. The aligned input data, $\hat{q}_{Aligned,t,l}^{NS_{loose}}$, and the target data $\hat{q}_{Target,t}^{NS_{tight}}$ are then compared in the same way as described above. Doing this, the orientation of the input data lies closer to the orientation of the target data and the error between them is reduced. The increase between the two baseline performances can be attributed to the removal of the average misalignment as outlined in Figure 1-1 in Section 1-4. To even further increase the performance, the error caused by the process and heading dependency should be addressed.

In order to implement the more sophisticated baseline performance, first the orientation difference between all samples,

$$\hat{q}_t^{S_{tight}S_{loose}} = (\hat{q}_{EKF,t}^{N_1 S_{tight}})^c \odot \hat{q}_{EKF,t,11}^{N_2 S_{loose}}, \tag{4-2}$$

is computed. The quaternion conjugate, recognizable by the subscript $(\cdot)^c$, is explained in Appendix B. The quaternion multiplication is only valid if the navigation frame $N_1$ of the target data aligns with the navigation frame $N_2$ of the input data. It is unlikely to happen, but differences between the two frames can occur. They can be caused by biases, faulty calibrations or externally caused differences in the magnetic field measured by the sensors. In this study it is expected that this will not pose a problem as identical IMUs are used and the same calibration procedures are applied to them. Also, the sensors are close to each other and no interfering material is used during testing. To verify this, the orientation difference between the two navigation frames is computed for data of experimental setup 1 described in Section 5-3. There is no difference between them if this orientation difference ($\hat{q}^{N_1N_2}$) is equal to

$$\hat{q}^{N_1N_2} = \begin{bmatrix} \pm 1 & 0 & 0 & 0 \end{bmatrix}^T. \tag{4-3}$$

Since the navigation frames are considered stationary, the algorithm presented in Appendix C can be applied [39]. The orientation difference between the two frames is

$$\hat{q}^{N_1N_2} = \begin{bmatrix} -1.000 & 0.00219 & 0.00122 & 0.0143 \end{bmatrix}^T \approx \begin{bmatrix} \pm 1 & 0 & 0 & 0 \end{bmatrix}^T. \tag{4-4}$$

Using Appendix A to convert this orientation difference to Euler angles for a more intuitive parameterization yields a orientation difference of $\hat{\phi}^{N_1N_2} = -0.00435$ [rad], $\hat{\theta}^{N_1N_2} = -0.00251$ [rad] and $\hat{\psi}^{N_1N_2} = -0.0285$ [rad]. There is hardly any difference in orientation between the frames and they are assumed to be aligned during this study: $N_1 = N_2 = N$. Note, the unit quaternion in (4-4) is not normalized. This is caused by rounding errors.

Unfortunately, the algorithm in Appendix C can not be applied to compute the average orientation difference between the two frames $S_{loose}$ and $S_{tight}$ because the orientation difference between those frames does not remain constant [39]. There is also no straightforward method to compute a physically meaningful average of a sequence of unit quaternions which complicates things. To solve this problem, the conversion to Euler angles is used (Appendix A) which range from $-\pi$ to $\pi$ and for $\theta$, the pitch angle, range from $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. If the rotation around an axis exceeds this range the angle gets 'wrapped' and information about how many rotations have been made gets lost. This could ruin the computation of the mean of the sequence of Euler angles representations. To retrieve this information, it is assumed that the rotation speed of the performed motions is not higher than $\pi$ [rad] in one sampling period, $T_s = \frac{1}{100[\text{Hz}]} = 0.01[\text{s}]$. That assumption makes it possible to unwrap the Euler angles and to compute their mean. This mean is wrapped to the usual range of Euler angles again if needed and an average orientation difference expressed using Euler angles is obtained. This average Euler angles parameterization is converted back to a unit quaternion parameterization and forms the constant average misalignment denoted by $\hat{q}_{Average\ misalignment}^{S_{tight}S_{loose}}$. Algorithm 2 describes the computation of this average misalignment step-by-step. The sequential sequences of orientation estimates computed with the EKF from the loosely attached IMU are rotated by this average misalignment and form the previously mentioned $\hat{q}^{NS_{loose}}_{Aligned,t,l}$. The last operation is compactly displayed by

$$\hat{q}^{NS_{loose}}_{Aligned,t,l} = \hat{q}^{NS_{loose}}_{EKF,t,l} \odot \hat{q}^{S_{loose}S_{tight}}_{Average\ misalignment}. \tag{4-5}$$

Note that subsequent aligned orientation estimates of the loosely attached IMU ($\hat{q}^{NS_{loose}}_{Aligned,t,l}$) still express the movements of the loosely attached IMU. Therefore, the notation $(\cdot)^{NS_{loose}}$ is still used here, however, the average orientation of these estimates is aligned with the average orientation of the orientation estimates of the tightly attached IMU. This might be confusing, as the quaternion multiplication seems invalid. Therefore, in this paragraph more detailed information on this is given.

**Algorithm 2:** Removing the average misalignment between orientation estimates of a loosely attached IMU and orientation estimates of the target IMU

```
INPUTS: A tensor of sequential sequences of orientation estimates constructed by the
  EKF, $\hat{q}^{NS_{loose}}_{EKF,t,l}$ and $\hat{q}^{NS_{tight}}_{EKF,t}$.
OUTPUTS: A tensor of aligned sequential sequences of orientation estimates, $\hat{q}^{NS_{loose}}_{Aligned,t,l}$.

1. Obtain the average misalignment between orientation estimates of the loosely attached
   IMU and the target IMU, parameterized with Euler angles.
    (a) Derive the orientation difference between all input and corresponding target
        orientation estimates using the unit quaternion parameterization.
            $\hat{q}^{S_{tight}S_{loose}}_{t} = (\hat{q}^{NS_{tight}}_{EKF,t})^{c} \odot q^{NS_{loose}}_{EKF,t,11}$.
    (b) Convert quaternion differences to the Euler angles parameterization.
    (c) Unwrap all sequential Euler angles.
    (d) Compute the mean of the Euler angles.
    (e) Wrap the mean Euler angles within range.
2. Remove the average misalignment from the orientation estimates of the loosely
   attached IMU.
    (a) Convert the mean Euler angles to the average orientation difference expressed as
        unit quaternion.
            $\hat{q}^{S_{tight}S_{loose}}_{Average\ misalignment}$.
    (b) Remove the average misalignment from the orientation estimates of the loosely
        attached IMU.
            $\hat{q}^{NS_{loose}}_{Aligned,t,l} = \hat{q}^{NS_{loose}}_{EKF,t,l} \odot (\hat{q}^{S_{tight}S_{loose}}_{Average\ misalignment})^{c}$.
```

At inference there is no access to all data from the loosely attached IMU beforehand or during the test. This problem can be solved by post-processing the data, making this algorithm a non-realtime algorithm. In addition, one can not be sure that the test subject will make movements distributed in the same way as the movements in the target dataset used for training were. This means that the performance of this algorithm can only be achieved in theory. In practice, its performance will be lower and probably the best way to determine the constant misalignment for realtime usage is to perform a calibration procedure for this. One could think of a certain stationary stance like suggested in [6] or performing several movements. Because the goal is to use this algorithm as an indication of the theoretical achievable performance without using ML techniques, there will not be any further elaboration on a possible calibration procedure to implement this algorithm for obtaining a realtime baseline performance.

#### Measures to cope with gimbal lock

As one may have noticed, the conversion to Euler angles can cause problems. Like discussed in Section 2-2, the Euler angles representation suffers from gimbal lock. This occurs when $\theta$, the pitch angle, approaches $-\pi$ [rad] or $\pi$ [rad]. In that case, two axes align and the orientation representation loses one degree of freedom, meaning that it does not contain enough information to infer the roll and yaw angle from it. Therefore, a solution is thought of. Since it is assumed that the orientation estimation will be applied for human motion, as outlined in Section 1-1, a simple solution to prevent gimbal lock is to position the sensor in such a way that gimbal lock will not occur due to the constraints of the body. However, this solution does not always work. For example, the arm can move in any direction and it is possible to get near a pitch angle of $-\pi$ [rad] or $\pi$ [rad] then. To cope with this, the assumption made in Section 4-3-1 is reused: the motions for this study will not rotate faster than $\frac{\pi}{0.01}[\frac{rad}{s}]$. Assuming this, the roll and yaw angle can be derived using an interpolation of the angles before and after the gimbal lock.

## 4-4 Removing the absolute heading direction dependency

This section explains more about the second preprocessing step that is designed to improve the orientation estimation using a loosely attached IMU, ML methods and physics-based knowledge. This step resolves the problem associated with the dependence of the direction in which any movement is recorded. As mentioned in Section 4-2, two methods have been designed to remove this dependency. They are discussed in Section 4-4-1 and Section 4-4-2.

### 4-4-1 Augmenting the dataset with different absolute heading directions

The goal of this method is to add data to the training dataset in order to include the same amount of information about the mutual movement between the loosely attached and the target IMU for all possible absolute heading directions. This is achieved by rotating all orientation estimates of the loosely attached IMU and their corresponding orientation estimate of the tightly attached IMU by a given rotation around the z-axis of the navigation frame. The augmented dataset containing input data for the RNN will be denoted by $\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l}$ and the augmented dataset containing the target data will be denoted by $\hat{q}^{NS_{tight}}_{\psi_{invariant},t}$. This way, at inference, movements facing any direction will be recognized by the RNN. For this, the rotation around the z-axis should be discretized. The smaller the discretization step, the more information is included. However, this also will cause a bigger growth of the dataset, due to which one might face the computational limitations of the used hardware. The extra demand for computational resources and time is a disadvantage of this method and has led to the decision to add only data that is rotated by the average absolute heading of the target data. This way the size of the training dataset only doubles but one does get a fair comparison between two datasets that are recorded facing a different absolute heading. A compact representation of the method of augmenting the training dataset in order to add information is presented in Algorithm 3.

**Algorithm 3:** Augment the dataset with data facing different absolute heading directions

INPUTS: A dataset containing sequential sequences of aligned orientation estimates of the loosely attached IMU, $\hat{q}^{NS_{loose}}_{Aligned,t,l}$, and a dataset containing sequential orientation estimates of the tightly attached IMU, $\hat{q}^{NS_{tight}}_{EKF,t}$.

OUTPUTS: An extended dataset containing sequential sequences of orientation estimates of the loosely attached IMU, $\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l}$, and a dataset containing sequential orientation estimates of the tightly attached IMU, $\hat{q}^{NS_{tight}}_{\psi_{invariant},t}$, both facing all added heading directions.

1. Declare the absolute heading angles ($n$) in the navigation frame that should be included.

$$\psi^{N_{rotated\ n}N}_{Extra\ absolute\ heading}.$$

2. Convert the Euler angles representations of the extra absolute heading to unit quaternions, $\phi^{N_{rotated\ n}N}_{Extra\ absolute\ heading}$ and $\theta^{N_{rotated\ n}N}_{Extra\ absolute\ heading}$ are 0.

$$q^{N_{rotated\ n}N}_{Extra\ absolute\ heading}.$$

3. Extend the recorded dataset with the rotated data.

   (a) Rotate the recorded data with $q^{N_{rotated\ n}N}_{Extra\ absolute\ heading}$.

$$\hat{q}^{N_{rotated\ n}S_{loose}}_{Aligned,t,l} = q^{N_{rotated\ n}N}_{Extra\ absolute\ heading} \odot \hat{q}^{NS_{loose}}_{Aligned,t,l} \text{ and}$$
$$\hat{q}^{N_{rotated\ n}S_{tight}}_{EKF,t} = q^{N_{rotated\ n}N}_{Extra\ absolute\ heading} \odot \hat{q}^{NS_{tight}}_{EKF,t}.$$

   (b) Add the extra data to the original datasets to form the extended datasets.

$$\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l} = \begin{bmatrix} \hat{q}^{NS_{loose}}_{Aligned,t,l} \\ \hat{q}^{N_{rotated\ 1}S_{loose}}_{Aligned,t,l} \\ \vdots \\ \hat{q}^{N_{rotated\ n}S_{loose}}_{Aligned,t,l} \end{bmatrix} \text{ and } \hat{q}^{NS_{tight}}_{\psi_{invariant},t} = \begin{bmatrix} \hat{q}^{NS_{tight}}_{EKF,t} \\ \hat{q}^{N_{rotated\ 1}S_{tight}}_{EKF,t} \\ \vdots \\ \hat{q}^{N_{rotated\ n}S_{tight}}_{EKF,t} \end{bmatrix}.$$

### 4-4-2 Removing the absolute heading dependency of the dataset

This section describes another method to make the data invariant to the absolute heading direction. Instead of adding information to the dataset, it is also possible to actually get rid of the information that makes the data dependent on the absolute heading direction.

The available data describes the orientation of the sensors with respect to the navigation frame $N$: $\hat{q}^{NS_{loose}}_{Aligned,t,l}$ and $\hat{q}^{NS_{tight}}_{EKF,t}$. The absolute heading of a sensor is defined as the angle of rotation around the z-axis in the navigation frame. In order to remove this information from the data, it should be isolated first. The Euler angles parameterization, discussed in Section 2-2-2, suits this goal best as it already uses the angles of rotation around the x, y and z-axis in its description. To make the yaw angle ($\psi$) independent from the other angles ($\phi$ and $\theta$) and coincide with the heading in $N$, the order of rotation around the axes is important here. There are in total 12 different valid sequences of rotations [38]. The sequence of rotation that is chosen to rotate from the reference frame to the sensor frame, first rotates around the z-axis with the corresponding yaw angle ($\psi$). Since this is the first rotation, the z-axis of both sensor frame and navigation frame align. The rotated frame is then rotated around its new y-axis by the pitch angle ($\theta$). Lastly, the frame is rotated around its new x-axis by the roll angle $\phi$ to align the orientation of the frame with the orientation of $S$ which the Euler angles parameterization described in the first place. This sequence of rotation is denoted by the Z-Y'-X" sequence. The rotation defining the whole rotation or orientation can be written as the three consecutive rotations around the orthogonal axes [38, 46]

$$R(\phi, \theta, \psi)^{NS} = R^Z_\psi R^Y_\theta R^X_\phi . \tag{4-6}$$

Further details on this rotation matrix and the conversion from Euler angles to the rotation matrix can be found in Appendix A-3.

