Static attitude determination using convolutional neural networks

sensors

MDPI

Article

Static Attitude Determination Using Convolutional Neural Networks

Guilherme Henrique dos Santos [1], Laio Oriel Seman [2,*], Eduardo Augusto Bezerra [1], Valderi Reis Quietinho Leithardt [3], André Sales Mendes [4] and Stéfano Frizzo Stefenon [5]

[1] Department of Electrical Engineering, Federal University of Santa Catarina, Florianópolis 88040-900, Brazil; guilherme.dos.santos@spacelab.ufsc.br (G.H.d.S.); eduardo.bezerra@ufsc.br (E.A.B.)
[2] Graduate Program in Applied Computer Science, University of Vale do Itajaí, Itajaí 88302-901 Brazil
[3] VALORIZA, Research Center for Endogenous Resources Valorization, Instituto Politécnico de Portalegre, 7300-555 Portalegre, Portugal; valderi@ipportalegre.pt
[4] Expert Systems and Applications Lab, Faculty of Science, University of Salamanca, Plaza de los Caídos s/n, 37008 Salamanca, Spain; andremendes@usal.es
[5] Faculty of Engineering and Applied Science, University of Regina, Regina, SK 3737, Canada; sfc079@uregina.ca
* Correspondence: laio@univali.br

Abstract: The need to estimate the orientation between frames of reference is crucial in spacecraft navigation. Robust algorithms for this type of problem have been built by following algebraic approaches, but data-driven solutions are becoming more appealing due to their stochastic nature. Hence, an approach based on convolutional neural networks in order to deal with measurement uncertainty in static attitude determination problems is proposed in this paper. PointNet models were trained with different datasets containing different numbers of observation vectors that were used to build attitude profile matrices, which were the inputs of the system. The uncertainty of measurements in the test scenarios was taken into consideration when choosing the best model. The proposed model, which used convolutional neural networks, proved to be less sensitive to higher noise than traditional algorithms, such as singular value decomposition (SVD), the q-method, the quaternion estimator (QUEST), and the second estimator of the optimal quaternion (ESOQ2).

Keywords: attitude determination; machine learning; neural network; measurement uncertainty

Citation: dos Santos, G.H.; Seman, L.O.; Bezerra, E.A.; Leithardt, V.R.Q.; Mendes, A.S.; Stefenon, S.F. Static Attitude Determination Using Convolutional Neural Networks. *Sensors* **2021**, *21*, 6419. https://doi.org/10.3390/s21196419

Academic Editors: Valerie Renaudin and Pasquale Daponte

Received: 7 August 2021
Accepted: 23 September 2021
Published: 26 September 2021

1. Introduction

The process of finding rotations between frames—although classic—is recurrent. It consists of providing information about the difference between some objects of interest that are seen from the perspective of the body frame and in the desired frame of reference. This difference is represented as a rotation operation; thus, the goal is to find the rotational axis and rotational angle that best rotate a frame given by ($\vec{\mathbf{a}}$, $\vec{\mathbf{b}}$, and $\vec{\mathbf{c}}$) to another frame given by ($\vec{\mathbf{x}}$, $\vec{\mathbf{y}}$, and $\vec{\mathbf{z}}$). Finding the rotation between frames is applied in many fields, such as robotics [1,2], navigation and control [3], and computer graphics [4,5], among others [6].

In spacecraft navigation, this process is called attitude determination, and typically, various sensors are used to collect information about the objects of interest. These data are returned as a vector, which is known as a measurement or observation vector. It is possible to obtain two orientation parameters from a given measurement vector. However, three orientation parameters are necessary in order to determine an attitude. This leads to an underdetermined problem if only one vector ($\mathbf{N} = 1$) is used, since the number of parameters is less than the minimum number that is required. On the other hand, the problem will be overdetermined if two measurement vectors or more ($\mathbf{N} \geq 2$) are used, since four orientation parameters will be used, i.e., there are too many variables.

Following this reasoning, some measurements are needed to determine a proper attitude such that $1 < \mathbf{N} < 2$. However, this is a problem because it is not possible to obtain

a non-natural number of measurements; thus, a set of measurement vectors are needed in order to estimate the proper orientation. The first proposed solution was called Wahba's problem [7], a least-square approximation given by:

$$L(\mathbf{A}) = \frac{1}{2}\sum_{\mathbf{i}=1}^{\mathrm{n}} \mathbf{a_i}\|\vec{\mathbf{b_i}} - \mathbf{A}\vec{\mathbf{r_i}}\|^2, \text{ such that } \mathrm{n} \geq 2 \quad (1)$$

where $\vec{\mathbf{b_i}}$ is an observation vector in the body frame, $\vec{\mathbf{r_i}}$ is a vector in the reference frame, $\mathbf{a_i}$ is a weight value for each observation vector, and $\mathbf{A}$ is the optimal attitude matrix. This first approach to attitude determination led to the development of several new algorithms derived directly from Equation (1), resulting in a trade-off, as more robust models are slower to converge, requiring more computational effort, and very fast algorithms may result in low accuracy [8], thus requiring careful fine-tuning.

To solve the problem, it is possible to use a combination of artificial neural networks (ANNs) with Kalman filters (KFs) [9] or even recurrent neural networks (RNNs) in order to estimate the new attitude [10]. These approaches require some previous attitude information in order to adjust the predictions, that is, they depend on time information. Therefore, these types of algorithms are known as attitude estimation or dynamic attitude determination algorithms. This paper focuses on another type of algorithm that does not need previous attitude information for the prediction, resulting in a static attitude determination.

The quaternion estimator (QUEST) method [11,12], singular value decomposition (SVD)-based method [13], second estimator of the optimal quaternion (ESOQ2) method [14], and q-method [15,16] belong to the static attitude determination class of algorithms, since their inputs are observation vectors that are taken simultaneously or close enough in time that they can ignore spacecraft motion between the measurements. Therefore, they can determine the attitude at that given instant. Conversely, the elimination of the time variable makes this class of algorithms susceptible to the quality of the input data, i.e., the precision of the sensors drastically affects the caliber of the results. Due to their nature, they have a crucial role in the "lost-in-space" scenario [17], where the spacecraft has no previous information about its attitude. Therefore, the navigation system must instantaneously infer an orientation by relying on the quality of measurements taken by its sensors, since this directly affects the performance of the algorithms [18].

In this context, this paper focuses on applying machine learning to the mitigation of the impact of the measurements' perturbations on the predictions. Specifically, this paper proposes a neural-network-based approach to determining the rotation difference between two coordinate frames based on a convolutional neural network (CNN) architecture. The contribution of this paper is based on the strategy of choosing the model by considering its uncertainty. Furthermore, this paper will show how the model's performance compares with that of traditional algorithms and if this data-driven approach to static attitude determination can mitigate the impact of input distortion.

2. Background

2.1. Attitude Representations

Rotation matrices can be parametrized in many different ways, and the most commonly used tools are the Euler axis/angle, Euler angles, and quaternions. The Euler axis/angle describes a rotation as a unit vector $\vec{\mathbf{e}}$ that lies along the rotation axis and a rotation angle ϕ about this axis. Although it has a clear physical interpretation, its axis is undefined for $\phi = \{0, 2\pi\}$, that is, it is not a continuous representation.

Euler angles are defined as a set of three parameters: ϕ, θ, and ψ. These angles break a single rotation into a sequence of three rotations using two intermediate frames throughout the process. Thus, it is relatively easy to interpret the problem. However, for $\theta = \pm 90°$, the solution degenerates, leading to a problem called "gimbal lock", which causes ϕ and θ to move in the same fashion. That is why one should be careful when using Euler angle representations.

Quaternions are four-dimensional vectors that embed the rotation axis and the rotation angle in a complex domain. Despite not having an apparent physical interpretation, they are the preferable form of parametrization for specialists, since they do not require trigonometric functions and avoid the "gimbal lock" issue. However, this type of representation has an interesting property where $\vec{\mathbf{q}} = -\vec{\mathbf{q}}$, that is, the antipodes represent the same rotation. Therefore, they are not a continuous representation.

In trying to find a valid continuous representation for $SO(3)$, a new method was proposed by [19]. They assumed that a representation space $\mathcal{R}$ produced by a neural network could be mapped into the original space $\mathcal{X} \in SO(3)$ through a function $\mathbf{f} : \mathcal{R} \to \mathcal{X}$ and mapped back through a function $\mathbf{g} : \mathcal{X} \to \mathcal{R}$.

