Pedestrian Dead Reckoning

using data-driven and physics-informed machine learning

L. Ligthart

Master of Science Thesis

TU Delft Delft University of Technology

Delft Center for Systems and Control

Pedestrian Dead Reckoning

using data-driven and physics-informed machine learning

Master of Science Thesis

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

L. Ligthart

August 21, 2025

Faculty of Mechanical Engineering (ME) · Delft University of Technology

The work in this thesis was supported by Leica Geosystems. Their cooperation is hereby gratefully acknowledged.

Abstract

Many applications such as indoor navigation, search and rescue operations, and surveying, require accurate localization in challenging environments. In environments where absolute positioning (using for example Global Navigation Satellite System (GNSS) receivers) fails due to dense obstruction of the required signals, positioning algorithms rely solely on dead-reckoning, often using measurements from an Inertial Measurement Unit (IMU). This dead-reckoning process suffers from integration drift due to the accumulation of errors in the sensor measurements. Pedestrian Dead-Reckoning (PDR) algorithms can reduce this integration drift in cases where the IMU is held by a walking person, by leveraging patterns from the periodic walking motion. This thesis investigates a state-of-the-art PDR algorithm from *Liu et al.* [1] that combines an Extended Kalman Filter (EKF) with a velocity-predicting neural network that corrects the filter in the measurement update to mitigate integration drift. The focus is on finding if its performance can be improved by adapting the neural network in the algorithm. First, adaptations of the network's parameters have been experimented with to investigate their effect on the algorithm's accuracy and search for a bottleneck that limits it. This bottleneck was found to be a bias in the predicted velocity by the network, as this violates the EKF assumption that the error in the measurement update is distributed with zero mean. The second part investigates how using ideas from the Physics-Informed Neural Network (PINN) as an alternative to the data-driven neural network in the PDR algorithm affects its performance. Four network architectures have been trained using a loss function that includes a penalty for errors in the physics of the predicted velocities of the system. Training these PINN-inspired networks required a much longer training time. The results show that using this physics loss does not show significant improvements in the accuracy of the PDR algorithm, as the bias in the velocity prediction is not addressed by the physics loss. This thesis concludes that the main limitation of the neural network in the PDR algorithm by *Liu et al.* [1] is a bias in the predicted velocity, and that a PINN is unable to provide significant improvements in the accuracy of the algorithm, whilst costing much more computational resources to train. It is therefore not recommended to use a PINN in this PDR algorithm, and the optimal configuration of the network was found to be a slightly adapted version of the network used by *Liu et al.*, using IMU and device tilt data sampled at 50 Hz as network input. Future research could focus on understanding the origin of the bias in velocity predictions and to mitigate it once its source is known.

Table of Contents

Preface

With this thesis, I am concluding my Master of Science in Systems & Control at the TU Delft faculty of Mechanical Engineering. The topic of this thesis was proposed by Henri van Bavel, Positioning Algorithm Engineer at Leica Geosystems and a former graduate of Manon Kok. When I was working with IMUs in the Eco-Runner student team of the TU Delft, I quickly learned of the struggles in IMU dead reckoning. This experience, along with my interests in deep learning through my education, led me to the decision to work on this project to contribute towards the research in dead reckoning algorithms.

I wish to thank my supervisor dr. Manon Kok for her valuable insights to this thesis and flexibility to my planning. I also want to thank my daily supervisor and contact with Leica Geosystems, Henri van Bavel, for taking the time to meet and discuss my thesis twice a week and even discuss questions I had outside of this time. I would like to thank both of them for making the collaboration of the TU Delft with Leica Geosystems available and providing me with the opportunity to do my graduation project at this company. Furthermore, I want to thank Henri's colleagues; Stefan Gächter for adapting and providing the simulation software of the PDR algorithm, and David Gräff for giving me access to a workstation to speed up my training process.

Delft, University of Technology — Liam Ligthart
August 21, 2025

Chapter 1

Introduction

1-1 Background and Motivation

1-1-1 Localization methods

The technology to provide accurate localization has become an indispensable part of modern daily life. Local news distribution, navigation during daily commutes, and smart home automation are all facilitated by the development of localization technologies [2]. Various techniques exist for determining the location of a device, which can generally be classified into two primary categories [3].

The first category, referred to as 'absolute' localization, relies on sensors and signals to determine a device's location relative to an origin. Examples of absolute measurement systems include Global Navigation Satellite System (GNSS) receivers and Ultra-Wideband (UWB) sensors [4]. While these methods can achieve accurate localization, their dependence on these external signals renders their availability dependent on the environment; GNSS signals are often weak or unavailable indoors or in areas with dense buildings or trees [5], and UWB positioning requires an environment with pre-placed signal emitters and receivers at known locations [6].

The second category of localization techniques, 'odometric' localization, utilizes one or more internal sensors of a device to determine its location, which enables usage in any environment [3]. These techniques frequently use an Inertial Measurement Unit (IMU) sensor, a combination of an accelerometer, gyroscope and sometimes magnetometer [7]. This device measures acceleration, rotational velocity, and the magnetic field strength in three directions. IMUs are generally compact, cost-effective, and capable of measuring at high sampling frequencies, allowing for rapid position updates. The position of an IMU-carrying device can be computed using a process known as dead-reckoning (Figure 1-1), where the accelerometer and gyroscope measurements are integrated from an initial position and orientation (pose) to determine the current pose. However, the sensor measurements are not perfectly accurate due to noise and bias. Integrating these measurements causes the errors to accumulate, leading to significant inaccuracies in the calculated pose, a phenomenon called integration drift [7]. Often, measurements from both categories are combined in an Extended Kalman Filter (EKF) to mitigate

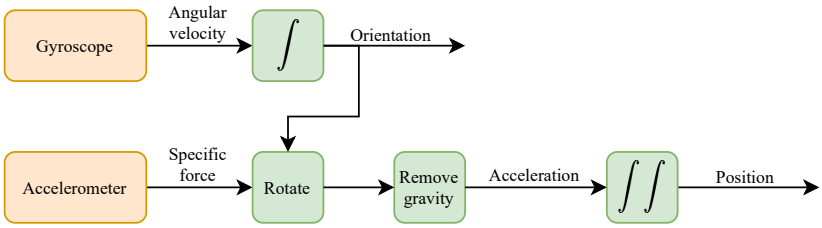

Figure 1-1: The process of dead-reckoning using accelerometer and gyroscope measurements, to obtain position and orientation estimates of a device. Image adapted from [7].

this integration drift and provide a smoother estimate than is achieved by just relying on the absolute measurements. This process is called sensor fusion [4]. Sensor fusion of absolute and odometric sensors still requires the environment to support the usage of external signals, which may not always be feasible.

Numerous applications require accurate localization techniques, including indoor navigation, search and rescue operations, and surveying. Such applications often encounter environments in which absolute positioning systems are unreliable or unavailable, making accurate localization a challenging problem. This thesis aims to enhance the accuracy of IMU-based localization algorithms. It is supported by Leica Geosystems, a company that develops, among other things, surveying equipment.

1-1-2 Surveying

Surveying involves the precise measurement of positions to either create an accurate map of an area or to transfer points of interest from a map to the area [8]. Modern surveying equipment uses absolute positioning, with two common examples being total station surveying and GNSS surveying (see Figure 1-2). The first example makes use of a separate stationary device called a total station, which measures the position of a small, handheld device called a surveying pole. A surveyor can carry the pole to a point of interest while the total station automatically tracks and records the position of the prism on the top of the pole. In GNSS surveying, the surveying pole uses signals from GNSS satellites to compute its absolute position without the need for a separate total station. Many surveying poles also contain an IMU, enabling the use of sensor fusion to obtain accurate and high-frequency position updates. Other sensors like a camera can be added to the surveying pole such that techniques as Simultaneous Localization And Mapping (SLAM) [9] can be used, but this thesis focuses on devices like Leica's AP20 (surveying pole for total station surveying) and GS18T (surveying pole for GNSS surveying) that only carry an IMU sensor for odometric localization.

Both total station and GNSS surveying require certain environmental conditions to work correctly. A total station requires a line of sight with the prism mounted on top of the surveying pole [8] (see Figure 1-2). In areas with many obstructions, maintaining this line of sight between the two devices can be difficult. GNSS surveying requires a strong signal from the GNSS satellites. Areas with large buildings or trees can attenuate and reflect these signals, reducing the accuracy of the GNSS location, and indoor environments often do not receive a strong enough signal at all [5], making GNSS surveying extremely difficult. When entering such areas, one has to rely on odometric localization using the IMU sensor in the

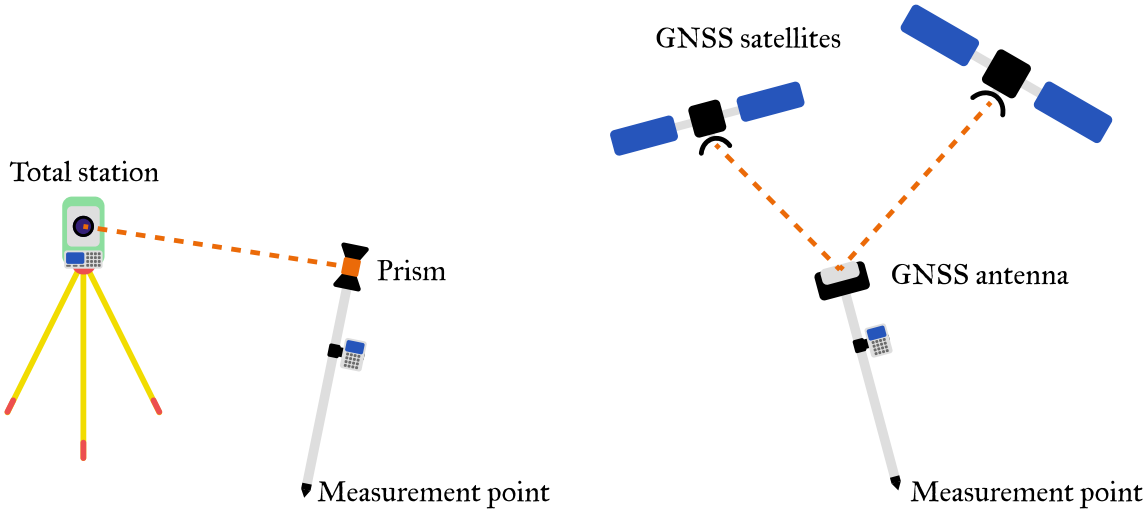

Figure 1-2: Left: Total station surveying. Right: GNSS surveying. The position of the bottom of the pole is the position of interest for the surveyor.

surveying pole. Although IMU dead reckoning algorithms have seen significant improvements over the last decades [1], their performance still is not accurate enough to be used in surveying (which requires an accuracy at centimeter level [8]). Consequently, total station and GNSS surveying is restricted to areas that allow line of sight with a total station or adequate GNSS signals. By aiding research in odometric positioning algorithms, Leica hopes to accelerate the improvement of the accuracy of these algorithms to extend the operational range of the total station and GNSS surveying instruments beyond these current limitations.

1-1-3 Pedestrian Dead Reckoning

In many applications, including surveying, the device to be tracked is often held by a walking person. This introduces motion characteristics that are limited by several constraints and symmetries that can be leveraged to improve the dead reckoning algorithm and reduce the integration drift [10]. Dead reckoning of a device being held by a walking person is commonly referred to as Pedestrian Dead-Reckoning (PDR), a topic that has been extensively studied over the past decades [2, 3, 11, 12].

Traditional PDR algorithms typically employ heuristic methods to constrain the pedestrian motion and thereby reduce integration drift [12]. Given that pedestrian motion has minimal vertical displacement during normal walking, many PDR algorithms restrict the motion to a two-dimensional horizontal plane [2, 11, 12, 13]. These algorithms often rely on a combination of step detection algorithms, orientation estimators, and step length models, to update the predicted position incrementally with each step taken [12, 13]. Other algorithms have been developed for the special case where the sensor is mounted to a person's foot, and use so-called zero-velocity updates (ZUPTs). ZUPTs use the stationary phase when the foot touches the ground during walking to correct velocity estimates and compensate the IMU bias [11, 14, 15].

However, these heuristic methods and sensor placement requirements limit the use of these algorithms to very specific circumstances [16] and cannot directly be applied to handheld

devices like the surveying pole. In an approach to make PDR algorithms more generalizable to different sensor placement locations, PDR algorithms using deep learning have been introduced [16]. When trained on a sufficiently large and diverse dataset, deep learning models can give accurate predictions of displacement or velocity solely based on pattern recognition [16]. Early algorithms relied completely on a neural network to update a horizontal position every few seconds from a predicted displacement by a neural network [17, 18]. Recent advancements in this field (for example by *Liu et al.* [1] and *Bajwa et al.* [19]) have improved this performance by using a two-component method, combining an EKF and a neural network. The EKF propagates the state using IMU data and dead-reckoning, while the neural network predicts velocity updates based on the learned data patterns, which are then used in the measurement update of the EKF to compensate for the integration drift.

1-2 Scope and Research Questions

The goal of this research is to explore the possibility of improving the current state of the art PDR algorithms from literature, to allow more accurate position estimation of pedestrians in areas which do not allow absolute position measurements. In particular, the algorithm from [1], which combines an EKF with a neural network, is investigated in this thesis. From the two components in this algorithm, the EKF is already decades old and extensive research has been done on its applicability [20], but deep learning with neural networks has seen huge developments over the previous years and is still seeing improvements [21]. Therefore, it is hypothesized that an improvement of the current PDR algorithm using an EKF and a neural network is more likely to be found in the neural network than the EKF. As such, this thesis focuses its scope on improving the current state of the art PDR algorithm from *Liu et al.* [1] by adapting the neural network.

This leads to the main research question of this thesis:

> *Can the accuracy of an existing state of the art PDR algorithm, using a combination of an EKF and neural network, be improved by adapting the neural network?*

This thesis is split into two parts to answer this question. The first part will test the baseline performance of the algorithms by *Liu et al.* [1] and *Bajwa et al.* [19] on the provided dataset of surveying data by Leica, and explore the limiting factors of the performance of the algorithm by experimenting with applying small changes to the existing network architecture and pre-processing method of its input data. This part will answer the research question

> *What influence do small adaptations of the neural network in an existing PDR algorithm from literature have on its accuracy, and is there a bottleneck to be found that limits the accuracy of the algorithm?*

Data-driven neural networks, like the one used in the algorithms by *Liu et al.* [1] and *Bajwa et al.* [19], tend to return accurate predictions for data similar to what it was trained on, but can be physically inconsistent. It has been shown that in other fields, this can lead to poor generalization of the network on data that is different than what it was trained on [22].

Raissi et al. [23] introduced the Physics-Informed Neural Network (PINN), a method to train a neural network not only on an observed dataset, but also on the underlying physical equations that govern the system which the network models. They have shown that in certain applications, a PINN could return more physically accurate predictions than a solely data-driven neural network, and therefore generalize better on unseen data. The second part of this thesis will explore how the ideas of a PINN can be applied to the neural network in the PDR algorithm by *Liu et al.* [1] and to test if using a PINN can lead to an improvement of the PDR algorithm in terms of accuracy in the predicted position. The second research question is therefore

> *Can the data-driven neural network in an existing PDR algorithm be adapted to accommodate for a physics-informed loss term, and does this adaptation lead to increased accuracy of the PDR algorithm?*

In order to aid this research, Leica has provided a dataset consisting of 25 hours of IMU data along with the estimated position, velocity, and orientation. This data was captured by 25 different persons using the AP20 surveying pole with the MS60 total station. The IMU is located in the AP20 surveying pole, and the position of the surveying pole is measured at 20 Hz using the MS60 total station. The velocity and orientation are estimated by an error-state EKF [24] using an internal state vector of the velocity, orientation, and IMU biases. The position of the pole is estimated by integrating the EKF velocity estimations. These estimations of position, velocity, orientation, and IMU biases are considered as the ground truth of this dataset.

1-3 Organization

This thesis is structured as followed: Chapter 2 will give an overview of the research related to this topic, which includes the IMU measurement model, details on the state of the art AI-aided PDR algorithms, and the theory behind the PINN. Chapter 3 is aimed towards answering the first research question, and after setting a baseline performance of the current AI-aided PDR algorithms from literature on Leica's dataset, attempts a variety of experiments on adapting the specifications of the neural network used in the AI-aided PDR algorithm. Chapter 4 will discuss the methodology and results to answer the second research question by applying the PINN to a variety of network architectures. Chapter 5 forms a conclusion on the main question of this research, and discusses recommendations for potential future research.

Chapter 2

Related Work

This chapter outlines the research this thesis builds upon. It begins by introducing the Inertial Measurement Unit (IMU) and its role in Pedestrian Dead-Reckoning (PDR), followed by a review of state-of-the-art methods that combine an Extended Kalman Filter (EKF) with neural networks to predict measurements used in the EKF measurement update step. Moreover, the theory behind the Physics-Informed Neural Network (PINN) is introduced, as its usage in the state-of-the-art PDR algorithm is the main focus of this thesis.

2-1 The IMU Measurement model

The IMU as used in this thesis provides six axes of measurements: three axes of accelerometer measurements of the specific force of the sensor, and three axes of gyroscope measurements of the sensor's rotational velocity. It is essential to specify the coordinate frame in which these measurements are expressed. In this thesis, three commonly used coordinate frames in PDR are distinguished [7, 1, 19]:

Body frame b [7] The coordinate frame of the IMU itself. Its origin is located at the center of the IMU sensor, and its three orthogonal axes are fixed relative to the sensor's orientation. As a result, this frame moves and rotates with the sensor. The raw IMU measurements are expressed in this frame.

Local navigation frame l [7] The local geographic frame in which the position and orientation of the body frame b are estimated. The origin of l is typically fixed to the earth, with its z-axis aligned with the local gravity vector.

Gravity-aligned body frame g [1, 19] The gravity-aligned body frame is obtained by rotating the body frame b such that the roll and pitch angle are zero with respect to the local frame l. The only remaining rotation relative to the local frame is the yaw around the z-axis. The origin of this frame is located at the origin of the body frame b.

The coordinate frame in which a vector is represented is indicated by a superscript: x^b. To express the vector in a different frame, its values can be transformed using a rotation matrix or unit quaternion [25]. The matrix $R^{lb} \in SO(3)$ is defined as the 3x3 rotation matrix that transforms a vector from its expression in the body frame b to its expression in the local frame l. Similarly, $q^{lb} \in \mathbb{R}^4$ is a unit quaternion that performs the same rotation.

The measurements obtained from an IMU can be described by the following models [7]:

$$y^b_{a,t} = R^{bl}_t(a^l_t - g^l) + \delta^b_{a,t} + e^b_{a,t}, \tag{2-1a}$$
$$y^b_{\omega,t} = \omega^b_{lb,t} + \delta^b_{\omega,t} + e^b_{\omega,t}, \tag{2-1b}$$

In these models, $y^b_{a,t}$ denotes the three-dimensional accelerometer measurement in the body frame, representing the specific force a^l_t acting on the sensor plus the gravity vector g^l, accelerometer bias $\delta^b_{a,t}$, and noise $e^b_{a,t}$. The term $y^b_{\omega,t}$ corresponds to the gyroscope measurement of the angular velocity $\omega^b_{lb,t}$ of the body frame b with respect to the local frame l, also expressed in the body frame and with the addition of gyroscope bias $\delta^b_{\omega,t}$ and noise $e^b_{\omega,t}$. The biases $\delta^b_{a,t}$ and $\delta^b_{\omega,t}$ are typically modeled as a constant, or slowly time-varying using a random walk process. The noise values $e^b_{a,t}$ and $e^b_{\omega,t}$ are often modeled as a zero-mean Gaussian noise on the measurement readings [7]. The subscript t represents the time index at which the measurement is taken.

2-1-1 Pose Estimation by Dead Reckoning using an IMU

To model the evolution of the state of an object equipped with an IMU, a discrete-time dynamic model can be formulated based on the IMU measurements [7]. This model uses the principle of dead reckoning, in which position and orientation are estimated through integration of accelerometer and gyroscope data over time.

The translational dynamics (of the position p^l_t and velocity v^l_t) as a function of acceleration a^l_t in the local frame l, are described as

$$p^l_{t+1} = p^l_t + \Delta\tau\ v^l_t + \frac{\Delta\tau^2}{2} a^l_t, \tag{2-2a}$$
$$v^l_{t+1} = v^l_t + \Delta\tau\ a^l_t, \tag{2-2b}$$

where $\Delta\tau$ denotes the time between two successive updates. The rotational dynamics (of the orientation) can be represented either as a unit quaternion or rotation matrix. The update equations as a function of rotational velocity $\omega^b_{lb,t}$ are given by

$$q^{lb}_{t+1} = q^{lb}_t \odot \exp_{\mathrm{q}}\left(\frac{\Delta\tau}{2}\omega^b_{lb,t}\right) \qquad \text{or} \qquad R^{lb}_{t+1} = R^{lb}_t \exp_{\mathrm{R}}\left(\Delta\tau\ \omega^b_{lb,t}\right), \tag{2-3}$$

where $\exp_{\mathrm{q}}(\eta)$ and $\exp_{\mathrm{R}}(\eta)$ are the exponential maps that convert a three dimensional rotation vector η into a quaternion q or a rotation matrix R, respectively [7]. The operation $\odot$ represents the quaternion multiplication [7].

By combining the state dynamics in (2-2) and (2-3) with the IMU measurement model in (2-1), the complete state dynamics are described by

$$\begin{bmatrix} p_{t+1}^l \\ v_{t+1}^l \\ q_{t+1}^{lb} \end{bmatrix} = \begin{bmatrix} p_t^l + \Delta\tau v_t^l + \frac{\Delta\tau^2}{2}(R_t^{lb}(y_{a,t}^b - \delta_{a,t}) + g^l + e_{p,a,t}) \\ v_t^l + \Delta\tau(R_t^{lb}(y_{a,t}^b - \delta_{a,t}) + g^l + e_{v,a,t}) \\ q_t^{lb} \odot \exp_{\mathrm{q}}\left(\frac{\Delta\tau}{2}(y_{\omega,t}^b - \delta_{\omega,t} - e_{\omega,t})\right) \end{bmatrix}. \tag{2-4}$$

For notational clarity, the superscript notations for the vector's coordinate frame has been dropped on the bias δ and noise e. The quaternion representation q_{t+1}^{nb} can be substituted with a rotation matrix R_{t+1}^{nb} if preferred, using the equivalent equation for the rotating dynamics using a rotation matrix in (2-3). Furthermore, the accelerometer and gyroscope biases may be included as states and could be propagated using a random walk process [7, 1].

2-1-2 Pose Estimation using Sensor Fusion of an IMU and an Absolute Position Sensor

The dead reckoning state propagation in (2-4) is prone to integration drift when used for more than a few seconds [7]. This drift arises due to the continuous accumulation of sensor bias and noise at each step in the dead reckoning algorithm, and creates and increasingly inaccurate estimate of the system's pose. To mitigate this integration drift, an additional sensor that measures absolute position can be added to the system, and its measurements $y_{p,t}^l$ can be fused in an EKF [4] (see Figure 2-1). In an EKF, the dead reckoning model in (2-4) that uses IMU data is incorporated in the time update to propagate the system's pose. The accumulated integration drift is corrected through measurement updates that use the position measurements from the absolute position sensor. Examples of this position sensor are Ultra-Wideband (UWB) systems [6], a Global Navigation Satellite System (GNSS) receiver [26], or a vision-based solutions like Simultaneous Localization And Mapping (SLAM) [9] using a camera to track points of interest in the surrounding. [27]. These extra sensors enable

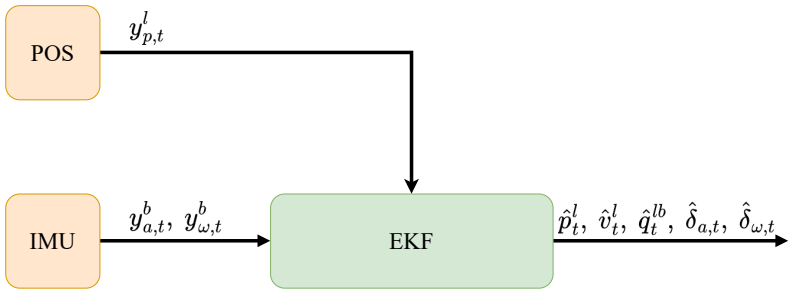

Figure 2-1: Ordinary EKF setup for a position tracking algorithm. IMU measurements are used in the time update, whilst an absolute position sensor is used in the measurement update to correct the EKF estimates. [4, 6, 26, 27]

accurate pose estimation by correcting the dead-reckoning prediction, reducing the effect of integration drift each measurement update.

