RBE/CS-549: Project 4 - Deep VIO

Dhrumil Sandeep Kotadia, Dhiraj Kumar Rouniyar, Krunal M. Bhatt

***Abstract*—During this phase, we investigated the application of deep learning techniques to address Visual Inertial Odometry (VIO). Initially, we generated the dataset using Blender. Subsequently, we employed this processed data to train separate networks for visual-only odometry and inertial-only odometry. Additionally, we proposed a network architecture designed to integrate both visual and inertial signals for odometry estimation. However, due to time constraints and unresolved issues within our visual-only and inertial-only networks, we were unable to train the VIO network as planned. Furthermore, we assessed the accuracy of our odometry predictions against ground truth data using the rpg trajectory evaluation toolbox, following a similar methodology as in Phase 1.**

I. Introduction

Traditional VIO implemented previously, depends largely on feature detection and tracking. It becomes difficult to identify similar set of features for inertial data. Pose tracking based on camera images has been shown to be sensitive to motion blur, occlusions, and lighting changes. Thus, a lot of work has been conducted over the last years on visual-inertial pose tracking using acceleration and angular velocity measurements from inertial sensor such as IMU, in order to improve the visual tracking.

A long short-term memory model (LSTM) model is used to provide an estimate of the current pose based on previous poses and inertial measurement, combined with output of visual tracking using Linear Kalman Filter gives a robust estimate [1]. Another approach extracts the image features using CNN and projects them to a visual manifold. Temporal features are extracted from IMU data on the platform using BiLSTM network [2]. This HVIO approach can be implemented to estimate a UAS position in real time.

Furthermore, replacing the feature matching with SuperPoint or SuperGlue models would work, but a similar set of features are hard to conceptualize for inertial data [3], [4]. Inertial navigation has some great work but a very few of them work with VI-fusion.

II. Dataset Generation

For this approach we have generated our own dataset using Blender. We use a down-facing camera and an 6-DOF IMU on an aerial robot looking at a planar surface. Data for such scenes is not readily available. Hence, we estimate the relative pose using the data generated from blender with such a setup. We also assume that the biases from the neural network are zero. The roll and pitch angles of the camera do not exceed 45°. Here, while generating the dataset, we ignore the realistic effects such as motion blur, depth of field or lighting changes.

The authors are with the Robotics Engineering Department of Worcester Polytechnic Institute, Worcester, MA 01609 USA(email: dkotadia@wpi.edu; dkrouniyar@wpi.edu; kmbhatt@wpi.edu)

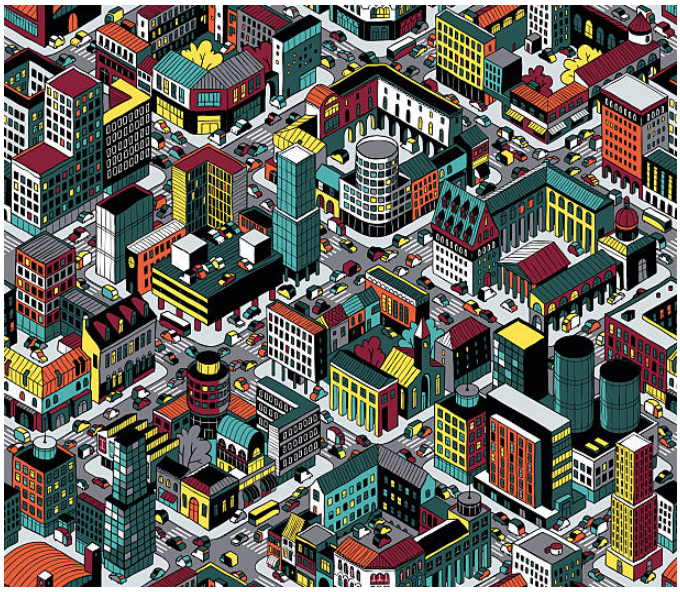

Fig. 1: Texture on Plane

The dataset was generated with a large textured plane resting at Z=0. The texture is shown in Fig. 1. We have a camera facing the plane and an IMU attached on the camera. We assume there is no relative R and T between the two. Now, given the trajectory data, there are several approaches to produce IMU data. One such technique involves using the data to compute the trajectory's first and second order derivatives, after which noise is injected (as demonstrated in [5]). An alternate technique is to use the IMU model in MATLAB, which gives acceleration, angular velocity, and orientation data in exchange for gyroscope and accelerometer inputs. The former of these two was utilized to produce the required IMU data. Moreover, for IMU data stream, only frames every 10^{th} are taken into account. Thus, 2000 photos and 20,000 IMU readings were employed for the model training. Below is an example scene where the camera moves in Blender along the suggested trajectory while collecting image data.

III. Deep VIO

This section contains 3 models which we will be discussing. They are VO Resnet, Drone LSTM and VIO. We have used a Quaternion loss function for angle and a MSE loss function for the translation errors. The loss functions are described as follows:

Translation Loss:

$$\text{Translation\_loss} = \frac{1}{3}\sum_{i=1}^{3}(o_i - l_i)^2$$

Rotational Loss:

$$\text{Rotational\_Loss} = \|\mathbf{Q}_1 \cdot \text{pinv}(\mathbf{Q}_2)\|$$

Total Loss:

$$\text{Total Loss} = \text{Translation\_loss} + \text{Rotational\_Loss}$$

The loss function described is composed of two distinct components designed to effectively train a model handling both translation and rotational predictions. The translation loss is calculated as the mean squared error between the predicted and actual translation vectors, focusing on the direct differences in positional elements. For the rotational aspect, the loss is computed using the norm of the quaternion product between the predicted rotation quaternion and the pseudo-inverse of the true rotation quaternion, which quantifies the discrepancy in orientation by assessing the rotational alignment. This combined loss function, by summing the translation and rotational losses, ensures a comprehensive assessment of both position and orientation errors, making it particularly suitable for applications in robotics and navigation where accurate spatial and angular estimations are critical.

A. VO Net

The VO_ResNet model is a sophisticated convolutional neural network tailored for visual odometry, leveraging a combination of convolutional layers, residual blocks, and fully connected layers to process and integrate diverse data inputs efficiently. It starts with a convolutional layer followed by batch normalization to extract preliminary features, which are further refined through strategically layered residual blocks to enhance the network's depth and learning capability without encountering the vanishing gradient problem. The architecture incorporates adaptive pooling and a series of fully connected layers that not only process the visual data but also integrate additional sensory inputs through concatenation, allowing for nuanced feature integration. This model uses dropout for regularization and ReLU activations to maintain non-linearity, supporting its robustness in learning complex patterns necessary for precise visual odometry tasks. This structured approach ensures the network is capable of handling the intricate requirements of visual processing applications, making it ideal for deployment in dynamic environments where precise motion detection and navigation are crucial. Fig. 2 below shows us the architecture of the described model.

