RBE549 Project 4 - Deep and Un-Deep Visual Inertial Odometry Phase 2

Yaşar İdikut
yidikut@wpi.edu

Harshal Bhat
hbhat@wpi.edu

***Abstract* — This report presents an end-to-end trainable framework for Visual Inertial Odometry (VIO) using deep learning to address the limitations of traditional VIO systems. Leveraging a custom Blender-generated dataset that simulates real-world scenarios, our model integrates deep Recurrent Convolutional Neural Networks (RCNNs) for monocular visual odometry and a Long Short-Term Memory (LSTM) network for processing inertial measurements. The visual network directly infers poses from sequences of raw RGB images, bypassing the conventional feature extraction and matching processes, while the inertial pathway enhances pose estimation with adaptive IMU data integration. We levergae existing models for transfer learning and optimize the state-of-the-art methods. Our experiments demonstrate that this approach shows better adaptability to various motion dynamics. The implementation details and the comprehensive evaluation on the Blender dataset highlight our method's potential in improving both the efficiency and robustness of VIO systems.**

I. Introduction

This research addresses the challenge of enhancing the accuracy and robustness of Visual Inertial Odometry (VIO) by integrating deep learning techniques. Traditional methods, reliant on feature detection and tracking, often falter in dynamic environments. Our study aims to:

1) Develop an end-to-end deep learning architectures that leverages both visual and inertial data for real-time pose estimation, reducing dependency on conventional feature extraction.
2) Utilize a synthetic dataset created in Blender, simulating realistic navigation scenarios with a down-facing camera and a 6-DoF IMU on an aerial robot, to train and validate our models.
3) Implement and assess three distinct models using:
    a) Only vision data from sequences of RGB frames.
    b) Only IMU data capturing 6-DoF measurements.
    c) Both vision and IMU data.
4) Evaluate the models on unseen test sequences, comparing their performance against ground truth trajectories derived from incremental poses.

This streamlined approach focuses on comparing the effectiveness of each data modality in isolation and in combination, aiming to advance VIO systems' capabilities in challenging environments.

II. Related work

Recent advancements in visual odometry (VO) leverage deep learning to enhance accuracy and computational efficiency. A novel end-to-end framework using deep Recurrent Convolutional Neural Networks (RCNNs) is presented in [1], which directly infers poses from raw RGB images, demonstrating competitive performance on the KITTI dataset. Similarly, [2] introduces DeepVIO, a self-supervised network that combines optical flow and IMU data, achieving notable improvements in trajectory estimation under challenging conditions. Additionally,[3] proposed an adaptive method that reduces computational redundancy by selectively disabling the visual modality, maintaining robust performance with significant complexity reduction.

We leverage networks for transfer learning and optimize them based on our motion dynamics on our custom dataset. Transfer learning from state-of-art-methods is meticulously achieved in this report.

III. Methodology

A. Architecture Details

Our architecture can be thought of as a modification of the network proposed in [3]. We remove the policy network as it was mostly proposed to improve the runtime considerations of the VIO network. We also take components of their network to create three separate networks to allow for odometry calculation given (1) only IMU data, (2) only camera data, and (3) both.

1) Intertial Only (IONet): The architecture for pose estimation combines fully connected linear layers (FC), CNN (one-dimensional) and RNN (LSTM) layers to process sequential data effectively. In our case, we set the IMU operating frequency to 100Hz. This input takes in the IMU measurements for the last 1.0 second (100 measurements). The IMU provides 6 DoF measurements (3 DoF for linear acceleration and another 3 DoF for angular velocity). Each of these 6 measurements are bundled into *subsequences* of size 10. Each of these subsequences go through the CNN layers followed by a fully connected layer that generates a feature vector encompassing the relative transformation information (translation and rotation). This CNN + FC portion of the IONet is the *inertial encoder* portion of the network (See

Figure 1). This encoder generates feature vectors for each given subsequence. These feature vectors are then sequentially passed through the LSTM layer. In total, 10 *sequences* are passed through the LSTM layer. The final hidden state of the LSTM layer then passed through another FC layer that generates transformation data as a tuple of linear and angular (euler) displacement. Hence, a single input to IONet is the last 100 IMU measurements. The expected output of the network is the relative transformation between the pose of the most recent measurement and the pose of IMU after a subsequence number of measurements (10 measuements, or 0.1 seconds). Therefore, this model is expected to take in the last 1 second of IMU measurements, and predict the relative transformation of the IMU in the next 0.1 seconds.