The function $\mathbf{f}$ is a mathematical function that is used as part of the forward pass of the network at both the training and inference time. For 6D representations, this function is guaranteed to return an orthogonal 3×3 matrix, and the loss function can then be applied to it. This mapping function is defined as follows for a 6D representation:

$$\mathbf{f}\left(\begin{bmatrix} | & | \\ \vec{\mathbf{a}}_1 & \vec{\mathbf{a}}_2 \\ | & | \end{bmatrix}\right) = \begin{bmatrix} | & | & | \\ \vec{\mathbf{b}}_1 & \vec{\mathbf{b}}_2 & \vec{\mathbf{b}}_3 \\ | & | & | \end{bmatrix} \tag{2a}$$

$$\vec{\mathbf{b}}_\mathbf{i} = \begin{bmatrix} \begin{cases} N(\vec{\mathbf{a}}_1) & \text{if } \mathbf{i} = 1 \\ N(\vec{\mathbf{a}}_2 - (\vec{\mathbf{b}}_1 \cdot \vec{\mathbf{a}}_2)\vec{\mathbf{b}}_1) & \text{if } \mathbf{i} = 2 \\ \vec{\mathbf{b}}_1 \times \vec{\mathbf{b}}_2 & \text{if } \mathbf{i} = 3 \end{cases} \end{bmatrix}^\mathbf{T} \tag{2b}$$

where $N(\cdot)$ stands for the normalization function, $\times$ represents the vector cross product, $\cdot$ represents the vector dot product, and the column vectors $\vec{\mathbf{a}}_1$ and $\vec{\mathbf{a}}_2$ are vectors from $\mathcal{R}$ that are estimated by the network.

2.2. Related Work

As mentioned before, if a spacecraft needs to know its orientation, some prior information must be gathered by sensors. They return the data as three-dimensional unit vectors representing the pointing direction of an object of interest with respect to the body frame. From these data, it is possible to find a proper rotation that will project the observations onto the target frame. There is an extensive field of study covering regression in rotation representations by using ANNs. In the computer vision field, there is a problem that is similar to the one described before, whose premise is to find the orientation of objects with respect to the camera. This task also shares the same goal as attitude determination, that is, it determines the orientations or poses of objects of interest from one coordinate frame to another. In the context of computer vision, this task is referred to as pose estimation.

Xiang et al. [20] tried to estimate the rigid transformation from the object coordinate frame $\mathcal{O}$ to the camera coordinate frame $\mathcal{C}$ given an input image. Each image contains a collection of objects, and the main task is to find their poses (location and orientation), which are represented by a 3D rotation $\mathbf{R}$ and a 3D translation $\mathbf{T}$ parametrized by quaternions. Thus, they can be written as $\mathbf{R}(\vec{\mathbf{q}})$. They used the mean squared error loss function to minimize the error, where they measured the distance between a set of 3D points $\vec{\mathbf{x}} \in \mathcal{O}$ rotated by both the ground-truth rotation $\mathbf{R}(\vec{\mathbf{q}})$ and the predicted rotation $\mathbf{R}(\hat{\vec{\mathbf{q}}})$. However, using this approach, the network performed poorly when the rotation angles were close to 0° and 180°.

Do et al. [21] used an architecture called mask R-CNN, which took an RGB image containing several objects as input and returned their classes, segmentation masks, and bounding boxes. They added a fourth output on top of this architecture, which tried to estimate the orientation and localization of each object. This output layer was a four-dimensional vector, where the first three scalar values represented the rotation and the last one represented the translation. However, instead of using representations, such as Euler angles and quaternions, they made use of Lie algebra $\mathfrak{so}(3)$ to represent the rotation, since a skew-symmetric matrix of an arbitrary element from $\mathfrak{so}(3)$ could be mapped to the

$SO(3)$ through exponential mapping. Hence, they only needed to regress a vector $\vec{\mathbf{x}} \in \mathbb{R}^3$ to define a proper rotation.

Point clouds can be interpreted as 3D vectors that describe a shape or object in some arbitrary coordinate frame; hence, this type of data can provide a lot of useful information, such as depth, volume, orientation, and location. In trying to explore this type of information, a novel approach [22] was proposed to address the 6D pose estimation problem: Instead of using RGB images, they used point cloud segments as the input to the system. The proposed neural network was an adaptation of the PointNet architecture [23], which takes a set of point clouds as input and returns a rotation matrix that best describes the orientation of the input data.

Those works addressed the 6D pose estimation task by considering different types of input data, such as RGB images and point clouds, as well as distinct attitude representations. Although all of them achieved good results for the task, they presented some limitations. First, their attitude representations were not adequate for the job because they showed discontinuities for certain angles. Second, the input data did not offer clear information about the orientation or relationship between the coordinate systems, thus demanding more effort from the network to detect those features.

In this context, this paper proposes a model that uses a different type of input data, which is denoted as an attitude profile matrix; this embeds the similarity between two coordinate frames and alleviates the model's effort. In addition, the proposed model uses another type of attitude representation that can be learned and does not suffer from discontinuities.

3. Methodology

PointNet is a neural network architecture that accepts raw point clouds as input and is robust with respect to input perturbation and corruption [24]. Taking into account the promising results of PointNet, several extensions have been developed [25–28]. Hence, PointNet was chosen as the base architecture for the experiments presented in this paper. Figure 1 shows the network structure.

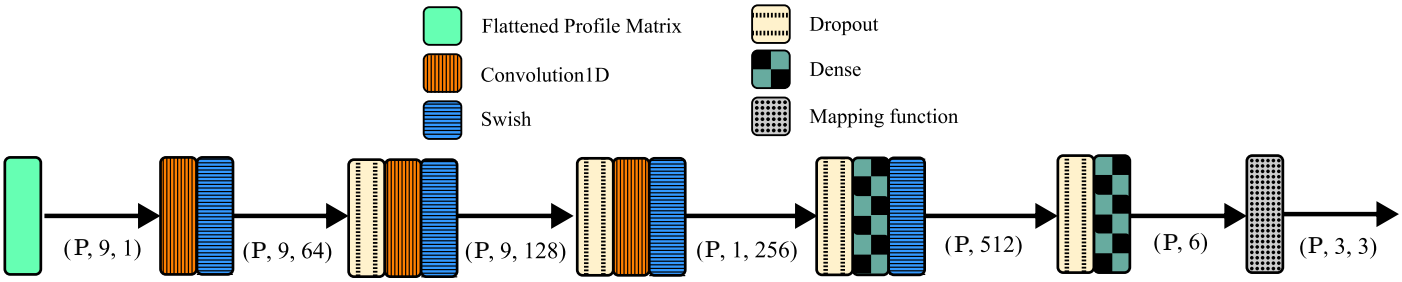

Figure 1. B-Swish (Modified PointNet).

Some modifications have been made to the traditional PointNet architecture in order to improve its performance. In the vanilla PointNet, a set of vectors in the original coordinate frame and another set in the target coordinate frame are used as input and fed into the model in pairs. However, in the proposed model, they were replaced by the attitude profile matrix. In fact, the same pairs of vectors were used to build the profile matrix according to Equation (3). By flattening this matrix, an array with the shape $(\mathbf{P}, 9, 1)$ was generated, where $\mathbf{P}$ represents the batch size. However, it was considered that all weights $\mathbf{a_i}$ were equal (star-tracking scenario) because it led to lower errors when compared to random weights, even in the evaluation phase, as can be seen in Section 4.1.

$$\mathbf{B} = \sum_{\mathbf{i}=1}^{\mathbf{n}} \mathbf{a_i} \vec{\mathbf{b}}_\mathbf{i} \vec{\mathbf{r}}_\mathbf{i}^\mathbf{T}. \tag{3}$$

All one-dimensional convolutions were built using a kernel of size of 9. However, in all but the last convolution, the shape of the feature maps was maintained and, as a

result, after applying the convolution operation, the second dimension of the tensor did not change. Next, all rectified linear unit (ReLU) [29] activation functions were replaced with Swish activation [29] because it turned out that ReLU led to more significant errors during training and evaluation. Finally, as the last operation, the mapping function $\mathbf{f}$, which maps a latent rotation representation $\mathcal{R}$ to $SO(3)$, was implemented, as performed in [19].

Dropout layers were added after each activation function [30]. The reasons were twofold: to act as a regularizer to prevent overfitting and to measure the model uncertainty. Since the dropout technique randomly drops some neurons with a probability $\mathbf{p}$ at each forward pass, each update to a layer during training is performed with a different view of the configured layer, forcing the knowledge to be spread out across all neurons equally, thus acting as a regularizer. This can be interpreted as if several neural networks are being trained simultaneously because at each forward pass, the model "changes" its architecture.

Notice that the regular dropout could be interpreted as a Bayesian approximation of a Gaussian process [31]. By applying the dropout during both training and inference, it is possible to analyze it as if predictions from many different networks have been made, that is, a Monte Carlo sample from the space of all possible networks. Hence, it is entitled Monte Carlo Dropout (MCD) [32]. Therefore, a distribution of predictions is gathered and a measure of uncertainty of the model can be evaluated.

In an actual application, the star-tracker algorithm returns two pairs of vectors from a star image: the body vectors, which represent each star in the body frame, and the reference vectors, which represent the same stars collected from the satellite's onboard database. The proposed model uses those two pairs of vectors to build the attitude profile matrix according to Equation (3). Afterward, the model reads a flattened version of this matrix and estimates the orientation matrix.