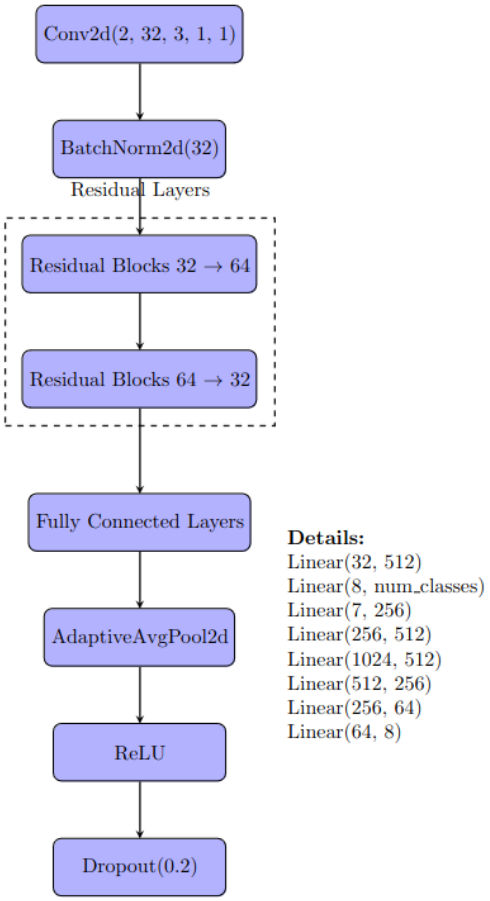

Fig. 2: VO ResNet

B. IO Net

The DroneLSTM model is an architecture specifically designed to address the complex requirements of drone navigation and control, using recurrent neural network principles. Constructed using the PyTorch library, this model incorporates an LSTM (Long Short-Term Memory) layer that processes sequential data, making it ideal for time-series inputs like those encountered in drone flight dynamics. The model begins with an LSTM layer configured to handle input sequences with a designated size and transform these through multiple recurrent layers, encapsulating the ability to remember patterns over time. This is crucial for maintaining stateful information across the drone's flight trajectory.

Following the LSTM layer, the architecture features several fully connected (dense) layers aimed at refining the LSTM outputs to specific tasks: one layer outputs the drone's positional data, while another focuses on its orientation, which are critical components for precise navigation and control. The outputs from these layers are then merged, forming a comprehensive output vector that includes both pose and orientation data. This setup ensures that the model not only captures complex dependencies within the input data but also translates these into actionable insights in real-time, making it highly effective for applications requiring dynamic response and acute accuracy, such as autonomous drone flight and real-time control systems. Fig. 3 shows the architecture of the model.

C. VIO Net

Here, Fig. 4 is the final architecture used in the implementation of the network.

Fig. 5 to Fig. 11 shows the trajectories we have used to train the network. Trajectories have been generated using the

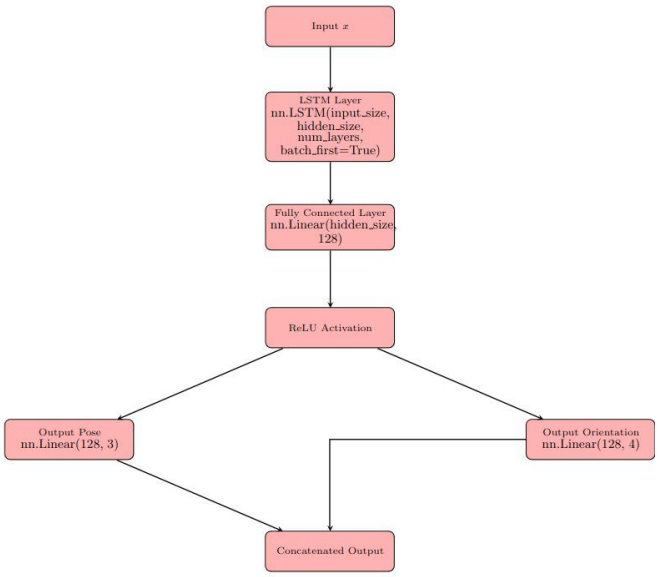

Fig. 3: LSTM network

method described in [6].

IV. Results

rotation loss plot, translation loss plot, training loss plot Fig. 12 shows the trajectory we used to test the network. We get a output similar to Fig. 11 as shown in the Fig. 13. We can observe that the output trajectory thus estimated is similar to the one in the test case.

We can see the individual output for each of the 3 models as follows:

VO Output Fig. 19, Fig. 17 shows the plot for output.
IO Output Fig. 26, Fig. 28 shows the plot for output. For the test set, we get the following outputs:

VIO Output The following output is of the VIO network. The trajectory is not followed and the performance leaves a lot to be desired.

V. Future scope

In the future scope of this work, we can improve the network by adding attention mechanism, using multi-modal fusion and designing a custom loss heuristic for the same. We can also improve on the data generation method and create a standard way to sync the time frame of for the same. Correct hyper-parameter tuning will help in getting better output.

References

[1] J. R. Rambach, A. Tewari, A. Pagani, and D. Stricker, "Learning to fuse: A deep learning approach to visual-inertial camera pose estimation," in *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2016, pp. 71–76.

[2] A. Y. A. Y. Muhammet Fatih Aslan, Akif Durdu, "Hvionet: A deep learning based hybrid visual–inertial odometry approach for unmanned aerial system position estimation," vol. 155, 2022, pp. 461–474.

[3] D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superpoint: Self-supervised interest point detection and description," 2018.