Because the average misalignment between the sensors is already removed, one can expect that the last sample of every input sequence has the absolute heading that is the closest to the absolute heading of the corresponding target orientation because they are samples of the same time step. Therefore, in order to remove the absolute heading direction from the input and target data, this angle is subtracted from the whole input sequence and corresponding target sample. This way, all data has an absolute heading direction or yaw angle ($\psi$) of around 0 [rad]. The deviation from this contains useful information about the mutual movement between the sensors and should not be removed.

To recover the absolute heading of the target sample, one should add the angle which was initially subtracted from the input sequence to the absolute heading of the orientation estimated by the RNN.

An overview of the implementation of the second method to the problem of the absolute heading dependency is given in Algorithm 4.

**Algorithm 4:** Remove the dependency on the data on the absolute heading direction

INPUTS: A dataset containing sequential sequences of aligned orientation estimates of the loosely attached IMU, $\hat{q}^{NS_{loose}}_{Aligned,t,l}$, and a dataset containing sequential orientation estimates of the tightly attached IMU, $\hat{q}^{NS_{tight}}_{EKF,t}$.

OUTPUTS: A dataset containing sequential sequences of aligned orientation estimates of the loosely attached IMU, $\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l}$, and a dataset containing sequential orientation estimates of the tightly attached IMU, $\hat{q}^{NS_{tight}}_{\psi_{invariant},t}$, both from which the absolute heading dependency is removed.

1. Convert all orientation estimates in both datasets to an Euler angles parameterization.
2. Subtract the yaw angle of the last samples of the aligned orientation estimate sequences of the loosely attached IMU, $\hat{\psi}^{NS_{loose}}_{Aligned,t,11}$ from the yaw angles of all orientation estimates contained in both datasets, $\hat{\psi}^{NS_{loose}}_{Aligned,t,l}$ and $\hat{\psi}^{NS_{tight}}_{EKF,t}$.
3. Wrap the resulting yaw angles to their range.
4. Convert all orientation estimates parameterized by Euler angles back to unit quaternions.

$$\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l} \text{ and } \hat{q}^{NS_{tight}}_{\psi_{invariant},t}.$$

5. After training the network with this data, the yaw angles of the last samples of the sequential sequences with aligned orientation estimates of the loosely attached IMU, $\hat{\psi}^{NS_{loose}}_{Aligned,t,11}$, should be restored to the yaw angles of the sequential orientation estimates of the RNN to obtain the sequential orientation estimates of the tightly attached IMU, $\hat{q}^{NS_{tight}}_{EKF,t}$.

This second step of preprocessing the data can be found after the first step, which removes the average orientation difference from the input. The reader is referred back to Section 4-2, where Figure 4-5 can be found, which describes the model with all three preprocessing steps.

## 4-5 Removing ambiguity and discontinuities from unit quaternions

In this section, the last step, which preprocesses the data before feeding it to the RNN, is addressed. As discussed in Section 3-3 and Section 3-4, the data should be unique and sequential data should posses a certain level of continuity. Here, first a solution to the problem of the ambiguity of unit quaternions together with a method that will lower the amount of needed data, is presented. Next, it is explained how subsequent samples in sequences are processed to stimulate the capability of detecting time related behavior in the process.

### 4-5-1 Ambiguity in data

A simple solution to the problem of possible ambiguity, shown in Section 3-3, is to shape the loss function that is used for training the network. When it is designed such that it is indifferent to the two possible orientation representations, the network can decide by itself which of these is easier to estimate given the input data. This adjustment can be implemented by comparing the estimate of the network to the target sample $q$ but also to its counterpart -$q$. Taking the representation which yields the minimum loss will solve the problem of ambiguity. The performance measures presented in (6-3) and (6-4) already incorporate this indifference to both orientation representations which describe identical orientations. There is, however, a disadvantage to this solution. It is likely that the network will learn to estimate both orientation representations, dependent of the sign of the input representation. Both positive and negative inputs will be present in the data, and thus also both positive and negative outputs will be estimated. This requires twice as much data.

One commonly used way to overcome the issue of the non-uniqueness of unit quaternions is to enforce the first component of all unit quaternions, both input and target data, to be non-negative. This is a simple method called 'canonizing' and is easy to implement. Despite that this is a simple method, no studies are known which use unit quaternions and ML methods and apply this method. Often the data is left untreated and only a loss function that is indifferent to a positive or negative unit quaternion describing a certain orientation is introduced [1, 47].

Applying these two measures solves the problem of ambiguity and reduces the amount of data needed. Given a limited amount of data to train the network on, it is expected that the performance of the network is higher than when this step is not applied. The argument for this expectation lies in the fact that the search space is reduced by a factor 2.

### 4-5-2 Sequential time-dependencies

In order to stimulate the ability of detecting time dependencies, it is ensured that all data samples in the input sequences have minimal Euclidean distance with respect to the previous sample. This way the data samples of a sequence start with a unit quaternion having positive first component, and successive samples have minimal Euclidean distance. One could say that there are no 'jumps' between data points in a sequence, apart from the effects of discretization of course. In order to apply this, the Euclidean distance ($\delta$) from one data point ($q_A$) to the next data point ($q_B$) is computed. Because the orientation representation $q$ represents the same orientation as $-q$, two distances are computed for this. The Euclidean distance from data point ($q_A$) to data point ($q_B$) is computed using

$$\delta = ||q_A - q_B||_2. \tag{4-7}$$

For data point $q_B$, the unit quaternion variant ($q$ or $-q$) is determined by selecting the one that has the smallest Euclidean distance with respect to data point $q_A$. This way of treating the data was inspired by [24]. It is supposed to remove discontinuous jumps between consecutive data points. This minimal Euclidean distance between consecutive samples is a naturally inherited characteristic of unit quaternions estimated by an EKF. However, due to canonizing the data, there could be a jump between components of the first (canonized) data sample and components of the rest of the sequence. This problem does not occur frequently but it is taken care of this way if it does.

A compact representation of this step is presented in Algorithm 5.

**Algorithm 5:** Removing ambiguity and discontinuities from unit quaternions

Inputs: A dataset containing sequential sequences of heading independent orientation estimates of the loosely IMU, $\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l}$, and a dataset containing sequential orientation estimates of the tightly attached IMU, $\hat{q}^{NS_{tight}}_{\psi_{invariant},t}$.
Outputs: Unique and continuous sequential sequences of orientation estimates of the loosely attached IMU, $\hat{q}^{NS_{loose}}_{Unique,t,l}$, and Unique sequential orientation estimates of the tightly attached IMU, $\hat{q}^{NS_{tight}}_{Unique,t}$.

```
1. Canonize all orientation estimates of both input and output.
   If $\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l} < 0$ Then
       $\hat{q}^{NS_{loose}}_{Unique,t,l} = -\hat{q}^{NS_{loose}}_{\psi_{invariant},t,l}$,
   Else
       $\hat{q}^{NS_{loose}}_{Unique,t,l} = \hat{q}^{NS_{loose}}_{\psi_{invariant},t,l}$.
   If $\hat{q}^{NS_{tight}}_{\psi_{invariant},t} < 0$ Then
       $\hat{q}^{NS_{tight}}_{Unique,t} = -\hat{q}^{NS_{tight}}_{\psi_{invariant},t}$,
   Else
       $\hat{q}^{NS_{tight}}_{Unique,t} = \hat{q}^{NS_{tight}}_{\psi_{invariant},t}$.
2. Ensure all sequences of the input are continuous.
   For l = 2, ... 11 do
   (a) Compute the Euclidean distance between orientation sample $\hat{q}^{NS_{loose}}_{Unique,t,l-1}$ and $\hat{q}^{NS_{loose}}_{Unique,t,l}$.
           $\delta_{pos,t} = ||\hat{q}^{NS_{loose}}_{Unique,t,l-1} - \hat{q}^{NS_{loose}}_{Unique,t,l}||_2$,
           $\delta_{neg,t} = ||\hat{q}^{NS_{loose}}_{Unique,t,l-1} + \hat{q}^{NS_{loose}}_{Unique,t,l}||_2$.
   (b) If $\delta_{neg,t} < \delta_{pos,t}$ Then
           Replace orientation sample $\hat{q}^{NS_{loose}}_{Unique,t,l}$ by $-\hat{q}^{NS_{loose}}_{Unique,t,l}$.
```

This step is implemented just before the RNN, so that the other preprocessing steps can not spoil the point of this step by altering the data. Figure 4-5 shows where the preprocessing step is applied.

## 4-6 Using physics-based input features as opposed to inertial data

This section describes how the belief is verified that using physics-based input features as opposed to inertial data is beneficial for the problem. So far, the proposed input and output data has been orientation estimates computed by an EKF and preprocessed with three steps. This way, knowledge about physics in the form of a physics-based method is used. In order to verify the belief that this is beneficial, the orientation estimates forming the input to the RNN are replaced by the IMU measurement data. To compute the orientation estimates, only this inertial data was used. Therefore, it should be possible to achieve equal performance with a network using the inertial input features, if the network is capable of estimating the orientation estimates and then map these to the target orientation estimates. This is considered a bigger task than merely learning the latter and, therefore, it is expected that when using the same amount of data and the same network, the performance will be worse. If true, the choice to use physics-based knowledge made at the beginning of the study, is considered beneficial.

### 4-6-1 Feature scaling the inertial data

When replacing the input orientation estimates by the gyroscope, accelerometer and magnetometer sensor data, one loses the advantages of the unit quaternion representation. Specifically, the new input data is not scaled between -1 and 1 anymore. Presenting a network with input features that have different scales can cause the network not to weight them equally. This means that the input features that are distributed on a bigger scale become the dominant source of information for the network. Scaling the features to a given range is beneficial for the convergence rate of the network [32]. Leaving out this operation sometimes even leads to a network that fails to learn. Two common ways to scale input data are normalizing and standardizing [1, 48]. Standardizing the data means centering the data around 0 with a standard deviation of 1, assuming the data fits a Gaussian distribution. Normalizing the data means rescaling it to a range of 0 to 1. For this method, the gyroscope data will have its bias already removed as described in the calibration procedure Section 4-1-1. Because of this, possible non-zero mean contains information about the movement of the sensor. Therefore, it is chosen to only normalize the data and rescale it to a range from -1 to 1 using

$$\begin{aligned}
y^{S_{loose}}_{\omega_{rescaled},t} &= 2\frac{y^{S_{loose}}_{\omega,t} - y^{S_{loose}}_{\omega_{min}}}{y^{S_{loose}}_{\omega_{max}} - y^{S_{loose}}_{\omega_{min}}} - 1, \\
y^{S_{loose}}_{a_{rescaled},t} &= 2\frac{y^{S_{loose}}_{a,t} - y^{S_{loose}}_{a_{min}}}{y^{S_{loose}}_{a_{max}} - y^{S_{loose}}_{a_{min}}} - 1 \text{ and} \\
y^{S_{loose}}_{m_{rescaled},t} &= 2\frac{y^{S_{loose}}_{m,t} - y^{S_{loose}}_{m_{min}}}{y^{S_{loose}}_{m_{max}} - y^{S_{loose}}_{m_{min}}} - 1.
\end{aligned} \tag{4-8}$$

Here, the subscript $_{min}$ and $_{max}$ denote the minimum and maximum value of the data used for training. The subscript $_{rescaled}$ denotes that the data is rescaled.

# Chapter 5

# Experiments

*This chapter first describes the hardware that was used to implement the methods. Next, it discusses the experiments that are designed to assess and reveal the advantages of the preprocessing steps presented in this study. To this end, a simulated setup is designed, as well as four experimental setups. They are discussed in Section 5-2 and Section 5-3. Lastly, this chapter describes details about the different datasets that are recorded with these setups.*

## 5-1 Hardware

In order to record the inertial data, two IMUs were used. The wireless Xsens 'MTw Awinda' is deployed for this purpose [49]. It offers highly accurate time-synchronized measurements and is very easy in use. The standard software that Xsens provides, MT Software Suite, is used to record the inertial data. As a sampling frequency $f_s$, 100 Hz is used throughout the whole study.

In order to implement and train the RNN, Python (version 3.8.5) was used together with the Tensorflow package (version 2.2.0) and the Keras package for deep learning (version 2.4.3). All coding and computations have been performed on a HP ZBook Studio G4 with an Intel(R) Core(TM) i7-7700HQ (8 CPUs) processor running at 2.80GHz, a NVIDIA Quadro M1200 GPU and 8 GB RAM.

Using this hardware setup, it takes on average 7.5 hours to train 20 networks in order to compute a single result. This illustrates the limitations which were imposed by the hardware during this study.

## 5-2 Simulated setup

In order to create a so-called 'worst-case' scenario to show the essence of the third step, a simulated setup was designed. It comprises a disc rotating around one axis with an IMU attached to it at the center. In this experiment the z-axis is used. A rod connected to this disc moves a lever in different orientations around a hinge joint. Another IMU is attached to the rotation point of this lever and its rotation range is limited by the x-axis. The IMU on the disk generates both orientation representations $q$ and $-q$ due to its full rotations. The setup is used to generate synthetic gyroscope data from which orientation estimates are computed using the EKF. A picture of the setup is shown in Figure 5-1.

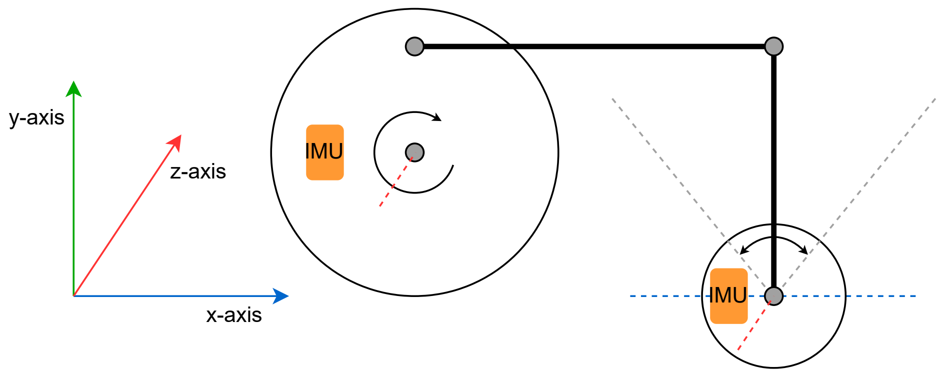


**Figure 5-1:** The setup of the simulated experiment designed to show the benefits of preprocessing step 3.

## 5-3 Experimental setups

Then, there are four experimental setups that use real IMUs to collect inertial measurements. The degree of movement that is possible between the input and output sensors differ, so that the proposed model can be tested on different setups with loosely attached IMUs. The first setup has no movements between the two sensors. A picture of the actual setup is shown in Figure 5-2. With this setup, the average misalignment is the only error source. It will be used to verify the working of the first preprocessing step.

