ARTIFICIAL SATELLITES, Vol. 59, No. 2 – 2024
DOI: 10.2478/arsa-2024-0004

NEURAL NETWORK APPLIED TO TELESCOPE POINTING INACCURACY MODEL

Vitaliy ZHABOROVSKYY, Myhailo MEDVEDSKY, Vasyl CHOLIY, Victor PAP, Viachelsav SEMENENKO

Main Astronomical Observatory, Kyiv, Ukraine

e-mail: zhskyy@mao.kiev.ua

ABSTRACT. In the course of satellite observations using satellite laser ranging (SLR), a key task is pointing the telescope with high precision. Positioning the steering system's mechanical parts with zero error is impossible. Accordingly, we must analyze and account for pointing errors by incorporating the telescope mounting errors themselves into the modeling error. Such models are far from trivial owing to the factors such as satellite azimuth, altitude, perhaps distance, or meteorological data.

In this article, we explain how the data for the telescope pointing inaccuracy model (TIM) was collected and how a neural network was used to build a very precise TIM for the Golisiiv 1824 SLR station in Kyiv.

We have focused our efforts on the suggested approach's positive aspects based on our experience of using it to find practical solutions. Our practical recommendations may also be interesting for anyone working with hardware, especially in analyzing their errors. The key proof of the effectiveness of the approach is the serious increase in the number of satellites successfully tracked, especially for "blind" paths, when the satellite is not visible to the observer through the telescope guide.

Keywords: telescope pointing model, neural network, satellite laser ranging

1. INTRODUCTION

The Golisiiv 1824 SLR station in Kyiv uses three reflectors and one collimator on a common setup. The direction of the telescope axis is given in horizon coordinates with its azimuth A – accounting for the northern bearing, – and height or altitude h – being the angle between the horizon and the axis direction.

The steering subsystem uses absolute digital angle sensors with 21 binary digits. This means that for a whole 360° circle, the sensor readouts give us (without any additional steps and calculations) the azimuth and altitude of the telescope axis with an approximate precision of 0.6 arcsec ($360^\circ/2^{21} * 3600 \sim 0''.62$).

Hardware issues and shifting of the sensor zeroes add some errors in pointing. Thus, if we need to point the telescope axis at A_e, h_e, we must instead point it toward A_o, h_o. The differences $\Delta_A = A_e - A_o$ and $\Delta_h = h_e - h_o$ are called the telescope inaccuracy model (TIM). The model depends on Ae, he, the satellite orbit, and some other parameters such as meteorological data.

Preselection of the model parameters may not be obvious, so iterations are essential. In the first step, the model should provide us with the shifting of a sensor's zero, after which we will then be able to analyze finer details.

Over time, due to technical maintenance issues and lower telescope mount and steering system quality, the model loses accuracy, and blurs and other issues may appear. We must keep our attention on the model and constantly work on its precision. Maintaining the model, including its rebuilding after collecting new data, is a continuous task.

The main criteria for pointing quality are the ability to receive signals from non visible satellites along with the number of successful observations. There are plenty of visible satellites during summer, with non visible satellites mostly being observed during winter. As such, the main idea is to collect data from visible satellite observation and have updated TIM ready for the new winter season.

The first generation of inaccuracy models for SLR telescopes was created based on a classical instrument theory. Bending, axes non perpendicularity, and zero shift were successfully modeled (Butkiewicz et al., 1994).

The first TIM for the Golisiiv 1824 SLR station in Kyiv was based on stellar observations (Medvedsky and Suberlak, 2002). The model worked well enough as a first approximation, but there were two obvious drawbacks: special time needed to be devoted for observations of the stars, cutting out time for satellites, and the model was built for observations of stars despite the telescope being assigned for satellite observation. In the latter case, there is a planetary aberration that needs special attention and is absent in stellar observation.

The second version of TIM was the model based on satellite observations (Zhaborovskyy et al, 2013). In this case, the major disadvantage in the first place is the necessity of serious preliminary data analysis and the need to build polygonal and trigonometric formulas. Despite being quite trivial from a mathematical point of view, the model proved to be quite nontrivial during preparation and maintenance.