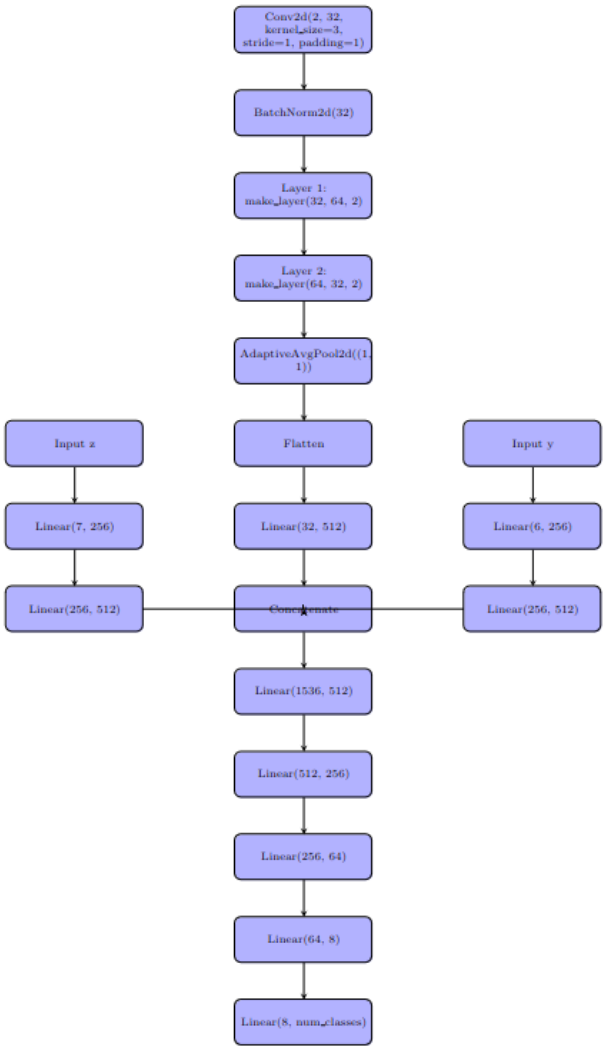

Fig. 4: VIO Net

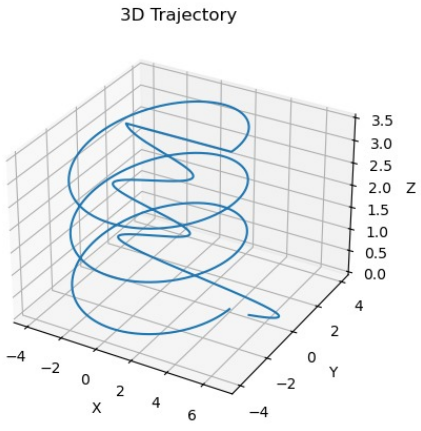

Fig. 5: Training Trajectory 1

[4] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue:

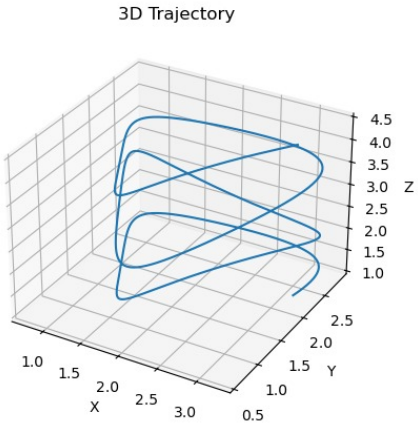

Fig. 6: Training Trajectory 2

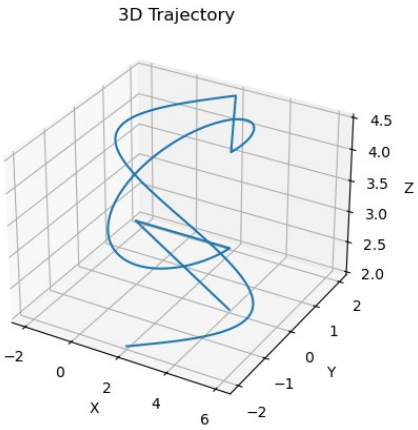

Fig. 7: Training Trajectory 3

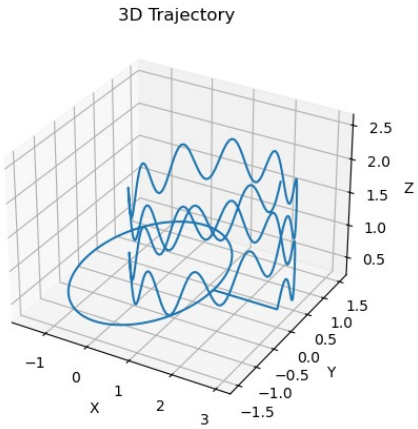

Fig. 8: Training Trajectory 4

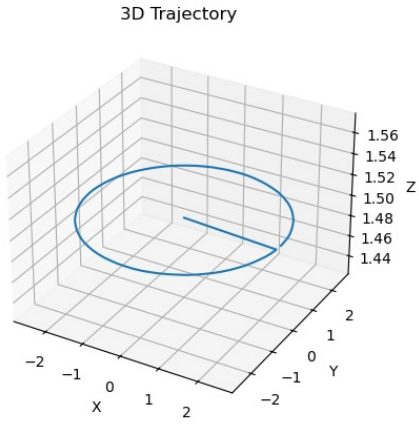

Fig. 9: Training Trajectory 5

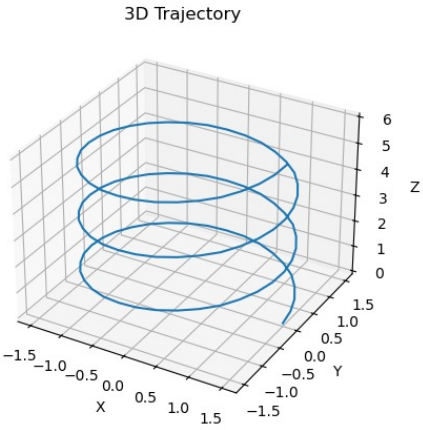

Fig. 10: Training Trajectory 6

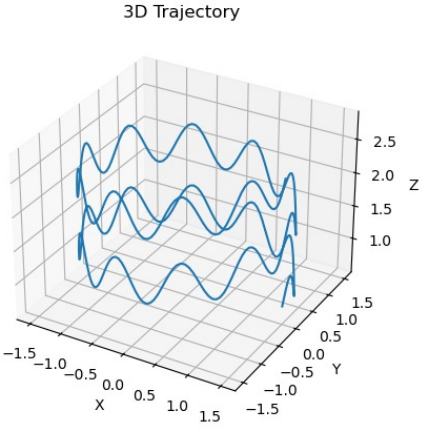

Fig. 11: Training Trajectory 7

Learning feature matching with graph neural networks," 2020.
[5] "prgumd/oystersim," GitHub, 10 2023. [Online]. Available: https://github.com/prgumd/Oystersim
[6] [Online]. Available: https://rbe549.github.io/rbe595/fall2023/proj/p0/