The network inputs feature vectors and uses an LSTM configuration with 1024 hidden units across two layers, incorporating a dropout rate of 0.2 to prevent overfitting. Outputs are refined through a series of linear transformations and LeakyReLU activations to produce the final pose estimates. Complementing this, a series of 1D convolutional layers process inertial data by expanding input channels progressively from 6 to 256, stabilized with batch normalization and dropout. These features are then used for pose prediction through the recurrent network.

IONet is trained from scratch using the simulated IMU readings. After training is complete, the encoder portion of the network is reused in VIONet.

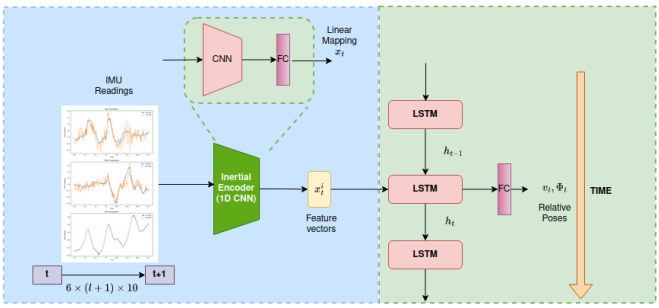

Fig. 1. IONet Architecture

2) Visual Only: VONet: Similar to IONet, the architecture of VONet for pose estimation combines fully connected linear layers, CNN (two-dimensional) and RNN (LSTM) layers. We set the camera operating frequency to 10Hz. We use 10 sequences of pair subsequent of images. Hence, this network also takes in the last 1 second's measurements and outputs the predicted relative transformation in the next 0.1 seconds. This modification is done by changing the *visual encoder* portion of the network (See Figure 2).

CNN portion of the network is initialized with the weights of FlowNet. The rest of the network is trained from scratch. LeakyReLU activation functions are employed to introduce non-linearity, helping the network learn more complex patterns. Batch normalization is applied at each layer to stabilize learning and accelerate the convergence of the network. These components, combined with dropout of 0.2, ensure robust performance by preventing overfitting, enhancing the model's ability to generalize across diverse scenarios.

VONet's encoder portion consists of CNN layers that are initialized to the FlowNet weights, fully connected layers that are trained from scratch. VONet is trained on the Blender generated data. After training is complete, the encoder portion of the network is reused in VIONet.

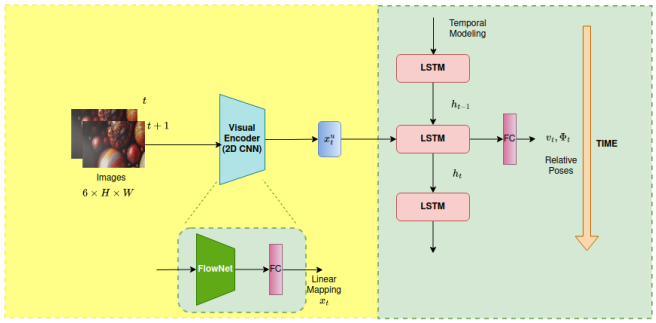

Fig. 2. VONet Architecture

3) Visual and Inertial Network: VIONet: To reduce training time and re-usability, we use the *inertial encoder* portion of the network from IONet (See Figure 1), and the *visual encoder* portion of the network from VONet (See Figure 2). Feature vectors generated from these encoders are concatenated before passing through the LSTM + FC layers. In this network, the encoders are frozen, i.e., their parameters aren't updated by the training loop. Only the LSTM + FC layers are trained from scratch (See Figure 3).