The idea of the third (the subject of the article) model arose from our assumption that no particular precision is gained by the model from its mathematical content, so we decided to concentrate on the most valuable practical items: the model must be as easy and fast as possible during its preparation, usage, and derived calculations. Maintaining and changing the model if a new observation is added must not be difficult either. Our opinion is that the neural network is good for this – we care little about which formulary is inside the model, and we are just interested in having result values for given input parameters and in immediate utilization (maybe in real time) of the model's latest sensor readouts. There is no reason to compare the precision of previous models with that of the newest one because in practice, the most important criterion is ease and usage of model building.

2. DATA PREPARATION

Observation of the satellites starts from building its ephemeris. It is done once per day, mostly in the evening before the observation session starts. Having calculated it, we can point the telescope using the calculated data. During evening and morning sessions, most of the satellites are in sunlight and can be seen through the telescope guide. If necessary, an operator can manually correct the telescope pointing. For each successful observation the sensor readout A_o, h_o, their time changes, and ephemeris satellite position A_e, h_e is saved. We get the satellite's position according to the ephemerides and the position sensors, and the discrepancy of these positions makes up the telescope pointing error. These data are the basis for TIM creation and can be collected while the main observing program is running.

The accuracy of the measurement of pointing errors depends on the accuracy of the sensor (0.6 arcseconds), the accuracy of the ephemeris (1 – 2 arcseconds), and the width of the laser beam (10 arcseconds), hitting a satellite that guarantees signal reception. Based on the given values, the width of the laser beam is dominant and more precise modeling is not required. Of course, a situation is possible when the accuracy of the ephemeris is lower, but only for specific satellites/cases, and we do not use such data for modeling errors.

Most satellites are not visible in the middle of the night, and an operator must scan the sky in the vicinity of the ephemeris point until the satellite is caught. The better the TIM, the less time wasted scanning. Ephemeris data and sensor readouts are the source of the data for the TIM. The TIM itself is an approximation of their differences. An early model (Zhaborovskyy, 2013) consisting mostly of trigonometric series was used at our SLR station but had poor precision (near 150 arcsec for both axes). Its level of precision was only satisfactory for some particular satellite paths. The model was quite hard to manage.

In Figure 1, the differences in azimuth and altitude for the year 2023 are presented for all paths of all satellites. The figure shows approximately 13,000 points with the mean value removed and scaled to $[0...1]$. The SLR setup can only work with altitudes $[20°...75°]$ so that the gap around the zenith is clearly visible. Absolute corrections for azimuth: $[-0.24...0.13]$, altitude: $[-0.09..0.07]$ degrees.

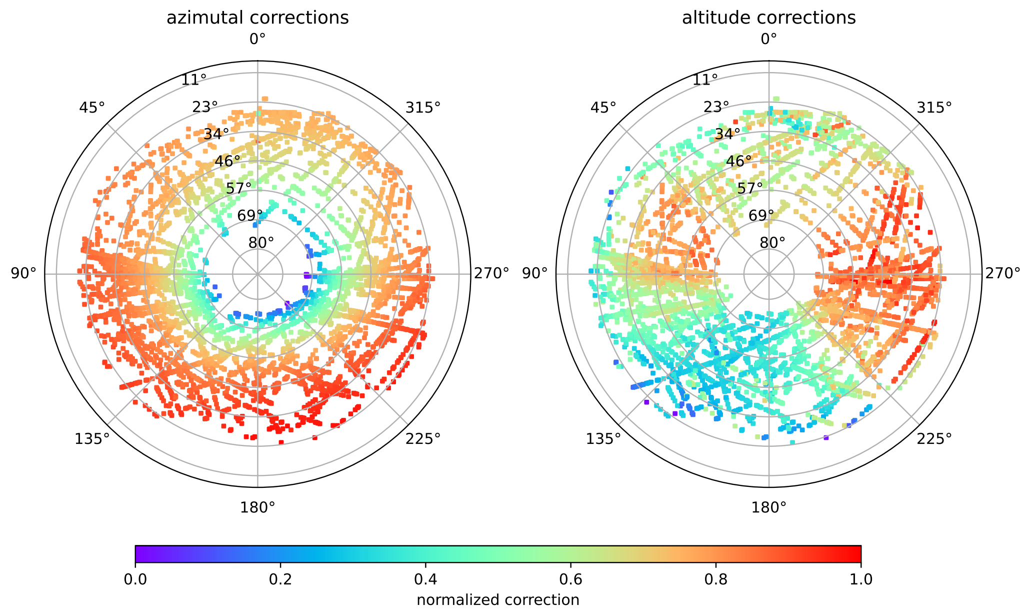

Figure 1. Raw source data for the model. Approximately 13,000 points are plotted in the horizon coordinate system. Normalized corrections are shown in different colors.

