From Images to Inertia: Deep Learning's Take on Visual Inertial Odometry

Computer Vision (RBE549) Project 4 - Deep Visual-Inertial Odometry

Hrishikesh Pawar
MS Robotics Engineering
Worcester Polytechnic Institute
Email: hpawar@wpi.edu

Tejas Rane
MS Robotics Engineering
Worcester Polytechnic Institute
Email: turane@wpi.edu

Abstract—**Accurate pose estimation remains a critical challenge in the field of robotics and autonomous systems, particularly in GPS-denied environments. Visual-Inertial Odometry (VIO) offers a promising solution by combining visual information with inertial measurements to estimate the pose of a device. However, traditional VIO systems are often limited by their dependency on environmental conditions and the robustness of feature extraction techniques. This paper proposes a novel deep learning-based approach to enhance the robustness and accuracy of pose estimation in VIO systems. We introduce a hybrid neural network architecture that integrates convolutional neural networks (CNNs) for image processing with Long Short-Term Memory networks. (LSTMs) for effective handling of sequential inertial data. A comprehensive synthetic dataset, generated using Blender, provides a controlled environment for both training and evaluating the proposed model.**

I. Introduction

The advent of autonomous vehicles and robotics has heightened the necessity for robust and precise navigation systems. Traditional methods often rely on GPS or extensive sensor suites, which can be hindered by environmental factors or high costs. Visual-Inertial Odometry (VIO) presents a compelling alternative by fusing camera imagery and inertial measurements to estimate the pose of a device relative to its environment. However, VIO systems confront significant challenges in terms of robustness, especially in dynamic and visually complex environments.

In this paper, we explore an approach to VIO that integrates deep learning to improve pose estimation accuracy and robustness. Traditional VIO systems often struggle with feature degradation in adverse conditions, such as low light or high dynamic scenes. By employing deep learning models, we aim to extract and utilize complex features more effectively than conventional algorithms, potentially overcoming some of the inherent limitations of traditional methods.

Our approach is heavily inspired by recent advancements in the field, particularly the integration of neural networks for feature extraction and pose prediction. This work builds upon foundational theories in both computer vision and inertial measurement analysis.

A. ***Organization of the paper***

The paper is structured as follows:

- **Section II: Data Generation** - Discusses the creation of a synthetic dataset using Blender for realistic simulation of visual and inertial data, which serves as a foundation for training and evaluating our models.
- **Section III: Deep Learning-Based Odometry Stack** - Details the architecture, training, and integration of deep learning models for enhancing the performance of odometry systems.
- **Section IV: Experiments and Discussions** - Presents experimental setups, results, and a thorough discussion on the performance of the proposed models compared to traditional VIO systems.
- **Section V: Conclusion** - Summarizes the findings, discusses the implications of this research, and suggests directions for future work.
- **Section VI: Acknowledgements** - Offers thanks to those who provided support or contributions that were significant to the research.

II. Data Generation

A. *Simulation of Dynamic Trajectories in Blender*

This section describes the setup and methodology for generating realistic visual and inertial data through simulation. The environment for these simulations was established in Blender. A large 3D mesh plane, textured with a high-resolution image, was set as the visual backdrop, serving as the visual ground truth. A virtual camera and an inertial measurement unit (IMU), which have no relative rotation (R) or translation (T) between them, were spawned within this environment to accurately capture the scene and generate corresponding sensor data.

The trajectory design involved programming nine distinct trajectories to simulate various flight patterns of an aerial robot. Refer to Fig. 1 for visualizations of these trajectories.

Dynamic calculations for each trajectory were conducted to model the physical movements realistically. The linear accelerations and angular velocities were derived based on the temporal spacing (*delta t*) between each rendered frame, ensuring temporal coherence and physical accuracy. Velocities were calculated using the following equation:

$$v = \frac{\Delta \text{location}}{\Delta t} \tag{1}$$

Accelerations were computed with the equation:

$$a = \frac{\Delta v}{\Delta t} \quad (2)$$

Additionally, orientation changes were computed using sinusoidal functions to simulate the roll, pitch, and yaw of the camera throughout the trajectory. Specifically, the equations for pitch, roll, and yaw are given by:

$$\text{pitch}(t) = A \cdot \sin(\theta(t)) \cdot \cos(\theta(t)) \quad (3)$$

where A is the amplitude (maximum angle of pitch in radians), and $\theta(t) = 2\pi\frac{t}{T}$ is the phase of the cycle at time t, with T being the total duration of the cycle.

$$\text{roll}(t) = A \cdot \cos(\theta(t)) \cdot \cos(\theta(t)) \quad (4)$$

where A is the amplitude (maximum angle of roll in radians), and $\theta(t)$ is defined in the same manner as for pitch.

$$\text{yaw}(t) = 0 \quad (5)$$

During the data capture and storage phase, each frame rendered by the camera and the synthetic IMU data were precisely synchronized and recorded. This approach ensured the creation of a dataset that included perfectly aligned visual and inertial data, capturing accelerations and angular velocities without any initial biases or noise, thereby representing an ideal dataset.

B. *Integration of Sensor Noise Models*

To enhance the realism of the simulated dataset, sensor noise was artificially introduced to the synthetic IMU data. This was crucial for preparing our models to handle real-world uncertainties and variabilities in sensor readings. We employed OysterSim, to generate realistic IMU noise patterns.[1]

Utilizing the low accuracy model from OysterSim, we aimed to approximate the noise characteristics typical of less precise, commercially available IMU devices. This choice was driven by our objective to develop robust algorithms capable of performing well under practical operational conditions where high-grade IMUs may not always be available or feasible.