2-2 Pedestrian Dead Reckoning using an Extended Kalman filter with an Artificial Neural Network

In PDR, the EKF using an absolute measurement in Figure 2-1 cannot be applied due to the absence of an absolute position sensor. An algorithm by *Liu et al.* [28] called TLIO demonstrated the potential of using a modified EKF setup, where the missing external sensor measurements are replaced with predictions generated by a neural network, as shown in Figure 2-2. The network receives one second of IMU data as input, which is rotated to the local frame l using the estimated orientation from the EKF. This results in an input matrix of shape $(6, T)$ with T the amount of time samples, dependent on the sampling rate of the IMU. The network returns the predicted positional displacement during the time of the input window along with its uncertainty. These predicted measurements can be used in the measurement update of the EKF as described in [1], correcting the pose which was integrated in the time update.

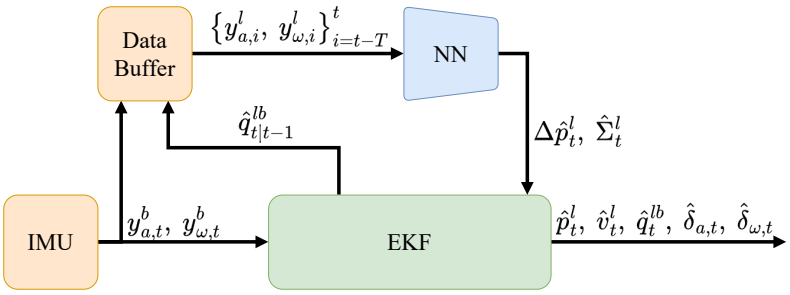

Figure 2-2: Global architecture of the TLIO algorithm [1]. The algorithm is governed by two components: an EKF and a neural network that produces measurement predictions to correct the EKF state. Image adapted from [1].

The neural network used in TLIO is a one-dimensional variant of the ResNet architecture by *He et al.* [29], which was originally created for image processing. The network, as depicted in Figure 2-3, uses several residual blocks that downsample the input data along the temporal dimension while increasing the number of features on the other dimension through the use of multiple convolution kernels. After the residual downsampling, two blocks consisting mostly of fully connected layers return a vector in $\mathbb{R}^3$ representing the predicted displacement over the time window, $\Delta\hat{p}^l_t$, and its predicted logarithmic uncertainty $\hat{u}^l$. This logarithmic uncertainty is converted to a covariance matrix by [1]

$$\hat{\Sigma}_{\Delta\hat{p}} = \text{diag}(e^{2\hat{u}^l_x}, e^{2\hat{u}^l_y}, e^{2\hat{u}^l_z}). \tag{2-5}$$

The input of the network in TLIO is 1 second of accelerometer and gyroscope data sampled at 200 Hz and represented in the local frame l. Since the raw IMU signals $y^b_{a,t}$ and $y^b_{\omega,t}$ are given in the body frame b, they must be rotated to the local frame l using the estimated

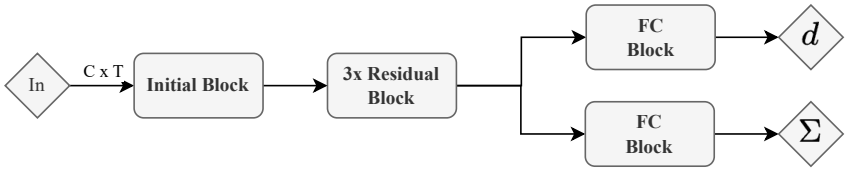

Figure 2-3: Global structure of the model architecture as used in TLIO [1], adapted from ResNet [29]. The model consists of an initial block, three residual blocks, and fully connected blocks to convert an input matrix of shape (C, T) to a displacement and uncertainty prediction. Detailed structures of the blocks are shown in Figure A-1 in Appendix A-1.

orientation from the EKF, $\hat{q}^{lb}_T$. Past orientations are obtained by integrating the current orientation backwards in time using the gyroscope measurements. This process of backwards integration is described in Algorithm 1.

Algorithm 1 Preprocessing of the input data for the neural network of TLIO

Input: $y^b_{a,0:T}, \quad y^b_{\omega,0:T}, \quad \hat{q}^{lb}_{T|T-1}$
$\bar{y}^b_{a,0:T} = \begin{bmatrix} 0 & (y^b_{a,0:T})^\top \end{bmatrix}^\top$
$\bar{y}^b_{\omega,0:T} = \begin{bmatrix} 0 & (y^b_{\omega,0:T})^\top \end{bmatrix}^\top$
for $t = T, T-1, \ldots, 2, 1$ **do**
  $\Delta q_t \leftarrow \exp_{\mathrm{q}}(\Delta\tau \; y^b_{\omega,t})$
  $\hat{q}^{lb}_{t-1} \leftarrow \Delta q_t \odot \hat{q}^{lb}_t$
  $\bar{y}^l_{a,t-1} \leftarrow \hat{q}^{lb}_{t-1} \odot \bar{y}^b_{a,t-1} \odot (\hat{q}^{lb}_{t-1})^{\mathrm{c}}$
  $\bar{y}^l_{\omega,t-1} \leftarrow \hat{q}^{lb}_{t-1} \odot \bar{y}^b_{\omega,t-1} \odot (\hat{q}^{lb}_{t-1})^{\mathrm{c}}$
end for
$y^l_{a,0:T} = \bar{y}^l_{a,0:T}[1:3]$
$y^l_{\omega,0:T} = \bar{y}^l_{\omega,0:T}[1:3]$
Returns: $y^l_{a,0:T}, \quad y^l_{\omega,0:T}$

Training of the TLIO network is done in two stages. Initially, a mean squared error of the predicted displacement $\Delta\hat{p}^l$ and its ground-truth value Δp^l over the entire dataset used for training, consisting of N samples, is used for 10 epochs to encourage convergence of the displacement prediction:

$$L_{\mathrm{MSE}} = \frac{1}{N}\sum_{i=1}^{N} ||\Delta\hat{p}^l_i - \Delta p^l_i||_2^2. \tag{2-6}$$

Then, a maximum likelihood loss is employed to optimize both the displacement prediction and its uncertainty:

$$L_{\mathrm{ML}} = \frac{1}{N}\sum_{i=1}^{N}\left(\log\det(\hat{\Sigma}_{\Delta\hat{p},i}) + ||\Delta\hat{p}^l_i - \Delta p^l_i||^2_{\hat{\Sigma}_{\Delta\hat{p},i}}\right). \tag{2-7}$$

During training, artificial perturbations are applied to the training data to improve generalization. A random bias is added to the IMU data, and the input is perturbed with small

rotations about random horizontal axes to simulate errors in the estimated gravity direction. Additionally, random yaw rotations are applied to enforce rotational invariance around the vertical axis. These perturbations are done separately every epoch. According to [1], these perturbations should reduce the effect of the positive feedback loop created by pre-processing the input to the neural network with the orientation estimate from the EKF, which itself is using the network output in its measurement update.

Bajwa et al. [19] developed an adaptation of TLIO, called DIVE. Their research focused on improving the accuracy of the algorithm from *Liu et al.* [1] for dead reckoning of quadcopter drones. Its general structure is shown in Figure 2-4. Unlike TLIO, which predicts a displacement, DIVE uses velocity predictions to update the EKF estimates in the measurement update. The network architecture is the same 1D ResNet as used in TLIO (see Figure 2-3). However, instead of using accelerometer and gyroscope data directly, DIVE replaces the gyroscope measurements with the three dimensional rotation vector, corresponding to the rotation from the body frame b to a gravity-aligned frame g. The g frame is redefined for each input window by decomposing the rotation quaternion q^{lb} into $q^{lg} \odot q^{gb}$ [19]. q^{lg} represents a rotation around the vertical axis of the l frame, and q^{gb} contains the remaining rotation. Moreover, the gravity vector $g^l = g^g$ is subtracted from the accelerometer measurements such that only the specific force remains. These preprocessing steps are detailed in Algorithm 2.

The same loss functions, (2-6) and (2-7), were used to train the model, with the distinction that these use the predicted and ground truth velocities rather than displacements:

$$L_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N} ||\hat{v}_i^l - v_i^l||_2^2, \tag{2-8}$$

$$L_{\text{ML}} = \frac{1}{N}\sum_{i=1}^{N} \left(\log\det(\hat{\Sigma}_{\hat{v},i}) + ||\hat{v}_i^l - v_i^l||^2_{\hat{\Sigma}_{\hat{v},i}}\right). \tag{2-9}$$

Using the gravity-aligned frame g to represent the input to the neural network in, the independence of the yaw angle of the device's orientation is maintained, as this frame rotates along the gravity axis with the device itself, therefore always representing the input data relative to the device's yaw angle. Random biases and gravity perturbations are also applied during training to ensure generalization of the network, just like in [1].

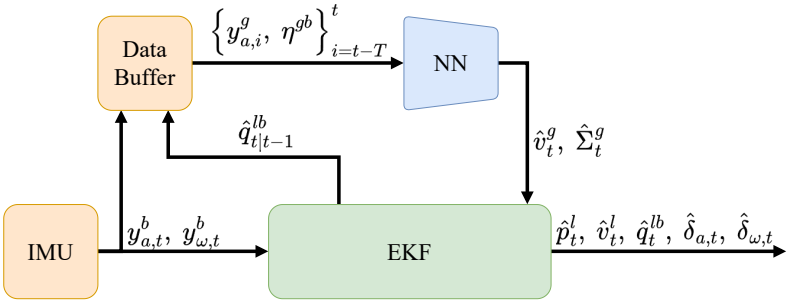

Figure 2-4: Block diagram of the DIVE algorithm [19]. Like TLIO [1], DIVE uses an EKF that is corrected by measurement predictions from a neural network, except the predicted measurements are velocities instead of position displacement, and the input of the network is pre-processed using Algorithm 2. Image adapted from [19].

Algorithm 2 Preprocessing of the input data for the neural network of DIVE

Input: $y^b_{a,0:T}, \quad y^b_{\omega,0:T}, \quad \hat{q}^{lb}_{T|T-1}$

$\bar{y}^b_{a,0:T} = \begin{bmatrix} 0 & (y^b_{a,0:T})^\top \end{bmatrix}^\top$

$\bar{y}^b_{\omega,0:T} = \begin{bmatrix} 0 & (y^b_{\omega,0:T})^\top \end{bmatrix}^\top$

$\hat{q}^{lg}_{T|T-1} \leftarrow \text{extract\_yaw}[\hat{q}^{lb}_{T|T-1}]$

$\hat{q}^{gb}_{T|T-1} \leftarrow (\hat{q}^{lg}_{T|T-1})^{\text{c}} \odot \hat{q}^{lb}_{T|T-1}$

$\eta^{gb}_T \leftarrow \log_q(\hat{q}^{gb}_{T|T-1})$

for $t = T, T-1, \ldots, 2, 1$ **do**

$\Delta q_t \leftarrow \exp_{\text{q}}(\Delta\tau \; y^b_{\omega,t})$

$\hat{q}^{lb}_{t-1} \leftarrow \Delta q_t \odot \hat{q}^{lb}_t$

$\hat{q}^{gb}_{t-1} \leftarrow (\hat{q}^{lg}_T)^{\text{c}} \odot \hat{q}^{lb}_{t-1}$

$\bar{y}^g_{a,t-1} \leftarrow \hat{q}^{gb}_{t-1} \odot \bar{y}^b_{a,t} \odot (\hat{q}^{gb}_{t-1})^{\text{c}} - g^g$

$\eta^{gb}_{t-1} \leftarrow \log_q(\hat{q}^{gb}_{t-1})$

end for

$y^g_{a,0:T} = \bar{y}^g_{a,0:T}[1:3]$

Returns: $y^g_{a,0:T}, \quad \eta^{gb}_{0:T}$

2-3 The Physics-Informed Neural Network

An increasingly popular method of including information on the physics of a system, for example a known Partial Differential Equation (PDE), into the training process of a neural network is the Physics-Informed Neural Network (PINN). The method of the PINN was published by *Raissi et al.* in 2019 [23]. The authors showed the competence of a PINN by letting it solve basic PDEs. Not only do these PINNs require less data to train a model similarly accurate to a solely data-driven network, they are also shown to be more generalizable to different conditions of the system [22].

Following [23], the formal definition of the PINN is as follows. Define $\hat{u}(x;\theta) : X \to \mathcal{U}$ as the neural network that predicts a physical variable $u \in \mathcal{U}$ from some input $x \in X$, with network parameters θ. Let $\mathcal{D}[\,\cdot\,]$ describe some (differential) operator that describes the physics of the system as

$$\mathcal{D}[u] = f(x), \tag{2-10}$$

with $f(x)$ some external driving force. For example, this can be the differential equation of a dampened harmonic oscillator [30],

$$\mathcal{D}[u] = m\frac{\delta^2 u}{\delta x^2} + \mu\frac{\delta u}{\delta x} + ku = 0, \tag{2-11}$$

but any equation that describes the physics of u is possible. Define $\hat{f}(x;\theta)$ as the physics-informed neural network that applies a sequence of differentiations on the network $\hat{u}(x;\theta)$, forming $\mathcal{D}[\hat{u}(x;\theta)]$ as a result:

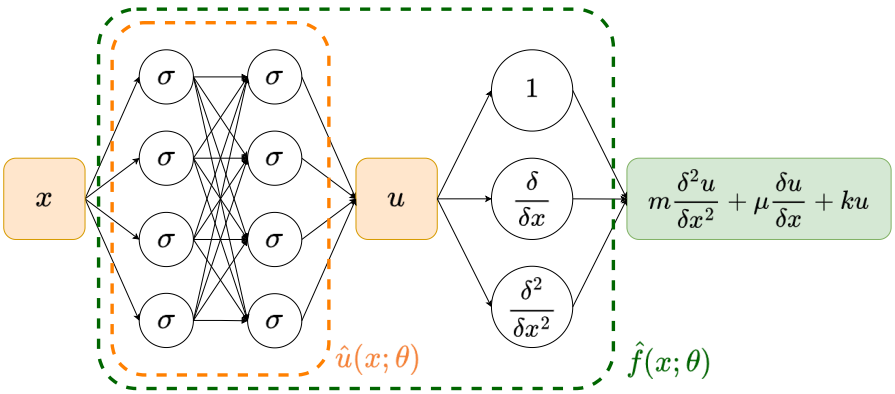

Figure 2-5: Structure of a Physics-Informed Neural Network (PINN). A neural network $\hat{u}(x;\theta)$ predicts the value of a physical variable u. This network can be differentiated to create a predictor $\hat{f}(x;\theta)$ for the Partial Differential Equation (PDE) that describes the system (in this example a damped mass-spring system). Both networks are used in separate parts of a loss function L. Image adapted from [22].

$$\hat{f}(x;\theta) := \mathcal{D}[\hat{u}(x;\theta)]. \tag{2-12}$$

These differentiations can be done using the automatic differentiation functions built in in packages like PyTorch [31] or TensorFlow [32]. The structure of this PINN is visually explained in Figure 2-5. As the physics-informed model $\hat{f}(x;\theta)$ requires a (double) differentiation, it is important that the activation functions σ in the network $\hat{u}(x;\theta)$ are continuously differentiable. Therefore, a popular activation function used in PINNs is the hyperbolic tangent function $\sigma(x) = \tanh(x)$, due to its smoothness and continuous differentiability [22, 23].

All the network parameters of $\hat{f}(x;\theta)$ lie in the network $\hat{u}(x;\theta)$ that predicts u, as the differentiation layer does not add any trainable parameters. Since $\hat{u}(x;\theta)$ and $\hat{f}(x;\theta)$ share the same network parameters θ, any information on both networks can be combined while training the parameters [23]. This results in a loss function that consists of two components:

$$L(\theta) = L_{\text{data}}(\theta) + \lambda_p L_{\text{physics}}(\theta), \tag{2-13}$$

where L_{data} is the data loss, obtained by using the known dataset $(x_i, u_i) \in X \times \mathcal{U}$ for $i = 1, ..., N_{\text{data}}$ on network $\hat{u}(x;\theta)$, and L_{physics} the physics loss of network $\hat{f}(x;\theta)$, using a different set of user-defined 'collocation points' $x_j \in X$ for $j = 1, ..., N_{\text{physics}}$. The hyperparameter λ_p defines the relative weight between the two loss functions. These loss functions are often defined as a mean squared error loss:

$$L_{\text{data}}(\theta) = \frac{1}{N_{\text{data}}} \sum_{i=1}^{N_{\text{data}}} ||\hat{u}(x_i;\theta) - u_i||_2^2, \tag{2-14a}$$

$$L_{\text{physics}}(\theta) = \frac{1}{N_{\text{physics}}} \sum_{j=1}^{N_{\text{physics}}} \left|\left|\hat{f}(x_j;\theta) - f(x_j)\right|\right|_2^2. \tag{2-14b}$$

Sometimes, information on the boundary and/or initial conditions of the physical system is known, which can be either included in the data, or added as an additional loss term in (2-13) with its own separate weight λ_b. Note that the collocation points x_j can be chosen anywhere on the domain X, and that any arbitrary amount N_{physics} of these points can be taken. As the output of $\hat{f}(x;\theta)$ is described by the PDE in (2-10), which is only a function of the input x, there is no need for an additional dataset as the value of this PDE can simply be calculated [22].

Using this augmented form of the loss function in (2-13), the training of the parameters θ is no different than training any other network, and for example the commonly-used stochastic gradient descent method [33] can be applied to solve the optimization problem.

Figure 2-6 shows an example of a solely data-driven neural network and a PINN trained to predict the excitation of a damped harmonic oscillator [25], when given a very small set of training data. By using a dense array of collocation points to compute the physics loss (2-14b) at, the accuracy of the network over the entire plotted interval improves significantly. The network is able to interpolate with high accuracy on the interior of the training dataset, and also extrapolation outside of the dataset is improved over the data-driven method.

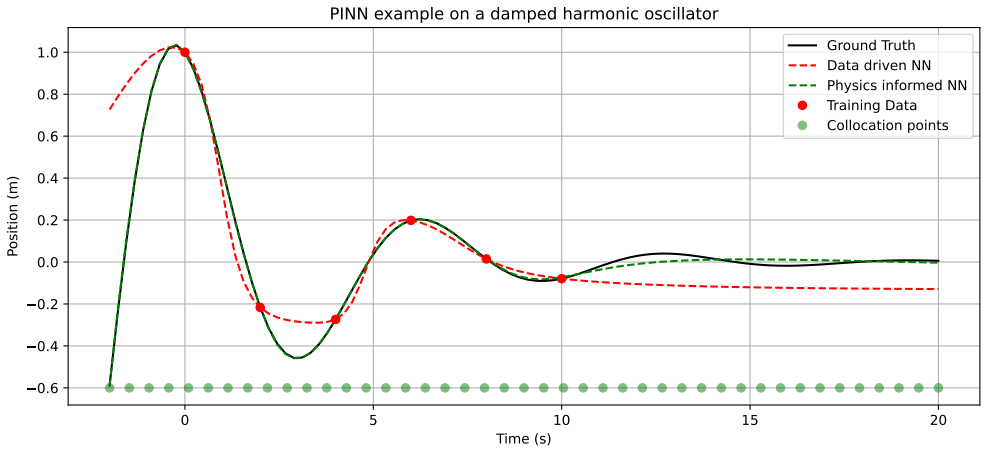

Figure 2-6: Example of a data-driven neural network and a physics-informed neural network trained to predict the location of a damped harmonic oscillator. Using a sparse dataset (red dots) and a dense array of collocation points (green dots), the PINN is able to predict the position much more accurately than the data-driven network. Figure adapted from [25].

A common problem that arises with training PINNs is the training time. The addition of the physics loss (2-14b) makes the loss function more complicated, which creates the need for more training epochs than required for a data-driven neural network [22]. Moreover, the computation of the partial derivative of the network output with respect to its input is often done using auto-differentiation through backpropagation of the network [23], which is a time consuming computation that adds extra computation time in the training process. The latter problem can be mitigated by using a faster method of computing this derivative. It has been shown [34] that approximating the differential operator using numerical methods can increase training time significantly, whilst still allowing convergence of the optimization

problem. This approximation could be done by computing the discrete derivative on a mesh of sampled outputs of the neural network [34].

At the time of writing this thesis and to the best of my knowledge, there is no literature to be found on using a PINN in the PDR problem. This thesis will focus on using the ideas from the PINN on the PDR algorithm by *Liu et al.* [1]. There will be slight alterations made on the fundamental PINN as in [22, 23]. First of all, the network architectures used are different. The PINNs in [22, 23] are commonly fully connected neural networks [21]. With the large amount of samples in the input data (a window of 1 second of 6D IMU data sampled at 200 Hz already consists of 1200 data samples), using a fully connected neural network for the PDR application would result in too large networks, and by using other network architectures that work well with sequential data, like the ResNet from *Liu et al.* [1], network size is reduced significantly without compromising performance. Secondly, the physics loss in [23] is solely based on the PDE of the modeled system. In PDR, the physics loss equations will contain sensor measurements (this will be explained in Chapter 4), which limits the choice of collocation points to time samples where IMU measurements have been taken. Therefore, the PINNs as referred to in this thesis are neural networks trained using a physics loss, as inspired by [23].

Chapter 3

Adapting the neural network of an existing data-driven PDR algorithm

This chapter is aimed towards answering the research question

> *What influence do small adaptations of the neural network in an existing PDR algorithm from literature have on its accuracy, and is there a bottleneck to be found that limits the accuracy of the algorithm?*

To answer this question, first a baseline performance of two similar algorithms from literature, by *Liu et al.* [1] and *Bajwa et al.* [19], has been evaluated on the dataset provided by Leica. This dataset contained Inertial Measurement Unit (IMU) measurements and estimated pose of the surveying pole when a total station was used for the Extended Kalman Filter (EKF) measurement updates. The pose, estimated by an error-state EKF [24], was considered as ground truth. The dataset was captured by different persons doing a variety of tasks, like walking in a predetermined path or doing measurements by placing the pole tip at different points of interest. The EKF as provided by Leica in their simulation software is similar to the one from *Bajwa et al.* [19]. This error-state EKF fuses IMU measurements with velocity measurement predictions from a neural network. The global structure of this algorithm is visualized in Figure 3-1.

By doing small experiments of adapting a specific feature in the network or the pre-processing of the IMU data, the influence of these features on the full PDR algorithm is tested. The EKF remains unchanged, as the scope of this project is to change the neural network only.

The structure of this chapter is as follows: first, the two networks from literature are trained and evaluated on Leica's dataset to set a baseline performance of the current state of the art methods in literature. Then, each following section describes the method and results of making small adaptations on the networks from the literature baselines. These adaptations include combining the input information from both baseline networks, removing the uncertainty prediction to create a simplified network, changing the amount of IMU samples in the input of the network, and using a different method of preprocessing the input data. The chapter ends with a conclusion on which adaptations worked well and which did not, and what the bottleneck is that limits the performance of the PDR algorithm.

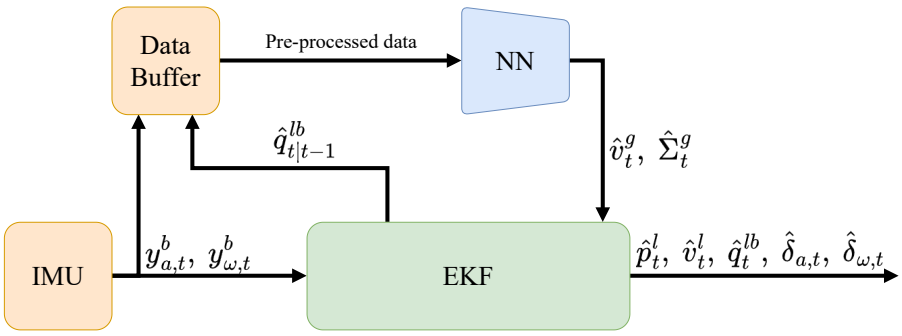

Figure 3-1: Global structure of the Pedestrian Dead-Reckoning (PDR) algorithm used in this chapter. The algorithm is adapted from *Liu et al.* [1] and *Bajwa et al.* [19]. The EKF is provided by Leica and fuses IMU measurements with an external velocity measurement to predict position, velocity, orientation, and IMU biases. The method of data pre-processing and specifications of neural network are experimented with, and are explained in each separate section.