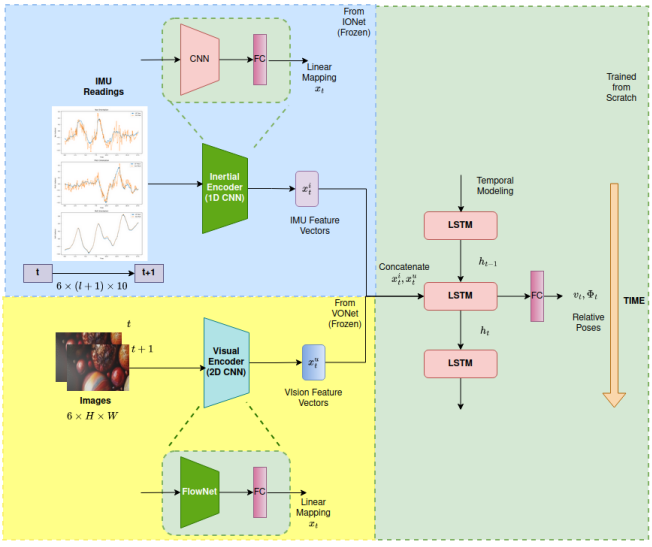

Fig. 3. VIONet Architecture

B. Loss Function

The model outputs the transformation prediction as a linear and angular (euler) displacement for each next 0.1s (*subsequence*) of the input. The ground truth is calculated by

summing the displacement through the ground truth poses for the IMU measurements over increments of 10 measurements (*subsequences*).

Initially, we experimented with cosine embedding loss for a custom CNN-based network for inertial and visual data. However, the lack of time-series tracking pushed us towards changing the architecture. After the change, we simply used the RMSE loss. It worked fine for simple cases as can be seen in Figures IV-B2 and IV-B2.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

Inputs being model's predicted pose difference and data ground truth pose difference between the next *subsequence* time. The total loss needs to be minimized for good trajectory tracking.

IV. Experimental Setup

A. Dataset

We initially used [8] to generate relative linear acceleration, angular acceleration, position and roll, pitch and yaw. However, there were issues like mapping timestamps to blender to sample images, unclear trajectory generation commands, etc. We switched to [9] which overcomes these issues. This generates the global poses in NED(North East Down) frame of reference. We defined waypoints using parametric equations for several curves. This gives us positions, velocities, quaternions and angular velocitites. We modified it to get relative poses and acceleration data as csv files. We generated 15 different trajectories of varying complexity and maneuverability as shown in Figures IV-A and 5.

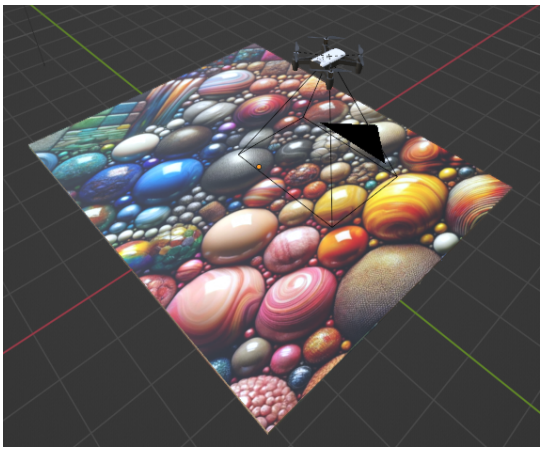

Fig. 4. Screenshot the Blender setup. Texture generated from DALL.E3[7]

B. Experiments

We use the Adam optimizer with a learning rate of 0.00001 for all the networks presented. We use a batch size of 16. We trained the models until we stopped them after we observed convergence of the test loss.