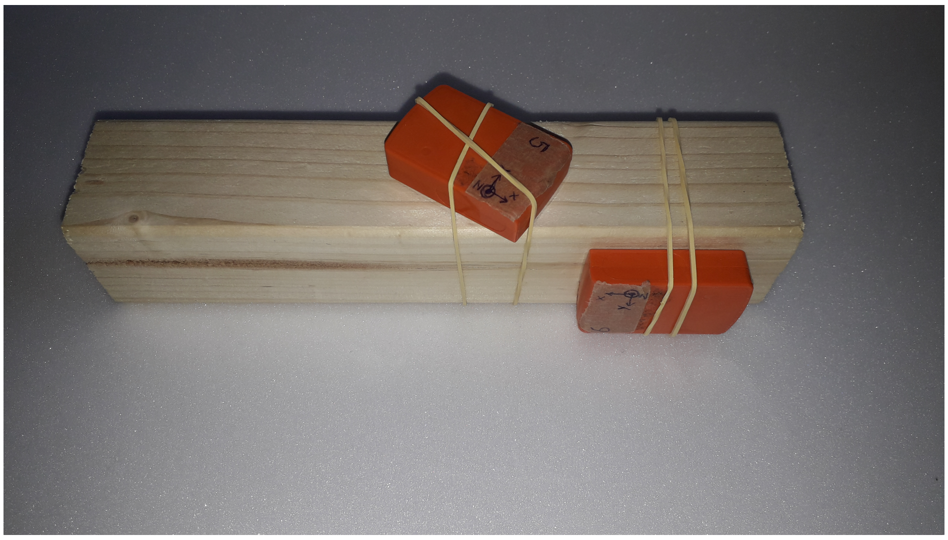


**Figure 5-2:** The setup of the first experiment designed to show the benefits of preprocessing step 1.

The second experiment involves two solid pieces (wood), which are connected via flexible tubes. The sensors attached to the solid pieces can move differently from each other but are dependent due to the connection of the tubes. A picture of the setup is shown in Figure 5-3.

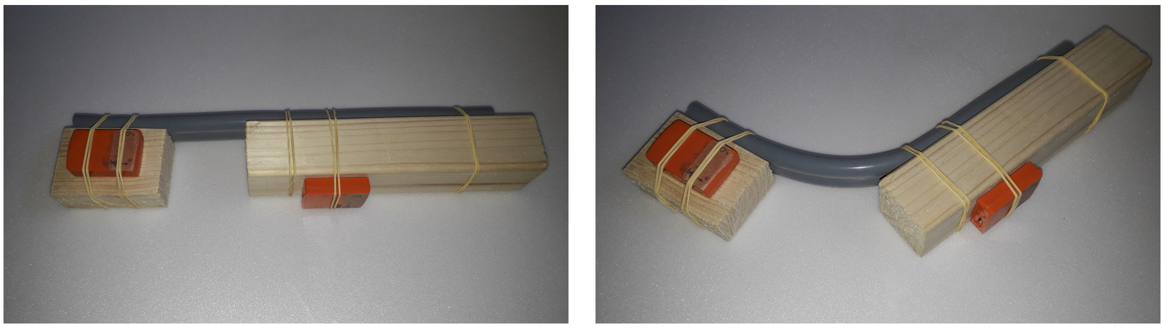


**Figure 5-3:** The setup of experiment 2 allows mutual movement between the sensors.

The third experiment also allows mutual movement between the input and output sensor but is constructed differently. Instead of connecting the two sensors using a flexible tube, an airbag is used. This resembles the situation of using wearable IMUs better. A picture of the setup is shown in Figure 5-4.

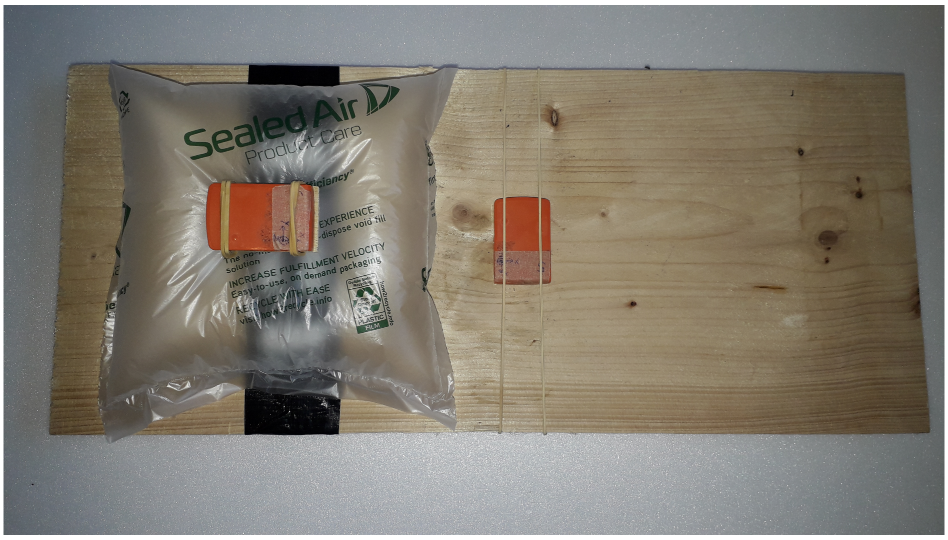
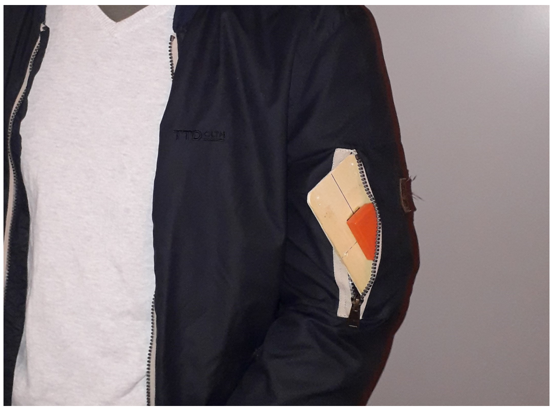
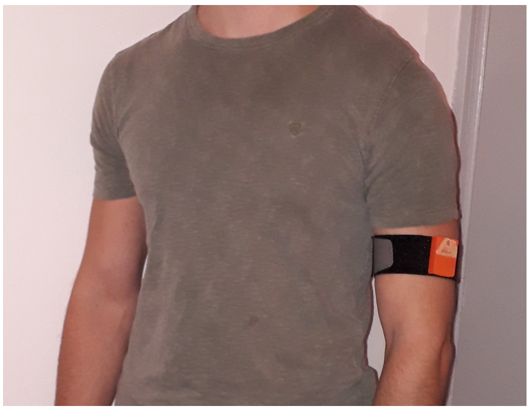


**Figure 5-4:** The setup of experiment 3 allows mutual movement between the sensors.

The last experimental setup involves a real human, wearing a loosely attached IMU in the pocket of his jacket. The target IMU is tightly attached to the upper arm using a rubber band. Two pictures illustrating the setup are shown in Figure 5-5.



**(a)** Imitation of a wearable IMU in a phone.



**(b)** An IMU tightly strapped to the body.

**Figure 5-5:** The setup of experiment 4 involving a human and a loosely attached IMU.

In all experimental setups described above, the loosely attached IMU will serve as the input sensor providing inertial input data. The true orientation data is constructed by the orientation estimates of the EKF which uses the inertial measurements from the tightly attached sensor. The orientation estimates computed by the EKF are successfully validated with the orientation estimates generated by the fusion algorithm of the Xsens sensors.

## 5-4 Datasets

This section describes details on the recorded datasets to reveal the working of the designed preprocessing steps.

The simulated setup, described in Section 5-2, is used to generate two datasets. Both datasets do not contain an average misalignment, neither have they different absolute heading directions. This way, the third preprocessing step can be solely employed in order to demonstrate its need. One of the datasets is used to validate the performance of the trained RNN. It contains both orientation representations $q$ and $-q$. The other dataset is used to train the RNN. It also contains both orientation representations $q$ and $-q$, however, not all RNNs that are trained using this dataset use the whole dataset. Multiple RNNs are trained and each of them uses a larger part of the dataset.

Using the first experimental setup, a dataset for training and two datasets for validation are recorded. The data from this experiment contains a constant misalignment between the input and target IMU. Therefore, this setup suits revealing the working of preprocessing step 1 well. The first dataset that is used to validate the performance is denoted by *Testset 1A* and contains the same average misalignment as the dataset used for training. The second dataset used for validating the performance is denoted by *Testset 1B* and contains a different average misalignment than the training data.

Recordings using the second experimental setup have provided three datasets. One which is used for training, and two that are used to validate performance. The dataset used for training contains an average misalignment, a process part and is recorded facing in one absolute heading direction. The datasets used for validation both contain the same average misalignment and also contain mutual movement between the input and target IMU, the process. The first dataset used for validation is referred to as *Testset 2A* and is recorded in the same absolute heading direction as the dataset used for training. The second dataset used for validation, referred to as *Testset 2B*, is recorded facing a different absolute heading direction.

With the third experimental setup, also three datasets are recorded. The first dataset is used for training and the others are used for validation of the preprocessing steps. All three datasets contain an equal average misalignment and mutual movement between the input and target IMU, the process. The training data is recorded facing one absolute heading direction. One of the datasets used for validation, called *Testset 3A*, contains data that is recorded in the same absolute heading direction as the training data. The other dataset used for validation the steps is recorded facing a different absolute heading direction.

The fourth experimental setup has provided six datasets which are split in two groups. The first group is data describing all kinds of motions performed by the arm. Therefore, this data is very diverse. The second group is data that only describes motions of the arm drawing circles in the air. Again, each group contains a dataset used for training which is recorded facing one absolute heading direction. Then, each group contains a dataset used for validation, with data that is recorded facing the same absolute heading direction as the training data. These datasets are referred to as *Testset 4A* for the dataset of the group containing the diverse data and *Testset 4C* for the dataset of the group containing data describing the motions of an arm drawing circles. Then, both groups contain another dataset which is used for validation and is recorded facing a different absolute heading direction. The dataset of the first group is referred to as *Testset 4B* and the dataset of the second group is referred to as *Testset 4D*.

Chapter 6

# Results

*In order to compare two sets of data and obtain a measure of the error between them, multiple performance measures can be used. This chapter starts off by addressing three performance measures used in this study. The rest of this chapter entails a quantitative overview and a discussion on the results of all preprocessing steps.*

## 6-1 Performance measures

This section discusses three different performance measures. First, the root mean squared error (RMSE) value of the Euler angles parameterization of the quaternion difference is discussed. After that, the mean Euclidean distance between estimated and target quaternions is explained and lastly the mean angle of rotation described by the quaternion difference as a performance measure is discussed. The orientation differences that are compared differ for each result. Therefore, in this section, the compared orientation representations parameterized as unit quaternions are denoted by $q_{A,t}^{NS_1}$ and $q_{B,t}^{NS_2}$.

The quaternion difference between two orientation representation parameterized by unit quaternion is determined by

$$\hat{q}_t^{S_1S_2} = (q_{A,t}^{NS_1})^c \odot \hat{q}_{B,t}^{NS_2}. \tag{6-1}$$

Note that $q_{A,t}^{NS_1}$ is considered to be the true orientation. If the estimated orientations, $\hat{q}_{B,t}^{NS_2}$, perfectly match the true orientations, then the orientation difference between them, $\hat{q}_t^{S_1S_2}$, should be all zeros.

Converting this to Euler angles using Appendix A, gives the Euler angles which define how the estimated orientations are oriented with respect to the target sensor orientations. Then, to compute a RMSE value for each Euler angle, the formula,

$$\begin{bmatrix} \hat{\phi} \\ \hat{\theta} \\ \hat{\psi} \end{bmatrix}_{RMSE}^{S_1S_2} = \sqrt{\sum_{t=1}^{N} \frac{\left(\left(\begin{bmatrix} \hat{\phi} & \hat{\theta} & \hat{\psi} \end{bmatrix}_t^{S_1S_2}\right)^T\right)^2}{N}}, \tag{6-2}$$

is applied where N is the total amount of samples. This definition of the error is only applicable if the quaternion difference is small enough and does not approach gimbal lock. This performance measure gives the most information about the performance as it is subdivided over three angles of rotation which are intuitive in the 3D space. However, due to averaging these angles obtained from individual samples that (can) have differently oriented coordinate systems, as well as the error being distributed over three angles, the final performance can be a bit misleading from this measure. For this reason it is best to look at performance differences as a whole rather than looking at individual angles.

An illustrative example of a simple case where this issue occurs is given in Appendix D.

The next and most dominantly used performance measure in this study is the Euclidean distance between two quaternions [1],

$$\hat{\delta}_{mean} = \frac{1}{N}\sum_{t=1}^{N} \min(||q_{A,t}^{NS_1} - \hat{q}_{B,t}^{NS_2}||_2, ||q_{A,t}^{NS_1} + \hat{q}_{B,t}^{NS_2}||_2). \tag{6-3}$$

This performance measure takes into account how far off each element of the quaternion is. Also, it takes into account that the representation $q$ is equal to $-q$ and uses the variant which has minimal distance. This performance measure is used as the loss measure during training of the neural network.

The last performance measure is the angle of rotation between two unit quaternions. To compute this angle of rotation $\alpha$,

$$\hat{\alpha}_{mean} = \frac{2}{N}\sum_{t=1}^{N} \arccos(|q_{A,t}^{NS_1} \cdot \hat{q}_{B,t}^{NS_2}|), \tag{6-4}$$

is used [1, 47]. Note that $\cdot$ is a dot product between the two quaternions. This performance measure is also indifferent to the two equal orientation representations $q$ and $-q$. The performance measure could be used as loss measure for training the neural network as well.

## 6-2 Results and discussion

In this section results of the different methods presented in Chapter 4 are presented and discussed. In response to Section 3-2, the learning curves of the trained networks discussed in this section have been examined. They have shown no signs of overfitting and, therefore, no regularization methods have been applied.

### 6-2-1 Removing ambiguity and discontinuities from unit quaternions

This section examines the working of preprocessing step 3. This step is validated first because it will be applied for all other tests which validate the working of the other preprocessing steps. This is done because all other tests benefit from a need for less data. An overview of the used model here is shown in Figure 6-1.