3-1 Performance of the baseline networks in the PDR algorithm

To compare the performance of the EKF for the adaptations to the neural network made in this chapter, two models from the literature have been selected as baseline: TLIO by *Liu et al.* [1] and DIVE by *Bajwa et al.* [19].

3-1-1 Method

TLIO [1] was the first to use an EKF with a predicted measurement update from a neural network, particularly a ResNet [29], to improve the performance of PDR algorithms. This method, however, used the displacement in position during a specific time window as a measurement for the EKF. In order to use a similar method as TLIO on the EKF from Leica, the output of the model has been adapted to be the velocity at the end of the input time window and follows the network structure in Figure 3-2.

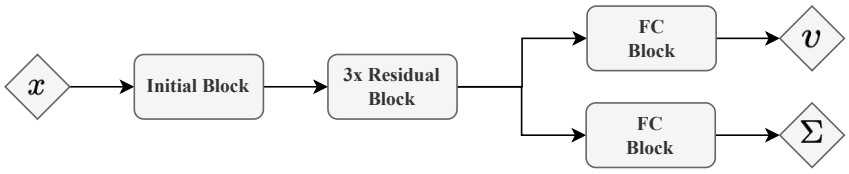

Figure 3-2: Global structure of the network architecture used in the PDR algorithm in Figure 3-1. The network has the same architecture as that of TLIO [1] in Figure 2-3, but returns a predicted velocity instead of displacement. Detailed structures of the blocks are shown in figure A-1 in Appendix A-1.

DIVE [19] was based on the idea of TLIO, but the field of application of their work is estimating the pose of a quadcopter, which is a fundamentally different field with different motion patterns than the tracking of pedestrians. Still, as this method showed more promising performance on their dataset compared to TLIO, this method has been chosen as a second baseline. The network used in DIVE is the same ResNet as in Figure 3-2, and already returns a velocity output, so nothing was adapted in this network.

Both networks received input data preprocessed with their own methods (Algorithm 1 and 2) for a window of 1 second at an IMU sampling frequency of 200 Hz. They were trained on the dataset as provided by Leica, which was split into 60% training data used to actively train the model on, 20% validation data that was used to monitor when the model started to overfit during the training process, and 20% testing data, which remained completely unseen during the training process, used to compare the performances of the trained models on. Careful attention has been taken to split the dataset such that data captured by one person was only used in one of the training/validation/testing splits, to prevent data leakage of similar walking patterns in different sets. Also, each set contained roughly the same variety in trajectory specifications, ranging from a scripted path of movement on a flat terrain to random movements on a surface with hills and stairs. The loss functions of velocity prediction ((2-8) and (2-9)) as used by *Bajwa et al.* [19] were used, where the Mean Squared Error loss (2-8) was used for 10 epochs, after which the Maximum Likelihood loss (2-9) was used until convergence. A learning rate of 5×10^{-5} was used and the magnitude of the random additive bias and gravity perturbations (as also used by [1], see Section 2-2) have been adapted to values that represent the measurements in Leica's dataset. This includes uniformly distributed random biases in $[-0.02, 0.02]$ m/s^2 for the accelerometer measurements, and $[-0.002, 0.002]$ rad/s for the gyroscope measurements, as well as a gravity perturbation sampled from $[0, 0.04]$ rad on a random horizontal axis. Moreover, pedestrian motion is symmetric along the yaw axis [10], and to make the network invariant to the direction of this motion, the in- and output data were rotated with a random yaw angle every training epoch like in [1].

3-1-2 Results

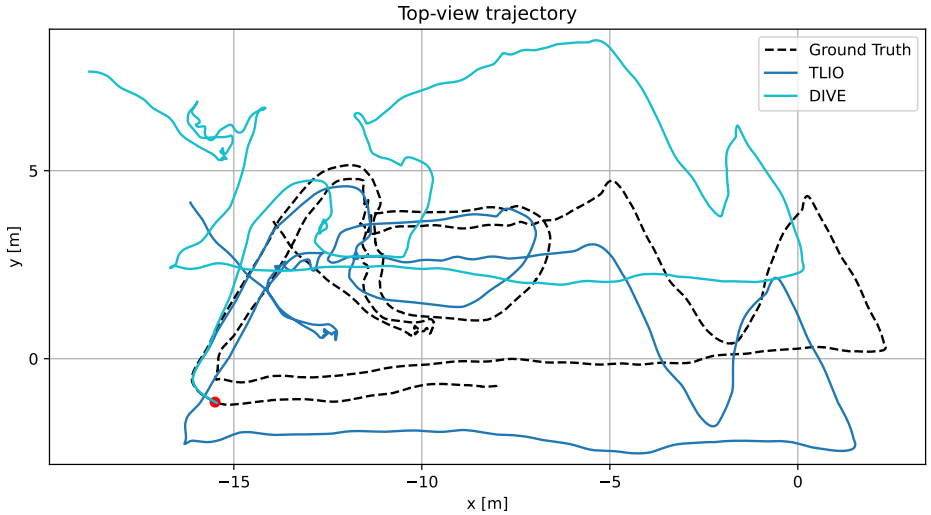

Figure 3-3: Top-down view of an example trajectory from the test set of 90 seconds of simulated dead reckoning. The black reference trajectory is the true position, the cyan and blue trajectories are the estimated positions using the adapted TLIO and DIVE networks in the PDR algorithm, respectively. The simulated dead reckoning started at the red dot.

Figure 3-3 shows an example of a 90 seconds dead reckoning simulation from the test set data. The dead reckoning has been simulated by artificially removing the availability of the

position sensor measurements and using the network's prediction for this. It is seen that in this experiment, the adapted TLIO algorithm remains the closest to the ground truth trajectory. To evaluate the overall performances of both algorithms, 135 trajectories as in Figure 3-3 from the test set have been evaluated, and the combined performance is summarized in the boxplots in Figure 3-4. These boxplots show, from left to right, the error distribution of the velocity prediction of the standalone network on IMU data rotated using ground-truth orientation, the error distribution of the network's velocity prediction using input data rotated with the estimated orientation from the EKF during dead reckoning experiments, the velocity estimation error by the EKF, and the drift rate of the position estimation. The drift rate is a measure of the absolute position error divided by the total traversed distance, a measure often used to determine the performance of PDR algorithms [1, 3].

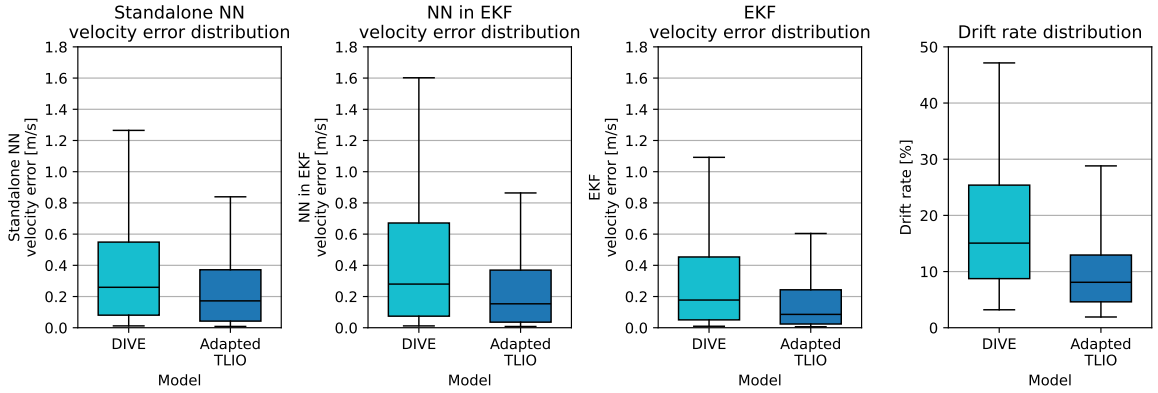

Figure 3-4: Comparison of the error distributions on the unseen testing set of the two baseline models for four metrics: velocity error of the standalone network using IMU input data rotated with the ground-truth orientation, velocity error of the network using IMU data rotated with the estimated orientation of the EKF, velocity error of the EKF estimate, and lastly the drift rate of the position estimation.

		DIVE	TLIO
Standalone network velocity error [m/s]	Median	0.26	**0.17**
	95%	1.26	**0.84**
Drift rate [%]	Median	15.1	**8.1**
	95%	47.1	**28.8**

Table 3-1: Median and 95th percentile of the standalone network velocity error and drift rate from Figure 3-4. Best values are highlighted.

Figure 3-4 shows there is a clear difference between the performance of the two models from literature on the dataset from Leica. Table 3-1 shows more detailed values from the left and right boxplots. The median velocity error of the network from DIVE on input data rotated using ground truth orientation from the testing set is 53% higher than the adapted TLIO network. Using input data rotated with estimated orientation during dead reckoning simulations in this same testing set, this difference increases as seen in the second plot in Figure 3-4, indicating that the DIVE network responds worse to the imperfections in the input data (poorly estimated biases on IMU data, errors in the orientation estimate used in preprocessing, even though both networks are trained with the same random perturbations during training. A reason for this difference might be the integration of the gyroscope data in the preprocessing of the input for the DIVE model (Algorithm 2); it may be more difficult

for the neural network to become robust to gyroscope bias when the input data is the device orientation (calculated from the raw gyroscope data) instead of the raw gyroscope data itself, and small errors in estimated gyroscope bias may result in larger prediction errors for the DIVE network compared to the adapted TLIO network.

The same difference in model performances is seen in the EKF, where the median error in velocity estimate by the EKF is twice as high using the DIVE network than when using the adapted TLIO architecture. This results in a median drift rate of 8.1% using TLIO, and 15.1% using DIVE.

At a surface level, this increase in error when using the DIVE network seems like a strange result, as the original publication [19] claims a better performance than TLIO on their dataset. However, the publication also shows that the model behaves differently on different datasets. Their algorithm was shown to perform well on quadcopter motion datasets and only tested on similar datasets. Quadcopter motion is very different from pedestrian motion (what TLIO was developed for), and this difference could explain the preference for the adapted TLIO network over DIVE on Leica's dataset containing only pedestrian motion.

3-2 Using device tilt as additional input to TLIO

Although the DIVE model showed worse performance than the adapted TLIO model on the Leica dataset, the idea of combining the input data of the two models has been pursued as this might give more information to the network.

3-2-1 Method

The only difference between the adapted TLIO model and the DIVE model as used in Section 3-1 lies in the input of the network. They use the same one-dimensional ResNet architecture (see section 2-2), the same output measures (velocity and its uncertainty), and the same training hyperparameters. The input of TLIO is a combination of 3D accelerometer and 3D gyroscope measurements, rotated to the local frame l using the orientation from the EKF. DIVE only uses the 3D accelerometer measurements rotated to a gravity-aligned frame g, along with a 3D rotation vector that represents the rotation from body frame b to the gravity-aligned frame g. As the IMU measurements are invariant of the absolute yaw (it is impossible to observe the absolute yaw orientation from IMU measurements, only the changes in it) [1], this 3D rotation vector essentially only has two degrees of freedom left. The device used to capture the dataset from Leica was symmetric along its own z-axis, so it could be held at any angle along this axis, without changing physical parameters such as moment of inertia. Therefore, it was speculated that the rotation around this axis did not include important information for the neural network, and there was only one rotation component that could include additional information on top of the accelerometer and gyroscope measurements: the tilt angle between the device and the z-axis of the l frame, as visualized in Figure 3-5. This tilt could give the model additional information on how the person is holding the surveying pole, which may influence the motion pattern of the device.

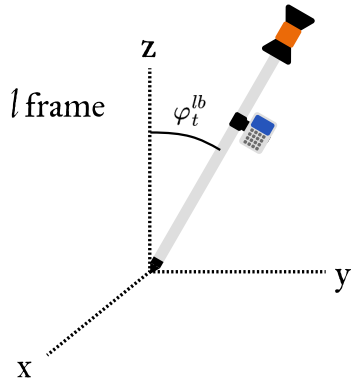

Figure 3-5: Definition of the tilt angle φ_t^{lb} between the device and the z-axis of the l frame.

Therefore, a new model was trained that used 7 channels of input data: 3D accelerometer measurements $y_{a,t}^l$, 3D gyroscope measurements $y_{\omega,t}^l$, and a tilt angle φ_t^{lb} of the device with respect to the z-axis in the local frame l. All measurements were still taken at 200 Hz for 1 second of input data, for a total of $T = 201$ measurements per channel.

3-2-2 Results

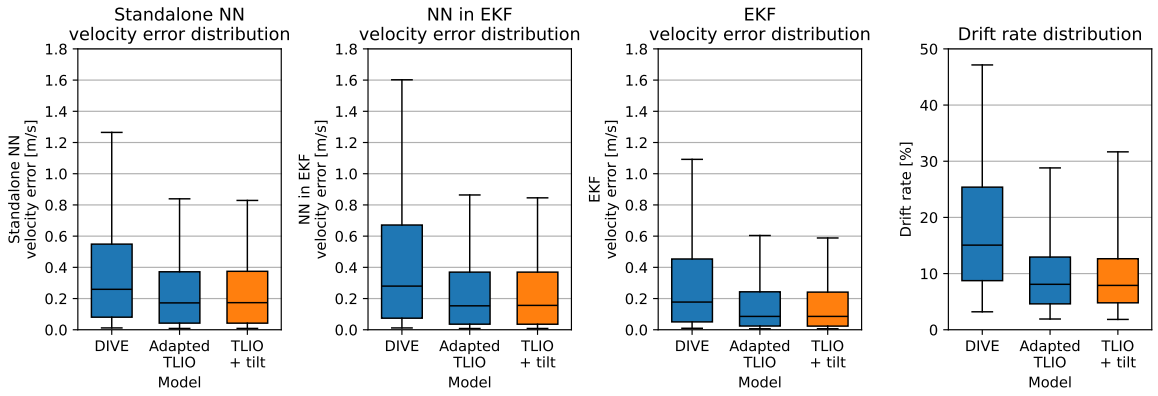

Figure 3-6: Comparison of the network using a combination of the input data from TLIO and DIVE. Numerical values of the standalone network prediction error (left) and drift rate (right) are placed in Table 3-2.

		DIVE	TLIO	TLIO + tilt
Standalone network velocity error [m/s]	Median	0.26	**0.17**	**0.17**
	95%	1.26	0.84	**0.83**
Drift rate [%]	Median	15.1	8.1	**7.9**
	95%	47.1	**28.8**	31.7

Table 3-2: Median and 95th percentile of the standalone network velocity error and drift rate from Figure 3-6. Best values are highlighted.

The performance of this combined model is shown in Figure 3-6. The figure shows there is not a significant difference between the adapted TLIO model using only accelerometer and

gyroscope data as input, and the new model using tilt as an additional input value. Table 3-2 reveals that using the tilt as an additional input to the adapted TLIO network slightly improves the median drift rate of the resulting PDR algorithm from 8.1% to 7.9% compared to the adapted TLIO network from Section 3-1. Therefore, the rest of this chapter will continue using the tilt angle as extra input channel.

3-3 Simplifying the network by using a linear model for uncertainty

Next to the predicted velocity measurement, the uncertainty of said measurement is also important for the EKF to function. In TLIO [1] and DIVE [19], this uncertainty is being predicted by the neural network along with the velocity prediction. However, in this experiment, the choice was made to remove this uncertainty prediction from the network, resulting in the network architecture as shown in Figure 3-7. This choice was made to reduce the amount of parameters in the network and simplify the loss function, therefore optimizing the training time that was needed to train one instance of this model.

Figure 3-7: Reduced network architecture from Figure 2-3. The uncertainty prediction has been removed, and replaced with a statistical approach in order to reduce model size and therefore training time.

3-3-1 Method

Instead of predicting the uncertainty in the network, a statistical approach has been taken where the uncertainty of the predicted velocity was modeled using the predicted velocity itself. After the network was trained, the errors e between the predicted and ground-truth velocities for all data samples in the test set were computed. A model will be fitted which assumes that the three directions of the errors, e_x, e_y, and e_z are distributed as independent normal distributions with zero mean and covariances σ_x, σ_y and σ_z as function of the velocity prediction $\hat{v}^l$:

$$e_x = \hat{v}_x^l - v_x^l, \qquad \text{assume} \quad e_x \sim N(0, \sigma_x(\hat{v}^l)), \tag{3-1a}$$
$$e_y = \hat{v}_y^l - v_y^l, \qquad \text{assume} \quad e_y \sim N(0, \sigma_y(\hat{v}^l)), \tag{3-1b}$$
$$e_z = \hat{v}_z^l - v_z^l, \qquad \text{assume} \quad e_z \sim N(0, \sigma_z(\hat{v}^l)). \tag{3-1c}$$

As the pedestrian motion is symmetric along rotations around the z-axis, it was assumed that $\sigma_x = \sigma_y$, and instead of using the xyz representation, the velocity predictions and their errors were represented in a horizontal and vertical component, based on their magnitude in the xy and z direction:

$$|\hat{v}_H^l| = \sqrt{\hat{v}_x^{l\,2} + \hat{v}_y^{l\,2}}, \qquad e_H = \sqrt{e_x^2 + e_y^2}, \tag{3-2a}$$
$$|\hat{v}_V^l| = |\hat{v}_z|, \qquad e_V = |e_z|. \tag{3-2b}$$

This way, the horizontal velocity error distribution e_H is also symmetric along the z-axis. Using $\sigma_H := \sigma_x = \sigma_y$ and $\sigma_V := \sigma_z$, the relation between these standard deviations and the predicted velocity $\hat{v}^l$ is assumed to be a linear function of the horizontal or vertical magnitude of the velocity prediction, as it was expected that larger predicted velocities would give larger prediction errors, and therefore uncertainties:

$$\sigma_H(\hat{v}^l) = a_H|\hat{v}^l_H| + b_H, \tag{3-3a}$$
$$\sigma_V(\hat{v}^l) = a_V|\hat{v}^l_V| + b_V, \tag{3-3b}$$

with a_H, b_H, a_V, b_V the parameters of the linear model. These standard deviations as a function of the velocity prediction were then used to compute the covariance matrix of the uncertainty of prediction $\hat{v}^l$:

$$\hat{\Sigma}(\hat{v}^l) = \text{diag}(\sigma_H(\hat{v}^l)^2, \sigma_H(\hat{v}^l)^2, \sigma_V(\hat{v}^l)^2), \tag{3-4}$$

which is used by the EKF along with the predicted velocity itself in the measurement update.

To obtain the parameters a_H, b_H, a_V, and b_V in (3-3), first the errors e_H and e_V were split into bins of increasing horizontal or vertical velocity magnitude, using a bin size of 0.2 m/s. For each bin, an error distribution has been fit to obtain the value for σ_H and σ_V. As the directional components of the errors (e_x, e_y, e_z) are assumed independent normal distributions with zero mean and standard deviations $\sigma_x = \sigma_y$ and σ_z, the distribution of e_H in (3-2a) follows a Rayleigh distribution [35] with probability density function

$$f(|\hat{v}^l_H|) = \frac{|\hat{v}^l_H|}{\sigma_H^2} \exp\left(-\frac{|\hat{v}^l_H|}{2\sigma_H^2}\right), \quad \hat{v}_H > 0. \tag{3-5}$$

The distribution of e_V in (3-2b) follows a folded normal distribution [35] with density function

$$f(|\hat{v}^l_V|) = \frac{\sqrt{2}}{\sqrt{\pi\sigma_V^2}} \exp\left(-\frac{|\hat{v}^l_V|}{2\sigma_V^2}\right), \quad \hat{v}_V > 0. \tag{3-6}$$

The parameters σ_H and σ_V in (3-5) and (3-6) represent the 39th and 68th percentile of each distribution, respectively [35].

Finally, a weighted least squares fit of the linear functions in (3-3) has been done on all obtained values for σ_H and σ_V from the fits in each velocity bin, with the amount of samples in each bin as the weights. Using the resulting linear functions, the covariance matrix of the uncertainty of the predicted measurement $\hat{v}^l$ could be computed as in (3-4).

With the uncertainty prediction removed from the neural network, the network was trained using the Mean Squared Error (MSE) loss in (2-8) for all epochs.

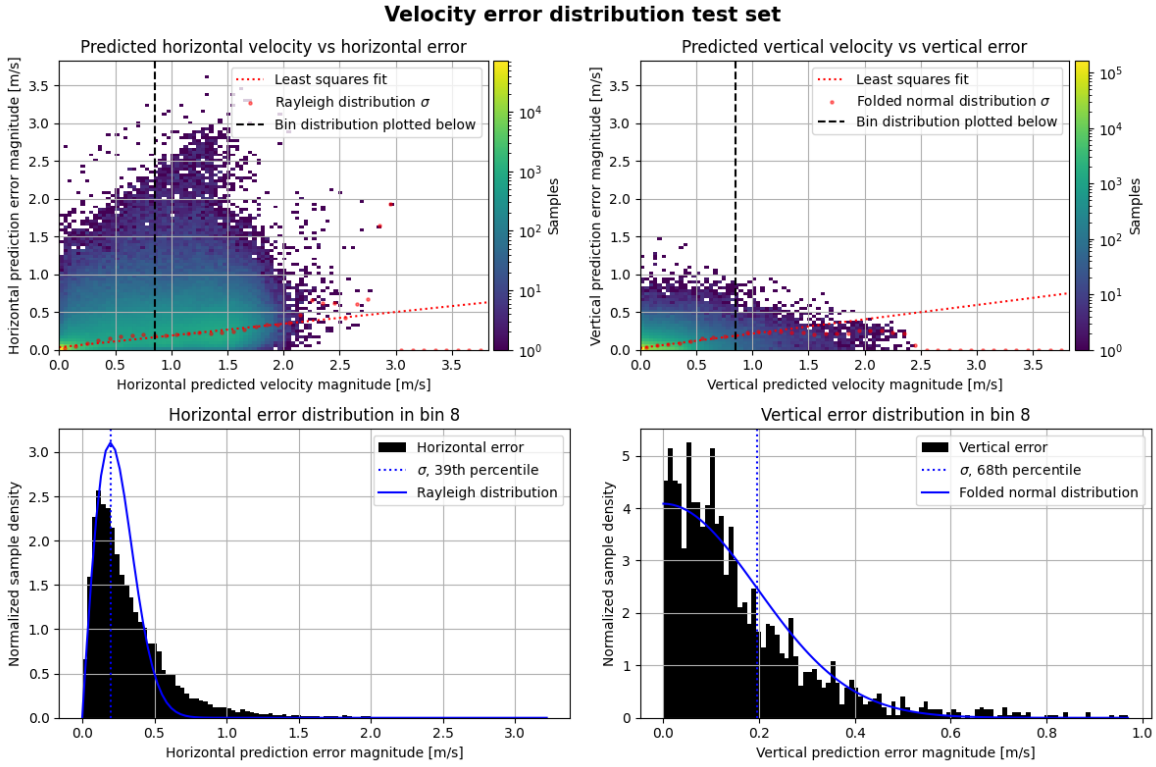

Figure 3-8: Distributions of horizontal (left) and vertical (right) velocity errors against their predicted values. The distribution has been split into bins of 0.2 m/s on the x-axis, and a Rayleigh (left) and folded normal (right) distribution have been fitted by computing the 39th and 68th percentile of the distribution in each bin. The linear uncertainty model was created doing a weighted least squares fit through the σ values of each bin.

3-3-2 Results

The simplified model from Figure 3-7 reduced its parameter amount from 5.4 million parameters to 4.6 million. Not a significant decrease, but combined with the usage of the simpler MSE loss function, the network was able to finish training in 1.9 hours. Compared to the original model, which took on average 2.5 hours to train, this is a reduction of 25% in training time.

Figure 3-8 shows the velocity error distribution for this simplified model, on data from the test set preprocessed using ground-truth orientation. The top plots show that indeed, there is a linear trend in the values of σ_H and σ_V (red dots) of the error distributions (y-axis) when plotted against the (horizontal and vertical) predicted velocity magnitudes (x-axis). Note the logarithmic scaling in the colorbar representing the density of samples. The bottom plots reveal that the vertical velocity error distribution in one of the velocity bins is indeed following the density function of a folded normal distribution (3-6) quite well. However, the bottom left plot, where the distribution of the horizontal component of the prediction error has been plotted, shows that the Rayleigh distribution (3-5) may not have been the right model to fit, as the true distribution has a lower peak and a longer tail. This means that the modeled value of σ_x and σ_y may be overly optimistic.