3. NEURAL NETWORK USAGE

According to the Cybenko theorem (Cybenko, 1989), arbitrary continuous function f of real argument can be approximated with any precision by the sum:

$$G(\vec{x}, \vec{w}, \alpha, \theta) = \sum_{i=1}^{N} \alpha_i \varphi(\vec{w_i}^T \vec{x} + \theta_i), \tag{1}$$

where

φ	–	some scalar function, see below;
$\vec{x}$	–	function arguments;
$\vec{w}, \theta, \alpha$	–	parameters;
N	–	level of approximations (layers).

Function $G(\vec{x}, \vec{w}, \alpha, \theta)$ can be implemented by a feed-forward artificial neural network (no loops allowed) with a minimum of one hidden layer. In the context of this work, the main advantage lies in the simplicity of usage without deep analysis of the neural network's mathematical model.

A neural network is built from layers of artificial neurons. The first of the layers is for input, and the number of neurons in it being the number of inputs. The final layer is for output, and it contains an arbitrary amount of neurons corresponding to as many outputs as we need. The neural network can contain as many hidden layers between the input and the final ones as needed. The output from one layer becomes the next layer's input. Achieving the ideal number of hidden layers is a matter of investigation.

It is worth noting that any preselected φ must be a nonlinear, smooth monotonous increasing function. Generally, it is called “activation function” and any non linearity of the final result results from their usage. Various layers of the neural network without activation functions can be joined, and the network may be represented only by the hidden layer.

Let us denote the number of neuron inputs as M. In this case, the neuron is representable as follows:

$$y = \varphi(\sum_{j}^{M} x_j w_j + \theta_i), \tag{2}$$

where

x_j	–	input data;
w_j	–	weights;
θ_i	–	shift;
y	–	output data;
φ	–	neuron activation function.

The general case (2) has nothing to do with the physics of the model, but we can use them given that we are solely interested in the numbers. The other task here is to preselect N– a number of layers and φ. There is no quick and universal fix for it. We have to do some investigation and find them from the results of numerical experiments. It adds the phenomenology to the neural network approach and shifts it away from the physics of the model.

Search for optimal w_j and θ is “model learning”. In general cases, optimizing some other “cost function” (or “lost function”) $C(f, G)$ in N-dimension parameter space will be required. The minimum of the cost function gives us some understanding of the approximation precision. The most trivial variant is standard deviation of ephemeris values from sensor readouts like in least squares method. However, the field of choice for cost function is quite huge. The method of cost function optimization is again a matter of choice, and it is most frequently gradient descent.

We use Python (version 3.9) with Keras library (version 2.11) (Chollet et al., 2015). All necessary instruments for creating the network, optimization, and precision control are included in the library.

To build the neural network for optimization of a given function, one should follow these steps: preselect a number of inputs, outputs, hidden layers, and number of neurons for each layer. For each layer, we should preselect “activation functions” and define “cost function”. The latter will help us determine the precision of the approximation. A mathematical algorithm for optimization need to be preselected too. Concerning neural networks, tasks for data fitting are called “regression”.

Let us discuss some recommendations for building networks:

- Use “dense” layers; layers where each neuron has connections with all neurons of the two neighboring layers; this is the best choice for regression case.
- Do not use too many neurons; these sorts of networks are prone to “overfitting”, an undesirable machine learning behavior that occurs when the machine learning model gives accurate predictions for training data but not for new data.
- Each neuron of the first layer resolving one feature of the input data; the next layer’s neurons combine the features found in the previous layer; the flexibility of the model depends on the number of layers with each neuron of the last layer resolving one feature of output data.
- More layers with an adequate number of neurons in them are generally better than an adequate number of layers with lots of neurons.
- Since this is regression, the neural network models data within the training interval better than outside it.