In the context of neural network training, estimating and compensating for sensor biases can pose significant challenges, particularly when biases vary unpredictably over time. To simplify our initial training processes and focus on core algorithmic performance, we assumed that the biases at the start of each data sequence were zero. This assumption aligns with common practices in preliminary stages of algorithm development, allowing us to primarily address the variance introduced by noise and later incorporate bias estimation strategies in advanced model iterations.

C. *Specification of Data Structure*

In our simulation, a total of 5000 samples of synchronized IMU data and images were generated to ensure a comprehensive evaluation of the pose estimation models. This data was meticulously organized and stored in `.txt` formats for each of the following.

[1] *OysterSim Source Code:* https://github.com/prgumd/Oystersim/blob/master/code/ImuUtils.py

a) Camera Pose Data: The camera poses were recorded for each frame, capturing both position and orientation in the scene.

b) IMU Data: The IMU data, encompassing both acceleration and angular velocities, was captured simultaneously with the camera images to maintain temporal alignment.

c) Trajectory Data: The actual trajectory data, representing the path of the camera through space, was also recorded.

d) Images: 5000 samples of images with the dimensions 640 x 480 pixels are saved in folder.

The format of each of the above are summarised in the following Table I

Data Type	**Format**
Camera Pose	Frame number, Position (x, y, z), Rotation (roll, pitch, yaw)
IMU Data	Frame number, Acc_X, Acc_Y, Acc_Z, Roll_Vel, Pitch_Vel, Yaw_Vel
Trajectory Data	X, Y, Z coordinates
Image Data	5000 - 640 x 480 samples

TABLE I: Summary of Data Storage Formats

This data storage format ensured that each dataset element is easily accessible and clearly defined, supporting both the training of our models and subsequent performance evaluations.

III. Deep Learning based Odometry Stack

In this section we talk about the three main components of our Deep Learning based Odometry stack - Inertial Odometry, Visual Odometry and Visual-Inertial Odometry. We also describe some of the other essential components of the pipeline which facilitate the training process. Te schematics of our model architectures is shown in Fig 2.

A. *Creating the dataset*

The drone pose data generated from Blender as described in the previous section is in the global frame of reference, which needs to be converted to relative poses. We also need enforce a $1:10$ ratio of images to the IMU data (on a realistic drone, the camera would run at around 100Hz whereas the IMU would run at around 1000 Hz). These operations are performed while creating the dataset for training.

We assume that we receive the images at time steps $t, t+1, t+2, \ldots$. Therefore, between time step t and $t+1$, we receive a total of 11 IMU readings and 2 images, both time steps inclusive. To simulate this, the ground truth camera poses corresponding to the images are first converted to relative poses; the pose corresponding to the second image is calculated with respect to the pose corresponding to the first image. The input IMU data is then divided into chunks of 11 and paired with the corresponding ground truth relative poses and images. These chunks are then split into training (80%) and validation (20%) sets at random and used for training.

We also exploit the collected dataset of 5000 images and corresponding IMU readings from one trajectory to increase the size of the dataset. Based on the description in the previous paragraph, if we enforce the $1:10$ ration for images and IMU