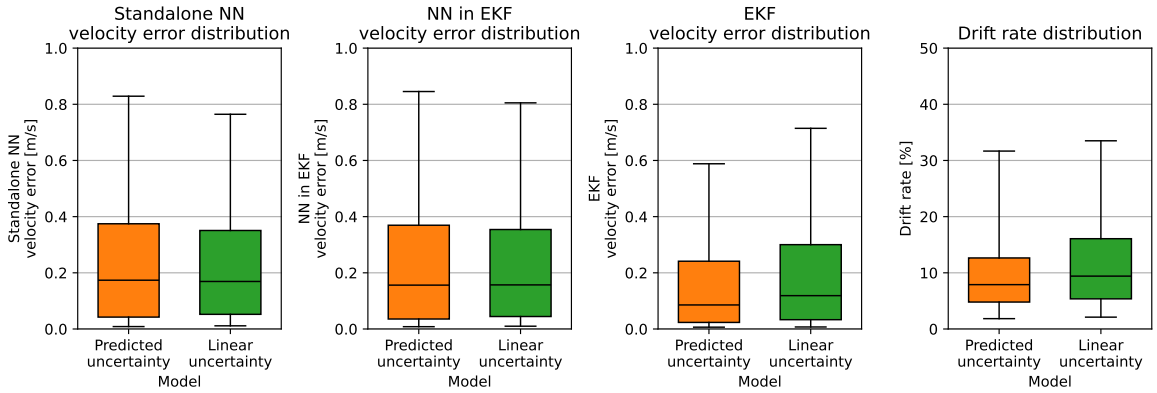

Figure 3-9: Performance comparison of the network that returns a predicted velocity and uncertainty (orange), versus a network that only predicts a velocity and uses a statistical approach to model the uncertainty as a linear function of the velocity magnitude (green). Numerical values of the median and 95th percentile of the standalone velocity error and drift rate are written in Table 3-3.

		Predicted uncertainty	Linear uncertainty model
Standalone network velocity error [m/s]	Median	**0.17**	**0.17**
	95%	0.83	**0.76**
Drift rate [%]	Median	**7.9**	9.4
	95%	**31.7**	33.5

Table 3-3: Numerical values of the most important statistics of the boxplots in Figure 3-9.

Figure 3-9 and Table 3-3 show the performance comparison of the network from the previous section, using a predicted uncertainty, and the network using the linear uncertainty model. The two left plots show that the standalone network using the linear uncertainty model yields a slight reduction in the larger error values, reducing the 95th percentile of all errors in the unseen test set from 33.5% to 31.7%. However, despite this increased performance of the velocity predictions, the EKF velocity estimate and drift rate become worse when using the network with a linear uncertainty model. The median drift rate increases from 7.9% for the network that predicts the uncertainty to 9.4% for the network with an uncertainty prediction based on the linear model. This indicates that using this simplified network architecture that trains faster is not beneficial for the performance of the PDR algorithm. Although the overall distribution in Figure 3-8 appears to scale linearly with $|\hat{v}^l|$, there may be situations in which the network can give a very accurate prediction even for large velocities (or vice versa), and the network that predicts the uncertainty may be able to react to these situations more accurately than the model that uses the linear model for uncertainty. Therefore, in the remainder of this chapter, the networks will predict both the velocity and its uncertainty.

3-4 Adapting the input data by using longer windows and downsampling

Both [1] and [19] show that the length of the input window into the neural network can have an effect on the performance of both the network and the resulting EKF estimates. *Bajwa et*

al. [19] also show that it depends on the dataset what the optimal input window length is. Therefore, several window lengths have been tested on the dataset provided by Leica, to find out which is the optimum for this set.

3-4-1 Method

Window lengths of 1, 2, 4, 8 and 16 seconds have been tested. However, as the window length and therefore amount of input samples increases, the network also becomes increasingly larger. To prevent extremely large models, a downsampling of the IMU data from 200 Hz to 50 Hz has also been tested. *Riddick et al.* [36] recommend a sampling rate of at least 35 Hz when capturing human motion with an IMU, so 50 Hz should still be a high enough sampling rate to extract all necessary information from the IMU signals. To support this claim, a frequency analysis of the captured accelerometer and gyroscope data was done on the dataset.

3-4-2 Results

Frequency analysis of the IMU data

By taking the Fourier transform [37] of an accelerometer and gyroscope measurement sequence, the frequency plots in Figure 3-10 have been created. A quickly decaying magnitude from the lower to higher frequencies is seen for both the accelerometer and gyroscope. At 25 Hz, both sensors show a small peak in their amplitude. For higher frequencies, the strength of the signal remains roughly the same, indicating these signals are mainly due to white noise [38]. With 25 Hz being the highest frequency that still contains information on the pedestrian motion, the Nyquist frequency [38] at which sampling can still extract all information from the data would be 50 Hz. Therefore, theoretically, downsampling the IMU data from 200 Hz to 50 Hz should not cause any issues.

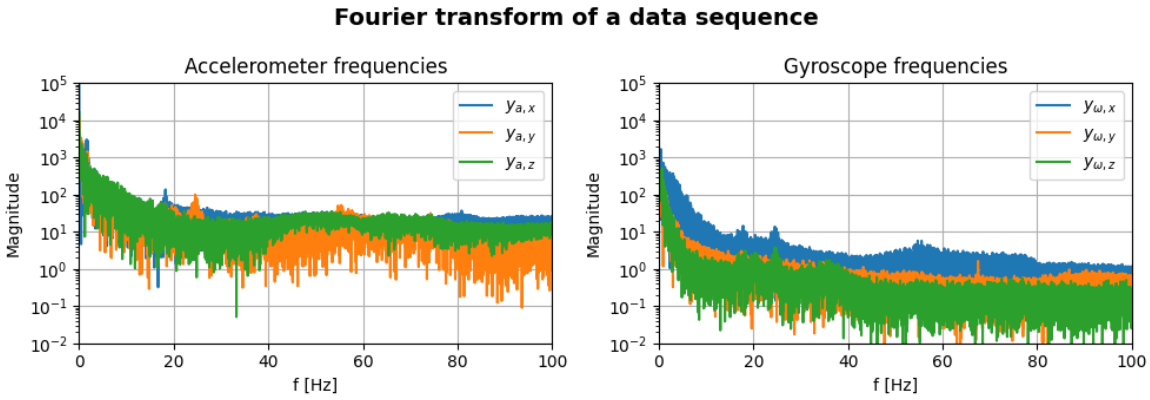

Figure 3-10: Magnitude of frequencies in one of the data sequences on the x, y, and z component of the accelerometer data (y_a) and gyroscope data (y_ω), computed using the fast Fourier transform [37].

Performance of the network and PDR algorithm using different input window characteristics

The resulting error distributions on the unseen testing set are shown in Figure 3-11, with window lengths on the x-axis and IMU frequency noted by the color of the boxes. It is seen that the networks that use a downsampled IMU data sequence as input (red) result in roughly the same error distribution as the networks using the full sampling rate of the IMU (orange). This is true for both the standalone network as well as the EKF velocity and drift rate errors, the latter actually showing improvements in median drift rate for longer input windows. It is therefore concluded that by downsampling the input data from 200 Hz to 50 Hz, no information on the pedestrian motion is lost, and this method will be used in the rest of the thesis to reduce model complexity.

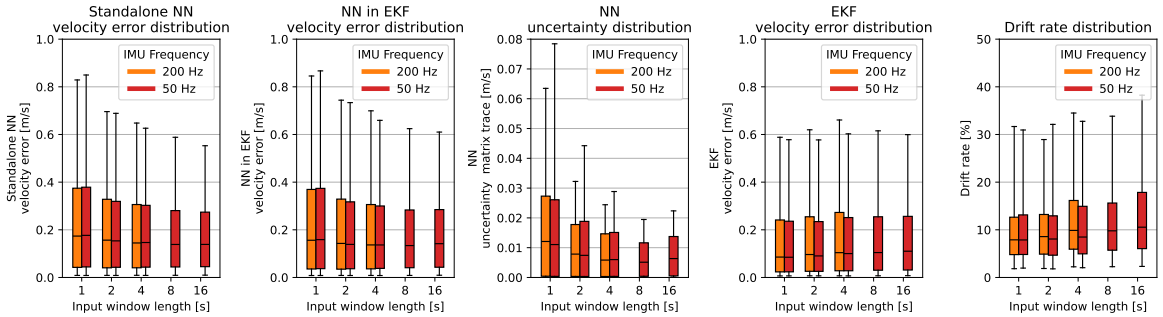

Figure 3-11: Performance comparison of networks using different lengths of input data at two different downsampling rates.

Window Length		1		2		4		8		16	
IMU frequency		200	50	200	50	200	50	200	50	200	50
Standalone network velocity error [m/s]	Median	0.17	0.18	0.16	0.15	**0.14**	0.15	-	**0.14**	-	**0.14**
	95%	0.83	0.85	0.70	0.69	0.65	0.63	-	0.59	-	**0.55**
Drift rate [%]	Median	**7.9**	**7.9**	8.6	8.0	9.9	8.5	-	9.8	-	10.6
	95%	31.7	30.9	**28.9**	32.1	34.5	32.8	-	33.8	-	38.2

Table 3-4: Numerical values of the most important statistics of the boxplots in Figure 3-11.

The standalone model performance on both IMU data preprocessed using ground truth orientation (left) and estimated orientation by the EKF (second from left) clearly improves with larger input windows. A reason for this may be the fact that longer windows carry more information on the pedestrian motion, and providing more of that information will make the network more accurate. It is also seen that there is virtually no difference in the performance of the networks that used a 50 Hz IMU frequency for their input, compared to the original 200 Hz. This was to be expected from the frequency analysis, and it also shows in the network performance.

However, although the network on its own produces more accurate predictions, the performance of the EKF seems to deteriorate for longer windows. The estimated velocity by the EKF shows a slight increase, and the drift rate increases much more. An additional plot has been shown in Figure 3-11 that shows the uncertainty distribution as predicted by the neural network. A rapid decrease in the trace of the uncertainty matrix Σ is visible, which

might make the EKF overconfident in the predicted velocity measurement that is used in its measurement update.

Scaling the predicted uncertainty

To find out if the overconfidence in the velocity predictions due to the lower predicted uncertainty could be the reason for the worse EKF performance, the network with an input window of 8 seconds has been re-evaluated on the EKF simulation using different scaling factors on its uncertainty output. The default scaling factor used in the EKF for all simulations is 0.2. Factors ranging from 0.1 to 4 have been tested, and the resulting error distributions of the network and EKF are plotted in Figure 3-12 with detailed values in Table 3-5.

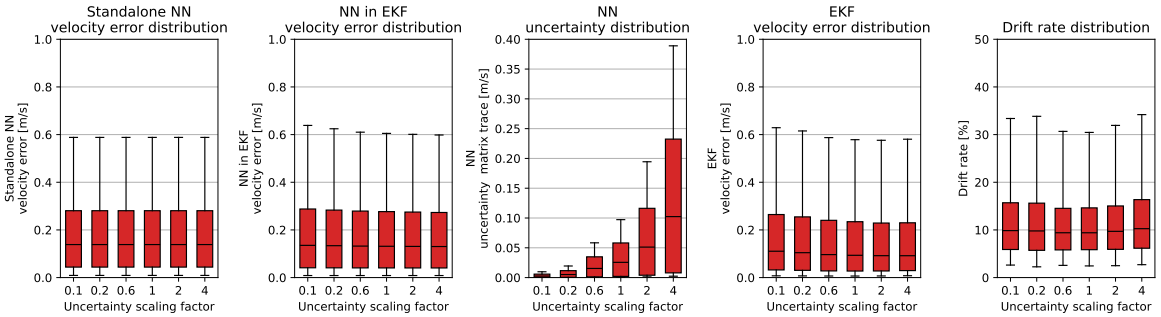

Figure 3-12: Performance comparison of the model using an input window of 8 seconds of IMU data sampled at 50 Hz, using different scaling factors of the uncertainty prediction.

Window Length [s]		8					
Uncertainty scale		0.1	0.2	0.6	1	2	4
Drift rate [%]	Median	9.7	9.8	**9.4**	**9.4**	9.7	10.3
	95%	33.4	33.8	30.6	**30.5**	31.9	34.2

Table 3-5: Numerical values of the most important statistics of the boxplots in Figure 3-12.

As this is the same network that is evaluated multiple times, its standalone performance on the unseen testing set (left plot of Figure 3-12) remains unaffected by the scaling of the uncertainty of the EKF. The velocity estimate of the EKF shows a slightly decreased error distribution for larger scaling factors, and the drift rate appears to have an optimum around a scaling factor of 1, based on the median and 95th percentile of this value. However, the changes are very minimal compared to the values of the drift rates in Figure 3-11 and Table 3-4, and still show an increased drift rate over the simulations that used networks with shorter input windows.

Investigating the bias in the velocity prediction

Another hypothesis for the worse performance of the PDR algorithm when using networks with longer input windows is proposed. An EKF assumes that the noise on the measurements used in its measurement update is normally distributed with zero mean [20]. Therefore, if measurements are given which are disturbed with a non-zero mean noise value, the EKF state

estimates could drift away from the true state of the system. The predicted velocity error distribution in Figure 3-11 only shows the overall magnitude of the error vector, which makes it impossible to see if there is any bias in the direction of this error. To get an impression of the behavior of the error on a smaller scale, the values of the predicted velocity error for one out of the 135 simulations of 90 seconds in the unseen testing dataset have been plotted in Figure 3-13. Only the horizontal error was considered here, as the vertical error was often much more accurate (the vertical motion patterns showed more similarity over the entire dataset and were therefore much more predictable).

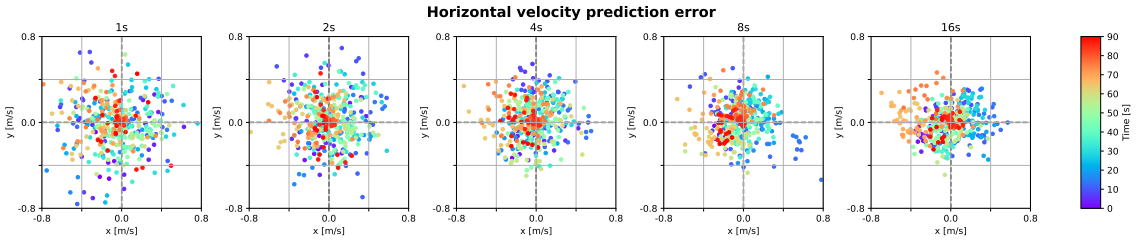

Figure 3-13: Scatterplots of the horizontal velocity prediction errors of the networks, evaluated at a frequency of 5 Hz, using a input window of 1, 2, 4, 8, and 16 seconds of IMU data sampled at 50 Hz. Data from one of the 135 experiments in the testing set is used. The color of the datapoint represents the time in the PDR simulation.

It is immediately obvious from Figure 3-13 that the precision of the velocity estimates increases as the network accepts a longer input window, since the scattered errors are grouped much closer to each other. By looking closer at the scatterplots, it can also be seen that for this particular experiment, it seems as if the models with a longer input window indeed show a small non-zero average when predictions of nearby time samples are considered. This is especially visible for the model with a 16 seconds input window, where the average error early in the window (cyan markers) is leaning towards a positive x value, and later in the window (orange markers), they center around a negative value of x.

To show this more clearly, figure 3-14 shows the moving average of the horizontal velocity prediction errors $e_{v,xy}$ using an averaging window of 20 seconds. It is seen that the models with longer input windows are showing a bias further from the origin than the models with short input windows.

$$MA(e_{v,xy}(t)) = \frac{\sum_{k=0}^{K-1} e_{v,xy}(t-k)}{K}, \tag{3-7}$$

where τ represents the sample index of the measurements during the 90 second simulation. K has been chosen to represent a moving average window of 20 seconds, as the evolution of this bias value seemed to change at timescales of around 10 to 30 seconds, and using 20 seconds of an averaging window was able to filter out most of the noise in the prediction error.

To quantify the average magnitude of this moving average in the prediction error of an entire experiment in one number, the Average Absolute Moving Average (AAMA) is defined:

$$\text{AAMA}(e_{v,xy}) = \frac{\sum_{t=0}^{t_f} |\text{MA}(e_{v,xy}(t))|}{t_f}, \tag{3-8}$$

with t_f the amount of samples in the experiment. In the experiment of Figure 3-14, it is seen that the AAMA increases from 0.085 m/s for the network with a short input window of only 1 second, to 0.124 m/s for the long 16 seconds input window network. The average AAMA values of all experiments are shown in Table 3-6 for each network with increasing input window length. This table reveals that the models with 2 and 4 seconds of IMU data as input have the smallest AAMA values, and therefore comply the most with the EKF assumption that the error in the measurements has a mean of zero.

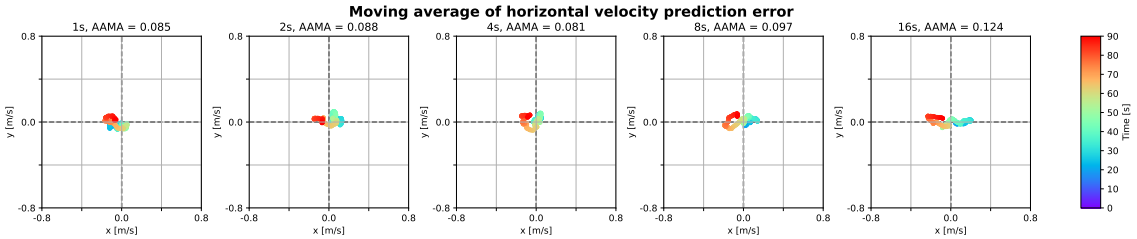

Figure 3-14: Scatterplots of the moving average of the velocity prediction errors in Figure 3-13, using a sliding window of 20 seconds.

Window length [s]	1	2	4	8	16
Overall AAMA [m/s]	0.104	**0.097**	**0.097**	0.102	0.117

Table 3-6: AAMA of the velocity prediction error for networks using 1 to 16 seconds of IMU data as input, sampled at 50 Hz. The AAMA as defined in (3-8) is a measure of the magnitude of the bias on the velocity predictions during a timescale of 20 seconds.

In conclusion, the larger bias in the error of the predicted measurements as shown in Table 3-6 is most likely the reason for the increase in drift rate when utilizing the networks with longer input windows. Therefore, although the velocity error of the standalone network is more accurate, the EKF doesn't produce a better result. The best performing network in this experiment uses 1 second of input data (IMU + tilt angle) sampled at 50 Hz.

3-5 Using orientation history instead of backwards integration

A concern could be raised about the preprocessing method of the IMU data, especially on the network with longer input windows. Algorithm 1 shows how the accelerometer and gyroscope data is rotated from body frame b to local frame l by using an orientation calculated by integrating the gyroscope measurements backwards in time. For short windows, this may not pose so much of an issue, but the longer the input window becomes, the longer this backwards integration is done as well, resulting in a larger accumulation of errors from the gyroscope measurements. The strength of this effect greatly depends on the precision and accuracy of the gyroscope. To find out if this is a problem, the models have been trained using a different preprocessing method that does not rely on this backwards integration, but instead uses the orientation history from the EKF to rotate the IMU data.

3-5-1 Method

In this new preprocessing method, the orientation history from the EKF is used to rotate the IMU data (see Algorithm 3). These orientations should theoretically be closer to the true orientation of the device, as they are corrected in the measurement update of the EKF, and therefore result in more accurate rotations of the sensor measurements. A concern that could be had about this method, however, is that the history of orientation estimates from the EKF is not a continuous signal, because the EKF measurement updates using total station measurements happen at a lower rate (20 Hz) compared to the time updates using IMU data (200 Hz). This causes the orientation history to follow a so-called sawtooth profile [39] creating a non-smooth or discontinuous signal. These discontinuities may have different characteristics in the training data and during inference; in the training data, the measurement update of the EKF uses total station measurements at 20 Hz, but during inference, these updates are governed by the velocity predictions of the neural network sampled at 7 Hz. Also, the network prediction error is most likely larger than the total station measurement error and this error can be distributed differently. This difference in discontinuity characteristic may negatively influence the network's performance during inference.

Algorithm 3 Preprocessing of the input data using the history of EKF orientations

Input: $y^b_{a,0:T}, \quad y^b_{\omega,0:T}, \quad \hat{q}^{lb}_{0:T}$

$\bar{y}^b_{a,0:T} = \begin{bmatrix} 0 & (y^b_{a,0:T})^\top \end{bmatrix}^\top$

$\bar{y}^b_{\omega,0:T} = \begin{bmatrix} 0 & (y^b_{\omega,0:T})^\top \end{bmatrix}^\top$

$\bar{y}^l_{a,0:T} \leftarrow \hat{q}^{lb}_{0:T} \odot \bar{y}^b_{a,0:T} \odot (\hat{q}^{lb}_{0:T})^{\mathrm{c}}$

$\bar{y}^l_{\omega,0:T} \leftarrow \hat{q}^{lb}_{0:T} \odot \bar{y}^b_{\omega,0:T} \odot (\hat{q}^{lb}_{0:T})^{\mathrm{c}}$

$y^l_{a,0:T} = \bar{y}^l_{a,0:T}[1:3]$

$y^l_{\omega,0:T} = \bar{y}^l_{\omega,0:T}[1:3]$

Returns: $y^l_{a,0:T}, \quad y^l_{\omega,0:T}$

3-5-2 Results

Contrary to the pre-processing method in Algorithm 1 and 2, the method in Algorithm 3 did not use a sequential for loop that was done for the backwards integration of the gyroscope data. This drastically decreased the computation time of the training process, as seen in Table 3-7. The increased efficiency in training time was a factor 5 for the network with a 1 second input window of IMU data, and increased up to almost a factor 30 for the network with the longest input window of 16 seconds. This highlights that that the pre-processing Algorithm 3 is significantly faster than the method in Algorithm 1.

Window Length	1		2		4		8		16	
Integration Method	BI	OH	BI	OH	BI	OH	BI	OH	BI	OH
Training time [h]	4	**0.8**	7	**0.8**	12	0.9	24	1.0	45	1.6

Table 3-7: Training times for the networks with different input window lengths and integration methods of the pre-processing.

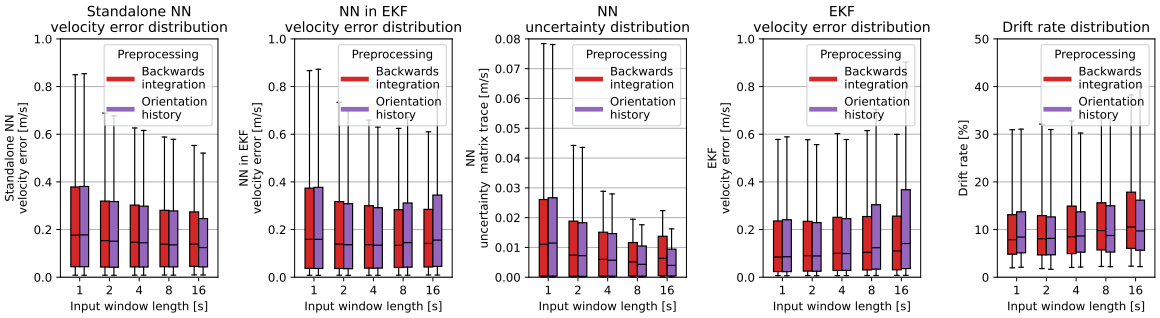

Figure 3-15: Performance comparison of networks using either the Backwards Integration preprocessing method (Algorithm 1) or the Orientation History preprocessing method (Algorithm 3), for different lengths of input window data. All models use IMU data sampled at 50 Hz.

Window Length		1		2		4		8		16	
Integration Method		BI	OH	BI	OH	BI	OH	BI	OH	BI	OH
Standalone network velocity error [m/s]	Median	0.18	0.18	0.15	0.15	0.15	0.14	0.14	0.14	0.14	**0.12**
	95%	0.85	0.85	0.69	0.68	0.63	0.62	0.59	0.58	0.55	**0.52**
Drift rate [%]	Median	**7.9**	8.4	8.1	8.2	8.5	8.7	9.8	8.8	10.6	9.7
	95%	30.9	31.0	32.1	31.0	32.8	**30.2**	33.8	33.3	38.2	40

Table 3-8: Numerical values of the most important statistics of the boxplots in Figure 3-15.