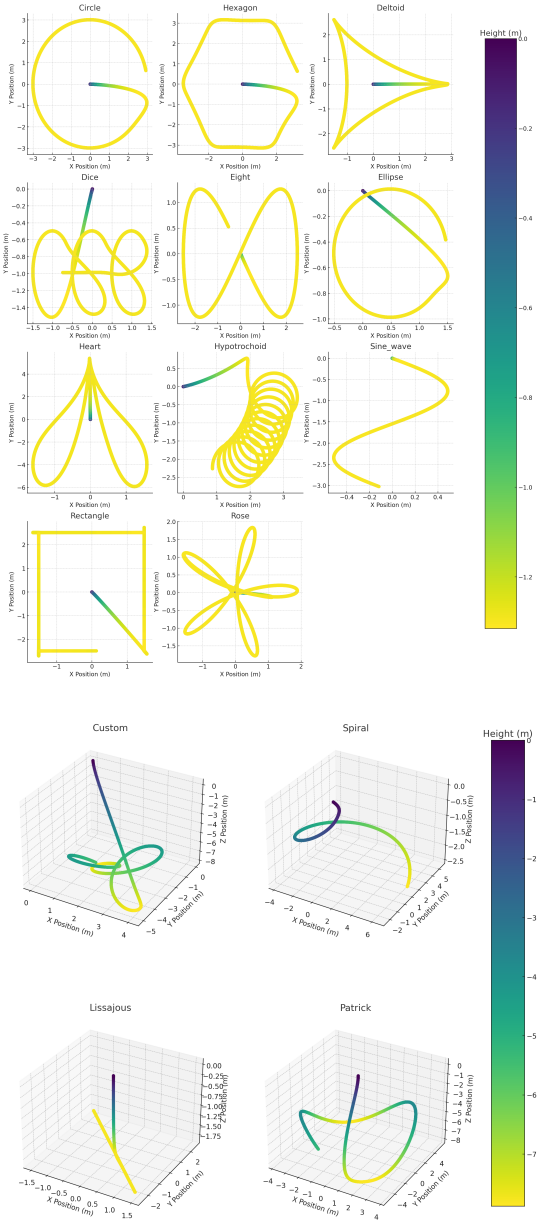

Fig. 5. Trajectory paths included in this dataset (Top Down)

1) Sequence Length (Look-back Time Horizon): We defined a parametric window for different time horizons that varies our data feed to our network. We experimented with 1s, 10s and 50s corresponding time data at once to IONet and assessed it's performance. The IONet performed well on 1s time horizon. Performance degraded as we increased the time horizon. This was in opposition to our expectations, as we thought that the LSTM layer would prefer more data. For VONet and VIONet, we weren't able to test time horizons more than 1 second due to memory constraints. The 1 second lookback period for VONet already was using 16/24GB of the available memory on the GPU.

2) Trajectory Tracking Performance Measurement : To evaluate the trajectory following performance of our networks,

we used the evaluation tool developed by Zhang et al. [10]. Of the 16 trajectories, where 13 was in the training set, and 3 in the testing set, the trajectories in the training set performed better. This implies that the network is overfitted into the training dataset. This can also be seen in the loss over epochs in Figures 6, 7, 8, 9, 10, 11.