3.1. Implementation Details

3.1.1. Dataset

The traditional algorithms were designed to determine the best attitude representation given a set of unit observation vectors and their respective weights. In essence, those vectors are random, and the only requirement is that they must be unitary and have a boresight axis, which is the axis that expresses the direction of the observed object.

Following this reasoning, the dataset was built according to the number of samples and the number of observation vectors per sample. First, random unit vectors with arbitrary boresight axes were generated, forming an array with a shape $(\mathcal{S}, \mathcal{O}, 3)$, where $\mathcal{S}$ stands for the number of samples, $\mathcal{O}$ is the number of observation vectors, and the last dimension is the number of axes of each vector. The last dimension was set to 3, since the data had components in $\mathbb{R}^3$. The first collections of vectors were chosen to be the reference vectors $\vec{\mathbf{r}}$.

The random rotation matrices $\mathbf{A_{true}}$ were generated through Equation (4), where the angles were drawn from a uniform distribution ranging from $-\pi$ rad to π rad, and the axis vectors were drawn from a uniform distribution.

$$
\begin{aligned}
\mathbf{A} &= \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \\
a_{11} &= \cos\phi + \mathbf{e_1^2}(1-\cos\phi) \\
a_{12} &= \mathbf{e_1 e_2}(1-\cos\phi) + \mathbf{e_3}\sin\phi \\
a_{13} &= \mathbf{e_1 e_3}(1-\cos\phi) - \mathbf{e_2}\sin\phi \\
a_{21} &= \mathbf{e_1 e_2}(1-\cos\phi) - \mathbf{e_3}\sin\phi \\
a_{22} &= \cos\phi + \mathbf{e_2^2}(1-\cos\phi) \\
a_{23} &= \mathbf{e_2 e_3}(1-\cos\phi) + \mathbf{e_1}\sin\phi \\
a_{31} &= \mathbf{e_1 e_3}(1-\cos\phi) + \mathbf{e_2}\sin\phi \\
a_{32} &= \mathbf{e_2 e_3}(1-\cos\phi) - \mathbf{e_1}\sin\phi \\
a_{33} &= \cos\phi + \mathbf{e_3^2}(1-\cos\phi).
\end{aligned}
\tag{4}
$$

The random direction cosine matrices (DCM) with shape $(\mathcal{S}, 3, 3)$ served as the ground-truth attitude matrices [33]. The reference vectors were simply rotated by each attitude matrix to generate the body vectors $\vec{\mathbf{b}}$ with a shape $(\mathcal{S}, \mathcal{O}, 3)$ as follows:

$$\vec{\mathbf{b}}_\mathbf{i} = \mathbf{A}_{\mathbf{true,i}} \vec{\mathbf{r}}_\mathbf{i} + \vec{\mathbf{n}}_\mathbf{i} \tag{5}$$

where $\vec{\mathbf{n}}_\mathbf{i}$ is a vector of measurement errors drawn from a zero-mean Gaussian distribution $\mathcal{N}(0, \sigma_\mathbf{i})$ with standard deviations $\{\sigma_\mathbf{i} \mid \mathbf{10}^{-6} \leq \sigma_\mathbf{i} \leq 0.01\}$ serving as additive white Gaussian noise (AWGN) [34].

3.1.2. Training

The dataset was generated with the total number of samples $\mathcal{S} = 2^{13}$, where 30% was used as the testing dataset and 4% as the validation dataset. The optimizer used was the Adam algorithm [35] with a learning rate of $\mathbf{lr} = 10^{-4}$ and a weight decay of 10^{-4} being applied every 500 epochs. The $\mathbf{lr}$ was also reduced by a factor of ten every 500 epochs.

The work was developed using the Tensorflow 2 framework, and the models were trained on a Nvidia GeForce GTX 1060 6 GB GPU (graphic processing unit). In order to train the network, the geodesic loss function defined by Equation (6) was used to minimize the error. The matrix $\mathbf{A}$ is the rotation difference given by the multiplication $\mathbf{A}_\mathbf{true}\mathbf{A}_\mathbf{pred}^\mathbf{T}$. This equation was also used as a metric, along with Wahba's loss, which is defined by Equation (1), where all weights $\mathbf{a}_\mathbf{i}$ were considered equal during the evaluation. Equation (6) was clipped to lie in the range $[-1 + \text{eps}, 1 - \text{eps}]$ in order to avoid numerical issues, where $\text{eps} = 10^{-7}$ is enough.

$$\phi(\mathbf{A}) = \cos^{-1}\left(\frac{\text{tr}(\mathbf{A}) - 1}{2}\right). \tag{6}$$

To validate the model's performance, several training routines were scheduled. For each routine, different values for $\mathcal{O}$ were used to check if this could bring a meaningful impact on the model's robustness. Different models were trained, considering a value for $\mathcal{O}$ that ranged from 3 to 7. Every training was performed by considering 2000 epochs with a batch size of 64. In addition, for each $\mathcal{O}$, three different dropout rates were considered: 10%, 15%, and 20%.

4. Results

4.1. Model Training

Initially, the model was trained with an earlier version of the architecture presented in Section 3, where the pairs of body and reference vectors, which were concatenated along the last dimension, were defined as the input to the network. The number of observation vectors was set as the kernel size parameter in the convolution layers. In all except the last convolution, the shape of the feature maps was maintained. In this assessment, the ReLU activation function was used instead of Swish.

Variations were made to improve the model, since this assessment did not achieve acceptable error values. The pairs of vectors were replaced by the attitude profile matrix, and the architecture presented in Figure 1 was formulated. However, ReLU was considered instead of Swish. Throughout this paper, this model will be called B-ReLU. This model had two versions: B-ReLU-A and B-ReLU-B. The former considered a star-tracking scenario, and the latter considered the actual weights for each pair of inputs when building the attitude profile matrices.

It was possible to notice that using the actual weights to build the $\mathbf{B}$ matrix led to poor results; thus, it was decided to use the star-tracking scenario. Since the Swish activation function showed more promising results than the ReLU activation function and its variations, this function was defined as the default in the following analyses. Then, the final model was built (see Section 3), which was called B-Swish.

After training the models, their performance was measured by using the validation set. The models whose parameters corresponded to the slightest error in Wahba's loss were tested. As shown in Figures 2 and 3, the hyperparameter $\mathcal{O}$ led to better results on the validation dataset when it was increased. This decrease in error indicated that the network was capturing more insights about the attitude profile matrix because it was using more observation vectors. Moreover, as the dropout rate increased, the network tended to have worse results, indicating that the parameters were over-penalized.

The network struggled to converge when using pairs of body and reference vectors as inputs. Moreover, when using the accurate weights to build the attitude profile, worse results were obtained. The Swish activation function improved the network performance; thus, it was chosen as the final architecture. It was still necessary to verify which of the B-Swish models was more reliable and if the results really reflected the system's expected behavior when subjected to unseen scenarios.

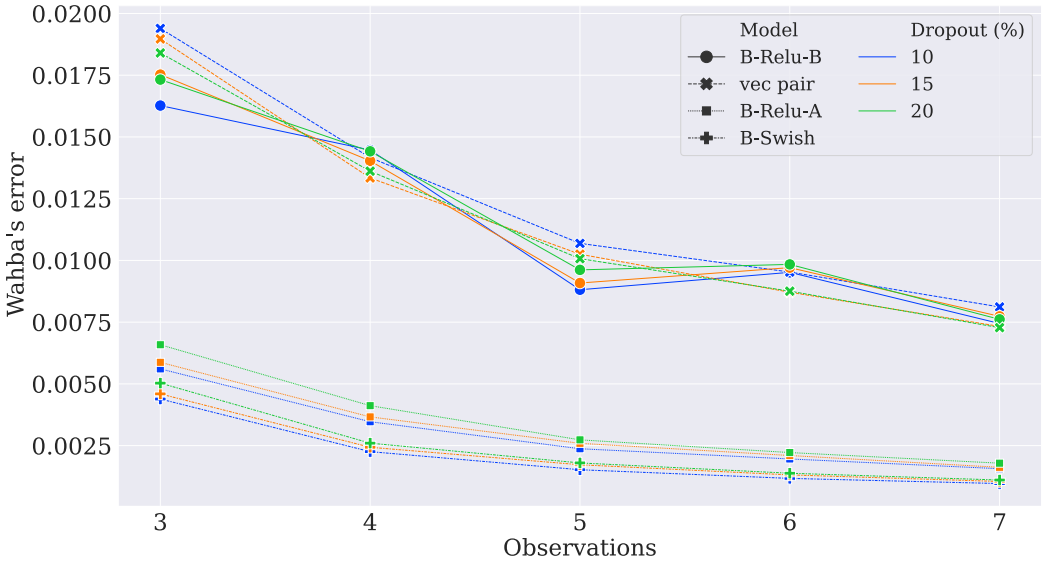

Figure 2. Wahba's loss.