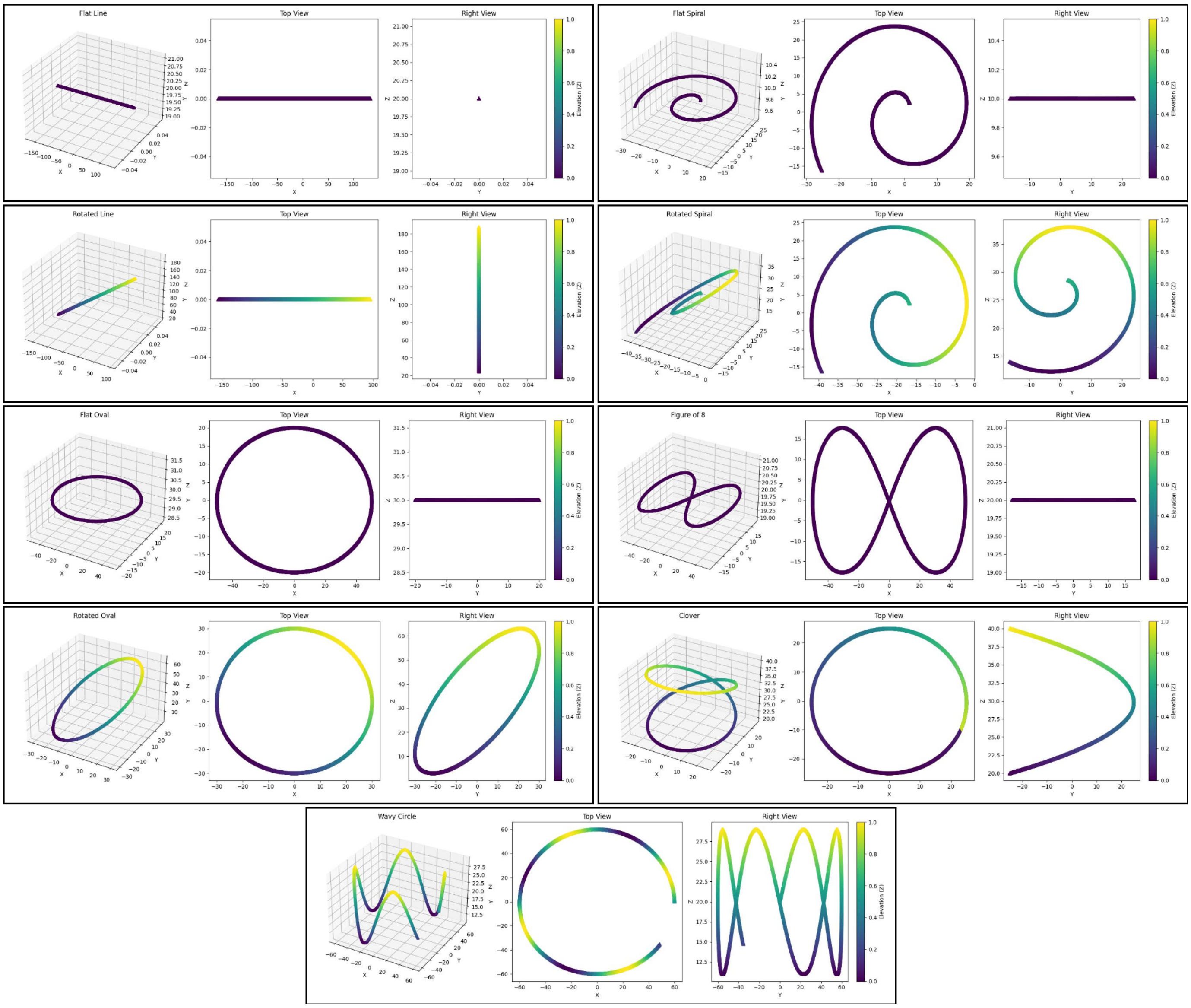

Fig. 1: Trajectories. Left Column From top: (1) Flat Line (2) Rotated Line (3) Flat Oval (4) Rotated Oval, Right Column From top: (5) Flat Spiral (6) Rotated Spiral (7) Fig of 8 (8) Clover, Center Bottom (9) Wavy Circle

readings, the total number of generated data samples from one trajectory would be only 500[2]. Instead, we generate our dataset using a sliding window of 2 images. Using this trick, we are able to extract a total of 4990 data samples from a single trajectory[3].

[2]The first data sample would be [image at time $t = 0$, image at time $t = 10$, and the IMU readings in between them, both time steps inclusive]. The next sample would be [image at time $t = 10$, image at time $t = 20$, and the IMU readings in between them, both time steps inclusive], and so on.

[3]The first data sample would be [image at time $t = 0$, image at time $t = 10$, and the IMU readings in between them, both time steps inclusive]. The next sample would be [image at time $t = 1$, image at time $t = 11$, and the IMU readings in between them, both time steps inclusive], and so on.

B. *Deep Inertial Odometry (IO)*

The Deep Inertial Odometry network is based on the Long Short-Term Memory (LSTM) layers, followed by fully connected linear layers. The input to the network is a sequence of IMU readings and the output prediction is the relative pose between the first and the last reading. We use LSTM layers because we want the network to understand the temporal relation between the IMU readings to predict the corresponding relative pose. For the activation function between the linear layers, we use the Parametric Rectified Linear Units (PReLU) activation function, as we observed that it performed better as compared to the Rectified Linear Units (ReLU) activation function.

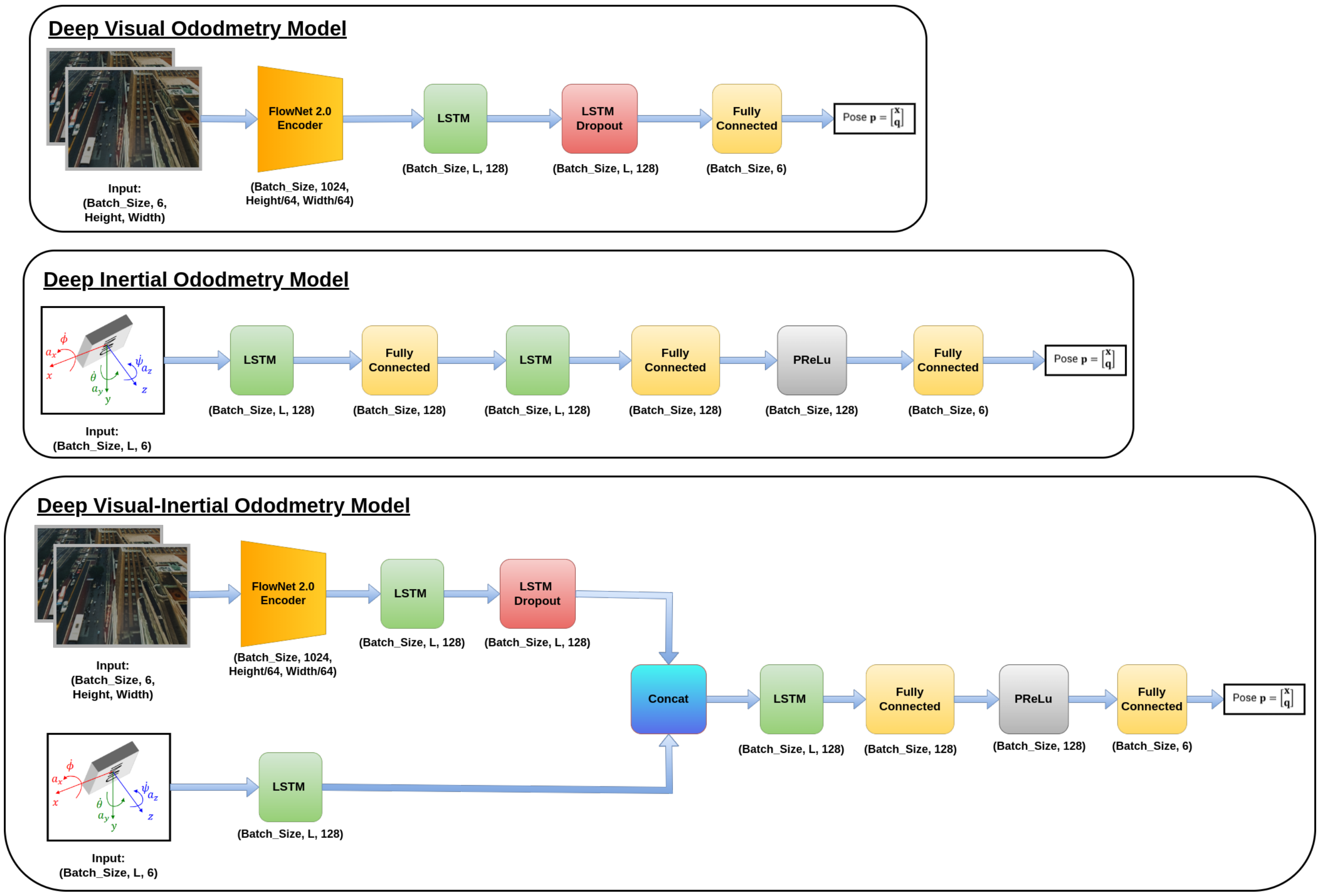

Fig. 2: Model Architectures. From Top: (1) Visual Odometry, (2) Inertial Odometry, (3) Visual Inertial Odometry, Note: L is sequence length