4. RESULTS AND CONCLUSIONS

4.1. Results

As a result, we want to create a model that uses azimuth and altitude as inputs and generates two corrections – one for azimuth and the other for altitude. Sines and cosines are used instead of angles to avoid ambiguity at the 0 – 360 degree azimuth break points; the model therefore has four inputs. Two parameters are expected as a result of the model, the azimuth and altitude corrections.

The general structure of the regression neural network used for TIM is shown in Table 1. This structure is the result of our experiments and was created based on the above recommendations. Each layer has a type, number of neurons (shape), and activation function.

The first layer is used as input and has four neurons. It is possible to raise the number of inputs in the case of velocities or other parameters being required as inputs. The activation function is not required for this layer.

The last layer has two neurons as a number of model outputs. The other layers are dense and hidden. The number of these layers and shapes was sampled at the base of recommendations from the previous section.

Nonlinearities are introduced in the model through activation functions for each layer. We used activation through:

- **ReLU (Rectified Linear Units)**: $\varphi(z) = max(0, z)$. It is recommended for use in hidden layers to avoid overfitting and as simple to implement.
- **Tanh**: $\varphi(z) = \tanh(z)$. Hyperbolic tangent. Very useful as the first activator.
- **Sigmoid**: $\varphi(x) = 1/(1 + e^{-x})$. Sigmoid function, activator of the last layer to build the final result. This is why scaling of the input data to $[0..1]$ interval and then unscaling the final result is necessary. Mean values are also to be subtracted from the input data.

The final decision on which activation function to choose is a creative decision and must be made as a result of experiments.

Keras library contains all necessary tools for layers, activation, and cost function creation.

Table 1. General structure of the TIM network, each row is a layer. Total number of trainable parameters is 19,762.

Layer type	Number of neurons	Number of parameters	Activation
InputLayer	4	0	None
Dense	64	320	tanh
Dense	64	4160	relu
Dense	64	4160	relu
Dense	64	4160	relu
Dense	32	2080	relu
Dense	32	1056	relu
Dense	32	1056	relu
Dense	32	1056	relu
Dense	16	528	relu
Dense	16	272	relu
Dense	16	272	relu
Dense	16	272	relu
Dense	8	136	relu
Dense	8	72	relu
Dense	8	72	relu
Dense	8	72	relu
Dense	2	18	sigmoid

For cost functions, we used standard deviation and as optimizer, and used the Nadam algorithm in the default configuration from the Keras library.

It is worth noting that we need a criterion for when to stop the teaching iterations. A quite simple and even trivial criterion is to limit the number of iterations, but we cannot be certain that the best result has been found. The better criterion is testing if the result at the end of the current iteration has a lower deviation than the one from the previous iteration.

Keras library has all the necessary functionalities to do this: at the end of the iteration, we can estimate the quality of the result and stop teaching if a criterion has been satisfied. The default cost function is the standard deviation of real values and the ones calculated by the model for azimuth and altitude separately. Training stops if the cost function fails to exceed a threshold value for a given amount of consecutive iterations.

Good practice is to use a portion of input data for testing. These data should not be used for training and are only intended for testing and quality estimation. When preparing TIM, every tenth point of the input set was left for testing. So, for 13,000 points of input data, 1,300 were used for testing and quality estimation. A very useful criterion for iteration is a comparison of cost function for test and training data. If these metrics diverge from the iterations, the model overfits and its parameter should be changed.

It is worth noting that we need a criterion for when to stop the teaching iterations. A quite simple and even trivial criterion is to limit the number of iterations, but we cannot be assured that we found the best result. The better criterion is testing if the result at the end of the current iteration has a lower deviation than the one from the previous iteration.

Keras library has all the necessary functionalities to do it: at the end of the iteration, we can estimate the quality of the result and stop teaching if a criterion is satisfied. The default lost function is the standard deviation of real values and the ones calculated by the model for azimuth and altitude separately. Training stops if for some amount of consecutive iterations the cost function does not exceed some threshold values.