The performance of the PDR algorithm using networks with this new preprocessing method can be seen in Figure 3-15. It can be seen that the standalone network performance using IMU data rotated by the ground truth orientation q^{lb} shows a consistent small improvement when the orientation history of q^{lb} is used in preprocessing compared to backwards integrating the gyroscope data to obtain the history of orientations. This improvement grows with the length of the input window, which indicates that the error caused from integrating the gyroscope values back in time indeed creates a problem in the network, as this error should also increase when this integration is done for a longer time. However, when these models are tested during inference using estimated orientation $\hat{q}^{lb}$ in the preprocessing algorithm, the error distribution appears to grow again for the longer input windows, with an optimum at the medium sized input windows around 4 seconds. This most likely happens because the measurement update during inference uses predicted velocities from the network instead of total station measurements in the training data, which creates different characteristics of the sawtooth profile on the predicted orientation, a concern raised earlier. This increased error causes a large increase in estimated velocity in the EKF.

Still, the networks using 8 and 16 seconds of input data achieve a lower median drift rate when the orientation history preprocessing method is used compared to the backwards integration method. To figure out why this happens, a closer look has been taken at the prediction error of the neural network during inference. Figure 3-16 shows scatterplots of the horizontal error in predicted velocity of the models using the orientation history preprocessing method, for one of the 135 dead reckoning experiments in the testing set. Compared to the errors of the networks using backwards integration for preprocessing in Figure 3-13, the magnitude of the error of the networks using 8 and 16 seconds input windows is much larger, which was already seen in the overall error distribution in the second plot of Figure 3-15. However, Figure 3-17 shows that the moving average of this error (see (3-7)) is much closer to zero than the moving

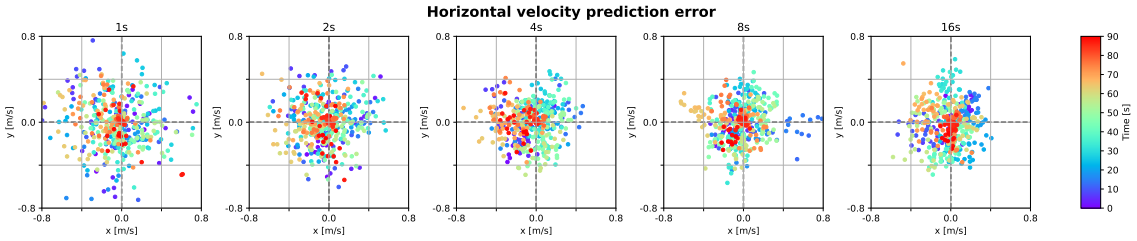

Figure 3-16: Scatterplots of the horizontal velocity prediction errors of the models using a input window of 1, 2, 4, 8, and 16 seconds of IMU data sampled at 50 Hz, using the Orientation History preprocessing method (Algorithm 3). The color of the data point represents the time in the PDR simulation.

average of the backwards integration networks (in Figure 3-14). Over the entire testing set, the AAMA (3-8) of both the networks using the two preprocessing methods is shown in Table 3-9. This table reveals that indeed the orientation history networks achieve lower prediction bias than the backwards integration networks.

However, only the networks with long input windows (8 and 16 seconds) achieve a lower drift rate when this orientation history preprocessing method is used, but the best performing networks are still the networks using short input windows, for which there is no significant difference in performance. The only advantage of the orientation history preprocessing method for these networks is that the networks can be trained faster, as shown in Table 3-7.

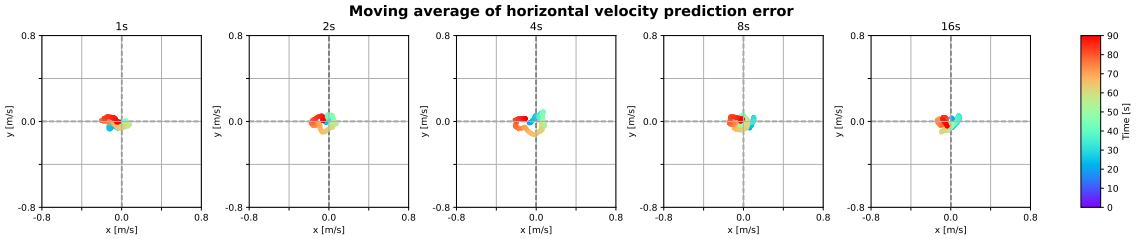

Figure 3-17: Scatterplots of the horizontal velocity prediction errors of the models using a input window of 1, 2, 4, 8, and 16 seconds of IMU data sampled at 50 Hz. The color of the datapoint represents the time in the PDR simulation.

Window length [s]	1		2		4		8		16	
Integration Method	BI	OH	BI	OH	BI	OH	BI	OH	BI	OH
Overall AAMA [m/s]	0.104	0.090	0.097	**0.085**	0.097	0.096	0.102	0.103	0.117	0.095

Table 3-9: Average Absolute Moving Average (AAMA) values of networks using input windows of 1 to 16 seconds of IMU data, preprocessed with either the Backwards Integration method (Algorithm 1) or the Orientation History method (Algorithm 3).

3-6 Concluding Remarks

This chapter was aimed towards answering the research question

> *What influence do small adaptations of the neural network in an existing PDR algorithm from literature have on its accuracy, and is there a bottleneck to be found that limits the accuracy of the algorithm?*

By testing the influence of several network parameters and in- and output processing strategies, their influence was understood as follows.

Baseline performance To start off, it was seen that the type of input data fed to the model matters a lot. The two baselines, an adapted version of TLIO [1] and DIVE [19] show great differences in both network prediction accuracy as well as eventual drift rate of the full system. It was found that on Leica's dataset, using only IMU data in the local frame l as in TLIO yields better results than using the pre-processed input from DIVE, which replaces gyroscope data with the orientation vector of the device.

Combining baseline models input Including the tilt of the device as extra input on top of the adapted TLIO model showed a small improvement in the median drift rate from 8.1% to 7.9%, as this may have given the neural network additional information on how the person was holding the device during data capture.

Downsampling A frequency analysis of the IMU data revealed that the useful signal these sensors capture is no higher than 25 Hz, and if there were any useful signal at higher frequencies, they are absorbed by the white noise in the sensor measurements. Therefore, the network's performance using a downsampled version of the input data was examined and the experimental results have shown that an IMU sampling rate of 50 Hz instead of 200 Hz was also sufficient to obtain equally accurate results.

Longer input windows Using a longer window of IMU and tilt data as the network's input created a network that could predict the velocity with higher accuracy, but increased the bias in the predicted velocity. This bias is a violation of the EKF assumption that the measurements in the measurement update are zero-mean, and thus creates a larger error in the EKF, resulting in a larger drift rate for the networks with longer input windows.

Pre-processing methods Using the history of estimated device orientations from the EKF to pre-process the IMU data (Algorithm 3), instead of using backwards integration (Algorithm 1) to do so, reduced the bias in velocity prediction error. However, it also introduced larger prediction errors for the models using long input windows when used in the EKF, causing this method to show mostly similar performance to the backwards integration method.

Conclusion and Discussion In the end, the largest bottleneck in the neural network was found to be its bias on the velocity predictions, as this is a direct violation of the EKF assumptions. Using longer input windows increased this bias, so the network resulting in the most accurate position prediction, with a median drift rate of 7.9% and 95th percentile of 30.9%, was the network that used only 1 second of input data, preprocessed using the backwards integration method (Algorithm 1) at a sampling frequency of 50 Hz, using a combination of IMU input data plus the tilt angle of the device with the vertical axis, as shown in Figure 3-11 and Table 3-4.

Although this chapter revealed the bias as the bottleneck of the networks, the source of this bias hasn't been discussed. Three potential causes of this bias are hypothesized. The first is that it is a result of the slowly changing bias in IMU data. The network may have learned to use integration of the input IMU data, creating integration drift [7] in predicted velocity due to integration of the IMU bias. The IMU bias generally changes slowly over time [7], so it would fit the pattern of the slowly changing bias in velocity predictions as seen in Figure 3-14. Secondly, it could be a result of a slowly evolving error in estimated heading of the EKF, which may create wrongly preprocessed input data to the network, and therefore a biased output. Another reason could be due to the difference in walking patterns in the training and testing set. The network was trained on a handful of people who all had their own walking pattern. In the testing set, different people, and thus different walking patterns, may not be entirely recognized correctly by the network. Therefore, the predicted velocity of the network will be slightly erroneous. These walking patterns don't necessarily change quickly (reasons for change may be changing from walking a straight path to a curved path, or walking uphill or downhill), and therefore the error due to the unrecognized walking pattern would also be similar, which therefore results in this bias. Further research in the evolution in IMU bias or the similarity of the walking patterns in the IMU data could be done to find out if any of these reasons could be the source of this bias.

Chapter 4

Physics-Informed Neural networks for pseudo-measurements for an EKF in PDR

The previous chapter has shown how solely data-driven neural networks behave on this Pedestrian Dead-Reckoning (PDR) problem. It showed that the biggest bottleneck to be overcome is the bias in the velocity prediction, which violates the assumption of the Extended Kalman Filter (EKF) that the measurement error is a distribution with a mean value of zero [20]. Using longer input windows (Section 3-4) or a different preprocessing method to rotate the input data (Section 3-5) was shown to have a negative effect on the bias in this velocity prediction, as indicated by the increased Average Absolute Moving Average (AAMA) value (3-8).

This chapter will focus on trying a different approach of training the neural network, and investigate the effects of this method on the performance of the PDR algorithm, as well as the prediction bias of the network and other sources of inaccuracies. A network inspired by the Physics-Informed Neural Network (PINN) (Section 2-3, [22, 23]) will be trained as the neural network to predict the velocity measurement used in the EKF, in order to answer the second research question

> *Can the data-driven neural network in an existing PDR algorithm be adapted to accommodate for a physics-informed loss term, and does this adaptation lead to increased accuracy of the PDR algorithm?*

All networks trained in this chapter use a 4 second window of Inertial Measurement Unit (IMU) data. Although the network using 1 second of input data performed best in Chapter 3, it is hypothesized that the PINN can retrieve more physical information from a longer window and show clearer results using a 4 second window as a baseline. A common issue with training PINNs is that their training time is much longer compared to data-driven neural networks, due to the complex nature of the physics loss function [22]. In an attempt to keep training times manageable, the orientation history method from Algorithm 3 is used as

preprocessing method for the input data, as Section 3-5 has shown the training time of this method is much shorter compared to the backwards integration method without impacting the performance significantly. To simplify the base model and data loss function for the PINNs further, the uncertainty prediction of the network is left out and replaced with the linear model from Section 3-3. The baseline against which the resulting models will be compared is the model from Section 3-4 using 4 seconds of input data sampled at 50 Hz, preprocessed with the backwards integration method in Algorithm 1.

This chapter is built up as follows: first, the method of training the PINNs will be explained in Section 4-1, after which the effect of the PINNs on the EKF and further investigations on the performance of the standalone networks will be discussed in Section 4-2. Lastly, the most promising network architectures and training methods have also been applied using an input window of 1 second, as this window length resulted in the best performing PDR algorithm in Section 3.

4-1 Method

The baseline 1D ResNet architecture from [1] as used in Chapter 3 only predicts the velocity at the final time index of the input window, $t = T$. However, the loss function as in (2-14) requires velocity predictions not just at the end of the time window ($\hat{v}^l_T$), but also in between ($\hat{v}^l_t \quad t \in [0, T]$). Thus, the network architecture was adapted to allow velocity predictions over the entire window.

Two methods have been tested to predict this velocity at multiple timestamps. The first is the classic approach as described in [22], where a scalar query time $\tau \in \mathbb{R}$ is fed to the network as additional input, specifying at which moment in the input window the velocity should be predicted. Therefore, the output of such a network is a continuous function of time τ: $\hat{v}^l(\tau)$ with shape $(3, 1)$. The derivative $\dot{\hat{v}}^l(\tau)$ that is required for the physics loss in (2-14b) can be computed using auto-differentiation that computes the partial derivatives of the three axes of output $\hat{v}^l(\tau)$ with respect to the time input τ. A second approach is based on [34], which describes that an approximation of the differential operator in the physics loss can also be sufficient for convergence of the training process of the network. For this method, the network returns an entire grid of predicted velocities $\hat{v}^l_t$ at (a subset of) the time indices $t \in [0, T]$ in the input window, therefore the output is of shape $(3, T_{\text{out}})$, with T_{out} the amount of velocity samples in the output window. Both methods, and the network architectures used, are explained in more detail below.

4-1-1 Query time method

Queried ResNet The 1D ResNet model as described in section 2-2 was already proven to work well in the PDR algorithm [1]. Even though most applications of a PINN use a standard fully connected network [22, 23], it was chosen to use the 1D ResNet architecture from [1] (see Section 2-2) as a base since it is known from [1] that this network can perform well in the PDR algorithm. The network input has been adapted to include a scalar value that represents the query time τ. As the original network, visualized in Figure 2-2, takes a matrix of shape $(7, T)$ as input (3D accelerometer, 3D gyroscope, and 1D device tilt data for T samples),

it has been chosen to add an additional row to this matrix that represents the query time, by feeding the scalar τ through a small fully connected layer that returns a matrix of shape $(1, T)$, which will be concatenated with the matrix containing IMU and tilt data to form a $(8, T)$ matrix at the input of the ResNet, see Figure 4-1. Scaling has been applied to the query time input τ such that it always stays between 0 and 1, as neural networks train better when input values are kept relatively low [21]. As mentioned earlier, to simplify the training process, the uncertainty prediction has been left out and the network only returns a vector of $(3, 1)$ representing the predicted velocity.

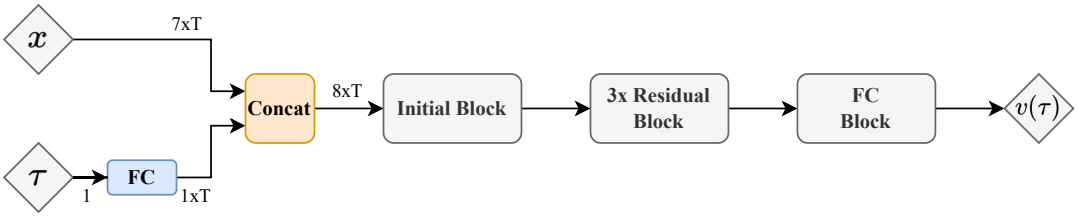

Figure 4-1: Schematic of the Query ResNet architecture. It is similar to the ResNet without uncertainty prediction in Figure 3-7, but with an extra channel in the input matrix representing the query time τ at which the velocity output $\hat{v}^l(\tau)$ should be predicted. Details of the blocks can be found in Appendix A-1.

Using the PyTorch backpropagation function, the time derivatives of the returned velocity prediction vector, $\frac{\delta \hat{v}^l(\tau)}{\delta \tau}$, can be computed after each forward pass through the network. This, however, is a time consuming process, but does yield accurate results. To compute this partial derivative accurately, a PINN requires continuously differentiable layers in the network that is trained [22, 23]. Two components of the ResNet have been modified to comply with this requirement. First of all, the ReLU activation function, which is used in all blocks (see Figure A-1 and A-2 for details), is not differentiable at an input value of zero and has a derivative of zero for any negative input values. This activation function has been replaced by the continuously differentiable hyperbolic tangent function $\tanh(\cdot)$. Also, the (1D) batch normalization layers [40] used in the initial and residual blocks (Figure A-1) cause a problem in the physics loss as the physical meaning, computed by the derivative over the entire network, will change depending on the batch statistics (mean μ and standard deviation σ) of the layer's input. Although this 'batch noise' can be beneficial for network generalization in many cases, it has been shown to pose a problem for some network architectures already [41]. To keep the derivative and therefore the physical meaning independent of the batch statistics, the normalization parameters (μ, σ) have been included as learnable parameters in the network instead.

4-1-2 Grid output method

Although the method with a query time input to the neural network is used in most papers that use a PINN [22], the process of auto-differentiation through backpropagation can be a very time consuming process. Therefore, inspired by [34], the method of approximating the differential operator through numerical derivatives has been explored. Three different network architectures that return an entire grid of velocity predictions $\hat{v}_t^l$ for $t \in [0, T_{\text{out}}]$ have been tested. These models took the same $(7, T)$ matrix of input data as the original network

described in Section 3-2, and returned a matrix of shape $(3, T_{\text{out}})$ consisting of T_{out} velocity predictions. Note that T_{out} and T do not necessarily have the same value, but in this chapter, the same sampling rate of 50 Hz was chosen for the input and output samples of the network, so they ended up to be the same.

Extended ResNet The first model that was tested was an extended version of the simplified 1D ResNet as in Figure 3-7, as this architecture was already proven to work well on PDR in [1]. The model used the same initial and residual layer structure, but had a separate fully connected block appended for every velocity sample in the output, as seen in Figure 4-2. The output sample rate was set at 50 Hz, identical to the sample rate of the input window. As the fully connected block contained the bulk of the parameters of the ResNet model, this approach increased the amount of trainable parameters of the model from 4.6 million (for the baseline model) to 163 million, an increase of over 35 times as large.

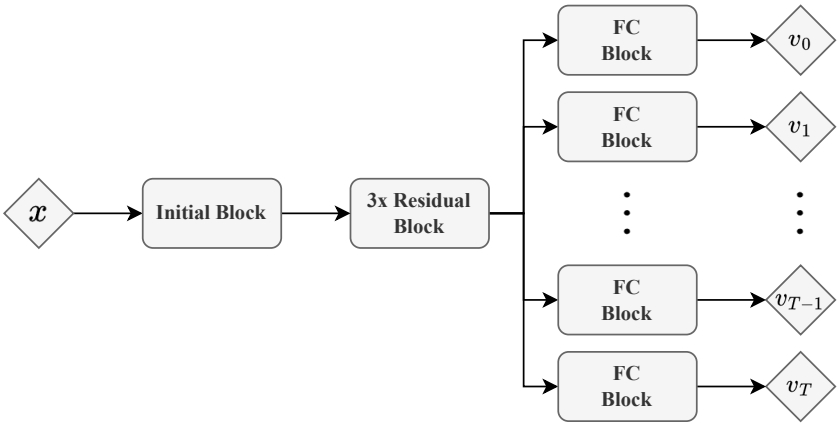

Figure 4-2: Schematic of the Extended ResNet architecture. The network is heavily based on the original 1D version of the ResNet from [1] without uncertainty prediction, as shown in Figure 3-7, except that multiple FC blocks are placed in parallel to predict the velocity for multiple timestamps in the window. Details of the blocks can be found in Appendix A-1.

UNet As the Extended ResNet is a network with excessively many parameters compared to the original 1D ResNet from [1], it was tested if a smaller network could achieve the same performance. A network architecture inspired by the UNet [42] has been created, see Figure 4-3. The first part of the UNet used in this thesis use the same residual downsampling layers as the 1D ResNet from [1]: it starts with several layers that use convolutions to downsample the input data. Instead of using separate fully connected layers to return the output samples as done in the Extended ResNet, an upsampling technique using transposed convolutions is used. The network also includes residual connections between the intermediate steps in the down- and upsampling that directly feed more detailed data to the other side of the network. The original UNet in [42] does not use the residual connections in the downsampling layers, but as the use of these layers has proven useful in [1, 19] and Section 3-1 on the PDR problem, the residual connections have been used in the downsampling part of the UNet as well. The network was set to predict a velocity at a sampling rate of 50 Hz again, similarly to the

Extended ResNet. By using this upsampling technique instead of a separate fully connected block for each sample in the output grid, the model size was kept at 5.8M parameters, only a small increase from the 4.6M in the original 1D ResNet.

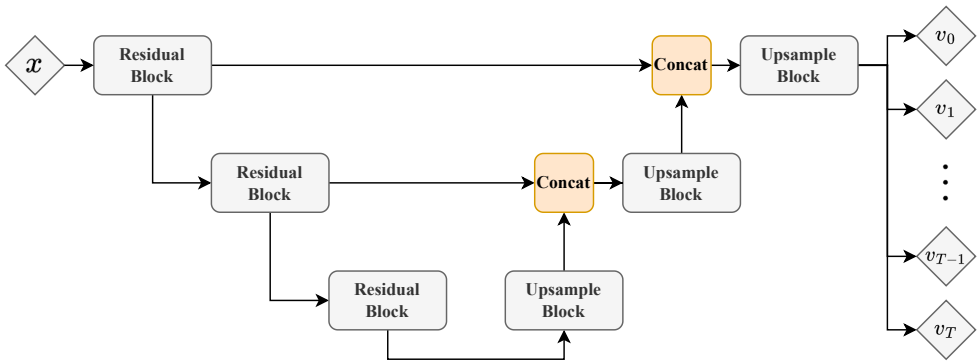

Figure 4-3: Simplified schematic of the UNet architecture used in this chapter. The structure is based on [42], but uses the residual downsampling blocks from [1]. Details of these residual blocks can be found in Appendix A-1. The upsample blocks are transposed convolutions that double the sampling rate on the time axis until the same sampling rate of 50 Hz is achieved at the output.

Transformer To test what performance a completely different network architecture could achieve compared to the commonly used ResNet, the third model architecture that was tested was a transformer [21, 43]. Although this architecture is most popular with large language models, it has been shown to work well on any form of sequential data [43], including time-series data [44], so therefore, this architecture was tried. There are also examples in literature to be found on combining the Transformer with a PINN for accurate predictions of sequential physical systems [45]. Figure 4-4 shows the top-level structure of the Transformer architecture used in this thesis. The 7 dimensional input samples are embedded to a higher dimension of 256, and are given a positional encoding that represents the time at which each sample is taken. This is added because the self-attention layers of the Transformer are invariant to permutations in the time dimension of the input data, and the important sequential information would be lost if there was no positional encoding. This encoding is included by adding a vector representing the time of the sample to the embedded input vector [43]. 16 layers of 32-headed self-attention with two fully connected layers of size 512 in between were used to make this model, resulting in 4.2M trainable parameters, a value similar to the original 1D ResNet. This resulted in a network that returned the predicted velocity at a sampling rate of 50 Hz.