To validate this step, datasets generated by the simulated setup, presented in Section 5-2, are used. After computing the orientation estimates using the EKF of the simulated IMU

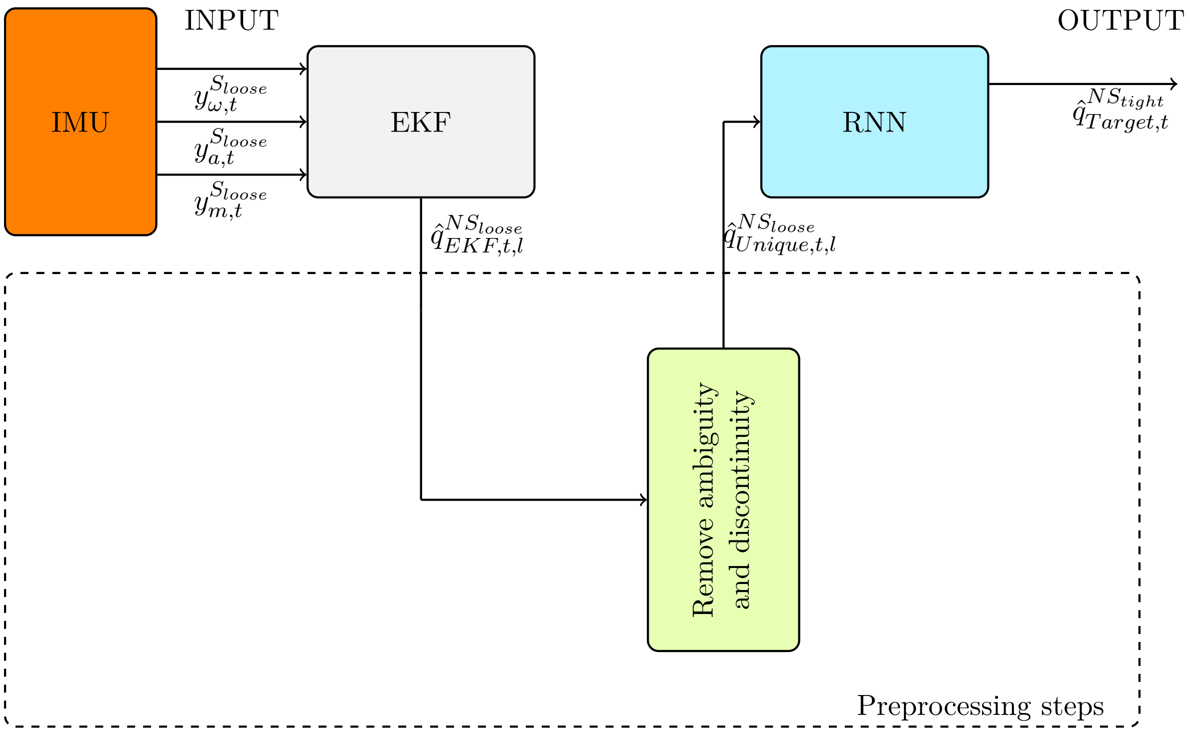

INPUT
OUTPUT
IMU
EKF
RNN
$y^{S_{loose}}_{\omega,t}$
$y^{S_{loose}}_{a,t}$
$y^{S_{loose}}_{m,t}$
$\hat{q}^{NS_{loose}}_{EKF,t,l}$
$\hat{q}^{NS_{loose}}_{Unique,t,l}$
$\hat{q}^{NS_{tight}}_{Target,t}$
Remove ambiguity and discontinuity
Preprocessing steps

**Figure 6-1:** Basic structure of the hybrid model with preprocessing step 1 included.

measurements, several networks are trained on a given number of samples of the trainset. Their performance on the validation set is denoted in Figure 6-2. The results are shown by a plot of the network performance expressed by (6-3) plotted against the amount of data points the network trained on. The performance is computed by averaging the performance of 20 differently initialized networks.

By inspecting the plot shown in Figure 6-2, it becomes clear that the expectation about the required amount of data is correct. When preprocessing the data using Algorithm 5, the error on the validation set decreases with fewer data points. In fact, adding information about the orientation representation $-q$ hardly improves the performance since optimal performance is already achieved. Without preprocessing the data with Algorithm 5, the error of the network on the validation set is high when trained on small amounts of data points. If the training data, however, contains both information about orientation representation $q$ and $-q$, the error decreases and equal performance is achieved.

For this reason, in all other experiments the applied loss function is indifferent to the non-uniqueness of unit quaternions and the data is preprocessed according to Algorithm 5.

### 6-2-2 Removing the average misalignment

In this section the working of Algorithm 2, removing the average orientation difference, is examined. This is the first preprocessing step that was designed in this study. The baseline performance using Algorithm 2, which was presented in Section 4-3-1, removes the average misalignment from the data and will often set a better baseline performance than when

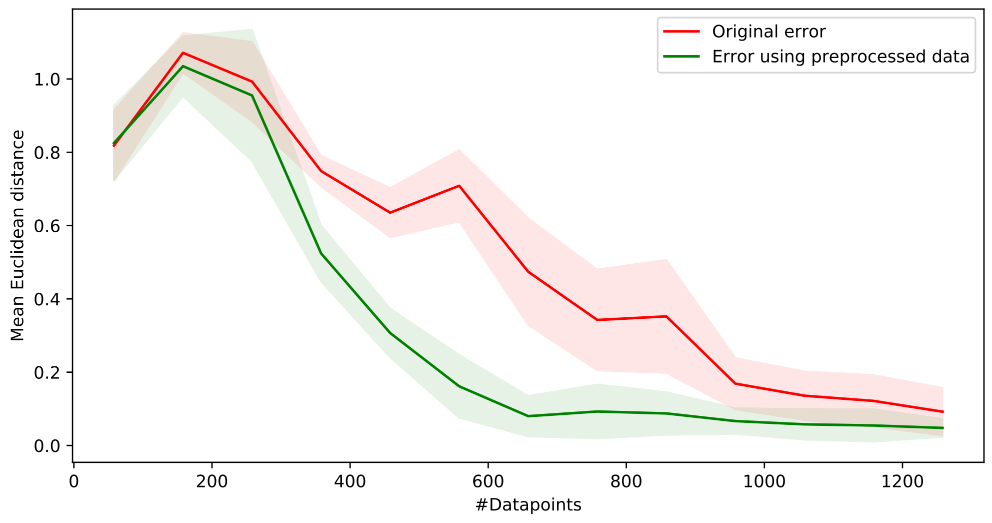


**Figure 6-2:** The average performance of 20 trained networks plotted against the number of data points.

ignoring the misalignment. In addition, applying this step before training a network, makes the network invariant to differently oriented attachments. As already mentioned in Section 4-3, this method is designed to serve two purposes. The results presented in this section will address the two purposes in order. Firstly, the partition of the error caused by a misalignment is identified. Secondly, it is verified that a trained network using this algorithm is robust to misaligned attachments.

#### Results on the identification of the partition of the error due to misalignment

To show results corresponding to this goal, *Testset 1A* and *Testset 2A*, described in Section 5-4, are used. Both contain errors caused by a misalignment between the sensors. The difference between the two datasets is that the sensors generating the data contained in *Testset 2A* are able to move differently from each other. Therefore, mutual movement between the sensors, a process, is also a part of the error between input and target data. For this cause, it is expected that a part of the error will remain after applying the method. Table 6-1 displays the error values of the baseline performance when ignoring the misalignment between the loosely attached and target sensor, as well as the baseline performance using Algorithm 2 which aligns the two sensors first. The actual percentages of partitions of the error derived from the data differ for every experiment and dataset. The results here do show that and when the use of Algorithm 2 is beneficial.

With these results it is possible to identify how big the part of the error was due to an average misalignment. Expected is that for *Testset 1A*, this is the main part. The error in *Testset 2A*, however, will also be composed of a part that is due to mutual movement between input and target sensor. The baseline performance without using Algorithm 2 indicates what the entire error is before removing the average misalignment or applying the RNN.

| | | $\hat{\phi}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\theta}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\psi}_{RMSE}$ [$10^{-3}$ rad] | Mean Euclidean distance, $\hat{\delta}_{mean}$ [$10^{-3}$] | Mean angle of rotation, $\hat{\alpha}_{mean}$ [$10^{-3}$ rad] |
|---|---|---|---|---|---|---|
| Baseline performance, using Algorithm 5 | *Testset 1A* | 1576 | 31.0 | 2470 | 1238 | 2670 |
| | *Testset 2A* | 1655 | 333 | 128 | 817 | 1685 |
| Baseline performance, using Algorithms 2 & 5 | *Testset 1A* | 22.0 | 25.0 | 32.0 | 21.3 | 42.5 |
| | *Testset 2A* | 114 | 267 | 135 | 153 | 306 |

**Table 6-1:** The error values before and after removing the misalignment.

For *Testset 1A*, the results using untreated data are compared to the results using data of which the average misalignment was removed by Algorithm 2. From this the conclusion can be drawn that 98.28% of the error was composed by an average misalignment. Figure 6-3 gives a clear overview of this error distribution. To perform calculations like this, the mean Euclidean distance is used.

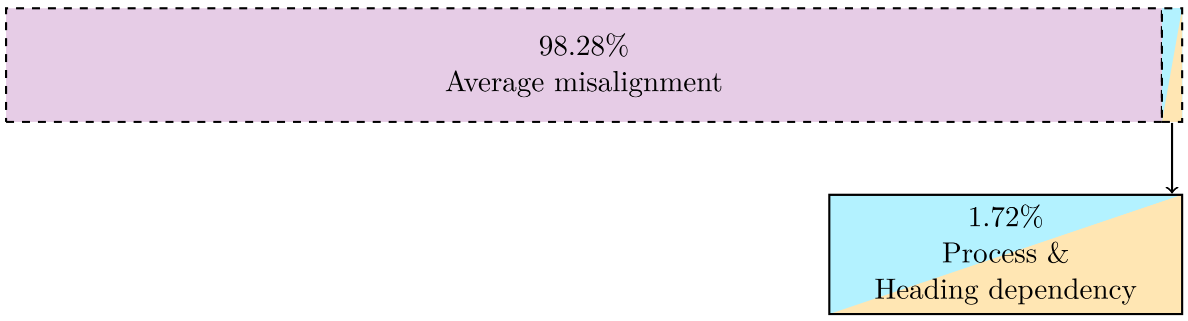


**Figure 6-3:** The error decomposition of *Testset 1A*.

For *Testset 2A*, the results from Table 6-1, yield a different error decomposition. For 81.27%, the error in the data can be contributed to a misalignment in the attachment. A visualization is shown in Figure 6-4.

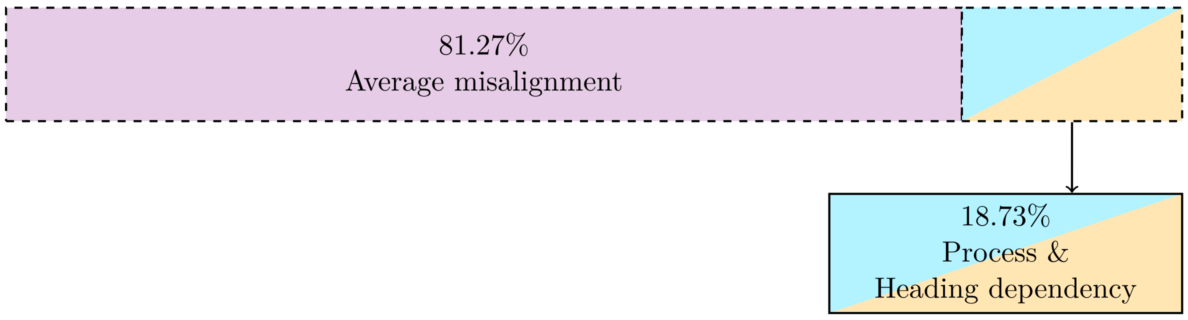


**Figure 6-4:** The error decomposition for *Testset 2A*.

These results are in line with the expectation as only *Testset 2A* are able to move differently from each other.

### Results on the invariance of a network to misalignment

The model displayed in Figure 6-5 is used to be able to identify the working of this purpose of Algorithm 2.

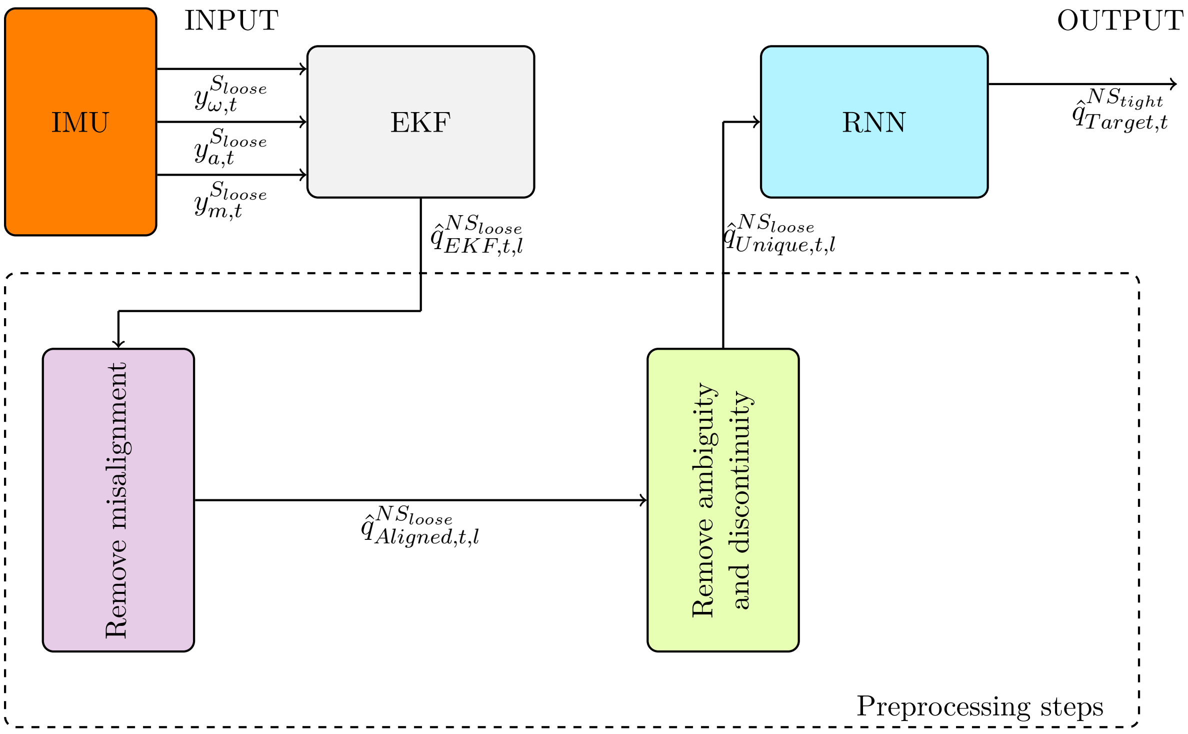

IMU
INPUT
$y^{S_{loose}}_{\omega,t}$
$y^{S_{loose}}_{a,t}$
$y^{S_{loose}}_{m,t}$
EKF
$\hat{q}^{NS_{loose}}_{EKF,t,l}$
RNN
OUTPUT
$\hat{q}^{NS_{tight}}_{Target,t}$
$\hat{q}^{NS_{loose}}_{Unique,t,l}$
Remove misalignment
$\hat{q}^{NS_{loose}}_{Aligned,t,l}$
Remove ambiguity and discontinuity
Preprocessing steps

**Figure 6-5:** Basic structure of the hybrid model with preprocessing steps 1 and 3 included.

Two neural networks are trained. For training and testing the second network, Algorithm 2 is applied to the data. The first network will use data to which Algorithm 2 is not applied and thus this data still contains misalignment between the input and output data. After training the networks, their performance is compared using two different datasets. *Testset 1A* contains the same misalignment as the network was trained on. *Testset 1B*, however, contains a different misalignment. This could occur easily in practice as described in Section 4-2 and the results are expected to deteriorate if not taken care of. For this reason, it is expected that the second network is robust to the differently misaligned data and will have a low error on *Testset 1B* compared to the first network. The data fed to the first network still contains the different misalignment for *Testset 1B*. All data used to compute these results are generated by experimental setup 1. This setup is sufficient to show the robustness of the network to misaligned attachments.