Good practice is to use a portion of input data for testing. These data should not be used for a model training. They are intended only for testing and quality estimation. When preparing TIM, every tenth point of the input set was left for testing, so from 13,000 points of input data, there were 1300 used for testing and quality estimation. The very useful criterion for iteration is a comparison of cost function for test and training data. If these metrics diverge with the iterations, the model overfits and its parameter should be changed.

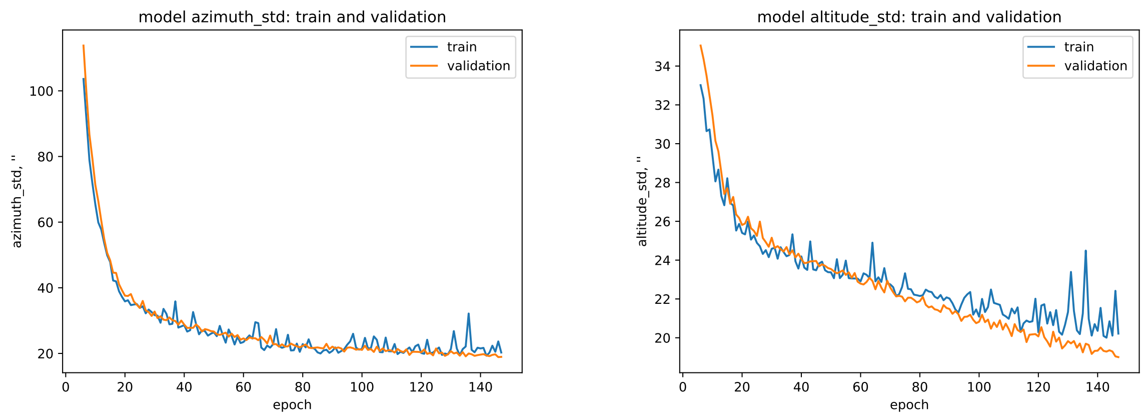

Figure 2. Cost function (standard deviation) for corrections to azimuth (left) and altitude (right) depending on learning iteration number (epoch)

In Figure 2, cost function depending on the iteration number is shown. The blue line connects the points for training data, and the orange ones show test data. No overfitting was detected in this case as both lines behave similarly.

For our station (Golisiiv 1824 in Kyiv), precision in pointing the telescope is limited by mechanical issues and laser beam width (10 arcseconds in our case). It sets the lower limit for model precision – 10 arcsec for each angle. We were very lucky and our model sometimes shows us much better precision in the sequence of iterations. The neural network from Table 1 achieves the necessary precision in less than 1,500 training iterations. It requires nearly 30 minutes on a usual laptop without CUDA. However, practical usage of the model led to high frequency fluctuations in the telescope guiding system. This indicated some overteaching, which is bad for precise pointing. That is why we added one empirical criterion and stopped teaching if the precision fell to 20 arcseconds. High-frequency fluctuations vanished but the precision of guiding left as well. We would lose the satellite for a very short time, but it could be easily found again because in this case, the corrections are very smooth and easily forecast by telescope mechanics. Only 150 iterations are necessary in this case (less than 10 minutes of machine time on a usual laptop).