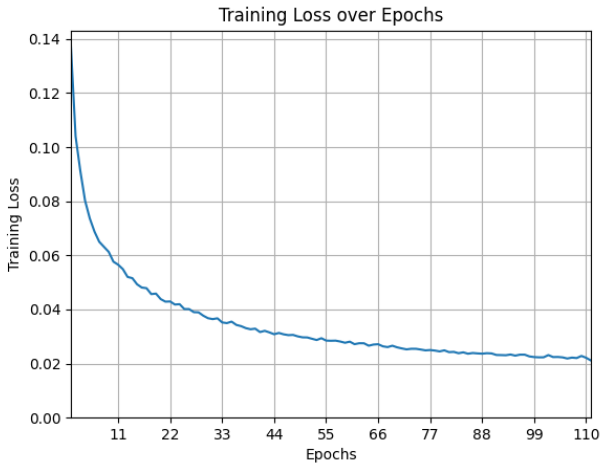

Fig. 6. IONet Training Loss vs Epochs

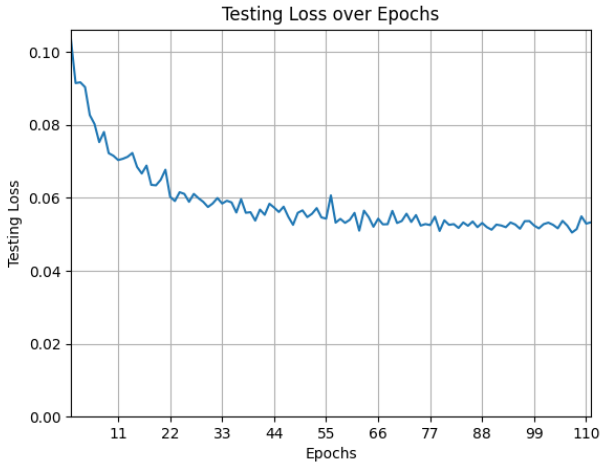

Fig. 7. IONet Testing Loss vs Epochs

V. Discussion and Conclusion

In this work, we presented a custom generated dataset and network architectures for odometry tracking given inertial and/or visual data. The provided graphs show that the model is able to perform this task up to a capacity. Given the time constraints and limited data, the trajectory tracking performance isn't great, but at least visible.

One possible research area for improving upon the classical VIO methods would be to combine RGB and event cameras together to obtain jello-free frames, thereby improving tracking of features across frames in fast moving bases. Replacing the encoder architecture with the encoder of a transformer could yield better results. Training LSTMs are time consuming because it is essentially a sequential operation. So, the last LSTM layers can also be replaced by a transformer architecture.

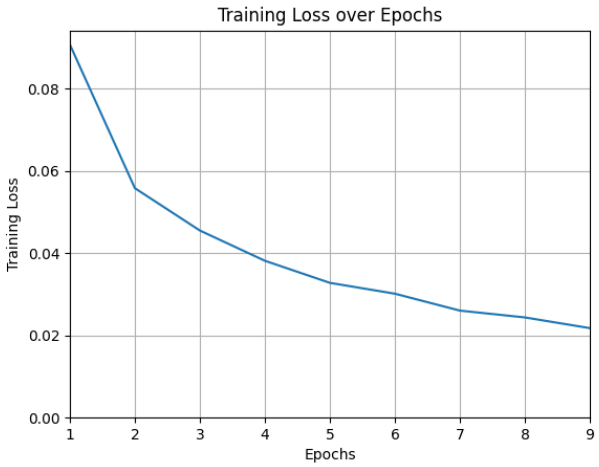

Fig. 8. VONet Training Loss vs Epochs

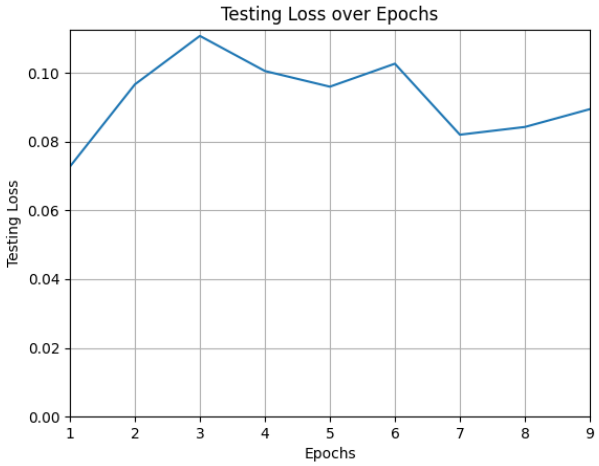

Fig. 9. VONet Testing Loss vs Epochs

References

[1] Wang, S., Clark, R., Wen, H., Trigoni, N. (2017, May). DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks. 2017 IEEE International Conference on Robotics and Automation (ICRA). doi:10.1109/icra.2017.7989236

[2] Han, L., Lin, Y., Du, G., Lian, S. (2019). DeepVIO: Self-supervised Deep Learning of Monocular Visual Inertial Odometry using 3D Geometric Constraints. arXiv [Cs.RO]. Retrieved from http://arxiv.org/abs/1906.11435

[3] Yang, M., Chen, Y., Kim, H.-S. (2022). Efficient Deep Visual and Inertial Odometry with Adaptive Visual Modality Selection. arXiv Preprint arXiv:2205. 06187.

[4] Sun, Ke Mohta, Kartik Pfrommer, Bernd Watterson, Michael Liu, Sikang Mulgaonkar, Yash Taylor, Camillo Kumar, Vijay. (2017). Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight. IEEE Robotics and Automation Letters. PP. 10.1109/LRA.2018.2793349.