Because at training, the weights of a neural network are initialized with random values, one could get a rather biased indication of the performance of this preprocessing step. To overcome this problem, 20 differently initialized networks are trained for every trainset. From the results of these networks, the mean and the standard deviation are computed. The results shown in Table 6-2 first show the mean error followed by the standard deviation around this mean.

Next, the importance of removing the average misalignment before training a network or

| | | $\hat{\phi}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\theta}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\psi}_{RMSE}$ [$10^{-3}$ rad] | Mean Euclidean distance, $\hat{\delta}_{mean}$ [$10^{-3}$] | Mean angle of rotation, $\hat{\alpha}_{mean}$ [$10^{-3}$ rad] |
|---|---|---|---|---|---|---|
| RNNs, using Algorithm 5, preserving misalignment | *Testset 1A* | 22.4 $\pm$ 3.62 | 23.7 $\pm$ 5.53 | 27.1 $\pm$ 4.7 | 19.1 $\pm$ 3.16 | 38.2 $\pm$ 6.32 |
| | *Testset 1B* | 3092 $\pm$ 5.05 | 36.9 $\pm$ 5.06 | 303 $\pm$ 5.93 | 1396 $\pm$ 1.93 | 3089 $\pm$ 5.37 |
| RNNs, using Algorithms 5 & 2, removing misalignment | *Testset 1A* | 20.9 $\pm$ 4.03 | 21.2 $\pm$ 3.61 | 26.8 $\pm$ 5.53 | 17.5 $\pm$ 2.56 | 35.0 $\pm$ 5.12 |
| | *Testset 1B* | 30.3 $\pm$ 5.71 | 35.2 $\pm$ 8.52 | 31.3 $\pm$ 9.07 | 26.0 $\pm$ 3.68 | 52.0 $\pm$ 7.37 |

**Table 6-2:** The average error of 20 trained RNNs on *Testset 1A* and *Testset 1B*.

inferring orientation estimates is identified. For this the results in Table 6-2 are used. The observations for the first RNN, for which the data was untreated, are denoted below.

1. The conclusion can be drawn based on results of *Testset 1A*, that a network is able to learn a constant misalignment rather well. This is expected as it is regarded as a simple task. Besides that, this is not the result why this test was conducted and it is left alone.
2. The performance deterioration is striking when *Testset 1B*, a dataset with a different average misalignment, is applied to the trained network. The performance is $73.1\times$ as bad for this different misalignment (using the mean Euclidean distance again). This illustrates the problems that can occur if the data is left untreated.

In order to test the robustness to different misaligned attachments, the second RNN was trained on data of which the average misalignment was removed. The possible non-zero average misalignment is removed by applying Algorithm 2 to the training data and test data before using it for the network. The results of *Testset 1B* on the network are much better than before removing the misalignment. In addition, the performance of *Testset 1A* does not seem to suffer from the removal of the misalignment. These results prove the necessity of applying Algorithm 2 before using any data on a network. For the rest of the tests that will be conducted, this step will be applied.

Another thing that can be noticed is that the performance measures mean Euclidean distance and the mean angle of rotation follow the same trend. If the one of them increases then the other also increases. The other experiments will confirm this and it serves as an indication that they both measure something that is related, which is expected because they both should give an impression of the performance, of how close the estimated data is to the target data.

### 6-2-3 Removing the absolute heading direction dependency

In order to examine the working of Algorithm 3 and 4, preprocessing step 2 which remove the absolute heading dependency, the model shown in Figure 4-5 is employed. Datasets *Testset 2A* and *Testset 2B* from experimental setup 2 are used for this. It is expected that *Testset 2B* does not perform well on a RNN that is not trained on data to which either Algorithm 3 or Algorithm 4 is applied. If the absolute heading dependency is removed, it is expected that the performance on *Testset 2B* improves.

| | | $\hat{\phi}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\theta}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\psi}_{RMSE}$ [$10^{-3}$ rad] | Mean Euclidean distance, $\hat{\delta}_{mean}$ [$10^{-3}$] | Mean angle of rotation, $\hat{\alpha}_{mean}$ [$10^{-3}$ rad] |
|---|---|---|---|---|---|---|
| Baseline performance, using Algorithm 5 | *Testset 2A* | 1655 | 333 | 128 | 817 | 1685 |
| | *Testset 2B* | 1652 | 383 | 120 | 818 | 1686 |
| Baseline performance, using Algorithms 2 & 5 | *Testset 2A* | 114 | 267 | 135 | 153 | 306 |
| | *Testset 2B* | 114 | 302 | 135 | 168 | 337 |
| RNNs, using Algorithms 2 & 5, absolute heading dependent | *Testset 2A* | 30.1 ± 4.67 | 38.3 ± 3.88 | 36.6 ± 2.54 | 27.2 ± 2.21 | 54.5 ± 4.42 |
| | *Testset 2B* | 98.6 ± 22.6 | 192 ± 48.7 | 134 ± 22.1 | 110 ± 17.6 | 220 ± 35.5 |
| RNNs, using Algorithms 2, 3 & 5, absolute heading independent | *Testset 2A* | 29.5 ± 4.78 | 40.2 ± 3.75 | 37.0 ± 2.76 | 27.8 ± 2.03 | 55.5 ± 4.06 |
| | *Testset 2B* | 43.1 ± 5.48 | 79.2 ± 5.68 | 54.4 ± 4.24 | 46.6 ± 3.2 | 93.1 ± 6.4 |
| RNNs, using Algorithms 2, 4 & 5, absolute heading independent | *Testset 2A* | 29.4 ± 4.72 | 40.1 ± 5.07 | 37.9 ± 3.69 | 27.9 ± 2.95 | 55.9 ± 5.89 |
| | *Testset 2B* | 37.1 ± 4.32 | 70.4 ± 4.98 | 52.4 ± 5.09 | 43.0 ± 3.06 | 85.9 ± 6.11 |

**Table 6-3:** The average error of 20 trained RNNs on *Testset 2A* and *Testset 2B*.

Note that when using Algorithm 3, the only additional data which is appended to the training set is the training data rotated to the mean absolute heading direction of *Testset 2B*. Since adding a mean absolute heading is not the same as adding every absolute heading, this means that only a portion of the effect is discovered rather than its full potential. Computational effort is the limiting factor here.

After inspecting the results for Algorithms 3 and 4, two major points stand out.

1. The error for a dataset which has the same absolute heading direction as the training data (like *Testset 2A*) is hardly affected when using Algorithm 3 or 4. This is because these algorithms try to remove an absolute heading dependency from the dataset that is identical to the absolute heading dependency of the training data. That won't make a big difference.

2. When using the algorithms on *Testset 2B*, containing data with different absolute heading direction, Algorithm 3 decreases the error by a factor 2.36. Algorithm 4 manages to decrease the error by a factor 2.56. The final obtained error for both algorithms lie very close to each other and it can be concluded that the algorithms work and also do about the same thing.

The obtained results were expected. An overview of the error decomposition of data in *Testset 2B* can be viewed in Figure 6-6. Only 5.26% of the error could not be estimated by the RNN when using Algorithm 4.

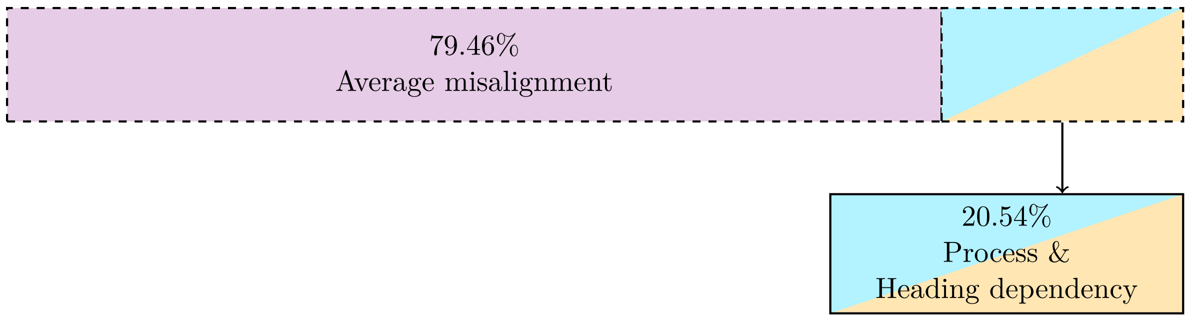


**Figure 6-6:** The error decomposition of *Testset 2B*

### 6-2-4 Using physics-based input features as opposed to inertial data

This section presents the error values of the orientation estimates of the RNN when it is fed with calibrated and scaled inertial data, rather than orientation estimates of the loosely attached sensor. For this, again, data from experimental setup 2 is used. The performance of datasets *Testset 2A* and *Testset 2B* is assessed. First, the results are displayed using physics-based input features, together with the use of Algorithms 2, 4 and 5. Next, the performance is assessed on the same datasets, however, this time using inertial input data.

| | | $\hat{\phi}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\theta}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\psi}_{RMSE}$ [$10^{-3}$ rad] | Mean Euclidean distance, $\hat{\delta}_{mean}$ [$10^{-3}$] | Mean angle of rotation, $\hat{\alpha}_{mean}$ [$10^{-3}$ rad] |
|---|---|---|---|---|---|---|
| RNNs, using physics-based inputs and Algorithms 2, 4 & 5 | *Testset 2A* | 29.4 $\pm$ 4.72 | 40.1 $\pm$ 5.07 | 37.9 $\pm$ 3.69 | 27.9 $\pm$ 2.95 | 55.9 $\pm$ 5.89 |
| | *Testset 2B* | 37.1 $\pm$ 4.32 | 70.4 $\pm$ 4.98 | 52.4 $\pm$ 5.09 | 43.0 $\pm$ 3.06 | 85.9 $\pm$ 6.11 |
| RNNs, using calibrated and scaled inertial input data | *Testset 2A* | 99.4 $\pm$ 32.2 | 107 $\pm$ 18.9 | 117 $\pm$ 21.7 | 77.6 $\pm$ 13.6 | 155 $\pm$ 27.2 |
| | *Testset 2B* | 706 $\pm$ 90.9 | 632 $\pm$ 40.5 | 1011 $\pm$ 119 | 555 $\pm$ 51.5 | 1131 $\pm$ 108 |

**Table 6-4:** The average error of 20 trained RNNs on *Testset 2A* and *Testset 2B*.

By inspecting the results presented in Table 6-4, it becomes clear that a network trained on calibrated and scaled inertial data is not able to outperform a network that uses physics-based inputs. This was presumed from the beginning of the study and is confirmed by this result. Although the network achieves 2.78× higher error with this amount of data, it is worth mentioning it is not totally failing to learn the process. From this, it can be concluded that by using physics-based inputs, a part of the process is already resolved and the remaining process is smaller or easier to learn.

Another important result is that the error difference for *Testset 2B* (12.9×) is remarkably higher than for *Testset 2A* (2.78×). As the main difference between the two datasets is that the data contains a different absolute heading direction, the reason for this should be sought there. The gyroscope measurements are indifferent to the heading direction, but the accelerometer and magnetometer measurements are not. This indicates that those input features are used by the network and to solve this issue, one should apply an operation to the accelerometer and magnetometer data similar to the removal of the absolute heading dependency described in Section 4-4.