C. *Deep Visual Odometry (VO)*

For Deep Visual Odometry, we employ a Flownet 2.0 (1) based encoder. The input images are concatenated in the channels dimension[4] and fed into the network. The output features of the encoder are then passed through a LSTM and a linear layer. The final prediction of the network is the relative pose of camera corresponding to the second image, with respect to the first image.

The Flownet 2.0 encoder is used in an attempt to induce the notion of homography in the network. We want the neural network to focus on the parts of the image which are changing (which have high flow) and find the matching features to calculate the odometry. We load pretrained weights of the Flownet 2.0 encoder, and trian only the last LSTM and linear layers.

D. *Deep Visual-Inertial Odometry (VIO)*

The Deep Visual-Inertial Odometry network is created as an amalgamation of the IO and VO networks describes in the previous subsections. The feature vectors from both the networks are concatenated into a single input vector to a LSTM layer, followed by some fully connected linear layers. The inputs to this network are the concatenated images for the Flownet 2.0 encoder and the sequence of IMU readings between the two images for the LSTM layers, and the output prediction is the relative pose of camera corresponding to the second image, with respect to the first image.

The motive behind this architecture is to use the initial layers of the IO and VO networks individually as the encoders for the inertial and visual data, and then fuse these feature vectors to generate the final prediction. The final LSTM and linear layers after concatenating the feature vectors should learn to give weightage to the features (IO or VO features) which provide a better estimate of the relative pose and predict the final relative pose accordingly.

E. *Loss Functions*

A custom loss function was created to train the deep neural networks of the Odometry stack. We use the Mean Square Error (MSE) to regress the position part of the predicted drone pose (`x`, `y`, `z`). For the angular part of the predicted drone pose (`roll`, `pitch`, `yaw`) part of the pose, we use the Geodesic Loss. Since the Geodesic loss is applied on rotation matrices, we convert the predicted and ground truth

[4]The final dimensions of the input vector is $6 \times 480 \times 640$.

Euler angles to rotation matrices using PyTorch3D functions. Both loss functions are given equal weightage for training the models. The mathematical formulation of the loss function is shown in Eq 6.

$$\mathcal{L} = \lambda_p \mathcal{L}_{MSE} + \lambda_a \mathcal{L}_{Geodesic},$$
$$\mathcal{L}_{Geodesic} = d(R_{pred}, R_{gt}) = \cos^{-1}\left(\frac{\text{tr}(R_{pred}^T R_{gt}) - 1}{2}\right) \tag{6}$$

where, λ_p and λ_a are hyper-parameters controlling weights of their corresponding loss terms.

IV. Experiments and Discussions

The training experiments for our Deep Learning based Odometry stack were performed in three stages.

a) Stage 1: We trained our models on a specific trajectory and performed testing on the same trajectory. We chose the Rotated Spiral trajectory to perform these experiments (the image of the trajectory can be found in Fig 1. We trained the Inertial Odometry model for 25 epochs, and both Visual Odometry and Visual-Inertial Odometry models for 50 epochs each. The Adam optimizer is used with default parameters to train the networks and the learning rate is set to $1e^{-3}$. The training was performed on the Turing cluster with a batch-size of 32. The input and output dimensions of the layers of the models is shown in Fig 2.

The Inertial Odometry model takes only 0.2 seconds per epoch to train. The loss curve for the training is shown in Fig 3. The loss curve is plotted in log scale to observe the training progress is more detail. The final results, tested on the Rotated Spiral trajectory itself, are shown in Fig 7. The top row shows the plots of different views of the predicted and ground truth positions, and the bottom row shows the plots for the predicted and ground truth rotations.

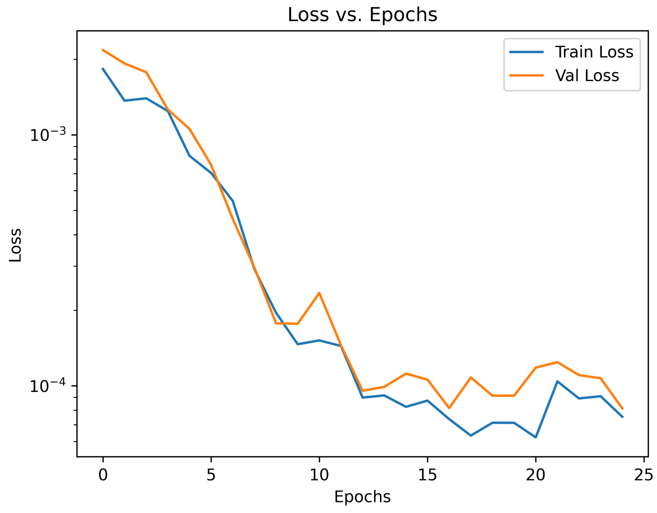

Fig. 3: The training and validation loss for Inertial Odometry trained on the Rotated Spiral Trajectory.