[5] Zichao Zhang, Davide Scaramuzza: A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS), 2018.

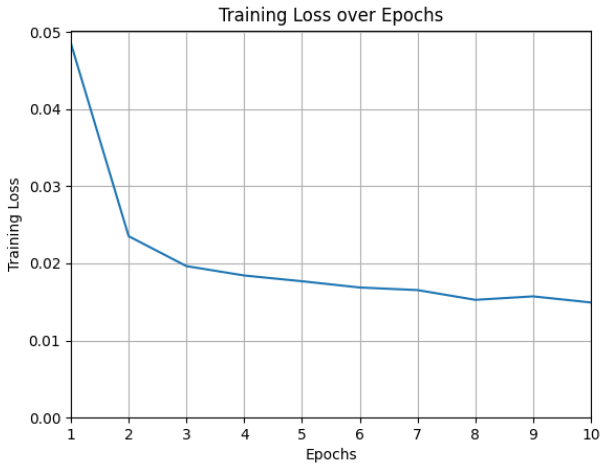

Fig. 10. VIONet Training Loss vs Epochs

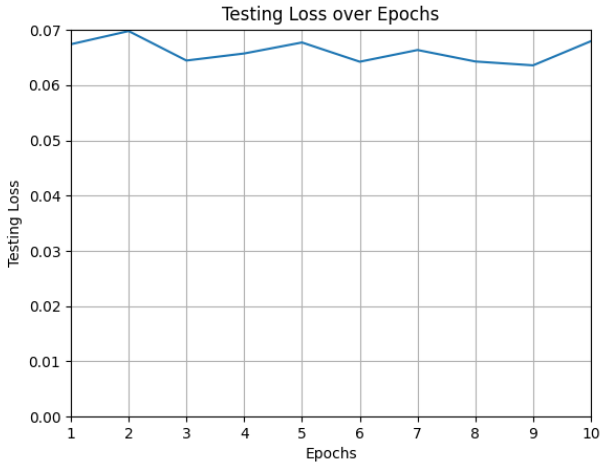

Fig. 11. VIONet Testing Loss vs Epochs

[6] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik and R. Siegwart, The EuRoC micro aerial vehicle datasets, International Journal of Robotic Research, DOI: 10.1177/0278364915620033, early 2016.

[7] Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., … Sutskever, I. (2021). Zero-Shot Text-to-Image Generation. arXiv [Cs.CV]. Retrieved from http://arxiv.org/abs/2102.12092

[8] https://github.com/Aceinna/gnss-ins-sim

[9] https://rbe549.github.io/rbe595/fall2023/proj/p0/

[10] Zhang, Z., Scaramuzza, D. (2018). A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry. IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS).

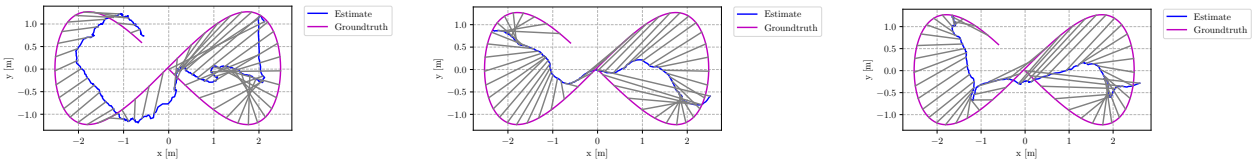

Fig. 12. Figure Eight position estimate vs. ground truth in top view for IONet, VONet, VIONet

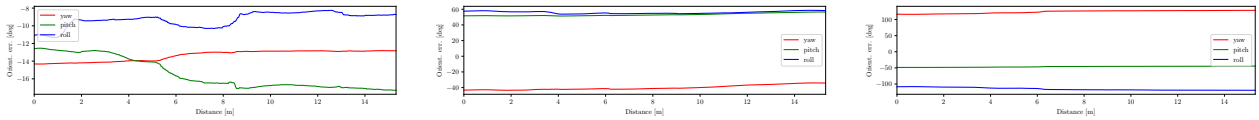

Fig. 13. Rotational tracking error (in degrees) over distance travelled (in meters) for Figure Eight