## 6-2-5 Results for data from experimental setup 3 and 4

So far, the effects of the designed preprocessing steps are shown for the simulated setup and for experimental setup 1 and 2. They are found to be beneficial to the problem of estimating orientation using a loosely attached IMU. In order to support these findings, two other experimental setups were designed (Section 5-3). In Table 6-5 the results of different steps are displayed for experimental setup 3. Results on experimental setup 4 can be found in Table 6-6 and Table 6-7.

| | | $\hat{\phi}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\theta}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\psi}_{RMSE}$ [$10^{-3}$ rad] | Mean Euclidean distance, $\hat{\delta}_{mean}$ [$10^{-3}$] | Mean angle of rotation, $\hat{\alpha}_{mean}$ [$10^{-3}$ rad] |
|---|---|---|---|---|---|---|
| Baseline performance, using Algorithm 5 | *Testset 3A* | 167 | 202 | 1583 | 780 | 1603 |
| | *Testset 3B* | 204 | 203 | 1575 | 778 | 1599 |
| Baseline performance, using Algorithms 2 & 5 | *Testset 3A* | 203 | 161 | 32.0 | 114 | 229 |
| | *Testset 3B* | 205 | 202 | 45.0 | 128 | 257 |
| RNNs, using Algorithms 2 & 5 | *Testset 3A* | 20.2 ± 1.36 | 21.7 ± 3.0 | 18.5 ± 1.86 | 15.0 ± 1.1 | 30.0 ± 2.21 |
| | *Testset 3B* | 342 ± 118 | 271 ± 115 | 1165 ± 133 | 609 ± 65.8 | 1239 ± 138 |
| RNNs, using Algorithms 2,4 & 5 | *Testset 3A* | 19.5 ± 1.02 | 21.3 ± 3.18 | 17.3 ± 1.23 | 14.4 ± 1.17 | 28.8 ± 2.33 |
| | *Testset 3B* | 24.8 ± 1.64 | 30.9 ± 2.74 | 24.3 ± 1.35 | 20.2 ± 1.17 | 40.4 ± 2.35 |
| RNNs, using calibrated and scaled inertial input data | *Testset 3A* | 23.0 ± 1.73 | 24.9 ± 3.35 | 46.1 ± 4.07 | 23.3 ± 2.14 | 46.6 ± 4.27 |
| | *Testset 3B* | 471 ± 28.4 | 333 ± 20.3 | 1191 ± 34.8 | 641 ± 15.9 | 1306 ± 33.6 |

**Table 6-5:** The average error of 20 trained RNNs on *Testset 3A* and *Testset 3B*.

As becomes clear from the difference in results displayed by the first two rows of Table 6-5, both datasets benefit from the removal of the average misalignment. The performance of *Testset 3A* improves by a factor 6.84 and that of *Testset 3B* by a factor 6.08. This improvement is dependent on the degree of misalignment between the sensors and on the initial error in the data. Next, the RNN is applied, which is supposed to estimate the process between the sensors caused by the movements of the object and the loosely attachedness of the input sensor. As *Testset 3A* is recorded facing the same absolute heading direction as the training data of the network, it is expected the performance will improve for this dataset. It does so by a factor 7.6. The data in *Testset 3B* is recorded facing a different absolute heading direction than the training data. As reported by [13] and considered to be common knowledge, neural networks perform bad on unseen data. This is in line with the results, as the performance deteriorates by a factor 4.76 again. In order to improve the performance on *Testset 3B* as well, the absolute heading difference is addressed. Applying Algorithm 4 and the RNN improves the performance by a factor 6.34. Also, the performance of *Testset 3A* does not suffer from removing the absolute heading dependency by Algorithm 3. In total, applying the preprocessing steps results in a performance increase of 54.2× for *Testset 3A* and 38.5× for *Testset 3B*. With that, the remaining error for both datasets is 1.85% for *Testset 3A* and 2.60% for *Testset 3B* of the initial error. This percentage can be contributed to a part of the process that was too complex, unobservable or undiscovered as explained in Section 1-3. The error distribution for *Testset 3B* is compactly visualized in Figure 6-7.

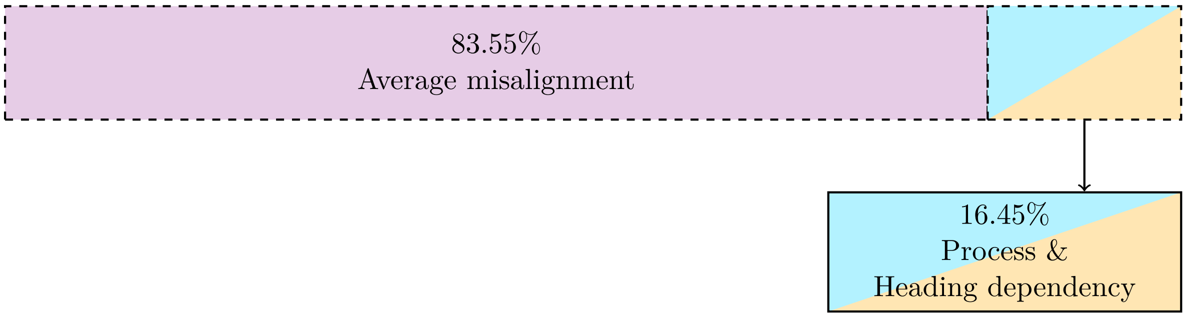


**Figure 6-7:** The error decomposition of *Testset 3B*.

Again, it can also be concluded that using calibrated and scaled inertial measurements as input features does not outperform the use of physics-based input features.

Next, the results of experimental setup 4 are displayed in Table 6-6. The datasets used to assess the performance to this setup are *Testset 4A* and *Testset 4B*. These contain a diverse set of motions.

| | | $\hat{\phi}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\theta}_{RMSE}$ [$10^{-3}$ rad] | $\hat{\psi}_{RMSE}$ [$10^{-3}$ rad] | Mean Euclidean distance, $\hat{\delta}_{mean}$ [$10^{-3}$] | Mean angle of rotation, $\hat{\alpha}_{mean}$ [$10^{-3}$ rad] |
|---|---|---|---|---|---|---|
| Baseline performance, using Algorithm 5 | *Testset 4A* | 893 | 119 | 44.0 | 442 | 892 |
| | *Testset 4B* | 795 | 122 | 37.0 | 396 | 797 |
| Baseline performance, using Algorithms 2 & 5 | *Testset 4A* | 128 | 30.0 | 38.0 | 59.0 | 118 |
| | *Testset 4B* | 111 | 33.0 | 34.0 | 54.5 | 109 |
| RNNs, using Algorithms 2 & 5 | *Testset 4A* | 120 ± 6.84 | 28.5 ± 3.11 | 35.6 ± 3.62 | 49.3 ± 2.24 | 98.7 ± 4.49 |
| | *Testset 4B* | 280 ± 73.7 | 317 ± 97.6 | 602 ± 141 | 346 ± 48.7 | 698 ± 99.5 |
| RNNs, using Algorithms 2,4 & 5 | *Testset 4A* | 163 ± 16.4 | 39.0 ± 3.63 | 53.8 ± 2.6 | 65.9 ± 4.64 | 132 ± 9.32 |
| | *Testset 4B* | 186 ± 11.5 | 52.1 ± 4.19 | 49.1 ± 3.01 | 81.3 ± 4.51 | 163 ± 9.04 |
| RNNs, using calibrated and scaled inertial input data | *Testset 4A* | 129 ± 6.04 | 34.6 ± 4.24 | 66.8 ± 3.08 | 57.0 ± 2.6 | 114 ± 5.21 |
| | *Testset 4B* | 503 ± 65.9 | 599 ± 44.4 | 891 ± 62.1 | 579 ± 25.1 | 1176 ± 52.5 |

**Table 6-6:** The average error of 20 trained RNNs on *Testset 4A* and *Testset 4B* containing a diverse set of motions.

As becomes clear from Table 6-6, the RNN and algorithm 4 are not able to reduce the estimation error for data obtained from this setup compared to the baseline performance using Algorithms 2 and 5. In search for an explanation for this, the suspicion grew that the process was much more complex and, therefore, the amount of recorded training data was not sufficient for the network to properly learn the process. In order to verify this, another dataset was recorded. This time, instead of trying to capture all possible movements, the amount of different motions of the subject was reduced. This way, more data of fewer motions was recorded. *Testset 4C* and *Testset 4D* contain data from motions performed by the arm which only draws circles in the air. Table 6-7 shows the obtained results.

| | | $\hat{\phi}_{RMSE}$ $[10^{-3}$ rad$]$ | $\hat{\theta}_{RMSE}$ $[10^{-3}$ rad$]$ | $\hat{\psi}_{RMSE}$ $[10^{-3}$ rad$]$ | Mean Euclidean distance, $\hat{\delta}_{mean}$ $[10^{-3}]$ | Mean angle of rotation, $\hat{\alpha}_{mean}$ $[10^{-3}$ rad$]$ |
|---|---|---|---|---|---|---|
| Baseline performance, using Algorithm 5 | *Testset 4C* | 1837 | 114 | 55.0 | 886 | 1837 |
| | *Testset 4D* | 1844 | 51.0 | 37.0 | 887 | 1839 |
| Baseline performance, using Algorithms 2 & 5 | *Testset 4C* | 137 | 74.0 | 36.0 | 75.9 | 152 |
| | *Testset 4D* | 145 | 75.0 | 64.0 | 81.2 | 162 |
| RNNs, using Algorithms 2 & 5 | *Testset 4C* | 78.4 ± 3.64 | 26.1 ± 3.5 | 18.1 ± 2.42 | 35.3 ± 1.43 | 70.7 ± 2.87 |
| | *Testset 4D* | 516 ± 50.8 | 228 ± 44.1 | 325 ± 44.9 | 279 ± 19.2 | 561 ± 39.1 |
| RNNs, using Algorithms 2,4 & 5 | *Testset 4C* | 109 ± 6.62 | 101 ± 4.62 | 85.2 ± 5.17 | 78.2 ± 2.48 | 156 ± 4.96 |
| | *Testset 4D* | 179 ± 10.9 | 61.7 ± 4.45 | 86.8 ± 2.68 | 75.8 ± 3.31 | 152 ± 6.65 |
| RNNs, using calibrated and scaled inertial input data | *Testset 4C* | 80.4 ± 2.67 | 51.5 ± 2.44 | 38.2 ± 1.77 | 45.3 ± 1.35 | 90.5 ± 2.71 |
| | *Testset 4D* | 633 ± 38.3 | 278 ± 54.7 | 701 ± 47.0 | 462 ± 25.7 | 935 ± 52.8 |

**Table 6-7:** The average error of 20 trained RNNs on *Testset 4C* and *Testset 4D* containing data from simpler motions.

As it can be observed, using the RNN and Algorithm 4 now show a slight reduction in error for *Testset 4D*. For *Testset 4C*, adding Algorithm 4 increases the error. A possible explanation for this is that the data in *Testset 4C* and the training data are so similar and that the network had used the absolute heading direction to compute the target orientation estimates. When removing this information, there is less information the network can rely on and the performance is dropped. The superior performance on *Testset 4C* can be explained by the fact that *Testset 4C* contains the same absolute heading direction information as the training data. However, it is undesirable to rely on that dependency. It is good to see that the error when using Algorithm 4 is about the same. This is a nice example of trading generalizability for performance. It is better to have a network performing well in many possible situations (*Testset 4C* and *Testset 4D*) rather than it being better for only a particular part of the possible data (*Testset 4C*). Another property of this experimental setup that could be the cause to these less convincing results is that the baseline error using Algorithms 2 and 5 is much lower. Put differently: There is just fewer process to be learned, making the RNN less necessary. The final error is not as low as it was for data obtained from experimental setup 2 and 3. This result supports the suspicion that the process is much more complex, since even if the data represents only a small portion of the possible motions, identical performance is not obtained. With this observation, the conclusion is drawn that in order to learn the process of experimental setup 4, at least a lot more data and computational power is required. This is beyond the scope of this study.

The last conclusion that can be drawn is that again, using calibrated and scaled inertial data does not outperform the employment of the RNN using physics-based input features.

The reduced error for experimental setup 1, 2 and 3 caused by the designed preprocessing steps sufficiently demonstrate that these steps are beneficial for estimating orientation using a loosely attached IMU.

# Chapter 7

# Conclusions and future work

*This chapter describes the general conclusions that are drawn from this study and its results. Next, possible future steps are discussed.*

## 7-1 Conclusions

In this work, four algorithms that constitute three preprocessing steps, were presented to improve orientation estimation using a loosely attached IMU, a RNN and physics-based knowledge. It was shown that the estimation of orientation using a loosely attached IMU can really benefit from these steps. These steps made it possible to decompose the error which has not been seen in literature yet. In addition, it was shown that using physics-based input features outperforms the use of calibrated and scaled inertial data.

The first preprocessing step, removing an average misalignment, answers to the first and second research goal, presented in Section 1-3. Using two experimental setups, this step has proven to be beneficial for two purposes. Firstly, by removing the average misalignment, it was possible to obtain a baseline performance that was not affected by a misaligned mounting of the sensor. This way, the part that contributed to the error due to this misalignment could be isolated. Secondly and possibly more important, by removing the average misalignment from data fed to the network during the training phase and inference phase, it was possible to train a network that is robust to misaligned attachments. This allows the proposed model to be deployed for an enlarged set of situations.

The second preprocessing step in the proposed model has aimed to remove the dependency on the absolute heading direction of the data. Two algorithms to do this have been designed, one which added artificially generated data to the dataset and one that removes the dependency from the data. The former showed good results, however, it was computationally much more demanding. The latter achieved equal results without the extra computational burden which makes it a powerful algorithm to address the second research goal.

The last preprocessing step answers to the second research goal. By removing ambiguity and discontinuities from data it was managed to reduce the amount of data required by a factor

2 for the worst-case scenario. Results on a simulated setup have shown such situations can occur and that this step is of crucial importance.

In response to the last research goal, the designed network was fed with calibrated and scaled inertial data. The goal was to verify that using physics-based input features is beneficial for the performance of the network. The results on experimental setup 2, 3 and 4 have confirmed that this is true. Another advantage of using the physics-based input features is that it is possible to reduce the task of the RNN. This way, there is more insight in what the network actually learns to do the estimation.

Algorithm 2 and Algorithm 4 make use of the conversion to the Euler angles parameterization. This parameterization suffers from gimbal lock. This does not occur too often but is a disadvantage of these steps. A way to cope with this is thought of, namely, interpolating the unobservable angles. This, however, remains an approximation of reality. Given the low frequency of occurring of gimbal lock, it is assumed that this approximation is not able to ruin the results.

Another issue that was encountered, was the difficulty of quantifying the effect of the designed algorithms. The results are highly dependent on the kind of setup that is used and also differ a bit dependent on the content of the recorded data. To address the dependency on the kind of setup, the use of the algorithms should be selected according to what a setup requires. It does not do any harm though, using the algorithms when they are not required. The problem of dependency on the recorded data could be addressed by using better hardware, which allows training the network with much more data. This way the dependency on the recorded data is reduced because ML methods draw their power from data, meaning more data could potentially result in better performance.

## 7-2 Future work

As the subject under investigation was rather broad, there are multiple steps that can be addressed by future studies. A number of them are discussed below.

As already described, the computational power of the hardware setup was a limiting factor in some cases. One could improve the effect of the preprocessing steps by recording more data and learning the process more thoroughly this way.

The robustness of the RNN could be examined for the case in which the orientation estimates from the EKF are slightly off or when a part of the test data is faulty. This could occur for example, when the magnetic field is disturbed but the measurements are still within bounds. Then the orientation estimates from the EKF are affected by this.

In this study, parameters of networks that had shown to be successful in literature are adopted in order to save time. These parameters could be optimized for this problem. While this is probably not the hardest task, the process will take a lot of time due to long training times. One could consider doing a grid search over the parameter space.