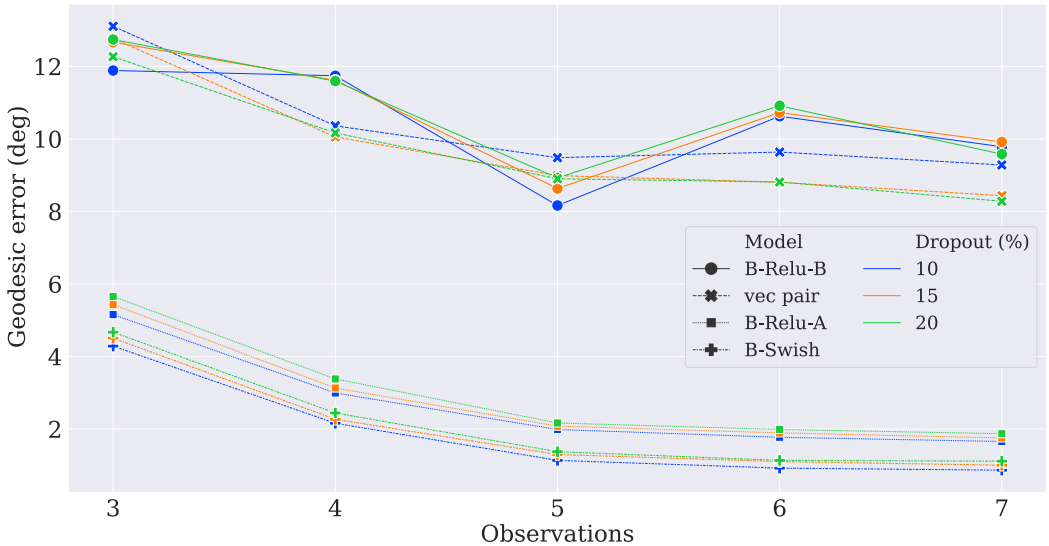

Figure 3. Geodesic error in degrees.

4.2. Model Choice

The models' performance was verified by using the twelve test cases specified by Markley [36]. Each test case contains a collection of vectors $\vec{\mathbf{r}}_\mathbf{i}$ and their corresponding

measurement errors $\sigma_{\mathbf{i}}$. These $\vec{\mathbf{r}}_{\mathbf{i}}$ vectors represent what, in practice, would be the output of the spacecraft's onboard sensors, such as star trackers, sun sensors, magnetometers, and others. Notice that these data would be returned as three-dimensional vectors. The $\sigma_{\mathbf{i}}$ errors represent how accurate the respective observation vectors are, that is, they hold the information about the precision of the sensors that generated these sets of vectors. The body vectors $\vec{\mathbf{b}}_{\mathbf{i}}$ were constructed by following Equation (5), where $\mathbf{A_{true}}$ is defined as

$$\mathbf{A_{true}} = \begin{bmatrix} 0.352 & 0.864 & 0.360 \\ -0.864 & 0.152 & 0.480 \\ 0.360 & -0.480 & 0.800 \end{bmatrix}. \tag{7}$$

The first four cases consider an ideal scenario, where all observation vectors lie along the boresight axes and are orthogonal to each other. The fifth one demonstrates a case where the vectors are orthogonal but do not lie along the boresight axes. From the sixth scenario onwards, they consider a narrow-field-of-view star tracker, where the rotation about the tracker's boresight is much less well determined than the pointing of the boresight. Despite the fact that they are handcrafted cases, the tests simulate some interesting scenarios that could happen in an actual application and give an idea about an algorithm's behavior when subjected to real situations.

The twelve cases were specified as follows:

Case 1 In this case, there are three measurement vectors with measurement noise $\sigma_1 = \sigma_2 = \sigma_3 = 10^{-6}$ rad as follows:

$$\begin{aligned} \vec{\mathbf{r}}_1 &= [1,0,0]^{\mathbf{T}} \\ \vec{\mathbf{r}}_2 &= [0,1,0]^{\mathbf{T}} \\ \vec{\mathbf{r}}_3 &= [0,0,1]^{\mathbf{T}}. \end{aligned} \tag{8}$$

Case 2 The same $\vec{\mathbf{r}}_1$ and $\vec{\mathbf{r}}_2$ vectors as those from **Case 1** are used with measurement noise $\sigma_1 = \sigma_2 = 10^{-6}$ rad.

Case 3 The same three vectors from **Case 1** are used, but the noise is increased to $\sigma_1 = \sigma_2 = \sigma_3 = 0.01$ rad.

Case 4 The same as **Case 2**, but the the noise is increased to $\sigma_1 = \sigma_2 = 0.01$ rad.

Case 5 Two reference vectors are used with measurement noise $\sigma_1 = 10^{-6}$ and $\sigma_2 = 0.01$ rad as follows:

$$\begin{aligned} \vec{\mathbf{r}}_1 &= [0.6,0.8,0]^{\mathbf{T}} \\ \vec{\mathbf{r}}_2 &= [0.8,-0.6,0]^{\mathbf{T}}. \end{aligned} \tag{9}$$

Case 6 In this case, there are three measurement vectors with measurement noise $\sigma_1 = \sigma_2 = \sigma_3 = 10^{-6}$ rad as follows:

$$\begin{aligned} \vec{\mathbf{r}}_1 &= [1,0,0]^{\mathbf{T}} \\ \vec{\mathbf{r}}_2 &= [1,0.01,0]^{\mathbf{T}} \\ \vec{\mathbf{r}}_3 &= [1,0,0.01]^{\mathbf{T}}. \end{aligned} \tag{10}$$

Case 7 The same $\vec{\mathbf{r}}_1$ and $\vec{\mathbf{r}}_2$ vectors as those from **Case 6** are used with measurement noise $\sigma_1 = \sigma_2 = 10^{-6}$ rad.

Case 8 The same three vectors as those from **Case 6** are used, but the noise is increased to $\sigma_1 = \sigma_2 = \sigma_3 = 0.01$ rad.

Case 9 The same as in **Case 7**, but the noise is increased to $\sigma_1 = \sigma_2 = 0.01$ rad.

Case 10 Three reference vectors are used with measurement noise $\sigma_1 = 10^{-6}$ rad and $\sigma_2 = \sigma_3 = 0.01$ rad as follows:

$$\begin{aligned}\vec{\mathbf{r}}_1 &= [1,0,0]^{\mathbf{T}}\\ \vec{\mathbf{r}}_2 &= [0.96,0.28,0]^{\mathbf{T}}\\ \vec{\mathbf{r}}_3 &= [0.96,0,0.28]^{\mathbf{T}}.\end{aligned} \tag{11}$$

Case 11 The same $\vec{\mathbf{r}}_1$ and $\vec{\mathbf{r}}_2$ vectors as those from **Case 10** are used with measurement noise $\sigma_1 = 10^{-6}$ rad and $\sigma_2 = 0.01$ rad.

Case 12 The same $\vec{\mathbf{r}}_1$ and $\vec{\mathbf{r}}_2$ vectors as those from **Case 10** are used with measurement noise $\sigma_1 = 0.01$ rad and $\sigma_2 = 10^{-6}$ rad.

For each case, the attitude profile matrix was built as defined in Equation (3) in order to serve as the input to the neural network, where a star-tracking scenario was considered, that is, each observation vector was equally weighted. Then, with the dropout layers enabled, **N** forward passes were executed through the network to sample a set of possible outputs. In those tests, the definition $\mathbf{N} = 1000$ was set.

After each forward pass, Wahba's error was evaluated by using Equation (1). The weights given by the test case were considered—and are defined in Equation (12b)—and $\mathbf{A_{pred}}$ was considered instead of $\mathbf{A_{true}}$ to measure the true distance. This led to an array of errors with size **N** for each test case. At the end of the **N** forward passes, the mean error was measured for each test case, as shown in Table 1. The error values are displayed on a logarithmic scale, where the bold entries represent the smallest values.

$$\sigma_{\mathbf{tot}} = \left(\sum_{\mathbf{i}=1}^{\mathcal{O}} \frac{1}{\sigma_{\mathbf{i}}^2}\right)^{-1} \tag{12a}$$

$$\mathbf{a_i} = \frac{\sigma_{\mathbf{tot}}}{\sigma_{\mathbf{i}}^2}. \tag{12b}$$

When choosing the most eligible model, if only the errors displayed in Table 1 were considered, it would still not be possible to notice any meaningful insights. The models were trained using four and five observations, and the use of a dropout rate of 10% was, in general, better than when using other rates. Thus, a procedure described in [37] was followed, which evaluated the rotation error with respect to the average rotation matrix of the predicted DCMs.

Table 1. Wahba's error for each model (the best result for each model is defined in bold).