As it is observed in the two figures the Inertial Odometry model is able to learn the approximate scale of the trajectory, but it drifts a lot and is not able to maintain the spatial information of the trajectory. The mean RMSE ATE error is $\mathbf{5.746083963093333}$ units and the median RMSE ATE error is $\mathbf{3.1583412212731288}$ units[5].

The Visual Odometry model takes about 18 seconds per epoch to train. The loss curve for the training is shown in Fig 4. The final results are shown in Fig 8.

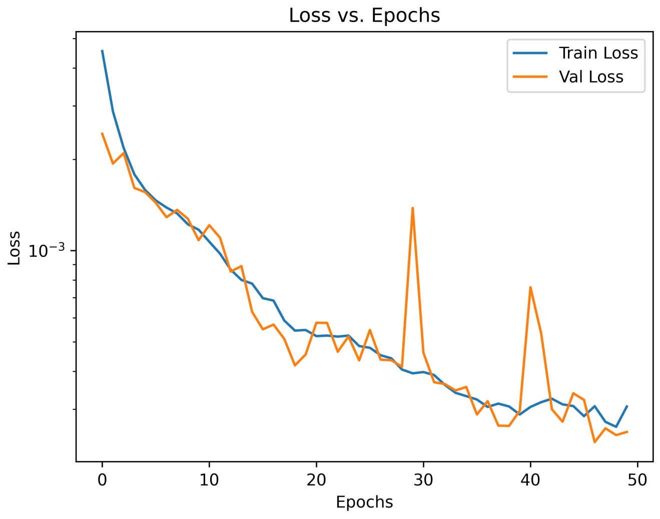

Fig. 4: The training and validation loss for Visual Odometry trained on the Rotated Spiral Trajectory.

As it is observed in the two figures the Visual Odometry model is able to learn the spatial information of the trajectory, but it loses the scale of the trjectory and drifts a lot. The drift in the predictions is observed to be much more than the Inertial Odometry model. The mean RMSE ATE error is $\mathbf{16.564576554050078}$ units and the median RMSE ATE error is $\mathbf{17.791302123576823}$ units.

Finally, the Visual-Inertial Odometry model takes about 20 seconds per epoch to train. The loss curve for the training is shown in Fig 5. The final results are shown in Fig 9.

As it is observed in the two figures the Visual-Inertial Odometry model is able to learn both the scale and the spatial information of the trajectory. There is still some drift observed in the trajectory, which would decrease if the model is trained for more number of epochs. The mean RMSE ATE error is $\mathbf{4.072604461574222}$ units and the median RMSE ATE error is $\mathbf{4.067260719385111}$ units.

Based in this experiment, we observe that the Inertial Odometry model performs better as compared to the Visual Odometry model. The Visual-Inertial Odometry model, takes the best parts of both these model - scale information from the Inertial Odometry and spatial information from the Visual Odometry - and performs better than both the individual models.

[5]The units of this metric is in Blender units as the trajectories are generated and predicted in Blender units.

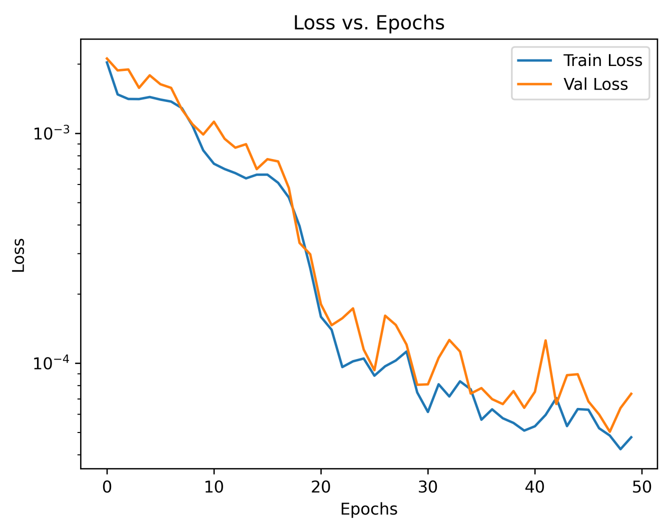

Fig. 5: The training and validation loss for Visual-Inertial Odometry trained on the Rotated Spiral Trajectory.

b) Stage 2: We trained the models on a specific trajectory and tested their performance on a different unseen trajectory. The three models trained on the Rotated Spiral trajectory were tested on Flat Spiral trajectory. The results of the Inertial Odometry are shown in Fig 10, Visual Odometry results are shown in Fig 11, and the Visual-Inertial Odometry results are shown in Fig 12.

We evaluated the RMSE ATE metrics for testing the performance of the models trained on the Rotated Spiral trajectory on the Flat Spiral trajectory. For the Inertial Odometry, the mean RMSE ATE error is $\mathbf{6.950821275971175}$ units and the median RMSE ATE error is $\mathbf{6.992217034006638}$ units. For the Visual Odometry, the mean RMSE ATE error is $\mathbf{33.03799379937866}$ units and the median RMSE ATE error is $\mathbf{22.954663928788815}$ units. Finally, for the Visual-Inertial Odometry, the mean RMSE ATE error is $\mathbf{8.471933159812883}$ units and the median RMSE ATE error is $\mathbf{8.05451989072098}$ units.

In this experiment, we observe that the Inertial Odometry actually performs better as compared to both the Visual Odometry and the Visual-Inertial Odometry models. We conclude that the Visual Odometry does not generalise on unseen data as compared to the Inertial Odometry model. This also affects the Visual-Inertial Odometry as the the visual part of the model generates poor estimates. But still the Visual Inertial Odometry model learns to give a higher weightage to the Inertial estimates as compared to the Visual estimates. We also conclude that it is easier for the models to show some generalisation on trajectories that are similar in scale and shape to the trajectories the model was trained on.

c) Stage 3: We tried training the models on multiple trajectories, in order to check the generalizability of the networks. We trained the model on eight trajectories (Flat Line, Rotated Line, Flat Oval, Rotated Oval, Flat Spiral, Rotated Spiral, Figure of 8, Clover), and held back one trajectory (Wavy Circle) for testing. The learning rate was decreased a lot and set at $1e^{-7}$ in order to help the model to train on the huge dataset.