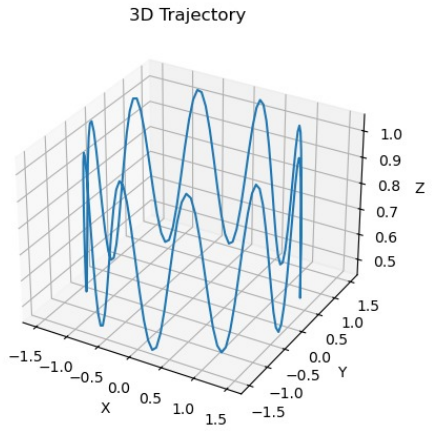

Fig. 12: Test Trajectory

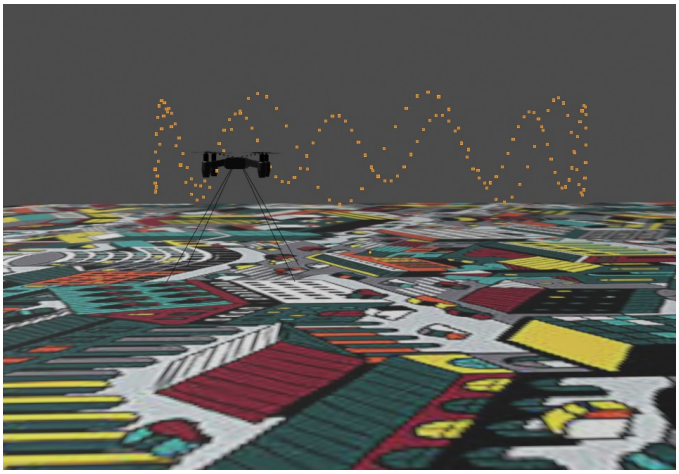

Fig. 13: Test Trajectory output

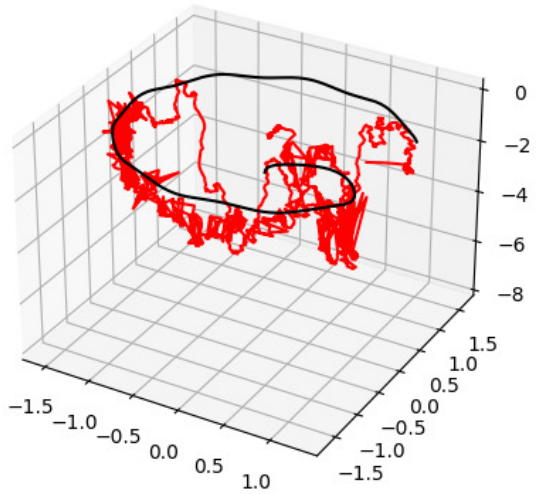

Fig. 14: Trainied Model overfitting on the train data

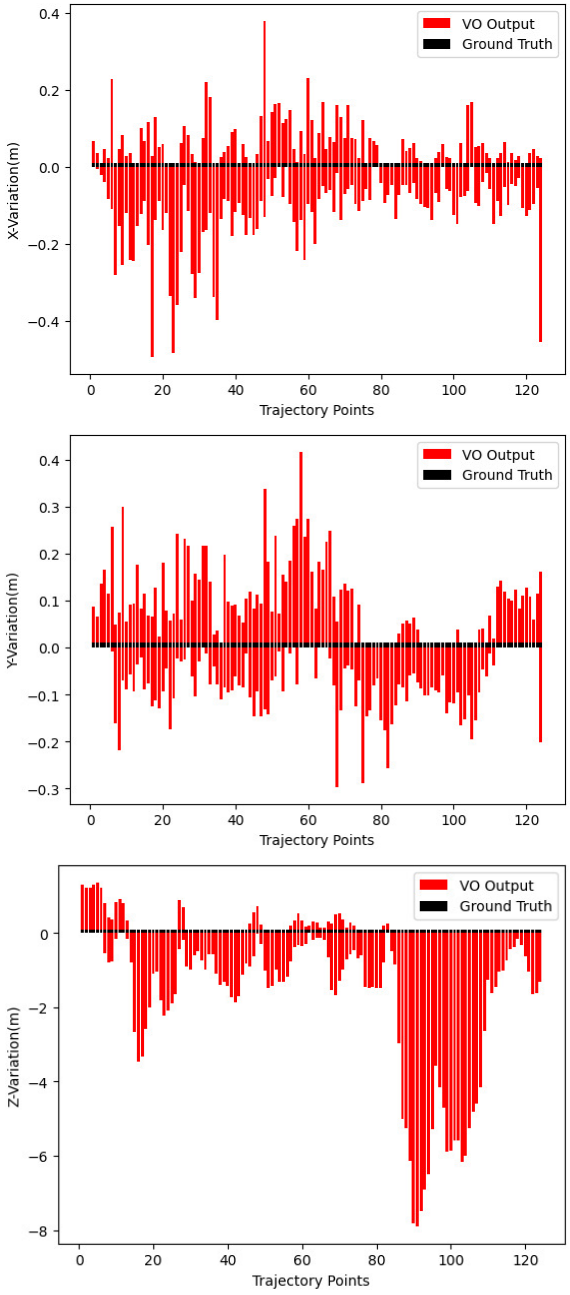