Model	**3obs**			**4obs**			**5obs**			**6obs**			**7obs**		
Dropout Rate (%)	**10**	**15**	**20**	**10**	**15**	**20**	**10**	**15**	**20**	**10**	**15**	**20**	**10**	**15**	**20**
Case															
1	−2.03	−1.96	−1.90	−2.04	−2.00	−1.93	**−2.17**	−2.02	−1.99	−2.10	−2.08	−1.99	−2.14	−2.07	−1.97
2	−1.81	−1.75	−1.68	−1.86	−1.78	−1.73	−1.92	−1.82	−1.76	−1.86	−1.85	−1.76	**−1.95**	−1.80	−1.76
3	−2.00	−1.94	−1.90	−2.06	−1.98	−1.92	**−2.12**	−2.02	−1.97	−2.08	−2.05	−1.98	−2.11	−2.06	−1.97
4	−1.80	−1.74	−1.69	−1.86	−1.78	−1.74	**−1.92**	−1.83	−1.75	−1.85	−1.84	−1.75	−1.91	−1.79	−1.77
5	−1.82	−1.75	−1.69	−1.81	−1.72	−1.71	**−1.94**	−1.79	−1.78	−1.90	−1.88	−1.79	−1.93	−1.79	−1.74
6	−1.94	−1.87	−1.85	**−2.05**	−1.97	−1.91	−1.89	−1.87	−1.71	−1.75	−1.78	−1.73	−1.83	−1.77	−1.75
7	−1.77	−1.67	−1.65	**−1.87**	−1.78	−1.72	−1.70	−1.68	−1.53	−1.58	−1.61	−1.53	−1.65	−1.60	−1.57
8	−1.92	−1.85	−1.83	**−2.02**	−1.96	−1.87	−1.89	−1.86	−1.69	−1.75	−1.77	−1.74	−1.82	−1.76	−1.74
9	−1.75	−1.68	−1.65	**−1.85**	−1.78	−1.71	−1.71	−1.69	−1.52	−1.56	−1.60	−1.55	−1.65	−1.60	−1.56
10	**−1.95**	−1.92	−1.83	−1.88	−1.91	−1.88	−1.92	−1.90	−1.81	−1.73	−1.73	−1.76	−1.73	−1.70	−1.64
11	**−1.75**	−1.68	−1.65	−1.65	−1.63	−1.62	−1.67	−1.65	−1.65	−1.64	−1.56	−1.54	−1.54	−1.58	−1.54
12	−1.56	−1.53	−1.53	−1.54	−1.58	−1.57	−1.49	**−1.64**	−1.53	−1.37	−1.44	−1.46	−1.39	−1.46	−1.35

Given the set of predicted DCMs $\{\hat{\mathbf{R}}_\mathbf{j} \mid \mathbf{j} = 1, 2, ..., \mathbf{N}\}$, it was possible to calculate an average rotation matrix $\mathbf{R_{avg}}$, that is, a matrix that represented the mean of a distribution of rotation matrices. This average rotation was found as the orthogonal projection of a

matrix $\overline{\mathbf{R}} = \mathbf{N}^{-1} \sum_{\mathbf{j}=1}^{\mathbf{N}} \hat{\mathbf{R}}_\mathbf{j}$ and can be further reviewed in [38], which used an SVD approach in order to evaluate $\mathbf{R_{avg}}$.

Once $\mathbf{R_{avg}}$ was determined, the noise matrices $\Delta\mathbf{R_j}$ could be evaluated. The new matrices held the orientation differences between each predicted DCM $\hat{\mathbf{R}}_\mathbf{j}$ and the average DCM $\mathbf{R_{avg}}$, which could be found by performing the following matrix multiplication:

$$\Delta\mathbf{R_j}(\vec{\mathbf{e}}_\mathbf{j}, \phi_\mathbf{j}) = \mathbf{R}_\mathbf{avg}^\mathbf{T} \hat{\mathbf{R}}_\mathbf{j} \tag{13}$$

where each $\Delta\mathbf{R_j}$ is parametrized by a rotation axis $\vec{\mathbf{e}}_\mathbf{j}$ and a rotation angle $\phi_\mathbf{j}$. After finding the noise matrices, their respective Euler vectors $\vec{\mathbf{u}}_\mathbf{j} = \phi_\mathbf{j} \vec{\mathbf{e}}_\mathbf{j}$ were extracted, and a covariance matrix was calculated as follows:

$$\mathbf{C}(\vec{\mathbf{u}}) = \frac{1}{\mathbf{N}} [\vec{\mathbf{u}}_1, \vec{\mathbf{u}}_2, ..., \vec{\mathbf{u}}_\mathbf{N}][\vec{\mathbf{u}}_1, \vec{\mathbf{u}}_2, ..., \vec{\mathbf{u}}_\mathbf{N}]^\mathbf{T}. \tag{14}$$

Since $\vec{\mathbf{u}}_\mathbf{j}$ is a representation of the difference rotation matrix $\Delta\mathbf{R_j}$, by taking the diagonal entries of $\mathbf{C}(\vec{\mathbf{u}})$, it is possible to have the variances in the **x**, **y**, and **z** axes of the predicted DCMs with respect to $\mathbf{R_{avg}}$. This procedure was performed for each test case to measure the uncertainty of the rotation axes predicted by each model. The measured variances (in degrees^2) for each dropout rate are displayed in Figures 4–6.

The models with a dropout rate of 10% had lower variance levels in the three axes for the majority of the test cases. It is worth mentioning that these models were tested with the same dropout rate as that with which they were trained, that is, the models that were trained with a dropout rate of 10% were tested with the same rate, and so on. By doing that, it was possible to verify whether the number of observation vectors presented the same behavior independent of the dropout rate used. For the $\vec{\mathbf{u}}_y$ and $\vec{\mathbf{u}}_z$ components, the values were similar, but the model that was trained using four observations had smaller variances for the $\vec{\mathbf{u}}_x$ component compared to the others.

These results led to the choice of the model that was trained with four observation vectors as the best option. Since the experiments dealt with directional data, it was not possible to use the standard equations of the mean and variance to calculate the uncertainty. Nonetheless, the dropout procedure was still applied during the inference, so it generated different outputs for the same input. Therefore, it was still possible to use those predictions and the covariance approach to estimate the uncertainty.

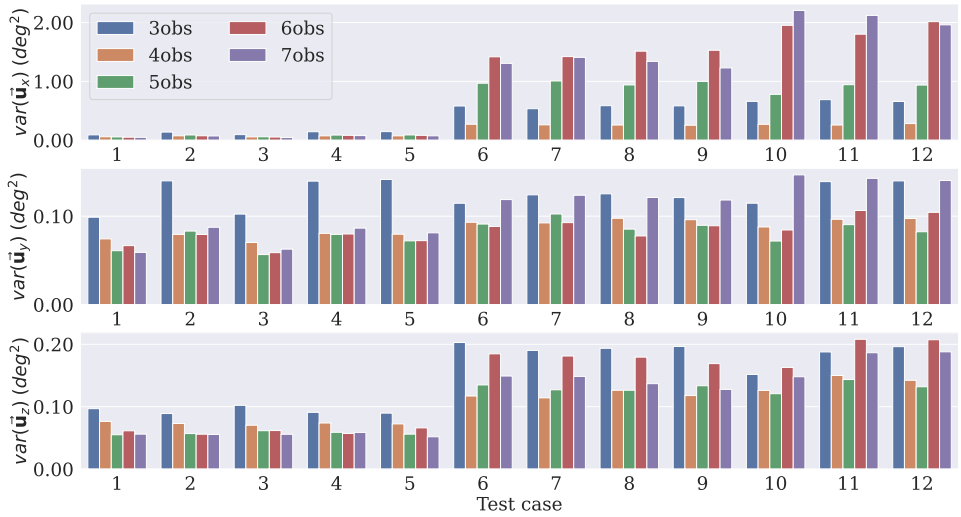

Figure 4. Uncertainties in the axes for each test case considering a dropout rate of 10%.

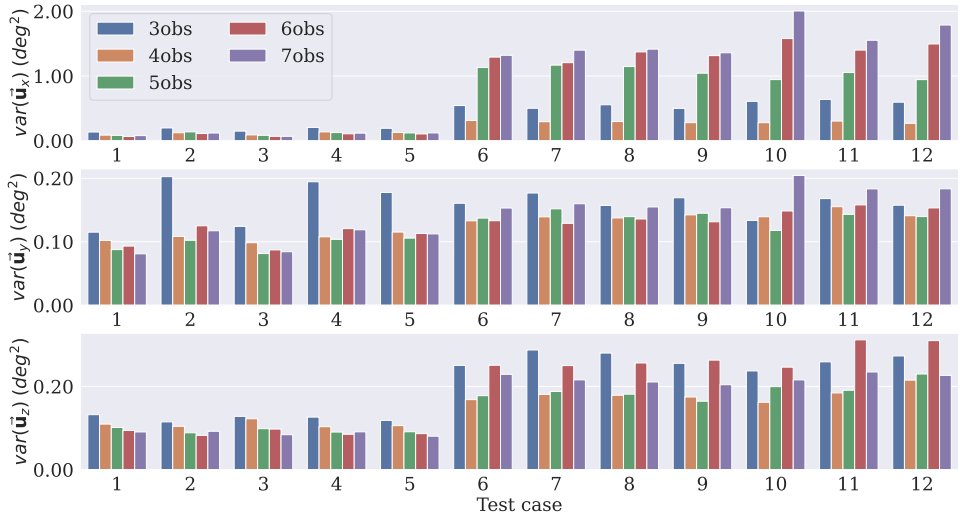

Figure 5. Uncertainties in the axes for each test case considering a dropout rate of 15%.