We used two methods for training the models on multiple trajectories. In the first method, we trained the model sequentially on the trajectories - we trained the model entirely on one trajectory, then re-trained the same weights on another trajectory and so on. Here, we observed that the model was able to train on the later trajectories, but the model forgot the initial trajectories. In the second method, we trained on each trajectory from the training set in each epoch. The training was performed for 25 epochs and the Visual Inertial Odometry model took about 12 hours to train. But, even after 12 hours, the model could not learn any of the training trajectories.

In this experiment, we conclude that the training strategy that we employed was inaccurate, an a different strategy could be explored. We also feel that the current models that we have designed are shallow and do now have enough parameters to learn and generalize on multiple trajectories at a time.

V. Conclusion and Future Scope

In this project, we explored Deep Visual-Inertial Odometry and its individual components - Deep Inertial Odometry and Deep Visual Odometry. We designed the three model architectures using different strategies to help the model achieve its task. To train these models, we created custom trajectories to simulate a quadrotor in Blender, and collected their inertial and visual data. Finally, we trained the three models on multiple trajectories and generated their simulations to compare the performance of the three methods.

We performed multiple experiments of training the models on different trajectories and testing the models in three stages - training on one trajectory and testing on the same, training one one trajectory and testing on unseen trajectory, and finally training on multiple trajectories. In the limited time frame, we could not perform a thorough analysis of the generalization performance of the models on multiple trajectories.

This work can extend into multiple research directions. One challenge that can be addressed is enhancing the robustness and real-time performance of the deep neural networks. A faster visual-inertial odometry pipeline can facilitate robust localization for high-speed robots and micro aerial vehicles (MAVs). Another potential research direction involves extending VIO capabilities to support long-term localization and mapping. This includes tackling issues such as map management and scalability, loop closure detection, and maintaining consistency over prolonged periods or in large-scale environments.

VI. Acknowledgements

Sincere gratitude is extended to Professor Nitin Sanket, whose guidance was invaluable throughout the progression of not only this project but the entire course. We are also deeply thankful to the teaching assistants, Yijia Wu and Aabha Tamhankar, whose support and insightful feedback were instrumental throughout the course. Further appreciation is given to our fellow students for their constructive discussions and collaborative spirit, which significantly enriched our learning experience.

References

[1] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," 2016.