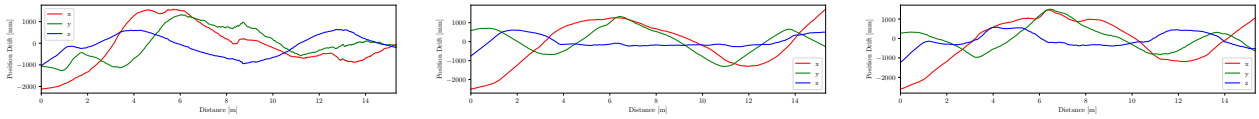

Fig. 14. Position tracking error (in percentage) over distance travelled (in meters) for Figure Eight

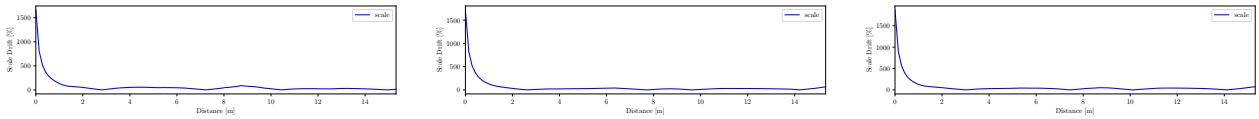

Fig. 15. Scale drift error (in percentage) over distance travelled (in meters)

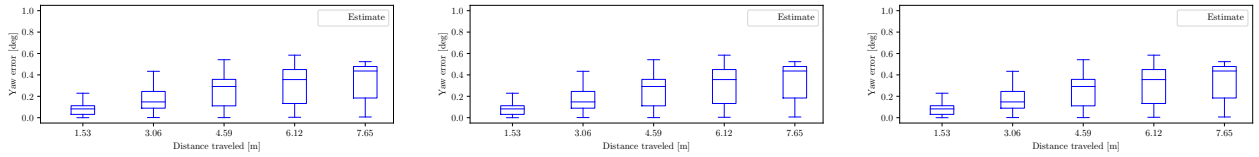

Fig. 16. Yaw tracking error (in degrees) over distance travelled (in meters)

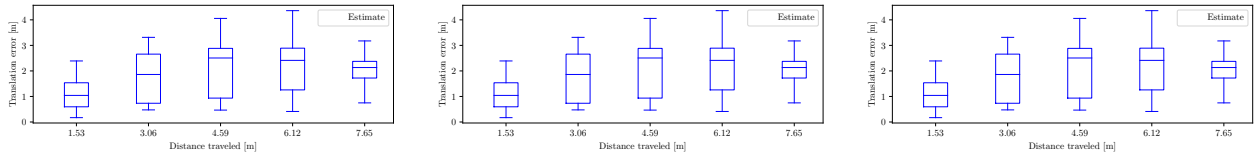

Fig. 17. Position tracking error (in meters) over distance travelled (in meters)