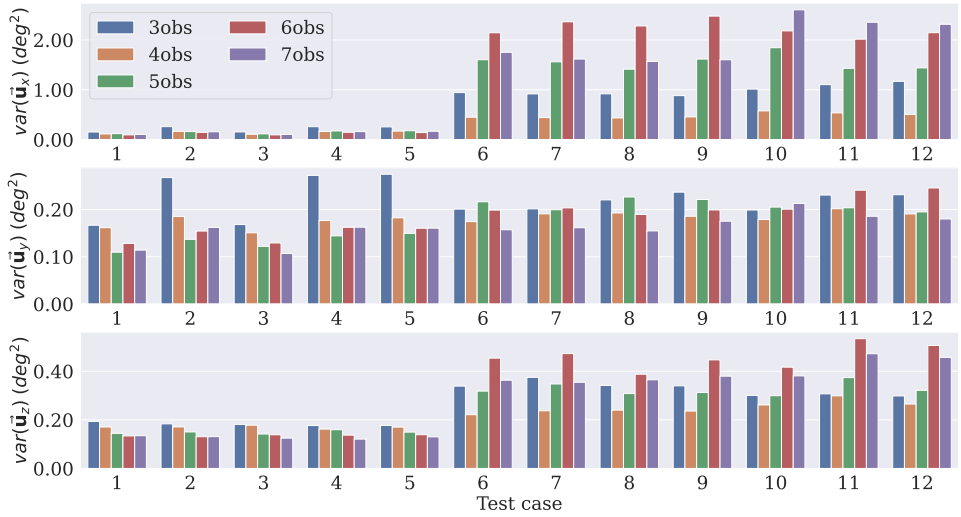

Figure 6. Uncertainties in the axes for each test case considering a dropout rate of 20%.

The distance between the true angle (extracted from Equation (7)) and the angles from the predicted attitude matrices $\mathbf{A_{pred}}$ was measured. For each test case, $\mathbf{N}$ predictions were taken, with their angles extracted using Equation (6). Figures 7–9 show how the absolute angular differences (in degrees) were spread for the three dropout values. It can be noticed that the dispersion of the differences was narrower for the model that used four observation vectors.

It was verified that the model trained with four vectors had the best results. To better understand the models, two new tests were performed. The first one evaluated the behavior of the models when using another dropout rate during the testing. The other estimated how capable the models were of retaining knowledge when the number of switched-off neurons was varied. Wahba's error and the variance of the B-Swish-4obs model were evaluated as they were previously, but the dropout rate was swept from 0% to 20% with steps of 2.5%. As shown in Figure 10, when the dropout rate increased, it was possible to notice that the model trained with 10% could not retain much knowledge compared with the other models, even though it achieved lower errors when the dropout layer was disabled (0%).

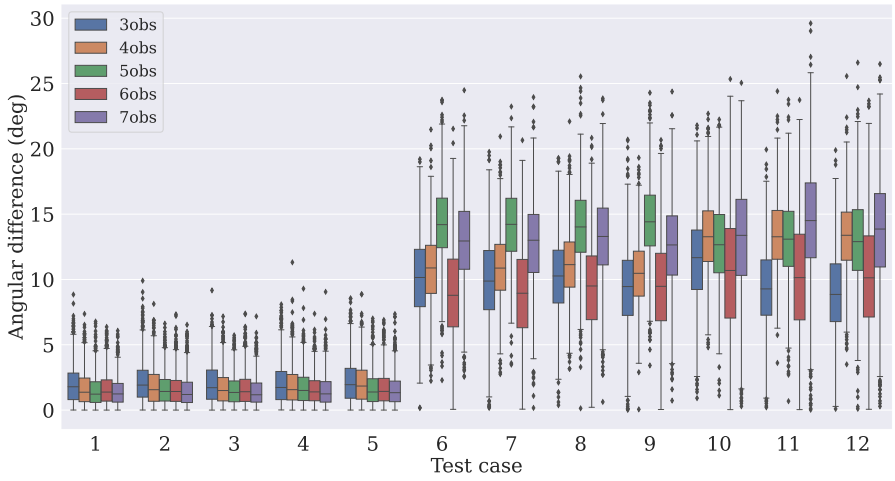

Figure 7. Dispersion of the absolute angular difference when using a dropout rate of 10%.

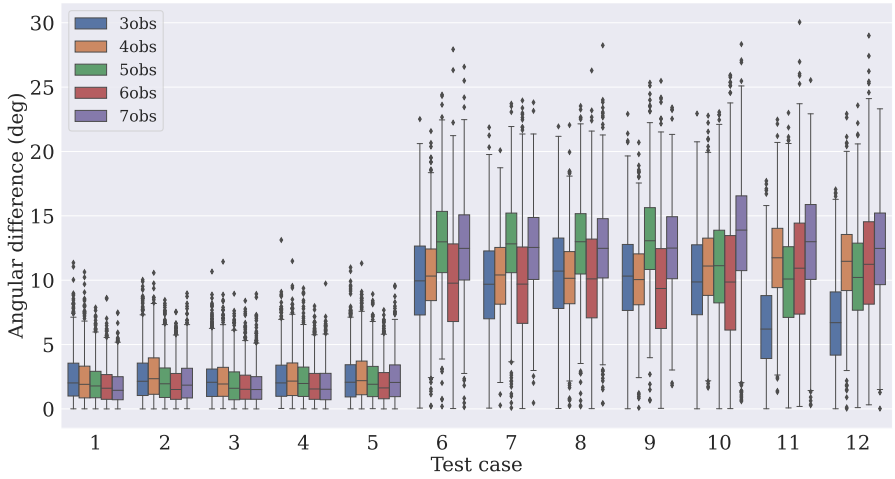

Figure 8. Dispersion of the absolute angular difference when using a dropout rate of 15%.

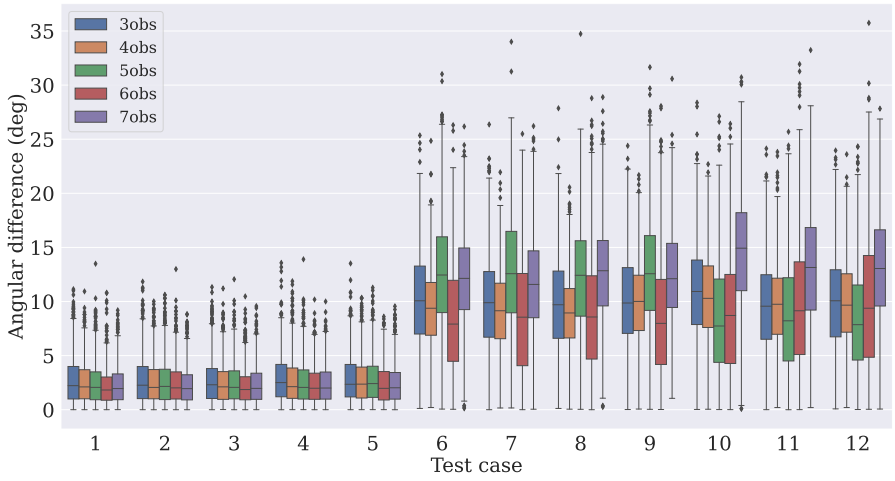

Figure 9. Dispersion of the absolute angular difference when using a dropout rate of 20%.

Similarly, the model trained with 10% became more uncertain of its predictions as the testing dropout rate increased, as can be seen in Figure 11. The upper bound refers to the variance in the **x**-axis, the lower bound refers to the **z**-axis, and the dashed lines refer to the variance in the **y**-axis.

Figure 10. The evolution of Wahba's error for test case 8.