Fig. 15: X,Y and Z VO Output overfitting for test trajectory

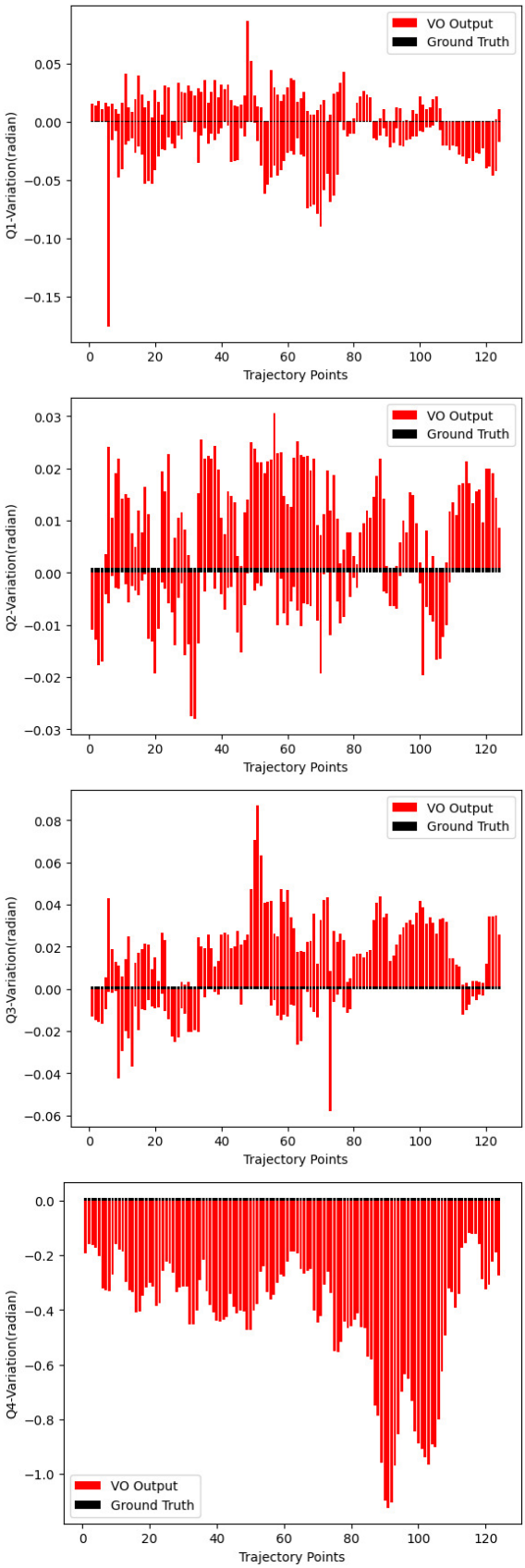

Fig. 16: Quaternion VO Output compared with the Ground Truth

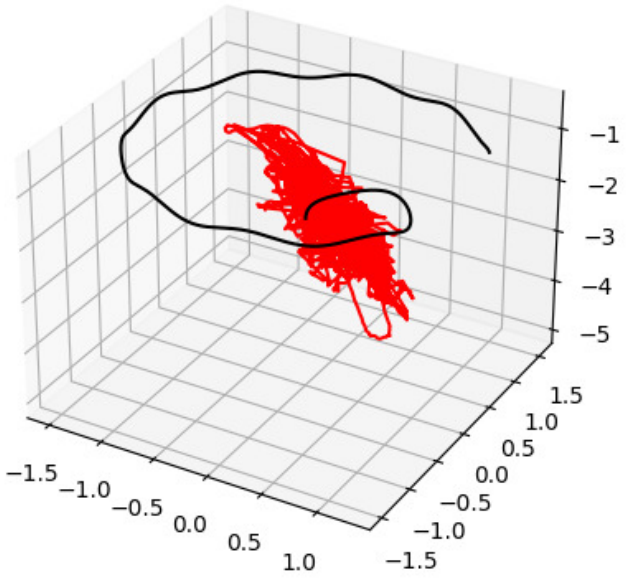

Fig. 17: VO Output for test trajectory

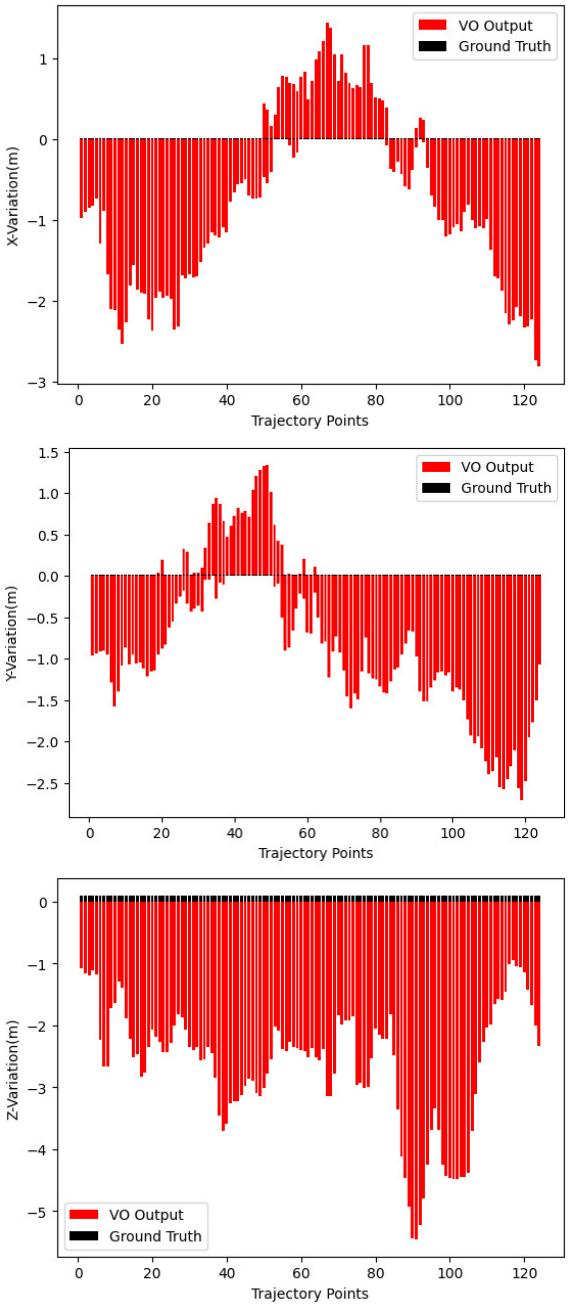

Fig. 18: X,Y and Z for VO Output for test trajectory