The orientation estimates computed by the EKF were compared with the orientation estimates from the fusion algorithm of the Xsens IMUs and it was concluded that they were sufficiently accurate. The orientation estimates computed by the algorithm of Xsens are considered to be correct. Although, this is an accurate orientation estimation algorithm, the orientation estimates of the EKF could be compared to the orientation estimates of an optical system.

The use of different IMUs has not been explored in this study. Future work could be to verify the working of the proposed model using a different kind of IMUs, which might have an effect due to different noise characteristics or sampling frequencies.

While this study focused on the orientation estimation using a single IMU on purpose, employing multiple loosely attached IMUs could be beneficial. This area is unaddressed and could be explored in future studies.

While there are good reasons to use RNNs for orientation estimation related problems, occasionally conducted research in this area have employed deep neural networks or convolutional neural networks. Comparing the performance using these networks and other ML methods are left for future studies as well.

# Appendix A

# Conversions

This appendix describes how unit quaternions are converted to Euler angles and visa versa. In order to do this an intermediate conversion to and from rotation matrices is applied. Below the conversions from a unit quaternion to a rotation matrix is described. Then the rotation from a rotation matrix to a unit quaternion. Next, the conversion from a rotation matrix to Euler angles is explained and lastly, the conversion from Euler angles to a rotation matrix will be discussed.

## A-1 Converting unit quaternions to rotation matrices

The conversion in (A-1) is used to convert unit quaternions to rotation matrices. In this case the orientation representation used as example describes the orientation of a sensor frame $S$ with respect to the navigation frame $N$. The quaternion that is converted should be normalized beforehand. If this is the case, the following conversion,

$$R(q)^{NS} = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 - 2q_0q_3 & 2q_0q_2 + 2q_1q_3 \\ 2q_0q_3 + 2q_1q_2 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 - 2q_0q_1 \\ 2q_1q_3 - 2q_02q_2 & 2q_2q_3 + 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}, \tag{A-1}$$

is used [37].

## A-2 Converting rotation matrices to unit quaternions

Having obtained a rotation matrix from an Euler angles representation, it can be converted to a unit quaternion representation. To do this, several measures are taken to prevent square roots having a negative argument [38, 37]. This way it is possible to convert unit quaternions to rotation matrices and converting them back yielding the same unit quaternions that was started with.

The rotation matrix is parameterized by

$$R^{NS} = \begin{bmatrix} r_{00} & r_{01} & r_{02} \\ r_{10} & r_{11} & r_{12} \\ r_{20} & r_{21} & r_{22} \end{bmatrix}. \tag{A-2}$$

If $r_{11} > -r_{22} \cap r_{00} > -r_{11} \cap r_{00} > -r_{22}$, then $R^{NS}$ is converted to $q^{NS}$ by

$$q(R)^{NS} = \frac{1}{2}\begin{bmatrix} 1 + r_{00} + r_{11} + r_{22} \\ r_{21} - r_{12} \\ r_{02} - r_{20} \\ r_{10} - r_{01} \end{bmatrix} / \sqrt{1+r_{00}+r_{11}+r_{22}}. \tag{A-3}$$

Otherwise if $r_{11} < -r_{22} \cap r_{00} > r_{11} \cap r_{00} > r_{22}$, then $R^{NS}$ is converted to $q^{NS}$ by

$$q(R)^{NS} = \frac{1}{2}\begin{bmatrix} r_{21} - r_{12} \\ 1 + r_{00} - r_{11} - r_{22} \\ r_{01} + r_{10} \\ r_{02} + r_{20} \end{bmatrix} / \sqrt{1+r_{00}-r_{11}-r_{22}}. \tag{A-4}$$

Otherwise if $r_{11} > r_{22} \cap r_{00} < r_{11} \cap r_{00} < -r_{22}$, then $R^{NS}$ is converted to $q^{NS}$ by

$$q(R)^{NS} = \frac{1}{2}\begin{bmatrix} r_{02} - r_{20} \\ r_{01} + r_{10} \\ 1 - r_{00} + r_{11} - r_{22} \\ r_{12} + r_{21} \end{bmatrix} / \sqrt{1-r_{00}+r_{11}-r_{22}}. \tag{A-5}$$

Otherwise $R^{NS}$ is converted to $q^{NS}$ by

$$q(R)^{NS} = \frac{1}{2}\begin{bmatrix} r_{10} - r_{01} \\ r_{02} + r_{20} \\ r_{12} + r_{21} \\ 1 - r_{00} - r_{11} + r_{22} \end{bmatrix} / \sqrt{1-r_{00}-r_{11}+r_{22}}. \tag{A-6}$$

## A-3 Converting Euler angles to rotation matrices

This section describes how an Euler angles representation is converted to a rotation matrix. This conversion is dependent on the order of rotation around the orthogonal axes of an object as described in Section 2-2-2. The order that will be presented here is the Z-Y'-X" order. This is in accordance with the constraint imposed on the sensor frame's z-axis, which need to align with the navigation frame's z-axis. This was described in Section 4-4-2. An Euler angles representation is converted to a rotation matrix representation using

$$\begin{aligned}
&R(\phi,\theta,\psi)^{NS}\\
&= R^Z_\psi R^Y_\theta R^X_\phi\\
&= \begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix}\\
&= \begin{bmatrix}\cos\psi\cos\theta & -\sin\psi\cos\phi+\cos\psi\sin\theta\sin\phi & \sin\psi\sin\phi+\cos\psi\sin\theta\cos\phi\\ \sin\psi\cos\theta & \cos\psi\cos\phi+\sin\psi\sin\theta\sin\phi & -\cos\psi\sin\phi+\sin\psi\sin\theta\cos\phi\\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi\end{bmatrix}. \qquad \text{(A-7)}
\end{aligned}$$

## A-4 Converting rotation matrices to Euler angles

The conversion from a rotation matrix representation to an Euler angles representation can be derived by inspecting (A-7). Using the parameterization of the rotation matrix as denoted in (A-2) the angles are defined as

$$\phi(R^{NS}) = \operatorname{atan2}\frac{R_{21}}{R_{22}}, \qquad \text{(A-8)}$$

$$\theta(R^{NS}) = -\operatorname{asin} R_{20} \text{ and} \qquad \text{(A-9)}$$

$$\psi(R^{NS}) = \operatorname{atan2}\frac{R_{10}}{R_{00}}. \qquad \text{(A-10)}$$

The range of the angles is from $-\pi$ [rad] to $\pi$ [rad] for the roll ($\phi$) and yaw angle ($\psi$) and from $-\frac{\pi}{2}$ [rad] to $\frac{\pi}{2}$ [rad] for the pitch angle ($\theta$). Note here that gimbal lock can occur as the pitch angle reaches near $-\frac{\pi}{2}$ [rad] or $\frac{\pi}{2}$ [rad]. Then the rank of the rotation matrix decreases by one and the roll and yaw angle cannot be distinguished from each other.

# Appendix B

# Quaternion algebra

This appendix provides a number of operations that can be applied to quaternions as well as a number of properties of quaternions [39, 31]. Most of them are used in Appendix C and in Algorithm 1.

## B-1 Quaternion operations

The quaternion consists of four elements $\begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T$. Operations that involve quaternions which are used in this work are listed below [39].

- The quaternion conjugate is denoted by $q^c = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^T$.
- The quaternion multiplication is denoted by $p \odot q = \begin{bmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ p_0q_2 + p_2q_0 + p_3q_1 - p_1q_3 \\ p_0q_3 + p_3q_0 + p_1q_2 - p_2q_1 \end{bmatrix}$, where $p$ and $q$ are both quaternions.
- The right multiplication operator $q^R = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix}$ and $(p \odot q) = q^R p$.
- The conjugate of a quaternion multiplication is given by $(p \odot q)^c = q^c \odot p^c$.

## B-2 Quaternion properties

In this section the proof of two properties is given.

1. $q + q^c = 2q_0$
2. $(a \odot b)_0 = a^T b^c$

Item 1 is derived by

$$q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}, q^c = \begin{bmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix} \text{ and } q + q^c = \begin{bmatrix} q_0 + q_0 \\ q_1 - q_1 \\ q_2 - q_2 \\ q_3 - q_3 \end{bmatrix} = 2 \begin{bmatrix} q_0 \\ 0 \\ 0 \\ 0 \end{bmatrix}. \tag{B-1}$$

Item 2 is derived by

$$a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}, b = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}, (a \odot b)_0 = a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3 \text{ and}$$

$$a^T b^c = \begin{bmatrix} a_0 & a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_0 \\ -b_1 \\ -b_2 \\ -b_3 \end{bmatrix} = a_0 b_0 - a_1 b_1 - a_2 b_2 - a_3 b_3. \tag{B-2}$$

# Appendix C

# Algorithm: Estimating orientation difference between stationary frames using unit quaternions

The author of [39] mentions an algorithm that can be used to estimate the orientation difference between two stationary frames using unit quaternions as orientation representations. This is often used in order to calibrate two reference systems measuring the same orientation. In this study, however, it was used to verify whether two reference frames are aligned or not.

## C-1 Theorem (Deriving the orientation difference between stationary orientation estimations)

Assume you have $\{q_n^{uv}\}_{n=1}^N$ and $\{q_n^{xw}\}_{n=1}^N$. So multiple (N) orientation estimations describing the orientation of frame $v$ with respect to frame $u$, and multiple (N) orientation estimations describing the orientation of frame $w$ with respect frame $x$. Then for every $n$ it should hold that

$$q_n^{uv} \odot q^{vw} \approx q^{ux} \odot q_n^{xw}. \tag{C-1}$$

Where $q^{vw}$ and $q^{ux}$ are the unknown rotations that are being sought. This should hold because at both sides of the equation one tries to calculate $q^{uw}$. In order to compute the rotation $q^{vw}$ and $q^{ux}$, the sum of the squared so-called residual errors is constructed from (C-1) [39],

$$e_n = q_n^{uv} \odot q^{vw} - q^{ux} \odot q_n^{xw}, \tag{C-2}$$
$$||e_n||_2^2 = (q_n^{uv} \odot q^{vw} - q^{ux} \odot q_n^{xw})^2. \tag{C-3}$$

This can be rewritten as

$$\begin{aligned}
||e_n||_2^2 &= (q_n^{uv} \odot q^{vw} - q^{ux} \odot q_n^{xw})(q_n^{uv} \odot q^{vw} - q^{ux} \odot q_n^{xw})^c, && \text{(C-4)}\\
&= (q_n^{uv} \odot q^{vw})(q_n^{uv} \odot q^{vw})^c - (q_n^{uv} \odot q^{vw})(q^{ux} \odot q_n^{xw})^c && \text{(C-5)}\\
&\quad - (q^{ux} \odot q_n^{xw})(q_n^{uv} \odot q^{vw})^c + (q^{ux} \odot q_n^{xw})(q^{ux} \odot q_n^{xw})^c, \\
&= ||q^{uw}||_2^2 - (q_n^{uv} \odot q^{vw})(q^{ux} \odot q_n^{xw})^c - (q^{ux} \odot q_n^{xw})(q_n^{uv} \odot q^{vw})^c + ||q^{uw}||_2^2, && \text{(C-6)}\\
&= 1 - (q_n^{uv} \odot q^{vw})(q^{ux} \odot q_n^{xw})^c - (q^{ux} \odot q_n^{xw})(q_n^{uv} \odot q^{vw})^c + 1, && \text{(C-7)}\\
&= ||q^{uu}||_2^2 - (q_n^{uv} \odot q^{vw})(q^{ux} \odot q_n^{xw})^c - (q^{ux} \odot q_n^{xw})(q_n^{uv} \odot q^{vw})^c + 1, && \text{(C-8)}\\
&= ||q^{uu}||_2^2 - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) - (q_n^{wx} \odot q^{xu})^c(q_n^{uv} \odot q^{vw})^c + 1, && \text{(C-9)}\\
&= ||q^{uu}||_2^2 - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c + 1, && \text{(C-10)}\\
&= ||q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}||_2^2 - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) && \text{(C-11)}\\
&\quad - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c + 1, \\
&= (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})(q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) && \text{(C-12)}\\
&\quad - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c + 1, \\
&= (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1)((q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c - 1), && \text{(C-13)}\\
&= (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1)(q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1)^c, && \text{(C-14)}\\
&= ||q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1||_2^2. && \text{(C-15)}
\end{aligned}$$

Equation (C-15) corresponds with the squared residual error stated in [39]. The sum of squared residual errors is denoted by

$$V(q^{ux}, q^{vw}) = \sum_{n=1}^{N} ||e_n||_2^2 = \sum_{n=1}^{N} ||q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1||_2^2. \tag{C-16}$$

Minimizing $V(q^{ux}, q^{vw})$ yields the best estimations of $q^{ux}$ and $q^{vw}$. This term is minimal if $\hat{q}^{ux}$ and $\hat{q}^{vw}$ equal the first left and right singular vectors of the singular value decomposition of matrix A.

$$\begin{aligned}
A &= U\Sigma V^T, && \text{(C-17)}\\
&= \sum_{n=1}^{N} q_n^{uv^{L^T}} q_n^{xw^R}. && \text{(C-18)}
\end{aligned}$$

## C-2 Proof

To get to this result, the argument of the loss function (C-16) is rewritten to

$$\begin{aligned}
||e_n||_2^2 &= ||q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1||_2^2, && \text{(C-19)}\\
&= (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1)(q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1)^c, && \text{(C-20)}\\
&= (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu} - 1)((q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c - 1), && \text{(C-21)}\\
&= (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})(q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c && \text{(C-22)}\\
&\quad - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c + 1,\\
&= ||q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}||_2^2 - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) && \text{(C-23)}\\
&\quad - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c + 1,\\
&= ||q^{uu}||_2^2 - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c + 1, && \text{(C-24)}\\
&= 2 - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) - (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c. && \text{(C-25)}
\end{aligned}$$

Now making use of (B-1) yields

$$\begin{aligned}
||e_n||_2^2 &= 2 - ((q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu}) + (q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})^c), && \text{(C-26)}\\
&= 2 - 2(q_n^{uv} \odot q^{vw} \odot q_n^{wx} \odot q^{xu})_0. && \text{(C-27)}
\end{aligned}$$

Now using (B-2)

$$||e_n||_2^2 = 2 - 2(q_n^{uv} \odot q^{vw})^T (q_n^{wx} \odot q^{xu})^c. \tag{C-28}$$