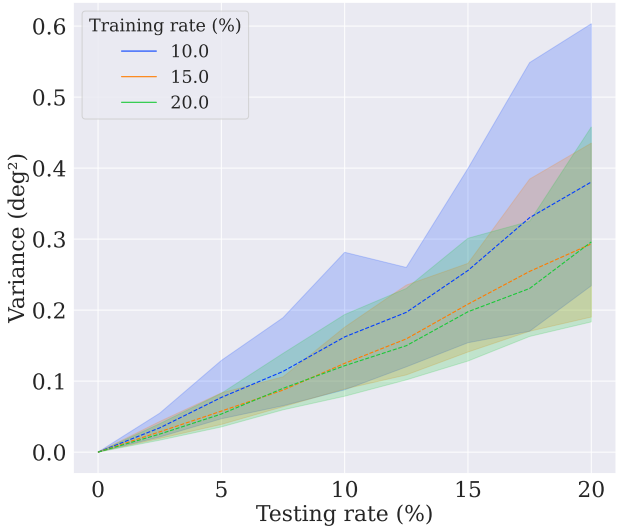

Figure 11. The evolution of the variance for test case 8.

The results presented in Figures 10 and 11 are only relevant if the model is running in an unstable environment, that is, it is expected that some neurons could "be lost" during inference; thus, the model has to be robust against the number of active neurons. To perform a comparison with the traditional algorithms, it was necessary to use the best scenario, that is, the B-Swish-4obs trained with a dropout rate of 10%, since it resulted in the lowest Wahba's error when the dropout layers were disabled.

4.3. Comparison with Traditional Algorithms

To have confirmation that the chosen neural network model is the most appropriate one for the problem in question, a benchmark was carried out by using QUEST [39], an SVD-based method [40], the q-method, and ESOQ2. To do that, the twelve test cases defined above were used, but now, $\mathcal{S}$ samples were generated for each case and their respective measurement noises $\mathcal{N}(0, \sigma_{\mathbf{i}})$ were applied. For the tests, $\mathcal{S} = 4000$ was considered.

For each sample used in the algorithm, Wahba's error was calculated by using Equation (1) with the value of $\mathbf{a_i}$ given by each test case. After all samples had been evaluated, the mean of all errors $\sum_{\mathbf{j}=1}^{\mathcal{S}} L(\hat{\mathbf{R}}_{\mathbf{j}})$ was evaluated for each case. This procedure was executed for all algorithms. The dropout layers were disabled during inference since there was

no need for evaluating the model uncertainty in this phase. The results are depicted in Figure 12 on a logarithmic scale for better visualization.

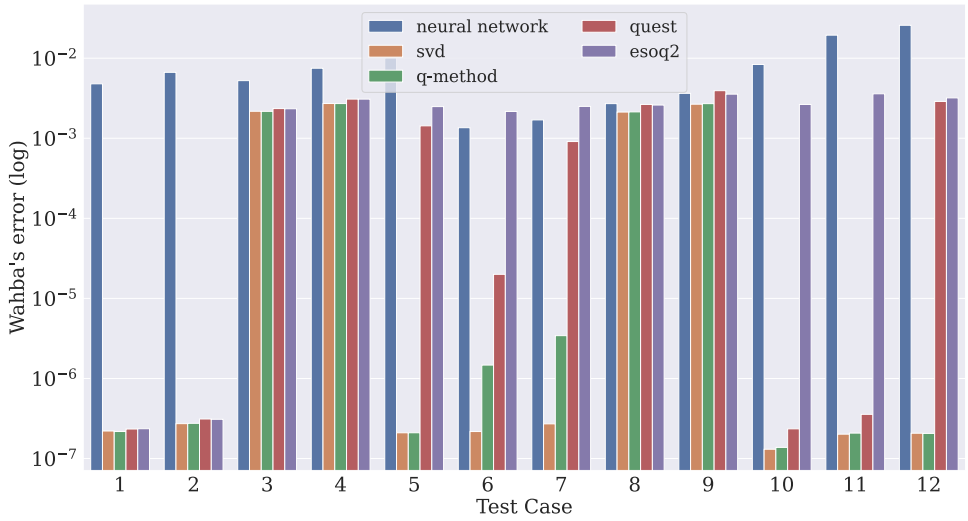

Figure 12. Comparison of the evaluation of Wahba's error among some algorithms.

The SVD-based and q-method algorithms achieved the lowest errors, since they were the most robust. In some situations, such as in cases **3**, **4**, **8**, and **9**, all algorithms had similar results, since the inputs to the system had more significant associated measurement noise. In spite of the fact that the neural network performed worse in most cases, the predictions were more constant, that is, when considerable noise was introduced, the system error did not increase proportionally in the same way as in the other algorithms.

The relationship between the measurement noise and Wahba's error is presented in Figure 13. This analysis considered a weighted average of the measurement noises for each test case by using Equation (12). It is possible to notice that Wahba's error remained constant in the neural network when the weighted average increased, unlike with the traditional algorithms. In addition, the presence of a single accurate observation vector (low-valued σ) in a test case was enough to prevent the conventional algorithm from performing poorly. Case number 10 is an example of this statement, since the traditional algorithms become less affected by the noise despite having two measurement noises with higher values.

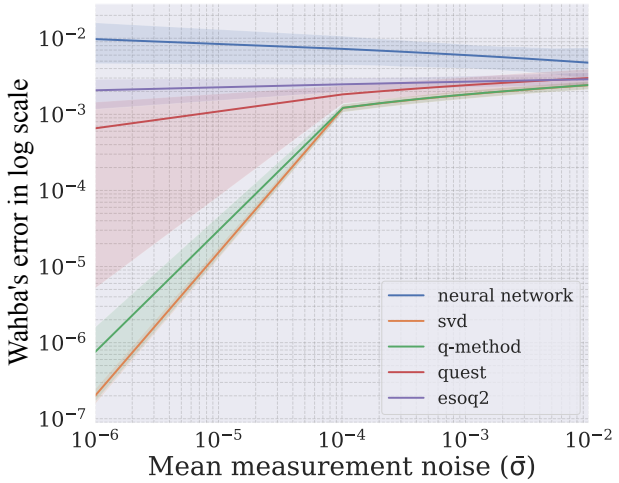

Figure 13. Impact of the average measurement noise on model performance.

5. Conclusions

This paper proposes a neural network model for the problem of static attitude determination based on an existing architecture called PointNet. Modifications to the system's layers, input shapes, and output shapes were performed due to the poor results in preliminary tests. It was noticed that changing the ReLU activation function to Swish resulted in an improvement in the loss with the validation set. The most noticeable gain happened when the input used in the system was changed, which was probably because the network was unable to obtain meaningful information about how the pairs of observation and reference vectors were correlated. In addition, traditional algorithms use calculation techniques that may require more computational resources, such as SVD, eigenvectors, and eigenvalues. With the proposed method, after the neural network is trained on Earth (offline training), the embedded system will rely on pre-established weights; thus, it will only require the active calculation of multiplications and additions.

Using a star-tracking scenario during the training stage resulted in smaller values of Wahba's error during the testing stage than those obtained when using the actual weights. This is interesting because it removes the need to store the actual weights $\mathbf{a_i}$ in order to build the attitude profile matrix $\mathbf{B}$. The error with the validation data indeed decreased when the number of measurement vectors increased. This did not reflect the actual behavior when the test case scenarios were run. This might be because the test cases used only up to three vectors, and the models that were trained using more observation vectors could not generalize very well for fewer vectors. However, for the same test cases, the variances of the predicted DCMs were not out of line around the **y** and **z** axes when measuring the uncertainty. On the other hand, they had significant differences for the **x** axis, especially for cases 6 to 12, where the input vectors had a boresight along the **x** axis.

While the chosen model could not surpass the traditional algorithms when the input vectors had more significant measurement noise associated with them, it did not suffer from this noise as the other algorithms did, achieving similar values of Wahba's error; therefore, it is less sensitive to the input noise. As future work, it is suggested Bingham's distribution concepts be incorporated into the model. As our output data are somehow a type of directional data, it may be possible to have a better estimation of the uncertainty, perhaps by reducing this value and obtaining some improvements in Wahba's loss. Thus, the effective use and testing of the proposed solution in real situations need to be considered.