Approximation of the differential operator

Each of the three model architectures used the same method of approximating the differential operator to compute the velocity derivative for the physics loss (2-14b) during training. For each velocity prediction on the interior of the predicted grid of velocities ($\hat{v}_t^l$ for $t \in (0, T) = [1, T-1]$), a second order polynomial fit was done through itself and its two

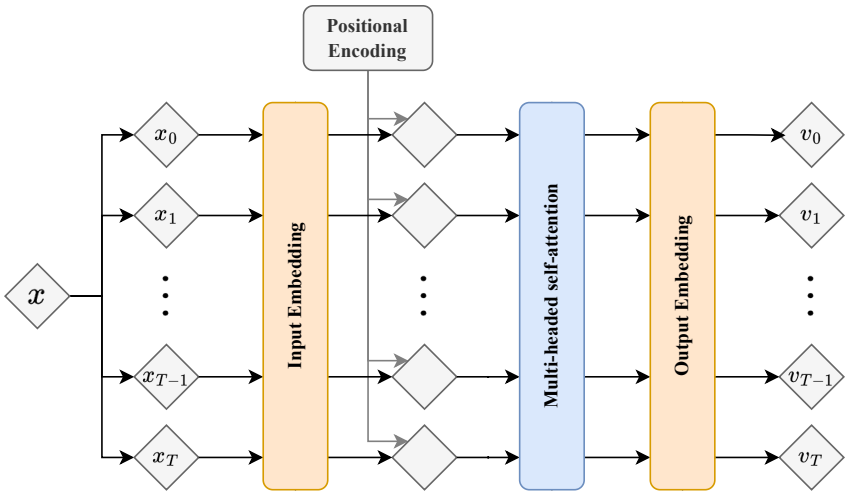

Figure 4-4: Simplified schematic of the Transformer model architecture [43, 21]. After embedding of the input to a larger dimension and adding a positional encoding that includes the time of each input sample in the embedding, the data is processed by the core of the Transformer consisting of 16 layers of multi-headed self-attention, each with two fully connected layers in between with hidden size 512. An output embedding decodes each sample to a 3-dimensional velocity.

neighboring velocity estimates $\hat{v}^l_{t-1}$ and $\hat{v}^l_{t+1}$. This polynomial fit assumes the three neighboring predictions are modeled as

$$\tilde{v}^l(\tau) = a\tau^2 + b\tau + c, \tag{4-1}$$

with $\tilde{v}^l \in \mathbb{R}^3$ the modeled value of the velocity prediction, $a, b, c \in \mathbb{R}^3$ the parameters of the polynomial, and $\tau \in \mathbb{R}$ the time in seconds. Setting $\tau = 0$ for the sample at which the derivative is computed ($\hat{v}^l_t$), and assuming a constant spacing of $\Delta\tau$ seconds with its two neighboring predictions $\hat{v}^l_{t-1}$ and $\hat{v}^l_{t+1}$, these predictions can be described as a system of equations

$$Y = F\Theta, \tag{4-2}$$

$$Y = \begin{bmatrix} (\hat{v}^l_{t-1})^\top \\ (\hat{v}^l_t)^\top \\ (\hat{v}^l_{t+1})^\top \end{bmatrix}, \quad F = \begin{bmatrix} \Delta\tau^2 & -\Delta\tau & 1 \\ 0 & 0 & 1 \\ \Delta\tau^2 & \Delta\tau & 1 \end{bmatrix}, \quad \Theta = \begin{bmatrix} a^\top \\ b^\top \\ c^\top \end{bmatrix}. \tag{4-3}$$

Because of the assumption that $\tau = 0$ for the sample to compute the derivative at, the derivative of the polynomial in (4-1) is simply $\frac{d\tilde{v}^l(\tau)}{d\tau}\Big|_{\tau=0} = b$, which can be computed using

$$\begin{bmatrix} a^\top \\ b^\top \\ c^\top \end{bmatrix} = \Theta = F^{-1}Y. \tag{4-4}$$

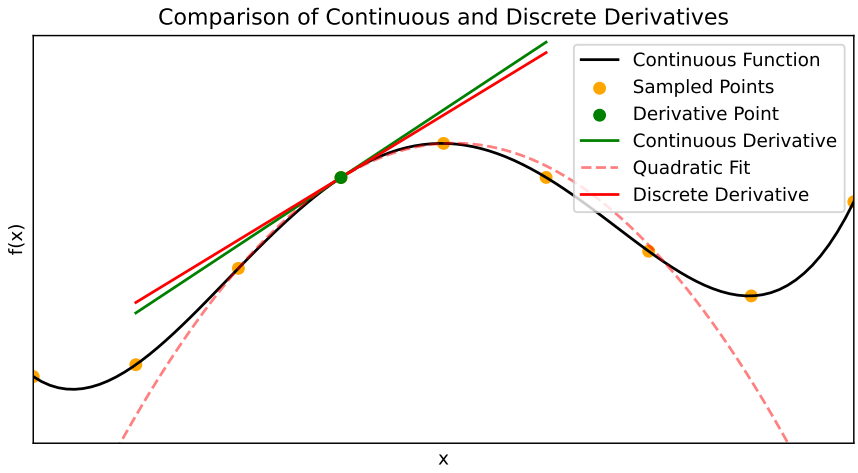

Figure 4-5: Example of a 1D function $f(x)$ from which the continuous derivative (green) has been taken using backpropagation, and a discrete derivative based on a quadratic fit using three neighboring samples of the function (red).

This operation can be done very efficiently for all time samples $t \in (0, T)$ on the interior of the window $[0, T]$, using batched tensor operations in PyTorch, which makes computing the numerical derivative a much faster approach than the method using backpropagation over the entire network. Note that as this numerical derivative can only be computed on the interior of the time window, the physics losses at time indices $t = 0$ and $t = T$ are not considered in the loss function. Figure 4-5 shows an example of this method compared to the backpropagation method. A slight approximation error is also seen in this figure, the size of which is dependent on the sampling rate of the discrete samples. As all the network architectures described above have a sampling rate of 50 Hz, it is assumed that this sampling rate is high enough and the approximated derivative is close enough to its continuous value such that it won't cause problems in the convergence of the physics loss.

These approximated derivatives can then be used in the physics loss function (2-14b) for training the three model architectures above as PINNs.

4-1-3 Loss function

As described in Section 2-3, the training of a PINN requires a loss function with a data loss and a physics loss.

A Mean Squared Error (MSE) between the predicted and true velocity has been used as data loss. As the network architectures as described in Section 4-1 are able to return velocity estimates at samples $t \in [0, T]$ in each time window $[0, T]$ for all data points $i \in [0, N]$, the loss to be computed for all these samples is

$$L_{\text{data},i,t} = ||\hat{v}^l_{i,t} - v^l_{i,t}||^2_2. \tag{4-5}$$

The physical equation that determines the physics loss is the accelerometer measurement model as in (2-1) [7],

$$y^b_{a,t} = R^{bl}_t(a^l_t - g^l) + \delta^b_{a,t} + e^b_{a,t}, \tag{4-6}$$

resulting in a physics loss at each sample of

$$L_{\text{phys},i,t} = ||\hat{y}^b_{a,i,t} - y^b_{a,i,t}||^2_2. \tag{4-7}$$

Again, i denotes the data sample ($i \in [1, N]$) and t the time sample within the window ($t \in [0, T]$). As the measurement $y^b_{a,t}$ is used in this loss, the amount of points at which this physics loss can be computed is limited to the amount of IMU measurements available, which were sampled at 50 Hz. The network architectures using a grid output all used a sampling rate of 50 Hz in the output window, whereas the architecture using query time as extra input was set to sample at only 25 Hz due to its large computational complexity. The prediction $\hat{y}^b_{a,t}$ for the measurement $y^b_{a,t}$ can be made using the measurement model (4-6),

$$\hat{y}^b_{a,i,t} = R^{bl}_{i,t}(\hat{a}^l_{i,t} - g^l) + \delta^b_{a,i,t}, \tag{4-8a}$$

$$\hat{y}^b_{a,i,t} = \left(R^{lb}_{i,t}\right)^\top \left(\dot{\hat{v}}^l_{i,t} - g^l\right) + \delta^b_{a,i,t}. \tag{4-8b}$$

During training, the rotation matrix $R^{lb}_{i,t}$ and accelerometer bias $\delta^b_{a,i,t}$ are known from the ground truth data of the dataset. The physics loss for each sample then becomes

$$L_{\text{phys},i,t} = ||\hat{y}^b_{a,i,t} - y^b_{a,i,t}||^2_2 = || \left(R^{lb}_{i,t}\right)^\top \left(\dot{\hat{v}}^l_{i,t} - g^l\right) + \delta^b_{a,i,t} - y^b_{a,i,t}||^2_2. \tag{4-9}$$

The total loss function used to train our PINN on then is the mean of the losses of each sample:

$$L = L_{\text{data}} + \lambda_p L_{\text{phys}} \tag{4-10a}$$

$$L_{\text{data}} = \frac{1}{3N(T+1)} \sum_{i=1}^{N} \sum_{t=0}^{T} ||\hat{v}^l_{i,t} - v^l_{i,t}||^2_2 \tag{4-10b}$$

$$L_{\text{phys}} = \frac{1}{3N(T+1)} \sum_{i=1}^{N} \sum_{t=0}^{T} || \left(R^{lb}_{i,t}\right)^\top \left(\dot{\hat{v}}^l_{i,t} + g^l\right) + \delta^b_{a,i,t} - y^b_{a,i,t}||^2_2 \tag{4-10c}$$

The factor $\frac{1}{3}$ has been added such that this computation can be done efficiently using a single PyTorch `mean(square(`e`))` function acting on the entire error tensor e of shape $(N, 3, T+1)$ that contains for each data sample, the x, y, and z error at all time samples in the window.

4-1-4 Implementation of the PINNs in the PDR algorithm

As the EKF in the PDR algorithm only uses the velocity prediction at the end of the time window ($\hat{v}^l(T)$ or $\hat{v}^l_T$) a small wrapper is placed around the network during its use in the EKF to ensure only this prediction is returned as output. For the Query ResNet, that means setting the query time τ, which is a value between 0 and 1 that indicates the fraction of time passed in the window, to 1. The Extended ResNet, UNet, and Transformer sliced all but the last velocity prediction from their output matrix. This way, the PINNs could still be implemented in the same EKF that only considered the latest velocity prediction. The wrapper also includes the predicted uncertainty based on the linear model from (3-4), which bases the values in the predicted uncertainty matrix on the magnitude of the horizontal and vertical predicted velocity.

4-1-5 Hypothesis of the effect of a PINN on the PDR algorithm

There are two ways in which the PINN architectures could be able to use additional information compared to the data-driven networks in Chapter 3. First of all, multiple velocity predictions in the 4-second window are made instead of one prediction of the velocity at the end of the time window. This requires the network to use more information on the state evolution over the entire window. Secondly, using a physics loss with $\lambda_p > 0$ forces the network to use physical relationships between the predicted velocities in the time window. Connecting multiple predictions to each other may improve the overall accuracy of these predictions. The largest bottleneck of the neural network was found to be its prediction bias in Chapter 3, and although this bias does not directly influence the loss function (4-10), using this physics loss and new model architectures may affect the bias anyway because the network is forced to use more information over the entire time window.

4-2 Results

This section presents and discusses the results of applying a PINN to the PDR problem. It is subdivided into four parts: the first part quickly summarizes the training process of the four networks, after which the performance of the full PDR algorithm using the EKF with the PINNs to predict the velocity for the measurement update is evaluated. The third part investigates the performance of the tested networks themselves, outside of the EKF, to explain their results on the full PDR algorithm. Lastly, the performance of the best PINNs from this section using a 1 second input window instead of 4 seconds is compared against the best network from Chapter 3.

4-2-1 Training specifications of the PINNs

Each of the four networks have been trained using four different physics loss weights λ_p in the loss function (4-10). The same 60/20/20 split into training, validation, and test data was used as described in Section 3-1, as well as the perturbations on the IMU data. With a learning rate of 5×10^{-5}, the networks were first trained on 10 epochs using only the data

loss ($\lambda_p = 0$ in (4-10)) to ensure convergence, after which the full loss function with physics loss was used until the validation loss reached a minimum.

A comparison of the size of the four tested networks, as well as the approximate time it took to train them until convergence of the loss value on the validation set is shown in Table 4-1. As mentioned earlier, it is seen that the Extended ResNet as in Figure 4-2 has much more trainable parameters than the baseline ResNet from TLIO [1], which causes the training time to increase greatly as well. The UNet (Figure 4-3) was made to be a more compact model, which resulted in a shorter training time compared to the Extended ResNet. The reason that the training time is still longer than the baseline ResNet is because of the additional complexity of the loss function, featuring the physics loss. This is in line with the statements from PINN literature [22, 46] that mention an increased complexity in training a PINN. The Transformer, because of its more complex architecture, also took longer to train, but the Queried ResNet took the longest to train, with 75 hours on average. This can be explained by both the computational intensity of the backpropagation to compute the derivative of the velocity output with respect to the time input, which is highlighted in [34], but also the fact that this network only returns one velocity estimate at the queried time in the window, whereas the other networks return an entire grid of velocity predictions that fill an entire window. This means that the grid output models require far less forward passes through the network to compute the loss of one epoch, which of course is far more efficient.

Output method	**Model type**	**Number of parameters**	**Approximate training time**
Single output at $t = T$	Baseline ResNet	4,6 M	0.8 h
Grid output	Extended ResNet	162 M	9 h
Grid output	Unet	5,9 M	3 h
Grid output	Transformer	4,2 M	12 h
Query time input	Query time ResNet	4,6 M	75 h

Table 4-1: Comparison of the network size and approximate training time of the baseline network (Section 2-2) and the networks described in Section 4-1.

4-2-2 Performance of the EKF using a PINN

Figure 4-6 shows the distributions of the standalone network, both using input IMU data rotated using ground truth orientation (left) and estimated orientation during a simulation (second from left), as well as the EKF velocity prediction error (third from left) and lastly, the drift rate (right). The numerical values of the important metrics in the plots are noted down in Table 4-2, which also contains the value for the AAMA as defined in (3-8). The performance of the different network architectures are discussed below.

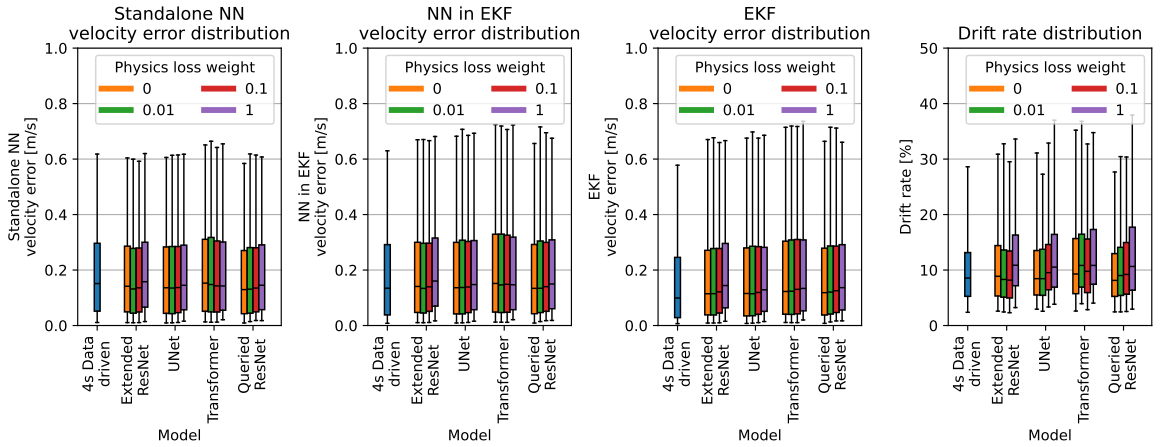

Figure 4-6: Distributions of the standalone network prediction error, the prediction error of the network during an experiment simulation, the EKF velocity error, and the drift rate, for the four PINNs described in Section 4-1, and four physics loss weights λ_p. The whiskers of the boxes represent the 95th percentile of the distribution.

Model Type		**4s data Driven**	**Extended ResNet**				**U-Net**				**Transformer**				**Queried ResNet**			
Physics Loss Weight		-	0	0.01	0.1	1	0	0.01	0.1	1	0	0.01	0.1	1	0	0.01	0.1	1
Standalone velocity error [m/s]	Median	.15	.14	.13	.14	.16	.14	.14	.14	.15	.15	.15	.14	.14	**.13**	.13	.14	.15
	95%	.63	.60	.60	.59	.62	.61	.61	.62	.62	.65	.66	.64	.66	**.58**	.62	.61	.61
Drift Rate [%]	Median	8.5	8.9	8.3	8.2	10.9	8.5	8.5	9.5	10.5	9.3	10.8	9.8	10.8	**8.1**	9.0	9.2	10.6
	95%	32.8	30.9	29.8	29.5	33.6	31.1	**27.2**	32.9	37.0	35.2	36.8	32.7	34.8	27.7	30.4	30.4	37.9
AAMA [m/s]		.104	.088	.088	**.086**	.102	.091	.093	.095	.102	.100	.104	.102	.108	.089	.094	.094	.100

Table 4-2: Numerical values of the median and 95th percentile in the boxplots in Figure 4-6, as well as the AAMA (as defined in (3-8)). Best values are highlighted in bold.

Extended ResNet The Extended ResNet (Figure 4-2) shows a slight decrease in median velocity prediction error from 0.14 m/s to 0.13 m/s when a small physics loss weight $\lambda_p = 0$ is applied instead of $\lambda_p = 0$, whilst median drift rate decreases from 8.9% to 8.3%. However, applying more weight to the physics loss, with $\lambda_p = 1$, results in increased median drift rate of 10.5%. This is not only explained by the increased velocity prediction error of the standalone model (to 0.16 m/s), but also by the larger bias in this error, as seen from the AAMA of 0.102 m/s. Therefore, these results show that for the Extended ResNet, using a physics loss can decrease the drift rate of the PDR algorithm, but only if its weight λ_p is kept in the orders of 0.01 and 0.1.

UNet The UNet (Figure 4-3) shows a different reaction to the physics loss as compared to the Extended ResNet. The network trained on a pure data loss (with $\lambda_p = 0$) shows the optimal performance, with a median velocity prediction error of 0.14 m/s and drift rate of 8.5%. Increasing the weight of the physics loss leads to slightly larger velocity prediction errors and an increase in prediction bias as seen from the AAMA increasing from 0.091 to 0.102 m/s. Both the increased prediction error and the larger AAMA contribute to the increased drift rate from 8.5% for the UNet without a physics loss to 10.5% for the UNet using $\lambda_p = 1$. Thus, using a physics loss negatively impacts the performance of the PDR algorithm when a

UNet architecture is used.

Although the UNet uses far less parameters compared to the Extended ResNet (5.9 M compared to 162 M), the performance of the two networks is very similar, so the upsampling technique used in the UNet shows to be an efficient way of reducing network size whilst retaining performance.

Transformer Using the Transformer (Figure 4-4) as a data-driven network, with $\lambda_p = 0$, resulted in a slightly higher velocity prediction error than the Extended ResNet and UNet, with a median value of 0.15 m/s, as well as a larger bias indicated by the AAMA of 0.100 m/s. This caused a higher drift rate of 9.3%. This indicates that the ResNet architecture proposed in *Liu et al.* [1] was indeed a better choice for this PDR problem. However, when the Transformer is used in combination with a physics loss, the velocity prediction error seems to decrease slightly to 0.14 m/s for the larger physics losses. This is different behavior than shown by the Extended ResNet and UNet, which have shown worse results when being trained using higher physics loss weights. Although the prediction error decreased, the bias in this prediction did not follow this trend, but showed sporadic jumps in the interval between 0.100 and 0.110 m/s. This resulted in the same jumps in drift rate, which varied between 9.3% and 10.8% without showing a clear relation to the physics loss weight λ_p. A plausible explanation for this behavior is that the Transformer architecture as used in this project is more susceptible to 'training noise' as a result of the randomness in the training process, and thus the differences in performances may be independent of the physics loss weight, but arise out of this noise.

Queried ResNet The Queried ResNet (Figure 4-1) shows the best performance when it is used with $\lambda_p = 0$, with a median drift rate of 8.1%. This result comes from a combination of an accurate velocity prediction (with a median error of 0.13 m/s) and a relatively low bias (with an AAMA of 0.089 m/s). When a physics loss is included in the training, this architecture shows the same behavior as the UNet: both the velocity prediction error and the AAMA become larger with increasing physics loss, and therefore, the drift rate increases significantly to 10.6%.

Thus, most networks show a decreased performance of the PDR algorithm when the weight of the physics loss is increased, with the exception being the Extended ResNet that benefits from a small weighed physics loss. Only the Extended ResNet and Queried ResNet are able to achieve a lower median drift rate than the baseline model from Chapter 3, which had a median drift rate of 8.5%. The Extended ResNet performs the best with a physics loss weight of $\lambda_p = 0.1$, achieving a median drift rate of 8.2%, and the Queried ResNet works better as a data-driven neural network, achieving a median drift rate of 8.1%.

In Sections 3-4 and 3-4, it was concluded that the EKF drift rate increases because of the prediction of the velocity prediction bias increases, which was shown using the AAMA. The same trend appears to show up here again, but when the four architectures are compared against the baseline from Chapter 3, it is seen that even with a high velocity prediction error (0.15 m/s) and AAMA (0.096) this baseline is able to outperform most of the new model architectures with its median drift rate of 8.6%. The data in Figure 4-2 and Table 4-2 cannot

explain why this happens. Moreover, it is also still unclear why the Extended ResNet is the only architecture that benefits from a small physics loss.

To explain these remaining questions, the next section will examine the predictions of the standalone network more thoroughly. This section only studied the integration of the PINN with the EKF, and its relevant network output (the predicted velocity $\hat{v}_T^l$ at the end of the time window) used in this EKF. However, this is not the only prediction returned by these PINNs, and therefore the next section will focus on the entirety of the network output.

4-2-3 Effect of the physics loss on the standalone networks

It has been shown that it depends on the choice of network architecture if the usage of a PINN shows increased performance of the PDR algorithm or not. A big difference these PINNs have compared to the original baseline is that the baseline only returned one prediction at the latest time of the input window ($\hat{v}_T^l$), whereas these PINNs return a velocity prediction over the entire window ($\hat{v}_t^l$ for $t \in [0, T]$), either by returning a grid of velocities at discrete points in time, or by using the query time τ as input to return a velocity prediction at that time. This section focuses on the performance of the network itself, to show the effect of the physics loss on its full output, not just the latest velocity prediction used by the EKF.

Velocity prediction error In the left plot of Figure 4-6, the velocity error distribution of the prediction at the end of the time window ($t = T$) has been shown. This distribution was also evaluated for all the other time samples in the time window, and Figure 4-7 shows this distribution for the Extended ResNet model with different physics loss weights, with the time in the input window on the x-axis. Two phenomena can be seen in these plots. First of all, the prediction error distribution at any moment in the time window does not seem to change significantly when a physics loss is added. Figure 4-6 already showed this was the case for the final sample in the time window, but Figure 4-7 confirms this for the rest of the window. Secondly, the error distribution is U-shaped, with the center of the window being more accurate than the sides. This effect is similar to the edge-effect in smoothing filters [47] where the edges of sequential data are more erroneous than data in the center.

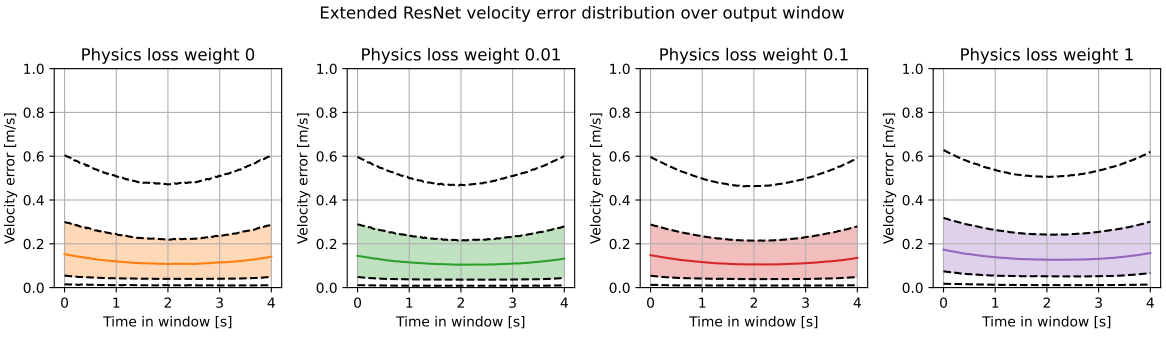

Figure 4-7: Velocity prediction error distribution over the entire time window as predicted by the Extended ResNet. The figures are made up of boxplots of the error distributions on all times separately, and a continuous line is drawn between the boxplot features. The whiskers of the boxplots, visualized as the highest and lowest dotted lines, show the 5th and 95th percentile.

The reason for this is that as the data beyond the edges of the window is unavailable, the network has less information to predict the velocity estimate close to the edges. As the error distribution remains similar over the entire time window when a physics loss of different weights is applied, it is concluded that either the physics loss has not learned the model well enough to use the physical relations for its prediction, or that using these physical relations does not give the network more information to generate a more accurate velocity prediction. The UNet, Transformer, and Queried ResNet architectures show the same behavior on the velocity prediction and are plotted in Appendix A-2-1. To figure out if the network did learn to use some physical information, the predicted acceleration of the networks has been analyzed as well.

Acceleration prediction error Although not a direct output of the PINNs from Section 4-1, the acceleration prediction during the entire window plays an important role in the physics loss (4-10) and can be computed as described in the same section; either by doing a backwards pass through the network with a query time as input, or by approximating the differential operator using a polynomial fit. The acceleration error distribution of the Extended ResNet for the entire window is plotted in Figure 4-8. In this plot, a first thing that is noticeable is that the error distribution decreases vastly when the physics loss weight is increased. This is not surprising, as the acceleration error is exactly what is used as physics loss, so increasing its weight in the loss function should decrease this error. A second, more interesting, phenomenon in this plot is the fact that the U-shaped error as in Figure 4-7 disappears with the introduction of the physics loss, and the error distribution is (roughly) constant over the entire time window. The useful physical information for this acceleration estimation at any point in the window is only contained in the IMU measurement at that specific time point, as the accelerometer measurement $y^l_{a,t}$ gives the acceleration directly, albeit with a little bias and noise. If the network were to use this information, there would be no difference in the amount of useful information wether the prediction is halfway the time window or close to the start or end, contrary to the pattern recognition that pure data-driven networks are based on, which suffer from less useful information close to the edges of the sequence and result in the edge effect [47]. This motivates the claim that by adding the physics loss, the network does in fact use this physical information that it does not use without the use of a physics loss. The effect of this, however, only reflects in an improvement

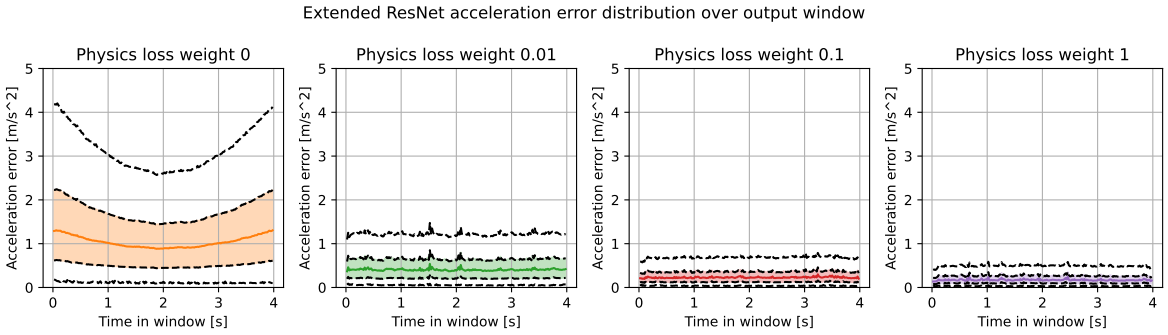

Figure 4-8: Acceleration prediction error distribution of the entire window as predicted by the Extended ResNet. Acceleration prediction is derived from the velocity prediction by using a polynomial fit as described in (4-1).

in the predicted acceleration, not the velocity as shown in Figure 4-7.