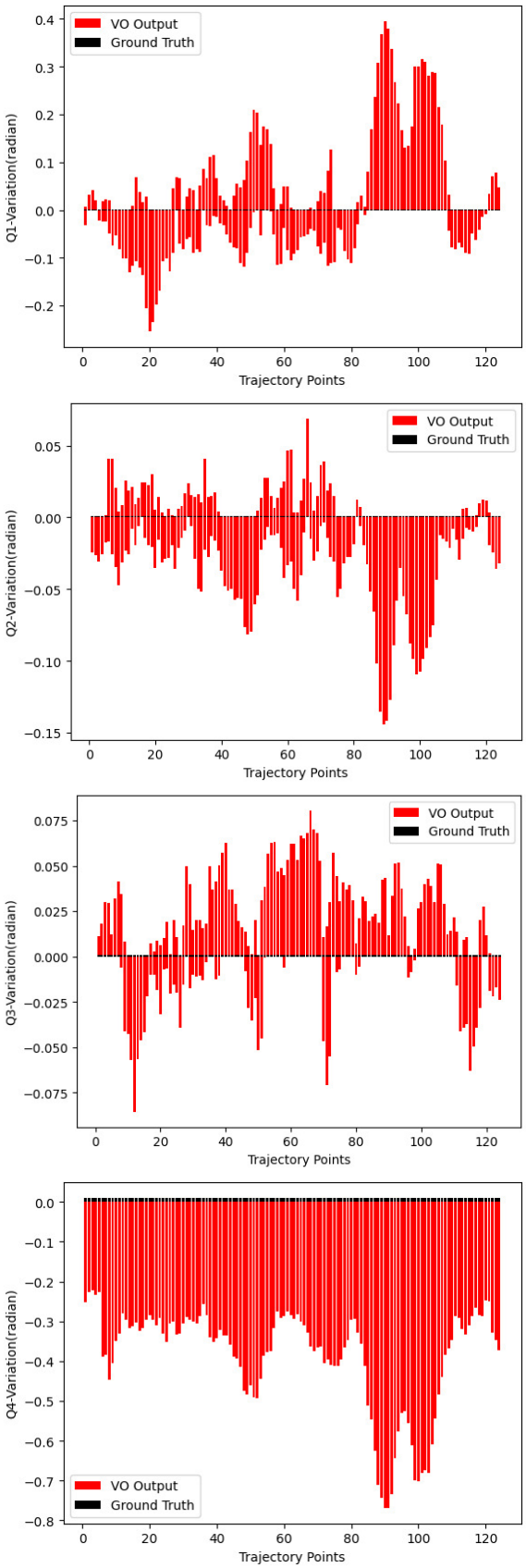

Fig. 19: Quaternion VO Output compared with the Ground Truth

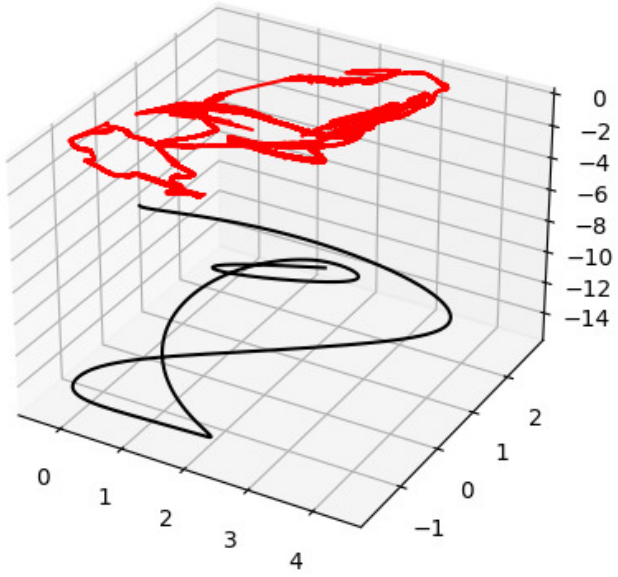

Fig. 20: IO output trajectory overfitting on train data

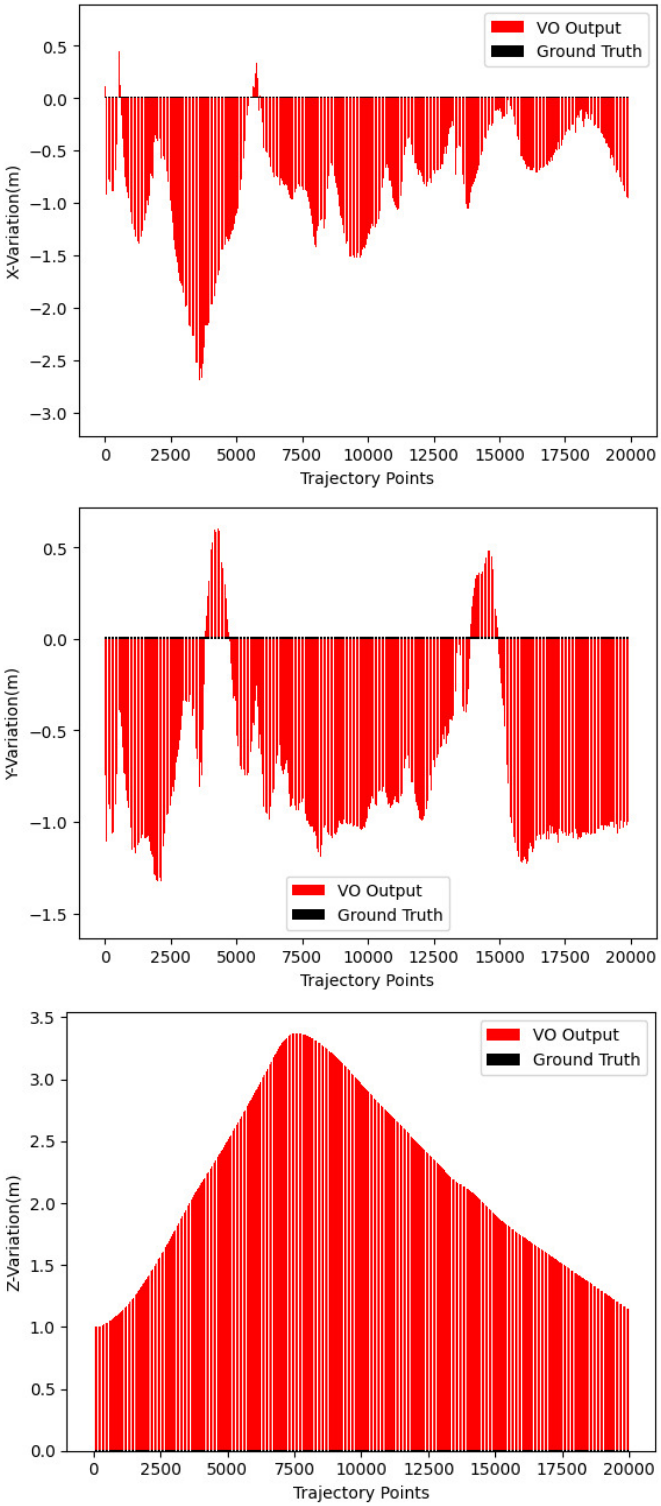

Fig. 21: Variation in X, Y and Z for IO Network