Author Contributions: Conceptualization, G.H.d.S.; investigation, G.H.d.S.; software, G.H.d.S.; validation, G.H.d.S.; supervision, L.O.S. and E.A.B.; writing—original draft preparation, G.H.d.S.; writing—review and editing, L.O.S., E.A.B. and S.F.S.; resources, V.R.Q.L. and A.S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by national funds through the Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) by the project UIDB/05064/2020 (VALORIZA—Research Center for Endogenous Resource Valorization). André Filipe Sales Mendes received a scholarship from the European Social Fund and Junta de Castilla y León (Operational Program 2014–2020 for Castilla y León, EDU/556/2019 BOCYL).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Le, T.T.; Le, T.S.; Chen, Y.R.; Vidal, J.; Lin, C.Y. 6D pose estimation with combined deep learning and 3D vision techniques for a fast and accurate object grasping. *Robot. Auton. Syst.* **2021**, *141*, 103775. [CrossRef]
2. Wang, G.; Liu, X.; Han, S. A Method of Robot Base Frame Calibration by Using Dual Quaternion Algebra. *IEEE Access* **2018**, *6*, 74865–74873. [CrossRef]

3. Suarez, A.; Heredia, G.; Ollero, A. Design of an Anthropomorphic, Compliant, and Lightweight Dual Arm for Aerial Manipulation. *IEEE Access* **2018**, *6*, 29173–29189. [CrossRef]
4. Yin, P.; Ye, J.; Lin, G.; Wu, Q. Graph neural network for 6D object pose estimation. *Knowl.-Based Syst.* **2021**, *218*, 106839. [CrossRef]
5. Tian, M.; Guan, B.; Xing, Z.; Fraundorfer, F. Efficient Ego-Motion Estimation for Multi-Camera Systems with Decoupled Rotation and Translation. *IEEE Access* **2020**, *8*, 153804–153814. [CrossRef]
6. Cariow, A.; Cariowa, G.; Majorkowska-Mech, D. An algorithm for quaternion-based 3D rotation. *Int. J. Appl. Math. Comput. Sci.* **2020**, *30*, 149–160. [CrossRef]
7. Wahba, G. A least squares estimate of satellite attitude. *SIAM Rev.* **1965**, *7*, 409. [CrossRef]
8. Guo, C.; Li, F.; Tian, Z.; Guo, W.; Tan, S. Intelligent active fault-tolerant system for multi-source integrated navigation system based on deep neural network. *Neural Comput. Appl.* **2020**, *32*, 16857–16874. [CrossRef]
9. Tan, T.N.; Khenchaf, A.; Comblet, F.; Franck, P.; Champeyroux, J.M.; Reichert, O. Robust-Extended Kalman Filter and Long Short-Term Memory Combination to Enhance the Quality of Single Point Positioning. *Appl. Sci.* **2020**, *10*, 4335. [CrossRef]
10. Esfahani, M.A.; Wang, H.; Wu, K.; Yuan, S. OriNet: Robust 3-D Orientation Estimation With a Single Particular IMU. *IEEE Robot. Autom. Lett.* **2019**, *5*, 399–406. [CrossRef]
11. Shuster, M.D. Approximate algorithms for fast optimal attitude computation. In Proceedings of the Guidance and Control Conference, Palo Alto, CA, USA, 7–9 August 1978; p. 1249.
12. Shuster, M.D.; Oh, S.D. Three-axis attitude determination from vector observations. *J. Guid. Control.* **1981**, *4*, 70–77. [CrossRef]
13. Markley, F.L. Attitude determination using vector observations and the singular value decomposition. *J. Astronaut. Sci.* **1988**, *36*, 245–258.
14. Mortari, D. Second estimator of the optimal quaternion. *J. Guid. Control. Dyn.* **2000**, *23*, 885–888. [CrossRef]
15. Davenport, P.B. *A Vector Approach to the Algebra of Rotations with Applications*; NASA Technical Note D-4696; NASA: Washington, DC, USA, 1968.
16. Keat, J. *Analysis of Least-Squares Attitude Determination Routine DOAOP (CSC/TM-77/6034)*; Technical Report CSC/TM-77/6034; Computer Sciences Corp.: Tysons, VA, USA, 1977.
17. Spratling, B.B.; Mortari, D. A survey on star identification algorithms. *Algorithms* **2009**, *2*, 93–107. [CrossRef]
18. Hashim, H.A. Attitude determination and estimation using vector observations: Review, challenges and comparative results. *arXiv* **2020**, arXiv:2001.03787.
19. Zhou, Y.; Barnes, C.; Lu, J.; Yang, J.; Li, H. On the continuity of rotation representations in neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Long Beach, CA, USA, 16–20 June 2019; Volume 1, pp. 5745–5753.
20. Xiang, Y.; Schmidt, T.; Narayanan, V.; Fox, D. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv* **2017**, arXiv:1711.00199.
21. Do, T.T.; Cai, M.; Pham, T.; Reid, I. Deep-6dpose: Recovering 6d object pose from a single rgb image. *arXiv* **2018**, arXiv:1802.10367.
22. Gao, G.; Lauri, M.; Zhang, J.; Frintrop, S. Occlusion resistant object rotation regression from point cloud segments. In Proceedings of the 15th European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 716–729.
23. Park, J.M.; Yoo, Y.H.; Kim, U.H.; Lee, D.; Kim, J.H. D3PointNet: Dual-Level Defect Detection PointNet for Solder Paste Printer in Surface Mount Technology. *IEEE Access* **2020**, *8*, 140310–140322. [CrossRef]
24. Yao, X.; Guo, J.; Hu, J.; Cao, Q. Using Deep Learning in Semantic Classification for Point Cloud Data. *IEEE Access* **2019**, *7*, 37121–37130. [CrossRef]
25. Li, Y.; Bu, R.; Sun, M.; Wu, W.; Di, X.; Chen, B. Pointcnn: Convolution on x-transformed points. *Adv. Neural Inf. Process. Syst.* **2018**, *31*, 820–830.
26. Jin, S.; Su, Y.; Gao, S.; Wu, F.; Ma, Q.; Xu, K.; Ma, Q.; Hu, T.; Liu, J.; Pang, S.; et al. Separating the Structural Components of Maize for Field Phenotyping Using Terrestrial LiDAR Data and Deep Convolutional Neural Networks. *IEEE Trans. Geosci. Remote Sens.* **2020**, *58*, 2644–2658. [CrossRef]
27. Hu, P.; Kaashki, N.N.; Dadarlat, V.; Munteanu, A. Learning to Estimate the Body Shape Under Clothing From a Single 3-D Scan. *IEEE Trans. Ind. Inform.* **2021**, *17*, 3793–3802. [CrossRef]
28. Peng, Y.; Chang, M.; Wang, Q.; Qian, Y.; Zhang, Y.; Wei, M.; Liao, X. Sparse-to-Dense Multi-Encoder Shape Completion of Unstructured Point Cloud. *IEEE Access* **2020**, *8*, 30969–30978. [CrossRef]
29. Yu, Y.; Adu, K.; Tashi, N.; Anokye, P.; Wang, X.; Ayidzoe, M.A. RMAF: Relu-Memristor-Like Activation Function for Deep Learning. *IEEE Access* **2020**, *8*, 72727–72741. [CrossRef]
30. Qian, L.; Hu, L.; Zhao, L.; Wang, T.; Jiang, R. Sequence-Dropout Block for Reducing Overfitting Problem in Image Classification. *IEEE Access* **2020**, *8*, 62830–62840. [CrossRef]
31. Zhang, Q.H.; Ni, Y.Q. Improved Most Likely Heteroscedastic Gaussian Process Regression via Bayesian Residual Moment Estimator. *IEEE Trans. Signal Process.* **2020**, *68*, 3450–3460. [CrossRef]
32. Mamo, T.; Wang, F.K. Long Short-Term Memory With Attention Mechanism for State of Charge Estimation of Lithium-Ion Batteries. *IEEE Access* **2020**, *8*, 94140–94151. [CrossRef]
33. Xu, Z.; Xie, J.; Zhou, Z.; Zhao, J.; Xu, Z. Accurate Direct Strapdown Direction Cosine Algorithm. *IEEE Trans. Aerosp. Electron. Syst.* **2019**, *55*, 2045–2053. [CrossRef]
34. Dytso, A.; Cardone, M.; Poor, H.V. On Estimating the Norm of a Gaussian Vector Under Additive White Gaussian Noise. *IEEE Signal Process. Lett.* **2019**, *26*, 1325–1329. [CrossRef]

35. Stefenon, S.F.; Kasburg, C.; Nied, A.; Klaar, A.C.R.; Ferreira, F.C.S.; Branco, N.W. Hybrid deep learning for power generation forecasting in active solar trackers. *IET Gener. Trans. Distrib.* **2020**, *14*, 5667–5674. [CrossRef]
36. Markley, F.L. Attitude Determination from Vector Observations: A Fast Optimal Matrix Algorithm. *J. Astronaut. Sci.* **1993**, *41*, 261–280.
37. Franaszek, M.; Cheok, G.S. Orientation uncertainty characteristics of some pose measuring systems. *Math. Probl. Eng.* **2017**, *2017*, 2696108. [CrossRef] [PubMed]
38. Curtis, W.D.; Janin, A.L.; Zikan, K. A note on averaging rotations. In Proceedings of the IEEE Virtual Reality Annual International Symposium, Seattle, WA, USA, 18–22 September 1993; pp. 377–385.
39. Paiva, L.P.S.; de Melo, F.M.S.R.; Menezes Filho, R.; Vieira, L.A.; Oliveira, F. Error analysis for alignment in AHRS: QUEST and SAAM algorithms. *An. Soc. Bras. Autom.* **2020**, *2*. [CrossRef]
40. Wu, J. Rigid 3-D Registration: A Simple Method Free of SVD and Eigendecomposition. *IEEE Trans. Instrum. Meas.* **2020**, *69*, 8288–8303. [CrossRef]