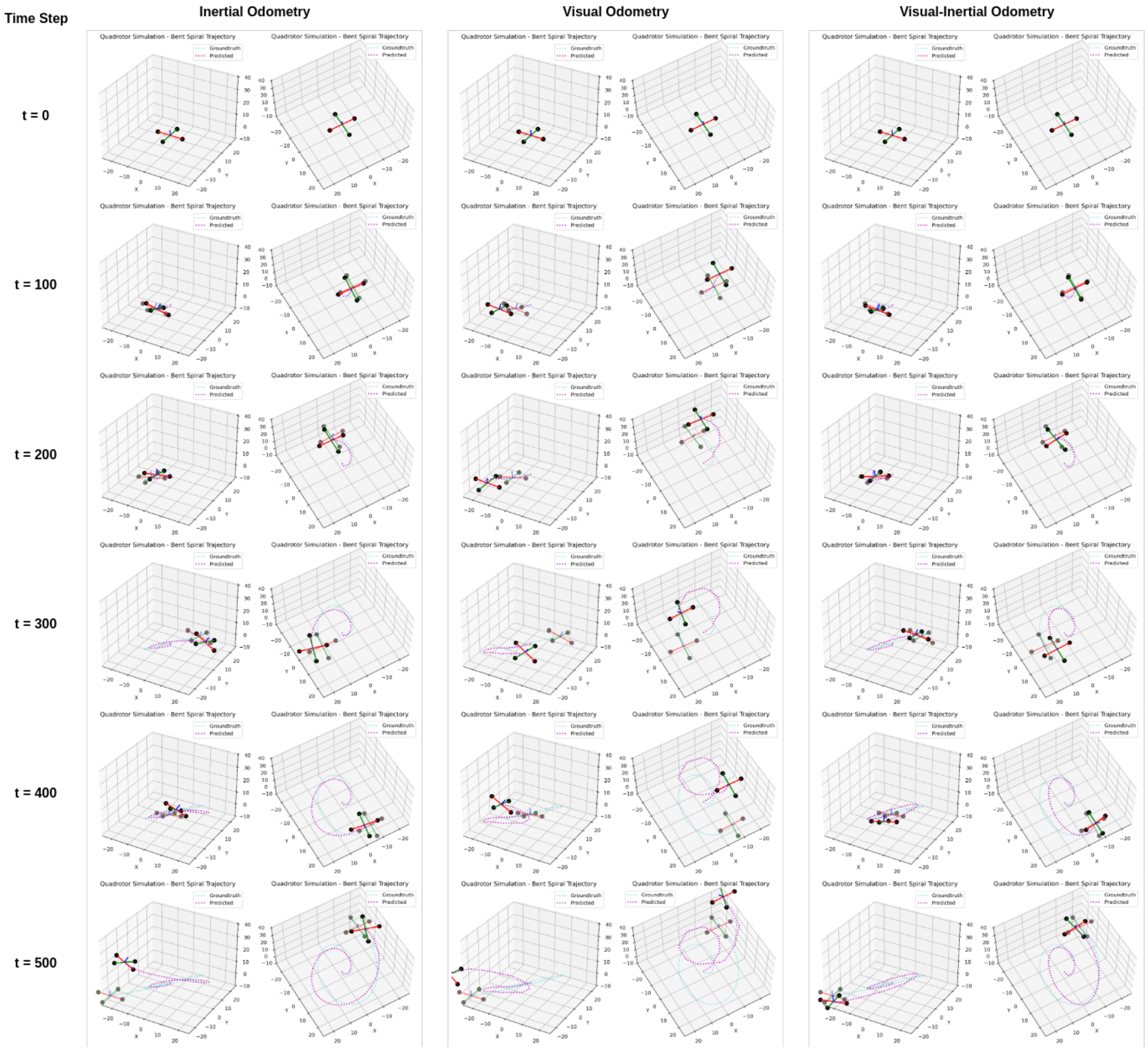

Fig. 6: Simulation of a Quadrotor estimating Odometry using the three methods - Inertial Odometry, Visual Odometry, Visual-Inertial Odometry. The simulation is performed using models trained on the Rotated Spiral Trajectory. The three columns show the predictions of these three models against the ground truth, and each row shows the time step in the trajectory. The red arms of the drone are initially along the X-axis, the green arms are initially along the Y-axis, and the initial Z-axis is shown by a blue marker in the center of the drone frame. The faded drone tracing the cyan trajectory shows the ground truth poses, and the other drone tracing the magenta trajectory shows the predicted poses. We show two different views of the simulation for each method, at each time step.

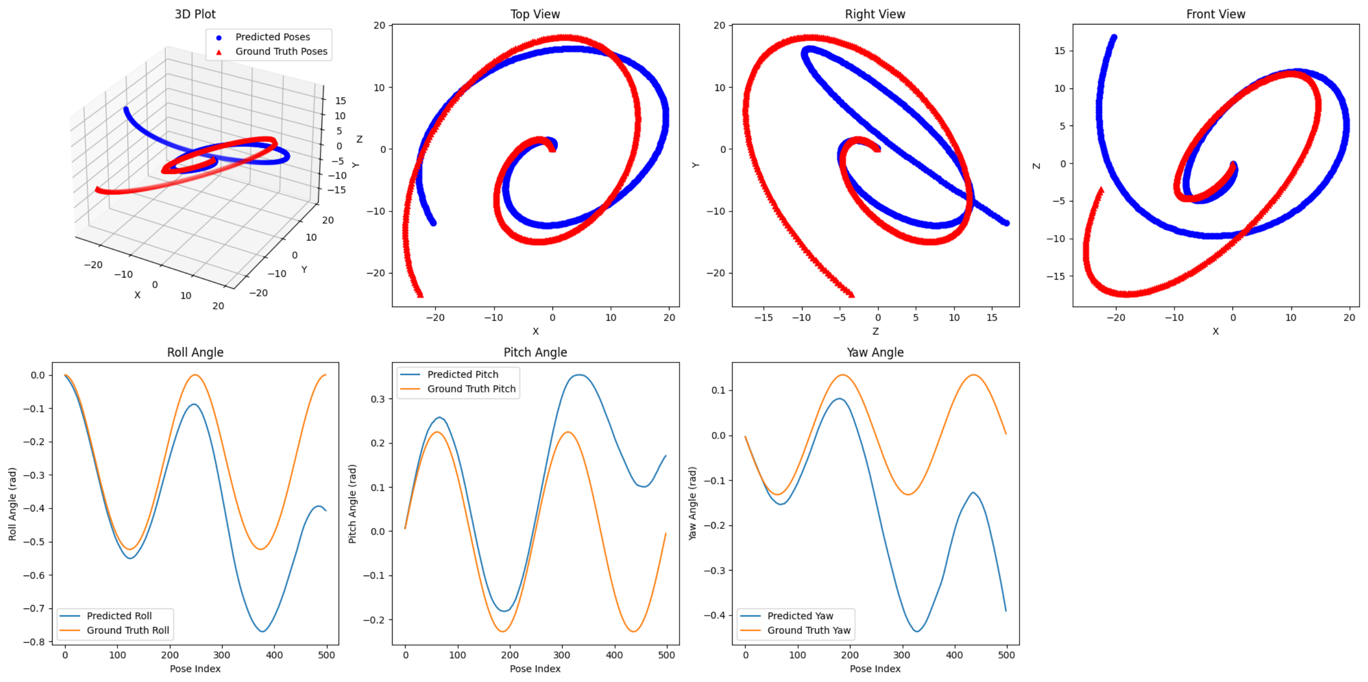

Fig. 7: Testing results of Inertial Odometry model on the Rotated Spiral Trajectory. Top row: Plots showing different views of the positions of ground truth (red) and the predicted (blue) trajectories. Bottom row: Plots showing the angles (roll, pitch, yaw) of the ground truth (orange) and the predicted (blue) trajectories.