The same error distribution plots are also made for the UNet architecture (Figure 4-9) using different physics loss weights. This time, there is a clear difference in the acceleration prediction compared to the Extended ResNet in Figure 4-8. The UNet appears to be much more accurate in acceleration estimation than the Extended ResNet, even without the physics loss in play. A reason for this may be the convolutional nature of the UNet [42]. In the Extended ResNet, each output in the window is the result of a separate linear block (see Figure 4-2), which makes it more likely for noise to appear between neighboring samples due to the linear blocks being independent. The UNet, however, uses convolutional layers to generate the output, causing the samples to be much more connected and less susceptible to individual noise. This also explains the slight increase in error at the edges due to the edge effect of the convolution [48]. Moreover, the edge effect that causes larger errors at the edges of the windows starts appearing much closer to the edge than for the Extended ResNet. Two reasons for this are speculated. One reason is that the UNet uses more local information to generate its prediction than the Extended ResNet, causing the loss of information to happen at a much closer point to the edges as the prediction is less dependent on input data in the far past and future. Another reason could be that the UNet, without physics loss, already uses the physical information of the input data in addition to the data-driven pattern recognition, and thereby reduce the pattern recognition errors which create the large edge effect errors. Increasing the physics loss weight λ_p does still further reduces the edge effect, just as seen for the Extended ResNet in Figure 4-8.

The error distribution of the Transformer (Figure A-6) and Queried ResNet (Figure A-7) show similar behavior as the UNet and are shown in Appendix A-2-2.

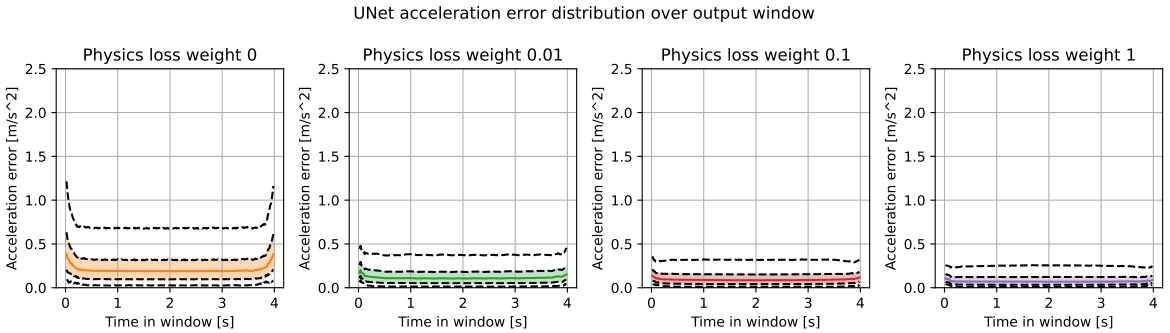

Figure 4-9: Acceleration prediction error distribution of the UNet.

Zooming in on one prediction window The improvement in acceleration as in Figure 4-8 does not translate into an improvement in velocity predictions as seen in Figure 4-7, but that doesn't mean there is no difference in how the velocity is predicted over the entire window. Figure 4-10 shows an example of one velocity window in the unseen test dataset, predicted by the Extended ResNet. It can be seen that the model trained without physics loss shows a lot of noise, whilst the model trained with the highest physics loss ($\lambda_p = 1$) returns a much smoother curve that follows the shape of the ground truth velocity quite well. To quantify this noise in the velocity prediction at the end of the time window, the standard deviation of the horizontal prediction errors of the entire test set has been computed and written down in Table 4-3, along with the prediction bias, quantified by the AAMA (3-8). It is seen that the

noise for the Extended ResNet is 0.041 m/s without using a physics loss, and reduces to 0.018 even when using a small physics loss weight in the loss function. This noise is the reason why the physics loss had such a large effect on the acceleration error. However, although the predicted velocity window is much smoother when a physics loss is added, the prediction itself still suffers from a slowly changing bias on the horizontal prediction, as the presence of a bias does not change the physics loss term that looks at the derivative of the velocity. This is seen clearly in Figure 4-10 on the x and y velocity predictions. The value of this offset at the end of the predicted window is the bias as measured by the AAMA, and it is seen in Figure 4-10 that the addition of a large physics loss in training does not reduce this bias, as can be confirmed by AAMA values in Table 4-3, which even shows an increasing value of this bias for larger λ_p, eventually leading to a worse performance of the EKF and drift rate, as already shown in Table 4-2.

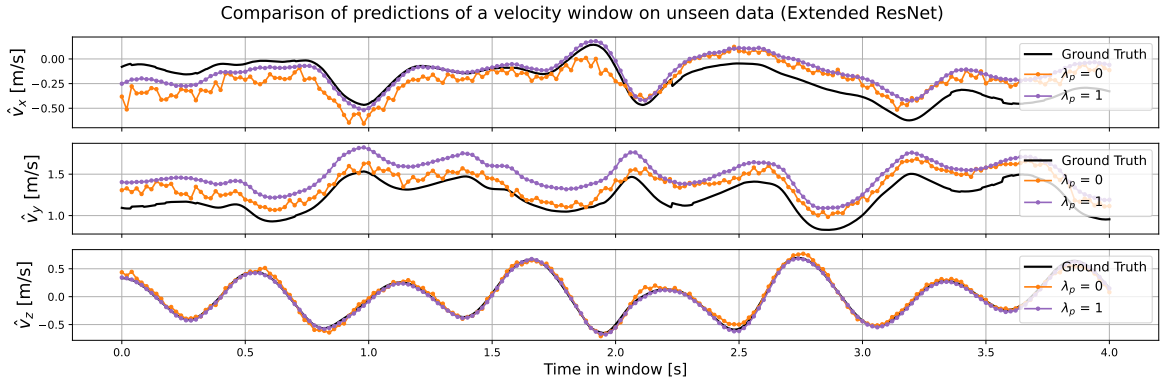

Figure 4-10: Ground truth and predicted velocity by the Extended ResNet of one data sample of a four seconds window in the unseen test set.

Model Type	**Extended ResNet**				**U-Net**				**Transformer**				**Queried ResNet**			
Physics Loss Weight	0	0.01	0.1	1	0	0.01	0.1	1	0	0.01	0.1	1	0	0.01	0.1	1
Bias (AAMA) [m/s]	.088	.088	**.086**	.102	.091	.093	.095	.102	.100	.104	.102	.108	.089	.094	.094	.100
Noise [m/s]	.041	.018	.017	.023	**.011**	.014	.015	.020	.027	.015	.013	.014	.020	.019	.019	.020

Table 4-3: Comparison of the average horizontal velocity prediction bias (AAMA) and noise (standard deviation) of the velocity prediction at the end of the time window, computed on the entire test set.

The predicted window for the UNet with and without physics loss is shown in Figure 4-11. Here, it is seen that the UNet without physics loss already has a lot less noise compared to the Extended ResNet, which is also visible from Figures 4-8 and 4-9 that show a great decrease in acceleration error between the Extended ResNet and the UNet. This shows why the addition of a physics loss was less effective in reducing the acceleration error of the UNet (Figure 4-9) than that of the Extended ResNet (Figure 4-8). The Transformer and Queried ResNet predictions of this one window are plotted in Figures A-8 and A-9. They also show that the models without physics loss include less noise than the Extended ResNet prediction, and therefore benefit less from a physics loss.

Ultimately, the magnitude of the horizontal standard deviation in the noise of the PINNs is usually around 0.02 m/s. Compared to the average horizontal bias in the prediction error

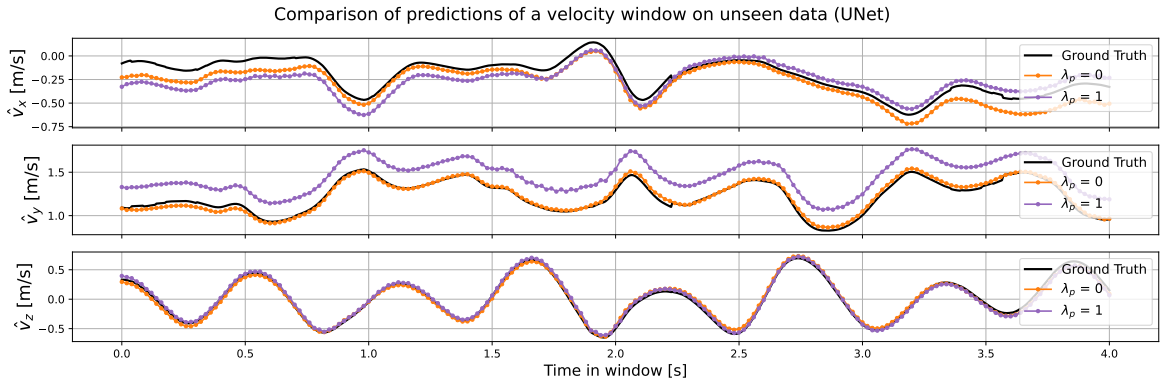

Figure 4-11: Ground truth and predicted velocity by the UNet of one data sample of a four seconds window in the unseen test set.

around 0.09 m/s, the noise is over four times lower. Moreover, the EKF is build to work with gaussian noise [20], and thus this noise should have less effect on the EKF estimates than the prediction bias. It is therefore concluded that this noise reduction, by either using a small physics loss during training of the Extended ResNet or by using a different model architecture like the UNet that inherently has less noise, does not impact the performance of the EKF much and therefore the drift rate doesn't see an improvement when a physics loss is used.

An exception of this statement is the Extended ResNet trained with $\lambda_p = 0$. Here, the standard deviation of the horizontal noise was significantly higher than all other network architectures: 0.041 m/s. Even though it is still low compared to the AAMA 0.088 m/s, adding a physics loss reduces this noise significantly to less than half this value. This could be the reason why the Extended ResNet was the only architecture to benefit from a slight physics loss component in the loss function.

4-2-4 Using a PINN on a network with 1 second of input data

Chapter 3 revealed that the best performing network uses 1 second of data as input. This chapter has shown in Table 4-2 that the Extended ResNet using $\lambda_p = 0.1$ and the Queried ResNet with $\lambda_p = 0$ are able to outperform the baseline network in terms of median drift rate. This baseline was the network using 4 seconds of input data and the Backwards Integration pre-processing method, as in Table 3-4. Therefore, these two best performing new architectures have been tested using 1 second of input data instead of 4, to see if the improvement over the 4 second baseline translates to a network using a shorter input window. Note that the Queried ResNet performed best without using a physics loss, and is therefore not a PINN, but a purely data-driven neural network.

Figure 4-12 shows the results these networks achieve, with detailed values of the standalone performance of the network and resulting drift rate of the full algorithm in Table 4-4. The error distribution of the standalone network using a window size of 1 second increases compared to the 4 seconds input window networks. The median value of this error (0.17 and 0.18 m/s for the Extended ResNet and Queried ResNet, respectively) is approximately the same as for the baseline network using 1 second of input data, which has a median of 0.18 m/s. However, where the baseline network achieves a median drift rate of 7.9% in the full algorithm, which is

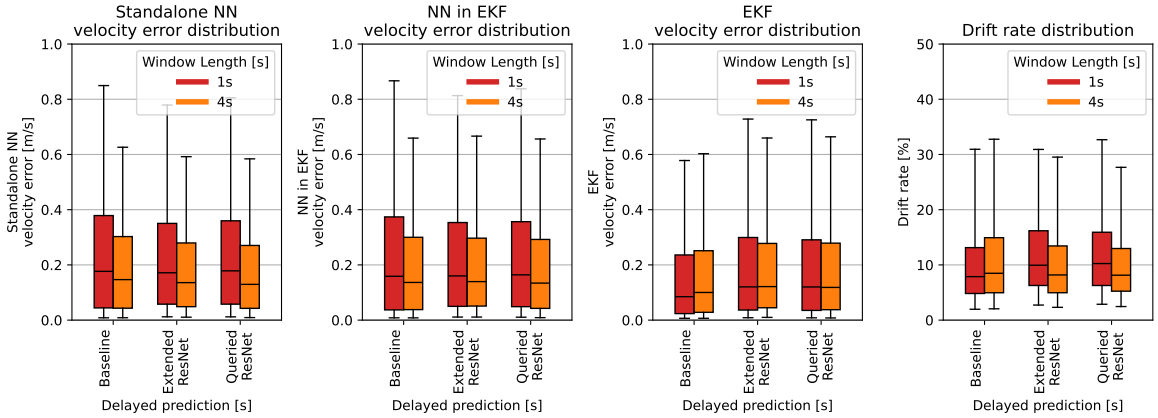

Figure 4-12: Distributions of the standalone network prediction error, the prediction error of the network during an experiment simulation, the EKF velocity error, and the drift rate, for the baseline models from Chapter 3, the Extended ResNet with $\lambda_p = 0.1$, and the Queried ResNet with $\lambda_p = 0$. The baseline models are trained with 50 Hz input data, preprocessed using Algorithm 1. The whiskers of the boxes represent the 95th percentile of the distribution.

Model Type		**Baseline (50Hz, BI)**		**Extended ResNet** $\lambda_p = 0.1$		**Queried ResNet** $\lambda_p = 0$	
Window Time [s]		1	4	1	4	1	4
Standalone velocity error [m/s]	Median	0.18	0.15	0.17	0.14	0.18	**0.13**
	95%	0.85	0.63	0.78	0.59	0.81	**0.58**
Drift Rate [%]	Median	**7.9**	8.5	9.9	8.2	10.2	8.1
	95%	30.9	32.8	30.9	29.5	32.7	**27.7**
AAMA [m/s]		0.097	0.104	0.096	**.086**	0.099	0.089

Table 4-4: Numerical values of the median and 95th percentile in the boxplots in Figure 4-12, as well as the AAMA (as defined in (3-8)). Best values are highlighted in bold.

less than the same network using 4 seconds of input, the Extended and Queried ResNets, on the contrary, see an increase in this drift rate. The Extended ResNet has a median drift rate of 9.9% and the Queried ResNet has 10.2%. Thus, even though the data driven network was found to be better when using shorter input windows in Section 3-4, the two new architectures only show improvement when a 4 seconds input window is used, but the best overall network is still the baseline using 1 second of input data. A reason why this drift rate is increasing is reflected in the AAMA, representing the bias of the velocity predictions. This bias also decreases for shorter windows using the baseline ResNet, but the Extended and Queried ResNets see an increase in this bias, resulting in a larger drift of the predicted position.

The reason why these two network architectures perform better for a longer input window cannot be deduced from these two examples alone. It can be speculated that it could be a cause of the nature of the training data. The baseline ResNet only compares the velocity prediction at the end of the time window with its ground truth in the loss function, whereas the two new architectures use these values from the entire time window. Therefore, using a larger window results in more training data, which could improve the performance of the prediction at the end of the window as well.

4-3 Concluding Remarks

This chapter aimed towards answering the research question

> *Can the data-driven neural network in an existing PDR algorithm be adapted to accommodate for a physics-informed loss term, and does this adaptation lead to increased accuracy of the PDR algorithm?*

To answer this question, two approaches have been tested to include this physics loss, which requires a time derivative of the output, in the loss function: using a queried network that returns a velocity prediction at a requested time and can be differentiated using backpropagation, and using a network that returns velocity predictions at a grid of multiple output times which can be used for a numerical derivative. The tested architectures have been trained as a purely data-driven neural network (with $\lambda_p = 0$) and as a PINN (with $\lambda_p \in \{0.01, 0.1, 1\}$) and were evaluated in the EKF to test their effect on the performance of the PDR algorithm.

Query time method An adapted version of the ResNet as in *Liu et al.* [1] has been tested, which requires an extra scalar as input that represents the time in the input time window at which the velocity should be predicted. It was shown that using this architecture as a PINN worsened the performance based on median drift rate of the resulting PDR algorithm, as the prediction bias increased when larger weights were used for the physics loss. The data-driven version of this architecture, however, did see an improved performance over the baseline model when a 4 seconds time window was used for the input data.

Grid output method Three models have been tested using this method: an extended version of the ResNet from *Liu et al.* [1], a UNet, and a Transformer. The Extended ResNet showed lower drift rates when trained as a PINN as opposed to being solely data-driven, because the physics loss reduced the high noise in the velocity predictions in one window. This architecture was also the only one of the three to achieve a better drift rate than the baseline network of 4 seconds. The UNet performed best as a data-driven network, and using any physics loss increased the bias in velocity prediction and therefore the drift rate of the full PDR algorithm. The Transformer did not show a consistent trend between the weight of the physics loss and the drift rate, but achieved the best performance as a data-driven network.

The data-driven Queried ResNet and physics-informed Extended ResNet have been trained on 1 second input and prediction windows as well, but contrary to the original data-driven ResNet in Chapter 3, these networks showed decreased performance of the PDR algorithm based on median drift rate, as the prediction bias on the velocity was larger.

The hypothesis in Section 4-1-5 said that the PINNs may affect the network's performance in two ways: because it uses additional information of the velocity at all times in the window, and because it links these velocities together using the laws of physics. The results of the four new network architectures that predict an entire window of velocities, trained with $\lambda_p = 0$, do not show any significant difference in performance of the PDR algorithm, so the first hypothesis is incorrect. Using a physics loss when training these networks ($\lambda_p > 0$) does

reduce the noise in the velocity predictions, but not the bias, and therefore did not improve the accuracy of the PDR algorithm. Thus, the second hypothesis is correct in the sense that the physical information is now used by the network, but this doesn't improve the algorithm. Therefore, as the training time of the PINNs is significantly longer than a data-driven neural network and it doesn't show significant improvements in accuracy, it is not recommended to use a PINN in the PDR application for surveying.

Another interesting observation was that the velocity prediction was seen to be more accurate in the center of the prediction window compared to the edges, as seen in Figure 4-10. Currently, the EKF in the PDR algorithm uses the velocity prediction at the end of the window, which is seen to have the highest error distribution. The decreased error distribution at the center of the window motivates an adaptation of the EKF, where it could use this velocity prediction in a 'delayed' measurement update. Further research on this topic could potentially lead to an improved PDR algorithm.

Chapter 5

Conclusion

The research in this thesis was done with the aim to explore if a state-of-the-art Pedestrian Dead-Reckoning (PDR) algorithm from literature can be improved. This algorithm uses an Extended Kalman Filter (EKF) that integrates Inertial Measurement Unit (IMU) measurements and corrects for integration drift by using predicted velocity measurements in the measurement update. This prediction comes from a data-driven artificial neural network, which uses a window of one second of IMU data as input. This thesis narrowed its scope to research the bottleneck of the neural network and possible methods to improve it.

Chapter 3 was focused on answering the first sub-question

> *What influence do small adaptations of the neural network in an existing PDR algorithm from literature have on its accuracy, and is there a bottleneck to be found that limits the accuracy of the algorithm?*

Two methods of training neural networks from similar existing PDR algorithms (an adapted version of *Liu et al.* [1] and the method from *Bajwa et al.* [19]) were applied to the dataset of surveying data from Leica, and it was found that the algorithm by *Liu et al.* outperformed the algorithm by *Bajwa et al.* considerably: the median drift rates using the two networks were 8.1% and 15.1%, respectively. This difference is thought to arise from the fact that *Bajwa et al.* made their algorithm for dead-reckoning of a quadcopter drone, which has inherently different characteristics than the pedestrian motion researched by *Liu et al.* and which Leica's dataset consists of. Several experiments have been done using adaptations of the network from *Liu et al.*, which included adding the tilt angle from the device orientation to the input data, simplifying the network by replacing the uncertainty prediction with a linear model based on predicted velocity, changing the input window lengths, changing the IMU sampling rate, and using a different method of pre-processing the input data. While most of these adaptations were able to produce a network that achieves a lower prediction error, the drift rate of the PDR algorithm often increased for these networks. The reason for this was found to be a bias in the prediction error of the neural network. This bias in prediction error violates the assumption of the EKF, which assumes a zero-mean Gaussian error on the measurements. The magnitude of this bias was measured by the Average Absolute Moving

Average (AAMA) (3-8), and it was concluded that this bias was the bottleneck that creates the largest errors in algorithm performance. The bias could be reduced by using another method of pre-processing of the input IMU data, where the data is rotated from the body frame b to the local frame l. Using the history of orientations of the EKF for this was not only faster, but also reduced prediction bias compared to using only the latest prediction and computing the history by backwards integrating the gyroscope data. This did, however, increase the prediction error during inference, resulting in similar values for the drift rate than the method using a backwards integration method to compute the history of orientations.

The network that performed best in the PDR algorithm was found to be the adapted network from *Liu et al.* [1] (returning a velocity prediction instead of displacement) using 1 second of accelerometer, gyroscope, and device tilt angle data sampled at 50 Hz. This network achieved a median drift rate of 7.9% in the full PDR algorithm.

Chapter 4 tested if the drift rate of the PDR algorithm could be improved using a Physics-Informed Neural Network (PINN) instead of a data-driven network to predict the velocity measurements for the EKF. This chapter aimed to answer the research question

> *Can the data-driven neural network in an existing PDR algorithm be adapted to accommodate for a physics-informed loss term, and does this adaptation lead to increased accuracy of the PDR algorithm?*

Two distinct methods have been tested to accommodate the network for a physics loss: the original ResNet from *Liu et al.* [1] with an additional query time as input, and three architectures that return an entire grid of predicted velocities in a time window. The first method used backpropagation of the network to compute the continuous derivative for the physics loss, and the second method used a numerical derivative based on polynomial fits of neighboring predictions. The four tested network architectures trained as data-driven networks achieved similar performance in the drift rate of the PDR algorithm as the best model from Chapter 3, and in most cases, using a physics loss showed an increase in drift rate because the bias in velocity prediction increased with larger physics loss weights. Two networks achieved increased performance over the baseline from Chapter 3, when trained using 4 seconds of input data: the Extended ResNet trained with $\lambda_p = 0.1$, and the Queried ResNet trained as data-driven network. However, training these networks on 1 second of input data created higher drift rates than the corresponding baseline using 1 second as well.

Answering the two research questions has led to a main conclusion from this thesis. The main question that this work was said to answer is

> *Can the accuracy of an existing state of the art PDR algorithm, using a combination of an EKF and neural network, be improved by adapting the neural network?*

This thesis has revealed that the main bottleneck of the neural network that limits the performance of the full PDR algorithm is the non-zero mean in the error of the velocity predictions from this network, which violates the assumption of the EKF that the measurement error distribution has a mean of zero. If this source of error were to be diminished, the PDR algorithm could potentially be more accurate than it is now. The thesis also revealed that using a PINN as neural network in the PDR algorithm does not reduce this error bias, but

rather increases it. If a data-driven network returns a noisy velocity prediction, it may benefit from a small physics loss as this reduces the noise quickly. Nevertheless, the best network found in this thesis was a data-driven network: a slight improvement in median drift rate from 8.1% to 7.9% was found when the tilt angle of the device with respect to the gravity direction is used as additional input along with the accelerometer and gyroscope data from the IMU. The one-dimensional ResNet from *Liu et al.* [1] using 1 second of this input data sampled at 50 Hz is able to achieve the best drift rate in this thesis. As the PINN as used in this thesis shows no significant improvement over using this data-driven network in the PDR algorithm from *Liu et al.* [1], but training it costs much more computational resources, it is not recommended to use a PINN in this algorithm.