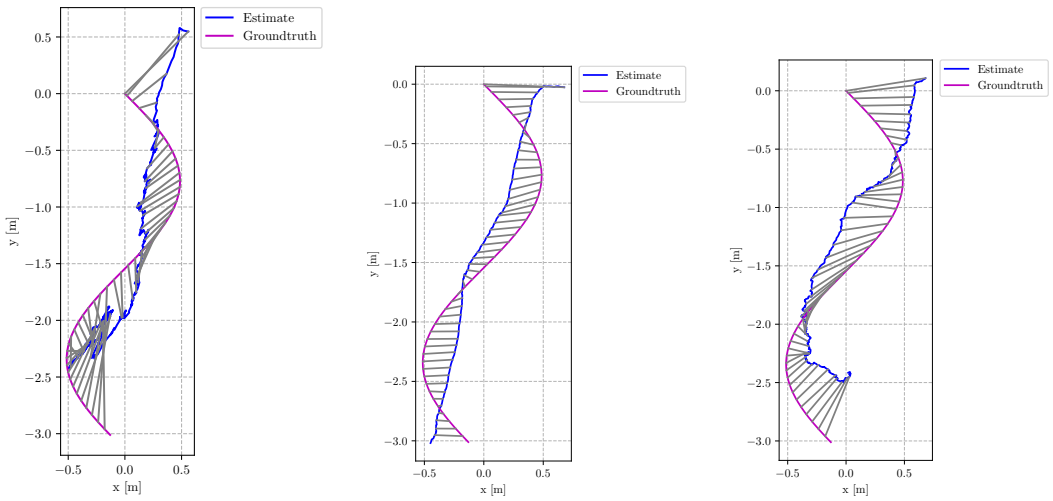

Fig. 18. Sine Wave Trajectory comparison for IONet, VONet, VIONet

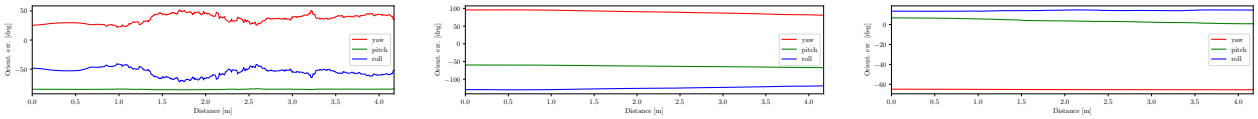

Fig. 19. Rotational tracking error (in degrees) over distance travelled (in meters) for Figure sine

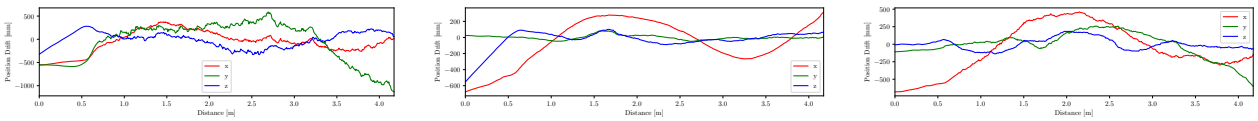

Fig. 20. Position tracking error (in percentage) over distance travelled (in meters) for Figure sine

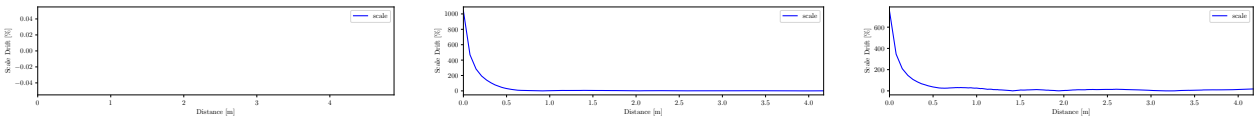

Fig. 21. Scale drift error (in percentage) over distance travelled (in meters)

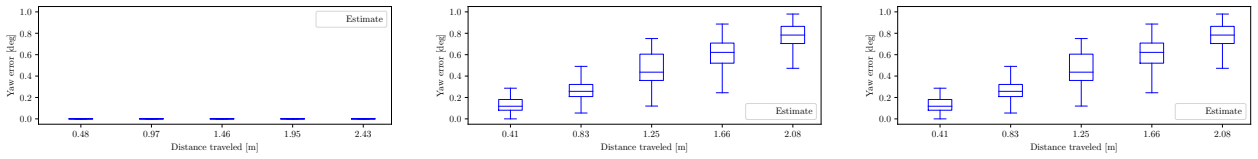

Fig. 22. Yaw tracking error (in degrees) over distance travelled (in meters)

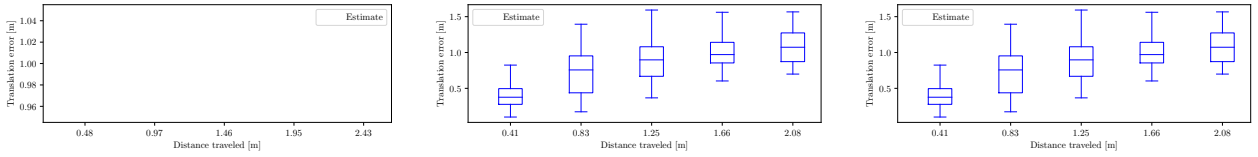

Fig. 23. Position tracking error (in meters) over distance travelled (in meters)