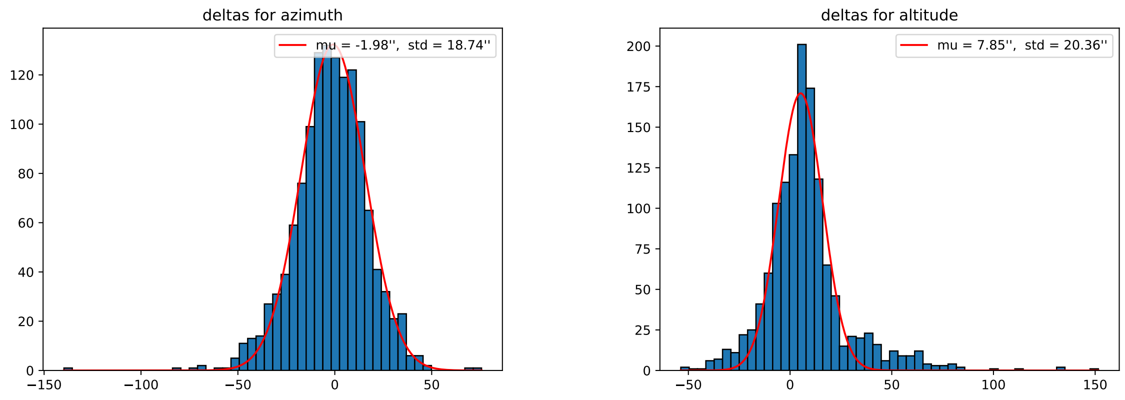

Figure 3. Residual distribution of modeled and original corrections

In Figure 3, histograms of differences between original and modeled corrections and the best fit of the normal distribution to them are shown. The histograms were built on the test data, so the standard deviations are not precisely 20 arcsec. The altitude histogram is slightly wider than the one for azimuth. We need to devote time considering the planetary aberration for satellites in our next steps. If the normality of the residuals is not fully satisfied, it means that there is some other error not being taken into account, refraction inconsistencies, for example. The practice of the model usage allowed us to conduct more observations of the satellites, especially those observed in a blind mode. International Laser Ranging Service statistics for station 1824 and Lageos and Lares show a sharp rise from 201 observations in 2021 and 46 in 2022 (station reconstruction) to 4327 in 2023 (`https://ilrs.gsfc.nasa.gov/network/system_performance/global_report_cards/monthly/`).

At the latest step, we save the model on an external HDD in the format generally used for works with neural models. The model is easily downloadable into any other code assigned to use the model. No dependency on programing language. Before observation, the corrections are calculated with the model, the ephemeris values are corrected, and those new values are in turn used for telescope pointing. The correction calculation requires less than 10 seconds for one ephemeride (300 — 1000 pairs of coordinates).

4.2. Conclusions

The neural network model is very practical and useful for SLR observations. The telescope steering system works with improved precision. This results in better pointing and a growing number of successful locations. Now, we can observe the satellites in a blind mode when the satellite is invisible on the operator console and manual correction is not possible.

The time for learning is the main time consumer of the neural models in general. This is the main area where the model needs polishing, perhaps by adding CUDA support. But that 30 min for model training on a usual laptop is not critical in our case.

Another problem with neural models is the absence of general recommendations on model structure, activation functions, optimization methods, goodness metrics, etc. It took serious time to find the optimal configuration by experimenting with the Keras user interface. This part of the work looks more like art than science.

The cost function has many dimensions, and sometimes, the optimization runs into a local minimum. Different solutions are generated in these cases. Sometimes, we cannot reproduce the same final numbers during two successive training experiments. But in any case, the statistical values, standard deviation, for example, are within 10 arcsec and are very similar.

We did not analyze the influence of source point distribution on the quality of the result. This is one of our next steps. Also, we did not determine the minimum amount of training data to have a good result. One can only note that 10k points are quite enough in our case.

Having some expertise in the usage of this type of telescope-pointing accuracy model, we recommend neural models for investigating the accuracy of your devices.

REFERENCES

E. Butkiewicz, S. Schillak, J.K. Latka (1994) Mount error model of BOROWIEC-2 SLR system *Artificial Satellites*, Vol.29., No 3., 119 - 128.

F.Chollet and team. (2015) Keras, *https://keras.io*.

G.Cybenko (1989) Approximation by Superpositions of a sigmoidal function, *Mathematics of Control, Signals and Systems*, V.2(4), 303-314.

M.Medvedsky, V.Suberlak (2002) Mount errors model for the Kyiv SLR station, *Artificial Satellites, Journal of Planetary Geodesy*, Vol. 37, No. 1, 3-16.

V.Zhaborovskyy, V.Choliy, M.Medvedskyy, V.Pap (2013) Telescope inaccuracy model based upon satellite laser ranging data, *Advances in Astronomy and Space Physics*, Vol.3, N.1, 63-65, 2013.

Received: *2024-02-07*

Reviewed: *2024-04-23 (undisclosed name); 2024-05-05 (K. Salmins);*

2024-05-14 (undisclosed name)

Accepted: *2024-06-17*