It is recommended that further research on reducing the drift rate of this PDR algorithm focuses on reducing the prediction bias of the neural network. Although the work in this thesis revealed this bias as the bottleneck of the neural networks, it is yet unknown where exactly this bias originates from. An initial step into reducing this bias is to find its origin, which could be done by for example studying the correlation of the network prediction bias with potential sources, like the IMU bias.

Another potential idea for further research is to use a delay in the EKF updates. Chapter 4 revealed that all four network architectures saw a decrease in error distribution of the standalone network predictions when the velocity halfway during the time window is predicted. Using the predictions halfway during the time window, instead of at the end of this window, could potentially improve the accuracy of the position prediction of the PDR algorithm. It has to be taken into account that the longer this delay becomes, the longer the EKF has to rely on dead reckoning to return the location at the current time, which may lead to larger errors due to integration drift. It is suspected that there is some optimum to be found in the duration of this delay.

Moreover, this thesis has narrowed its scope to investigating if changes on the neural network in the PDR algorithm from *Liu et al.* [1] could improve its performance. As the global structure of the algorithm wasn't changed, it was implicitly assumed that there was more information in the input data of the neural network to be extracted. This assumption could be incorrect, meaning that there was no more relevant information to be found in the IMU data. A broader research could be done by inspecting if changes to the network's input could lead to better predictions, for example by including the past EKF states (velocity, orientation) as additional input to the network.

Appendix A

Appendix

A-1 Detailed description of the TLIO network

The block schemes in Figures A-1 and A-2 show the detailed structure of the one-dimensional ResNet structure as used by *Liu et al.* [1] and *Bajwa et al.* [19], as well as this thesis. Figure A-1 shows the details of the initial, residual, and fully connected blocks. The Initial block contains two downsampling layers: a convolution with stride 2 and a MaxPool with kernel size 2, creating a 4× downsampling. The residual blocks use a separate residual downsampling component, creating 2× downsampling, followed by a residual convolutional component that don't change the shapes of the input tensor. The fully connected block which is found after all residual blocks (see Figure 2-3) consists of one convolution followed by two fully connected layers to return a 3-vector as output.

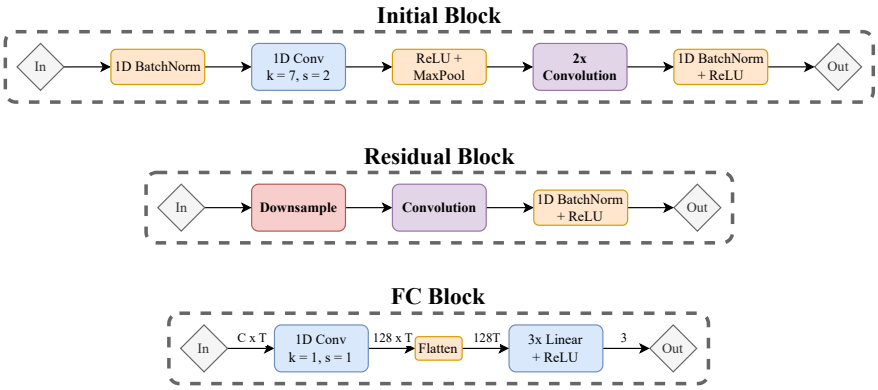

Figure A-1: Architecture of the initial, residual and FC blocks as used in the 1D ResNet model in 2-3. Details of the Downsample and Convolution blocks are shown in figure A-2.

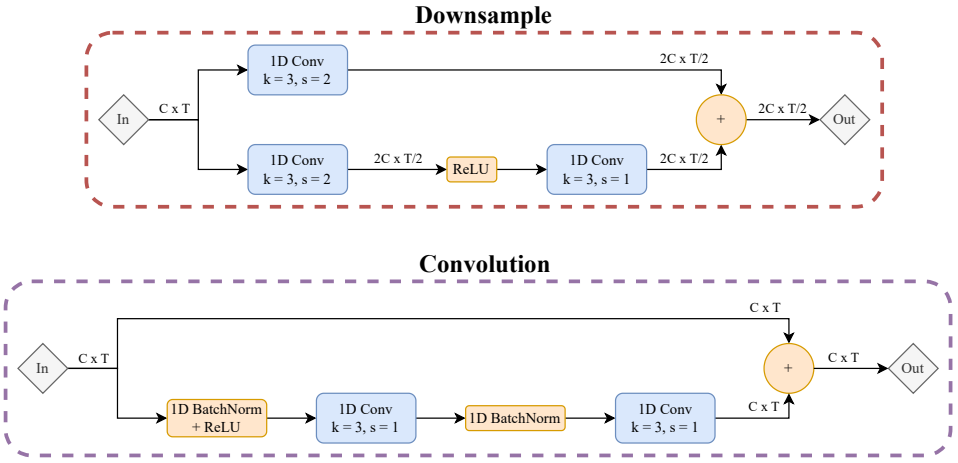

Figure A-2: Detailed structure of the Downsample and Convolution residual blocks as used in the 1D ResNet architecture shown in Figures 2-3 and A-1.

A-2 Error distributions of the remaining PINNs over the entire output window

This section shows the velocity and acceleration error distributions over the entire output window of the network architectures that have not been shown in Section 4-2 as they showed similar behavior to those that have been shown.

A-2-1 Velocity error distribution

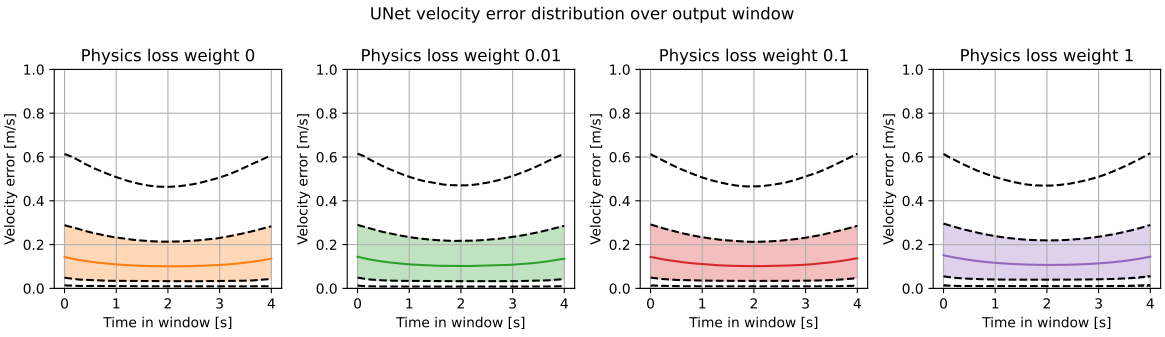

Figure A-3: Velocity prediction error distribution over the entire time window as predicted by the UNet.

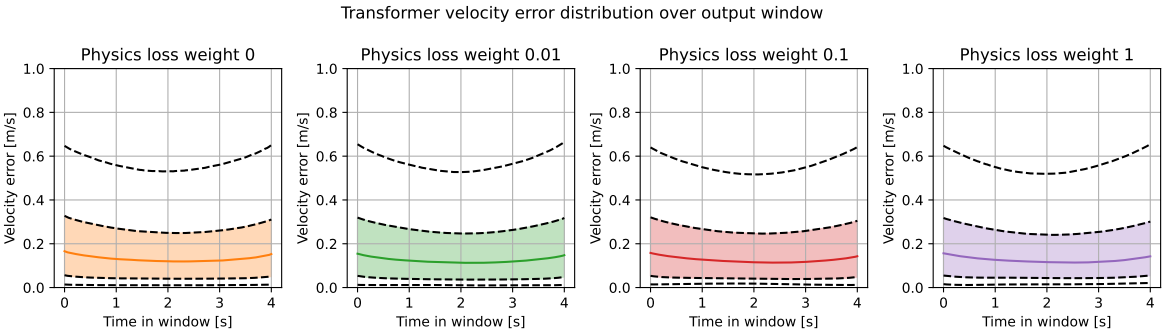

Figure A-4: Velocity prediction error distribution over the entire time window as predicted by the Transformer.

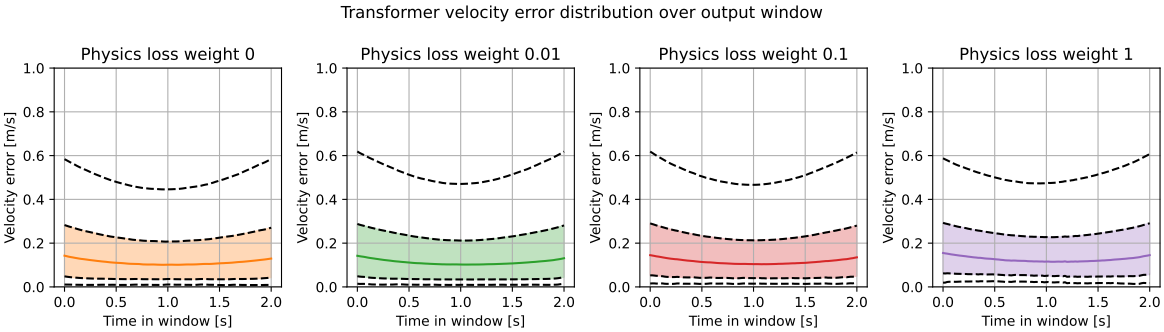

Figure A-5: Velocity prediction error distribution over the entire time window as predicted by the Queried ResNet.

A-2-2 Acceleration error distribution

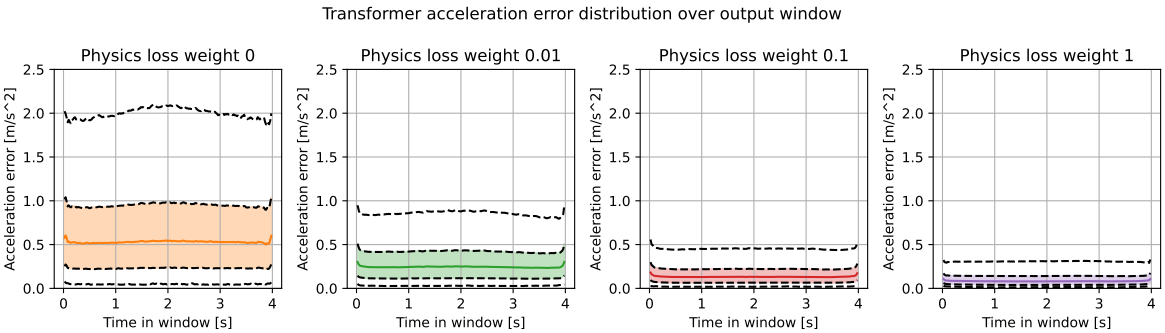

Figure A-6: Acceleration prediction error distribution over the entire time window as predicted by the Transformer.

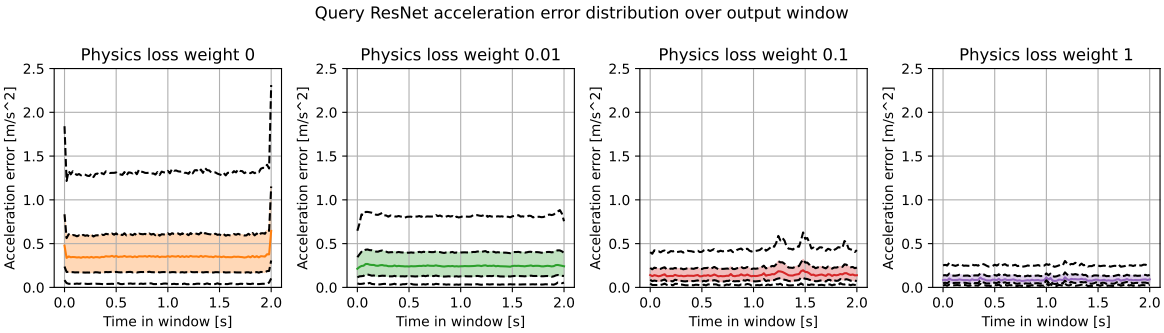

Figure A-7: Acceleration prediction error distribution over the entire time window as predicted by the Queried ResNet.

A-3 Velocity prediction of the remaining PINNs for one window

This section shows a velocity prediction of one output window from the test set of the network architectures that have not been shown in Section 4-2 as they showed similar behavior to those that have been shown.

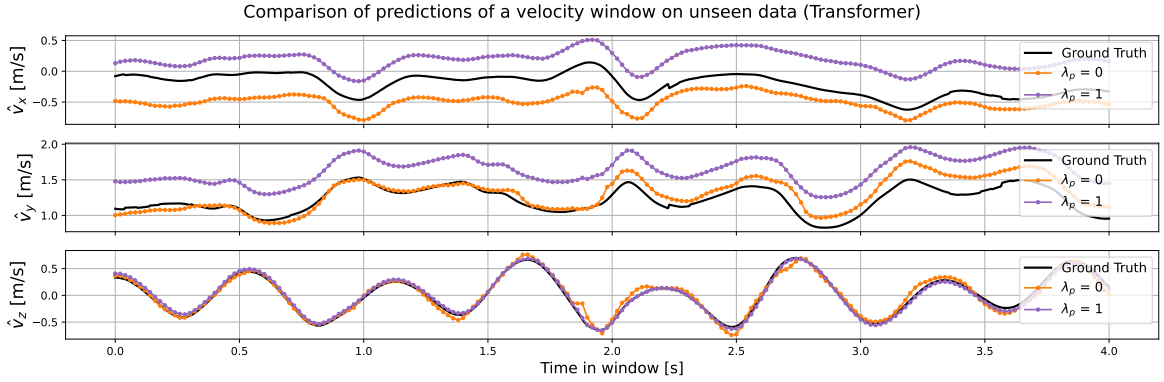

Figure A-8: Ground truth and predicted velocity by the Transformer of one data sample of a four seconds window in the unseen test set.

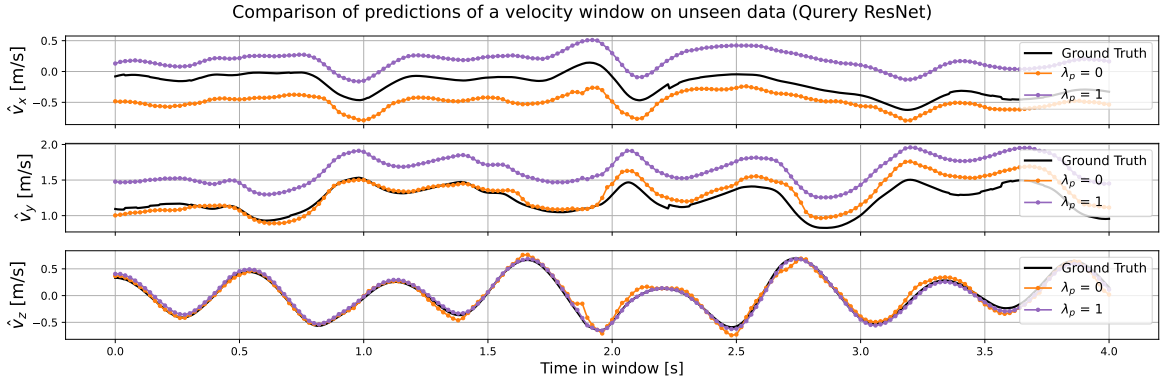

Figure A-9: Ground truth and predicted velocity by the Queried ResNet of one data sample of a four seconds window in the unseen test set.

Bibliography

[1] W. Liu, D. Caruso, E. Ilg, J. Dong, A. I. Mourikis, K. Daniilidis, V. Kumar, and J. Engel, "TLIO: Tight Learned Inertial Odometry," *IEEE Robotics and Automation Letters*, vol. 5, pp. 5653–5660, Oct. 2020.

[2] R. Harle, "A Survey of Indoor Inertial Positioning Systems for Pedestrians," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 3, pp. 1281–1293, 2013.

[3] X. Hou and J. Bergmann, "Pedestrian Dead Reckoning With Wearable Sensors: A Systematic Review," *IEEE Sensors Journal*, vol. 21, pp. 143–152, Jan. 2021.

[4] J. Hol, *Sensor fusion and calibration of inertial sensors, vision, ultra-wideband and GPS*. Linköping: Department of Electrical Engineering, Linköping University, 2011.

[5] P. Puricer and P. Kovar, "Technical Limitations of GNSS Receivers in Indoor Positioning," in *2007 17th International Conference Radioelektronika*, (Brno, Czech Republic), IEEE, Apr. 2007.

[6] A. G. Ferreira, D. Fernandes, A. P. Catarino, A. M. Rocha, and J. L. Monteiro, "A Loose-Coupled Fusion of Inertial and UWB Assisted by a Decision-Making Algorithm for Localization of Emergency Responders," *Electronics*, vol. 8, p. 1463, Dec. 2019.

[7] 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, pp. 1–153, Nov. 2017.

[8] L. Nadolinets, E. Levin, and D. Akhmedov, *Surveying instruments and technology*. Boca Raton: CRC Press, Taylor & Francis Group, 2017.

[9] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," *IEEE Robotics & Automation Magazine*, vol. 13, pp. 99–110, June 2006.

[10] J. Wahlstrom and M. Kok, "Three Symmetries for Data-Driven Pedestrian Inertial Navigation," *IEEE Sensors Journal*, vol. 22, pp. 5797–5805, Mar. 2022.

[11] J. Wahlström and I. Skog, "Fifteen Years of Progress at Zero Velocity: A Review," *IEEE Sensors Journal*, vol. 21, pp. 1139–1151, Jan. 2021.

[12] A. Jimenez, F. Seco, C. Prieto, and J. Guevara, "A comparison of Pedestrian Dead-Reckoning algorithms using a low-cost MEMS IMU," in *2009 IEEE International Symposium on Intelligent Signal Processing*, (Budapest, Hungary), pp. 37–42, IEEE, Aug. 2009.

[13] J. W. Kim, H. J. Jang, D.-H. Hwang, and C. Park, "A Step, Stride and Heading Determination for the Pedestrian Navigation System," *Journal of Global Positioning Systems*, vol. 3, pp. 273–279, Dec. 2004. Publisher: Springer Science and Business Media LLC.

[14] R. Stirling, J. Collin, K. Fyfe, and G. Lachapelle, "An innovative shoe-mounted pedestrian navigation system," in *Proceedings of European Navigation Conference GNSS*, pp. 1–15, 2003.

[15] E. Foxlin, "Pedestrian tracking with shoe-mounted inertial sensors," *IEEE Computer Graphics and Applications*, vol. 25, pp. 38–46, Nov. 2005.

[16] C. Chen and X. Pan, "Deep Learning for Inertial Positioning: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 25, pp. 10506–10523, Sept. 2024.

[17] C. Chen, X. Lu, A. Markham, and N. Trigoni, "IONet: Learning to Cure the Curse of Drift in Inertial Odometry," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, Apr. 2018.

[18] C. Chen, C. X. Lu, J. Wahlström, A. Markham, and N. Trigoni, "Deep Neural Network Based Inertial Odometry Using Low-Cost Inertial Measurement Units," *IEEE Transactions on Mobile Computing*, vol. 20, pp. 1351–1364, Apr. 2021.

[19] A. Bajwa, C. C. Cossette, M. A. Shalaby, and J. R. Forbes, "DIVE: Deep Inertial-Only Velocity Aided Estimation for Quadrotors," *IEEE Robotics and Automation Letters*, vol. 9, pp. 3728–3734, Apr. 2024.

[20] G. Welch, "An Introduction to the Kalman Filter," *University of North Carolina at Chapel Hill*, 1997.

[21] S. J. D. Prince, *Understanding Deep Learning*. The MIT Press, 1 ed., Oct. 2024.

[22] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, "Physics-informed machine learning," *Nature Reviews Physics*, vol. 3, pp. 422–440, May 2021.

[23] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, Feb. 2019.

[24] Q. Yang and X. Zhang, "Combined Location Method Based on Error State Kalman Filter," in *2023 4th International Conference on Computer Engineering and Application (ICCEA)*, pp. 481–485, Apr. 2023. ISSN: 2159-1288.

[25] R. S. Lab, "ETH Robot Dynamics Lecture Notes," 2017.

[26] S. Paul and T. K. Maiti, "Accurate Kinematic-Parameters Estimation Using IMU and GPS Sensors Fusion," *IEEE Sensors Journal*, vol. 24, pp. 35547–35554, Nov. 2024.

[27] M. B. Alatise and G. P. Hancke, "Pose Estimation of a Mobile Robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter," *Sensors*, vol. 17, p. 2164, Oct. 2017. Number: 10 Publisher: Multidisciplinary Digital Publishing Institute.

[28] Y. Liu, Y. Bao, P. Cheng, D. Shen, G. Chen, and H. Xu, "Enhanced robot state estimation using physics-informed neural networks and multimodal proprioceptive data," in *Sensors and Systems for Space Applications XVII*, vol. 13062, pp. 144–160, June 2024.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, (Las Vegas, NV, USA), pp. 770–778, June 2016.

[30] B. Moseley and S. Mishra, "AI in the Sciences and Engineering," 2024. ETH Zürich [Course Webpage]. Available: https://camlab.ethz.ch/teaching/ai-in-the-sciences-and-engineering-2024.html (accessed: Dec. 23, 2024).

[31] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[32] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Nov. 2015.

[33] S. Theodoridis, *Machine learning: a Bayesian and optimization perspective*. London San Diego: Academic press, 2nd edition ed., 2020.

[34] Z. Fang, "A High-Efficient Hybrid Physics-Informed Neural Networks Based on Convolutional Neural Network," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, pp. 5514–5526, Oct. 2022.

[35] K. Siegrist, *Probability, Mathematical Statistics, and Stochastic Processes*. 2021.

[36] R. Riddick, E. Smits, G. Faber, C. Shearwin, P. Hodges, and W. Van Den Hoorn, "Estimation of human spine orientation with inertial measurement units (IMU) at low sampling rate: How low can we go?," *Journal of Biomechanics*, vol. 157, p. 111726, Aug. 2023. Publisher: Elsevier BV.

[37] A. V. Oppenheim, A. S. Willsky, and S. H. Nawab, *Signals and systems*. Prentice-Hall signal processing series, Upper Saddle River, NJ: Prentice Hall, 2. ed ed., 1997.

[38] N. Storey, *Electronics: A systems approach*. Harlow: Pearson, 6th edition ed., 2017.

[39] T.-K. Chang and A. Mehta, "Optimal Scheduling for Resource-Constrained Multirobot Cooperative Localization," *IEEE Robotics and Automation Letters*, vol. 3, pp. 1552–1559, July 2018.

[40] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," 2015.

[41] S. Liang, Z. Huang, M. Liang, and H. Yang, "Instance Enhancement Batch Normalization: An Adaptive Regulator of Batch Noise," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 4819–4827, Apr. 2020. Number: 04.

[42] "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Lecture Notes in Computer Science*, pp. 234–241, Cham: Springer International Publishing, 2015. ISSN: 0302-9743, 1611-3349.

[43] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), pp. 6000–6010, Dec. 2017.

[44] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang, "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 11106–11115, May 2021.

[45] Z. Zhao, X. Ding, and B. A. Prakash, "PINNsFormer: A Transformer-Based Framework For Physics-Informed Neural Networks," *ArXiv e-prints*, May 2024. arXiv:2307.11833.

[46] A. Krishnapriyan, A. Gholami, S. Zhe, R. Kirby, and M. W. Mahoney, "Characterizing possible failure modes in physics-informed neural networks," in *Advances in Neural Information Processing Systems*, vol. 34, pp. 26548–26560, 2021.

[47] Y. Kondo, M. Numada, H. Koshimizu, K. Kamiya, and I. Yoshida, "Low-pass filter without the end effect for estimating transmission characteristics—Simultaneous attaining of the end effect problem and guarantee of the transmission characteristics," *Precision Engineering*, vol. 48, pp. 243–253, Apr. 2017. Publisher: Elsevier BV.

[48] M. Fisher, M. Leutbecher, and G. A. Kelly, "On the equivalence between Kalman smoothing and weak-constraint four-dimensional variational data assimilation," *Quarterly Journal of the Royal Meteorological Society*, vol. 131, pp. 3235–3246, Oct. 2005. Publisher: Wiley.

Glossary

List of Acronyms

AAMA	Average Absolute Moving Average
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite System
IMU	Inertial Measurement Unit
MSE	Mean Squared Error
PDE	Partial Differential Equation
PDR	Pedestrian Dead-Reckoning
PINN	Physics-Informed Neural Network
SLAM	Simultaneous Localization And Mapping
UWB	Ultra-Wideband