Using Appendix B-1 gives

$$\begin{aligned}
||e_n||_2^2 &= 2 - 2(q_n^{uv^L} q^{vw})^T (q^{ux} \odot q_n^{xw}),\\
&= 2 - 2(q^{vw^T} q^{uv_n^{L^T}})(q_n^{xw^R} \odot q^{ux}),\\
&= 2 - 2q^{vw^T} q^{uv_n^{L^T}} q_n^{xw^R} \odot q^{ux}. && \text{(C-29)}
\end{aligned}$$

Minimizing $||e_n||_2^2$ means maximizing $q^{vw^T} q^{uv_n^{L^T}} q_n^{xw^R} \odot q^{ux}$. We define

$$A = \sum_{n=1}^{N} q_n^{uv^{L^T}} q_n^{xw^R}. \tag{C-30}$$

The maximization problem then reads

$$\begin{aligned}
&\text{argmin} \sum_{n=1}^{N} ||e_n||_2^2 := \text{argmax } q^{vw^T} A q^{ux}, && \text{(C-31)}\\
&\text{subject to } ||q^{ux}||_2^2 = 1, ||q^{vw}||_2^2 = 1.
\end{aligned}$$

The solution for which $||e_n||_2^2$ is minimized then is

$$\begin{aligned}
A &= U\Sigma V^T,\\
\hat{q}^{ux} &= v_1, \text{ the first right singular vector},\\
\hat{q}^{vw} &= u_1, \text{ the first left singular vector}.
\end{aligned}$$

# Appendix D

# Example: The redistribution of RMSEs parameterized by Euler angles

This appendix provides the code and results to show that the RMSE of a single Euler angle can increase while the overall error decreases.

```
import numpy as np
from HelperFunctions import RToEulerAngles, qToR, quaternionConjugate
from HelperFunctions import eulerAnglesToR, RToQ, quatMultiply

# Define three starting orientation representations
Euler0 = np.array([0, 0, 0])*np.pi/180  # Euler angles [deg] sample 0
Euler1 = np.array([45, 0, 0])*np.pi/180 # Euler angles [deg] sample 1
Euler2 = np.array([0, 45, 0])*np.pi/180 # Euler angles [deg] sample 2

# Convert the Euler angles parameterization to unit quaternions
q0 = RToQ( eulerAnglesToR( Euler0 ) ) # unit quaternion sample 0
q1 = RToQ( eulerAnglesToR( Euler1 ) ) # unit quaternion sample 1
q2 = RToQ( eulerAnglesToR( Euler2 ) ) # unit quaternion sample 2

# Compute the quaternion difference between subsequent samples
diff1 = quatMultiply( quaternionConjugate( q0 ), q1 )
diff2 = quatMultiply( quaternionConjugate( q0 ), q2 )

# Convert the quaternion difference to Euler angles
EulerDiff1 = RToEulerAngles(qToR(diff1))
EulerDiff2 = RToEulerAngles(qToR(diff2))

# Compute the mean Euler angles of the orientation difference
EulerMeanDiff = (EulerDiff1 + EulerDiff2) / 2
print("The mean Euler angles [deg] of orientation difference are: ",
    EulerMeanDiff*180/np.pi)

# Convert the mean orientation difference to unit quaternions
```

```
qMeanDiff = RToQ(eulerAnglesToR(EulerMeanDiff))

# Rotate the orientation samples by the mean orientation difference
q1Rotated = quatMultiply(q1, quaternionConjugate(qMeanDiff))
q2Rotated = quatMultiply(q2, quaternionConjugate(qMeanDiff))

# Compute the original and rotated orientation differences
q1OriginaldifferenceQuaternion = quatMultiply(quaternionConjugate(q0), q1
    )
q1RotateddifferenceQuaternion = quatMultiply(quaternionConjugate(q0),
    q1Rotated)
q2OriginaldifferenceQuaternion = quatMultiply(quaternionConjugate(q0), q2
    )
q2RotateddifferenceQuaternion = quatMultiply(quaternionConjugate(q0),
    q2Rotated)

 # Convert original and rotated orientation difference to Euler angles
q1OriginalDifferenceEulerAngles = RToEulerAngles(qToR(
    q1OriginaldifferenceQuaternion))
q1RotatedDifferenceEulerAngles = RToEulerAngles(qToR(
    q1RotateddifferenceQuaternion))
q2OriginalDifferenceEulerAngles = RToEulerAngles(qToR(
    q2OriginaldifferenceQuaternion))
q2RotatedDifferenceEulerAngles = RToEulerAngles(qToR(
    q2RotateddifferenceQuaternion))

# Compute the RMSE of the Euler angles describing the mean orientation
    difference
RMSEOriginal = np.array([
        np.round(np.sqrt((np.square(q1OriginalDifferenceEulerAngles[0])
            /2) + (np.square(q2OriginalDifferenceEulerAngles[0])/2) ), 3),
        np.round(np.sqrt((np.square(q1OriginalDifferenceEulerAngles[1])
            /2) + (np.square(q2OriginalDifferenceEulerAngles[1])/2) ), 3),
        np.round(np.sqrt((np.square(q1OriginalDifferenceEulerAngles[2])
            /2) + (np.square(q2OriginalDifferenceEulerAngles[2])/2) ), 3),
    ])

RMSERotated = np.array([
        np.round(np.sqrt((np.square(q1RotatedDifferenceEulerAngles[0])/2)
             + (np.square(q2RotatedDifferenceEulerAngles[0])/2) ), 3),
        np.round(np.sqrt((np.square(q1RotatedDifferenceEulerAngles[1])/2)
             + (np.square(q2RotatedDifferenceEulerAngles[1])/2) ), 3),
        np.round(np.sqrt((np.square(q1RotatedDifferenceEulerAngles[2])/2)
             + (np.square(q2RotatedDifferenceEulerAngles[2])/2) ), 3),
    ])

# Print the results
print("Mean original orientation difference in Euler angles: ", np.round(
    RMSEOriginal*180/np.pi,3), "[deg]")
print("Mean rotated orientation difference in Euler angles: ", np.round(
    RMSERotated*180/np.pi,3), "[deg]")
```

The output of the program is denoted below:

```
> The mean Euler angles [deg] of orientation difference are: [22.5 22.5 0. ]
> Mean original orientation difference in Euler angles: [31.799 31.799 0.] [deg]
> Mean rotated orientation difference in Euler angles: [20.970 22.288 9.110] [deg]
```

The Euler angles in the output are ordered as follows: [roll, pitch, yaw]. The first line indicates that the yaw angle of the average orientation difference was zero. Also, the original average orientation difference had zero yaw, visible in line two. Despite this, the average misalignment of the rotated data has an increased yaw error, indicated by the last line.

# Bibliography

[1] A. Fransson and O. Lundström, "Learning a better attitude: A recurrent neural filter for orientation estimation," Master's thesis, Chalmers University of Technology, 2020.

[2] H. Fan, M. Jiang, L. Xu, H. Zhu, J. Cheng, and J. Jiang, "Comparison of long short term memory networks and the hydrological model in runoff simulation," *Water*, vol. 12, p. 175, 01 2020.

[3] S. Varsamopoulos, K. Bertels, and C. G. Almudever, "Designing neural network based decoders for surface codes," 2018.

[4] S. Sun, D. Melamed, and K. Kitani, "Idol: Inertial deep orientation-estimation and localization," 02 2021.

[5] D. Weber, C. Gühmann, and T. Seel, "Robust neural networks outperform attitude estimation filters," 2021.

[6] T. Beravs, P. Rebersek, D. Novak, J. Podobnik, and M. Munih, "Development and validation of a wearable inertial measurement system for use with lower limb exoskeletons," in *Proceedings of the International Conference on Humanoid Robot, IEEE-RASs*, pp. 212–217, 10 2011.

[7] R. Gurchiek, N. Cheney, and R. McGinnis, "Estimating biomechanical time-series with wearable sensors: A systematic review of machine learning techniques," *Sensors*, vol. 19, p. 5227, 11 2019.

[8] M. Trumble, A. Gilbert, C. Malleson, A. Hilton, and J. Collomosse, "Total capture: 3d human pose estimation fusing video and inertial sensors," 09 2017.

[9] X. Xiao and S. Zarar, "A wearable system for articulated human pose tracking under uncertainty of sensor placement," in *Proceedings of the 7th International Conference on Biomedical Robotics and Biomechatronics*, pp. 1144–1150, 2018.

[10] G. B. Guerra-filho, "Optical motion capture: Theory and implementation," *Theoretical and Applied Informatics*, vol. 12, no. 2, pp. 1–29, 2005.

[11] D. Nagaraj, E. Schake, P. Leiner, and D. Werth, "An RNN-ensemble approach for real time human pose estimation from sparse IMUs," in *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, pp. 1–6, 2020.

[12] S. Mohammed and I. Tashev, "Unsupervised deep representation learning to remove motion artifacts in free-mode body sensor networks," in *Proceedings of the 14th IEEE International Conference on Wearable and Implantable Body Sensor Networks*, pp. 183–188, 2017.

[13] X. Xiao and S. Zarar, "Machine learning for placement-insensitive inertial motion capture," in *Proceedings on the IEEE International Conference on Robotics and Automation*, pp. 6716–6721, IEEE, 2018.

[14] T. Zimmermann, B. Taetz, and G. Bleser, "IMU-to-segment assignment and orientation alignment for the lower body using deep learning," *Sensors*, vol. 18, no. 1, p. 302, 2018.

[15] T. Bikmukhametov and J. Jäschke, "Combining machine learning and process engineering physics towards enhanced accuracy and explainability of data-driven models," *Computers & Chemical Engineering*, vol. 138, p. 106834, 2020.

[16] Y. Guan and T. Plötz, "Ensembles of deep LSTM learners for activity recognition using wearables," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, vol. 1, no. 2, pp. 1–28, 2017.

[17] R. Khorrambakht, H. Damirchi, and H. Taghirad, "Preintegrated imu features for efficient deep inertial odometry," 07 2020.

[18] M. A. Esfahani, H. Wang, K. Wu, and S. Yuan, "OriNet: Robust 3-D orientation estimation with a single particular IMU," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 399–406, 2020.

[19] D. Weber, C. Gühmann, and T. Seel, "Neural networks versus conventional filters for inertial-sensor-based attitude estimation," 2020.

[20] M. Brossard, A. Barrau, and S. Bonnabel, "AI-IMU dead-reckoning," *IEEE Transactions on Intelligent Vehicles*, 2020.

[21] M. Brossard, A. Barrau, and S. Bonnabel, "Rins-w: Robust inertial navigation system on wheels," pp. 2068–2075, 11 2019.

[22] M. Brossard, S. Bonnabel, and A. Barrau, "Denoising imu gyroscopes with deep learning for open-loop attitude estimation," *IEEE Robotics and Automation Letters*, 06 2020.

[23] F. J. Wouda, M. Giuberti, G. Bellusci, and P. H. Veltink, "Estimation of full-body poses using only five inertial sensors: An eager or lazy learning approach?," *Sensors*, vol. 16, p. 2138, 2016.

[24] D. Pavllo, D. Grangier, and M. Auli, "Quaternet: A quaternion-based recurrent model for human motion," in *Proceedings of the 29th British Conference on Machine Vision*, 2018.

[25] Y. Huang, M. Kaufmann, E. Aksan, M. J. Black, O. Hilliges, and G. Pons-Moll, "Deep inertial poser: Learning to reconstruct human pose from sparse inertial measurements in real time," *ACM Transactions on Graphics*, vol. 37, no. 6, pp. 1–15, 2018.

[26] T. von Marcard, B. Rosenhahn, M. J. Black, and G. Pons-Moll, "Sparse inertial poser: Automatic 3D human pose estimation from sparse IMUs," *Computer Graphics Forum*, vol. 36, no. 2, pp. 349–360, 2017.

[27] K. Feng, J. Li, X. Zhang, C. Shen, Y. Bi, T. Zheng, and J. Liu, "A new quaternion-based Kalman filter for real-time attitude estimation using the two-step geometrically-intuitive correction algorithm," *Sensors*, 2017.

[28] A. Harindranath and M. Arora, "MEMS IMU sensor orientation algorithms-comparison in a simulation environment," in *Proceedings of the International Conference on Networking, Embedded and Wireless Systems*, pp. 1–6, 2018.

[29] 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*, pp. 1–7, 2011.

[30] Y. Xu, J. Quo, and Y. Guan, "EKF based multiple-mode attitude estimator for quadrotor using inertial measurement unit," in *Proceedings of the 36th Chinese Control Conference*, 2017.

[31] M. Kok, J. D. Hol, and T. B. Schön, "Using inertial sensors for position and orientation estimation," *Foundations and Trends in Signal Processing*, vol. 11, no. 1-2, pp. 1–153, 2017.

[32] Q. A. Dugne–Hennequin, H. Uchiyama, and J. P. S. Do Monte Lima, "Understanding the behavior of data-driven inertial odometry with kinematics-mimicking deep neural network," 2021.

[33] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, pp. 107–116, 1998.

[34] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: continual prediction with lstm," in *Proceedings of the Ninth International Conference on Artificial Neural Networks*, vol. 2, pp. 850–855, 1999.

[35] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[36] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *CoRR*, 2014.

[37] G. Terzakis, P. Culverhouse, G. Bugmann, S. Sharma, and R. Sutton, "A recipe on the parameterization of rotation matrices for non-linear optimization using quaternions," tech. rep., 2012.

[38] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, 2006.

[39] J. D. Hol, *Sensor fusion and calibration of inertial sensors, vision, ultra-wideband and GPS*. Dissertation no. 1368, Linköping University, Linköping, Sweden, 2011.

[40] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models," *CoRR*, 08 2017.

[41] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, pp. 1058–1066, 2013.

[42] "tf.keras.layers.lstm." https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM, 2021. Accessed: 2021-02-04.

[43] A. M. Sabatini, "Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation," *Sensors*, vol. 11, no. 10, pp. 9182–9206, 2011.

[44] M. Pedley, "High precision calibration of a three-axis accelerometer," 2013.

[45] C. Chen, X. Lu, A. Markham, and N. Trigoni, "Ionet: Learning to cure the curse of drift in inertial odometry," in *Proceedings of the 32'th AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[46] "Orientation output specifications." https://base.xsens.com/knowledgebase/s/article/Orientation-output-specifications-1605869709222, feb 2021. Accessed: 2021-03-12.

[47] D. Q. Huynh, "Metrics for 3d rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, 2009.

[48] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, "Supervised machine learning," 2020.

[49] "Mtw awinda." https://www.xsens.com/products/mtw-awinda, 2021. Accessed: 2021-04-28.