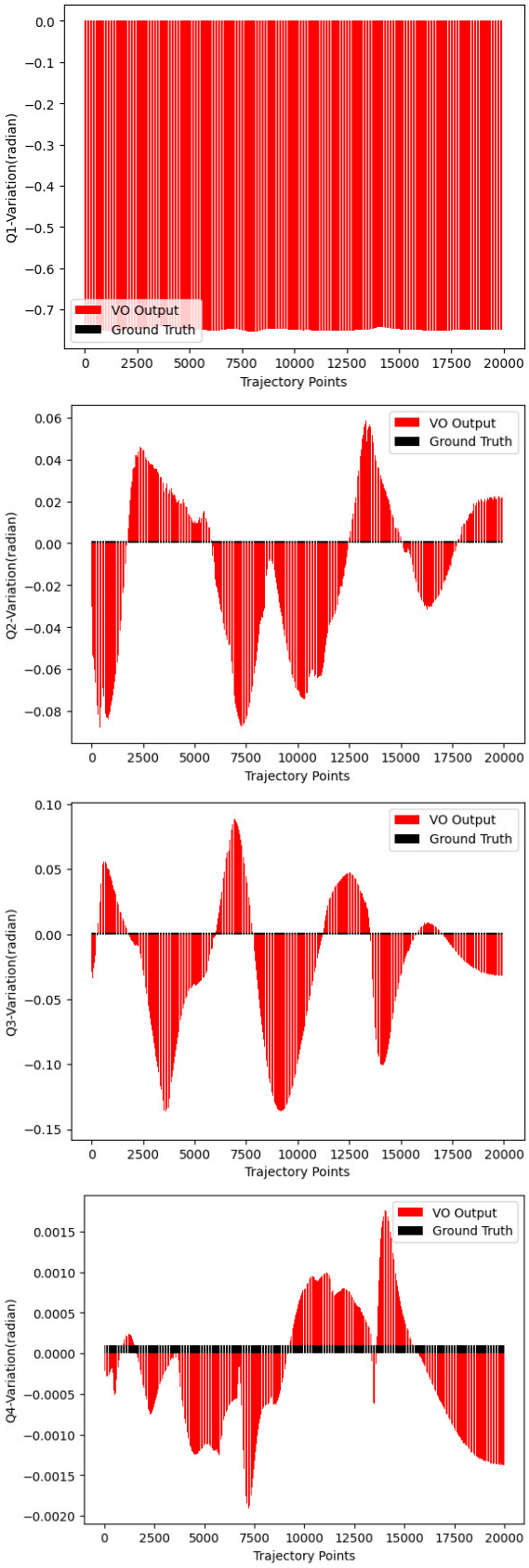

Fig. 22: Variation in Quaternions for IO Network

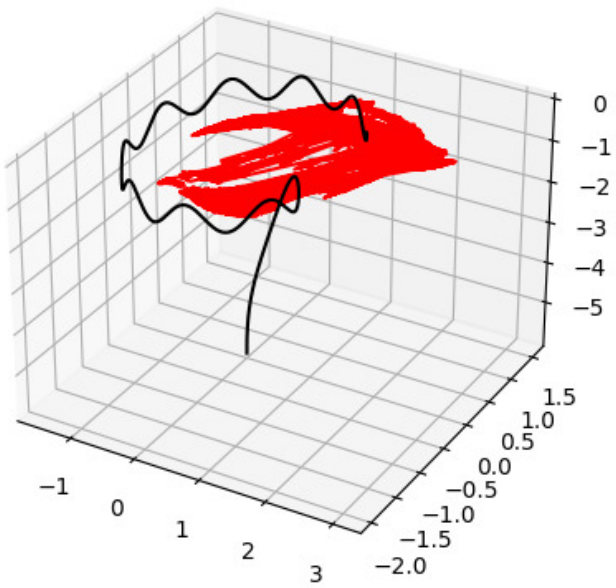

Fig. 23: IO output trajectory

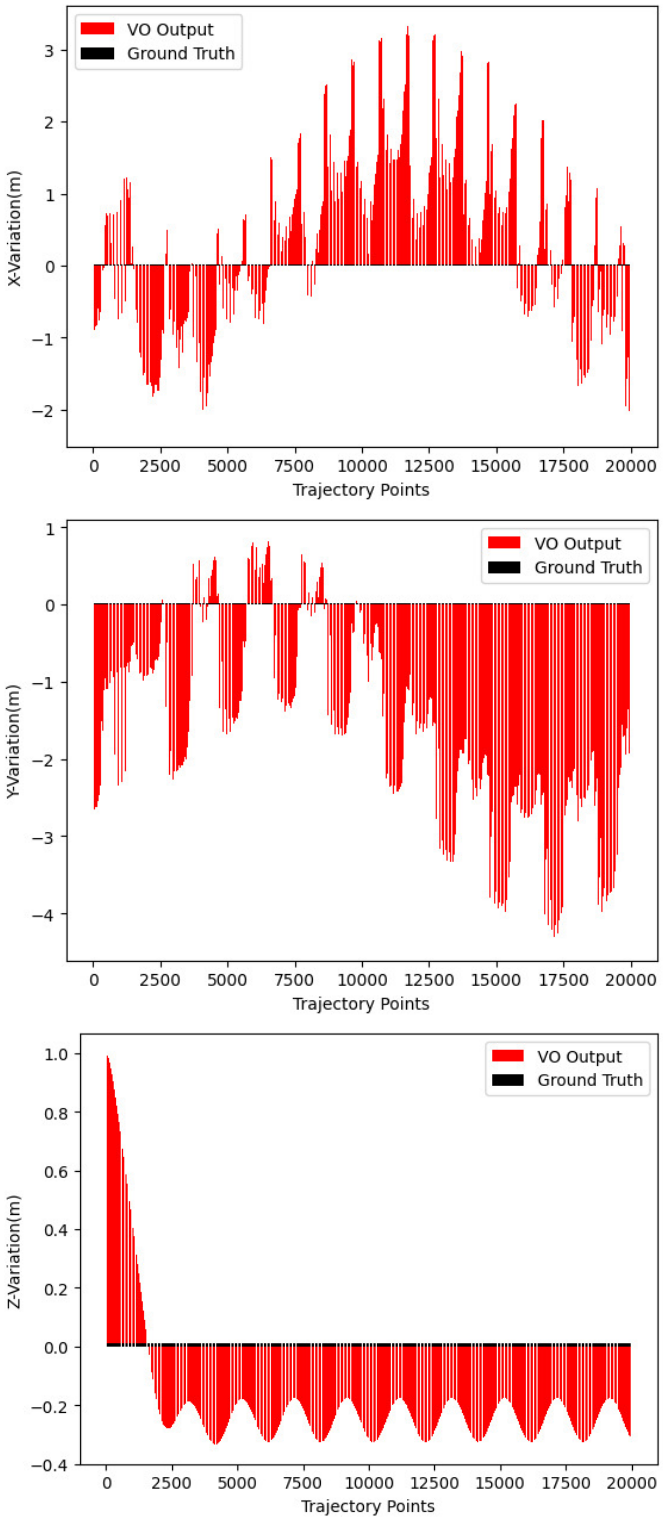

Fig. 24: Variation in X, Y and Z for IO Network

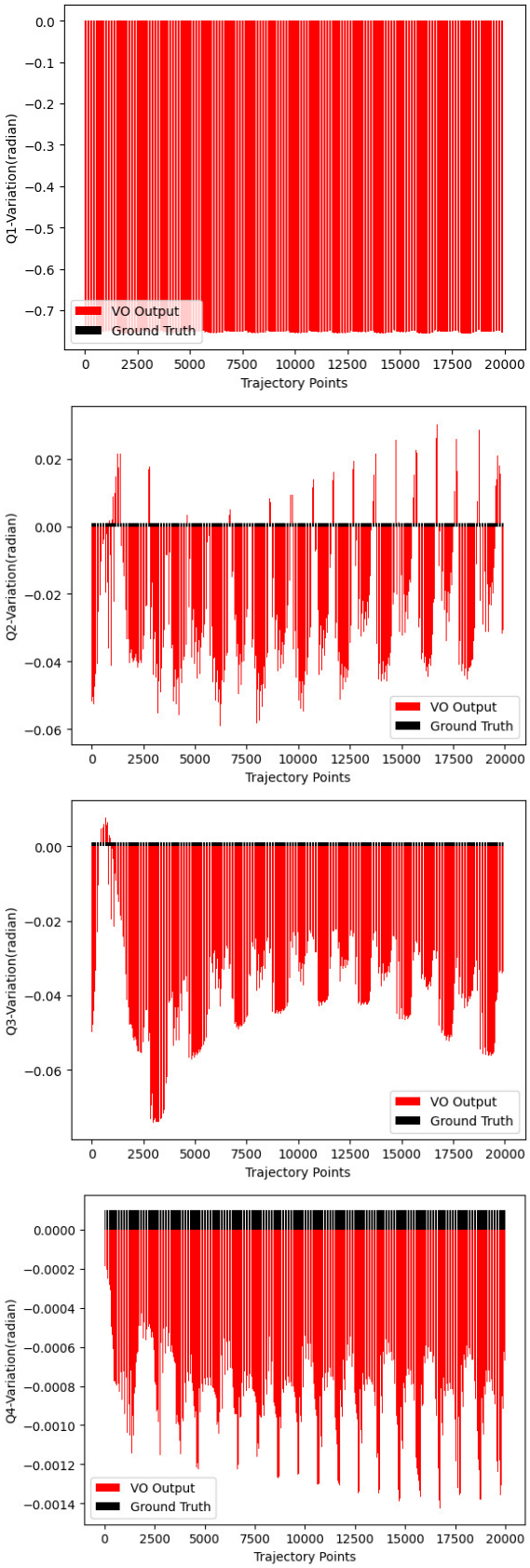

Fig. 25: Variation in Quaternions for IO Network

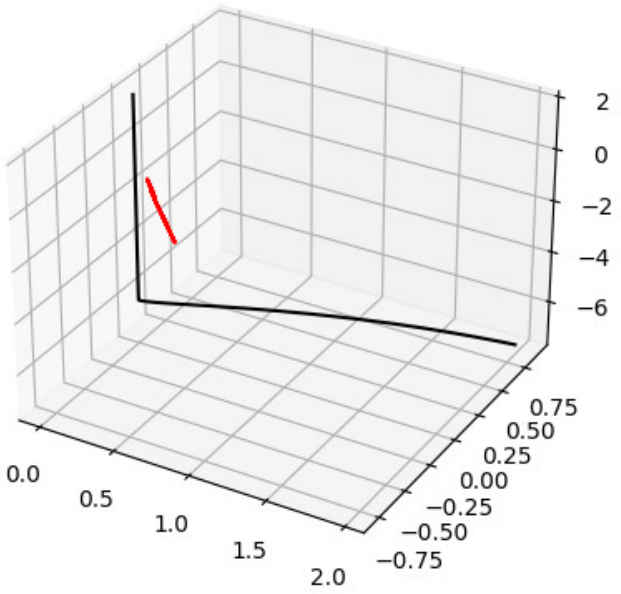

Fig. 26: VIO output trajectory

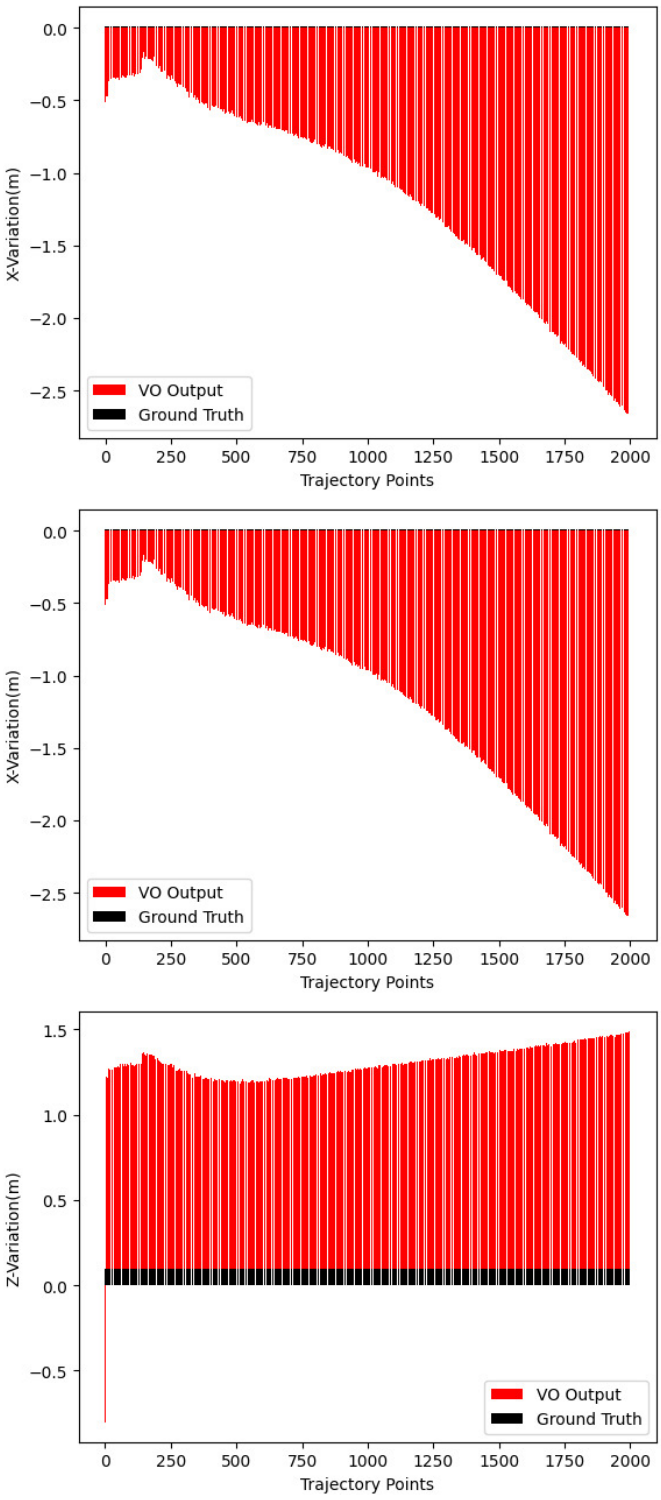

Fig. 27: Variation in X, Y and Z for VIO Network

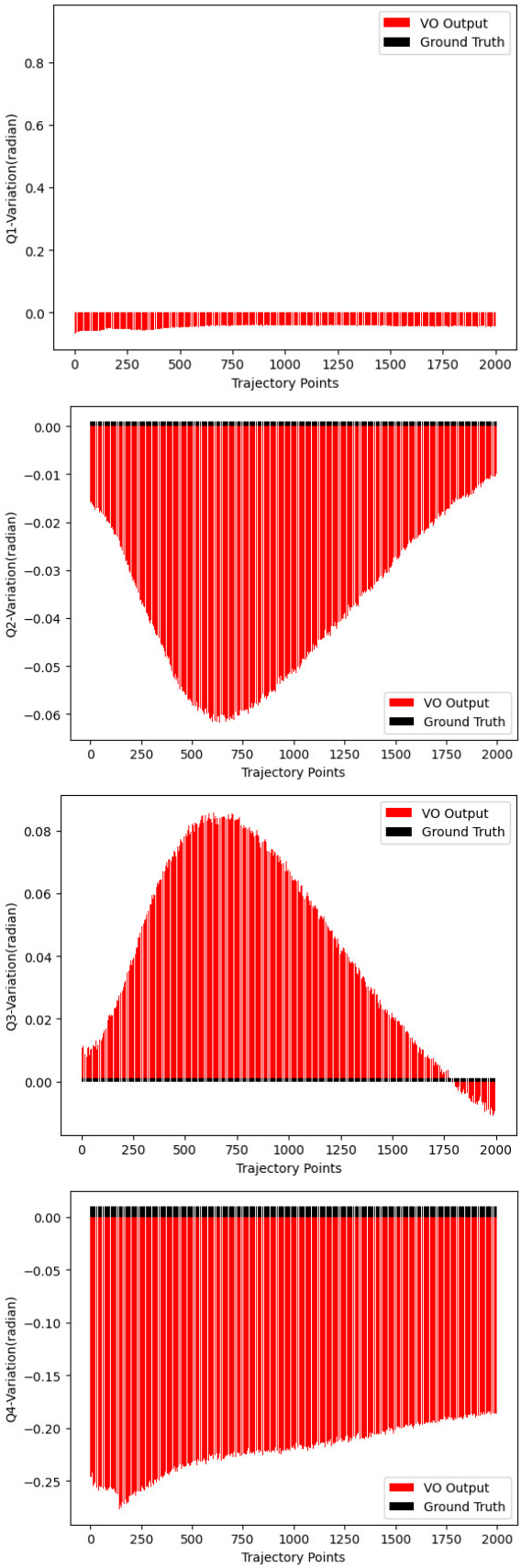

Fig. 28: Variation in Quaternions for VIO Network