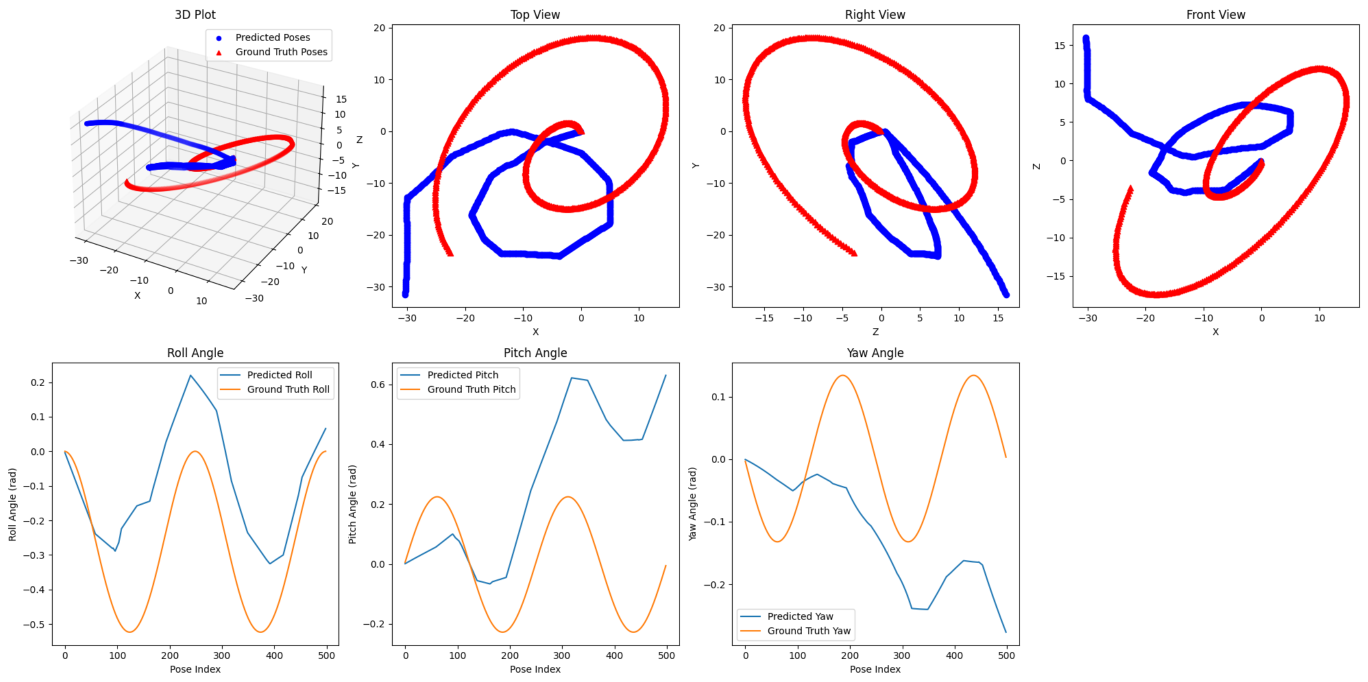

Fig. 8: Testing results of Visual Odometry model on the Rotated Spiral Trajectory. Top row: Plots showing different views of the positions of ground truth (red) and the predicted (blue) trajectories. Bottom row: Plots showing the angles (roll, pitch, yaw) of the ground truth (orange) and the predicted (blue) trajectories.

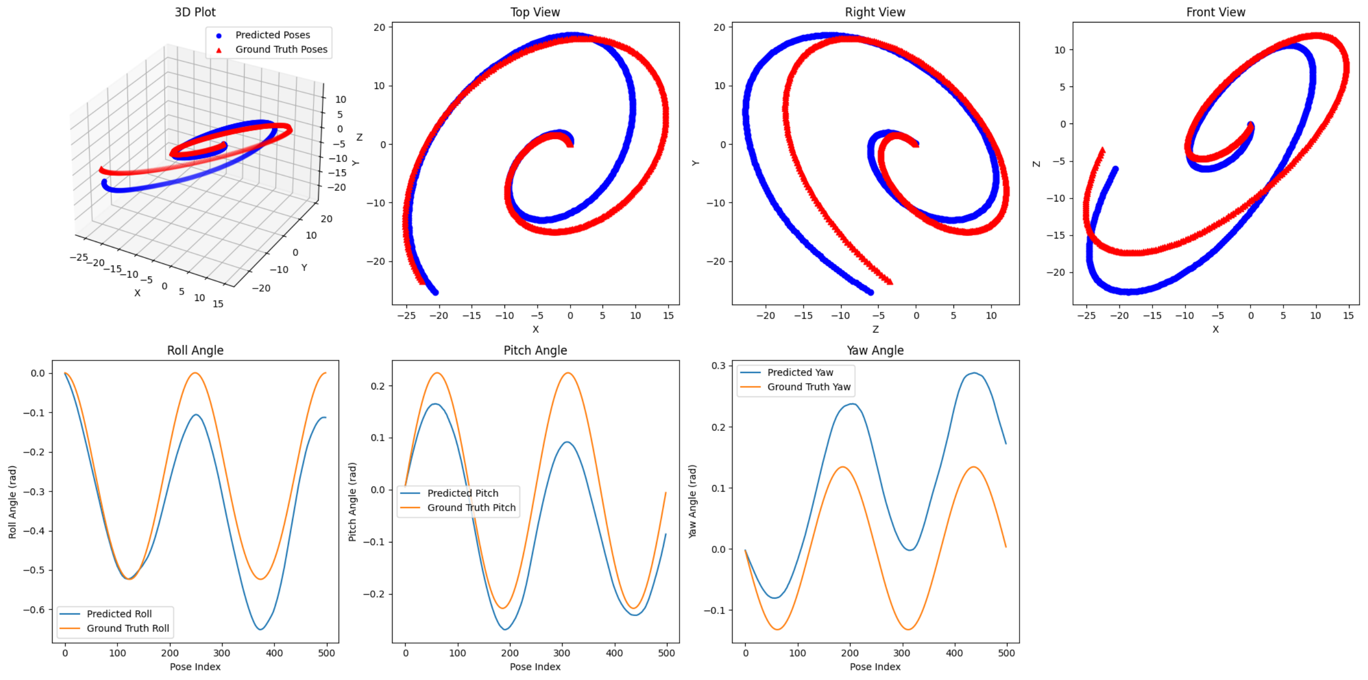

Fig. 9: Testing results of Visual-Inertial Odometry model on the Rotated Spiral Trajectory. Top row: Plots showing different views of the positions of ground truth (red) and the predicted (blue) trajectories. Bottom row: Plots showing the angles (roll, pitch, yaw) of the ground truth (orange) and the predicted (blue) trajectories.

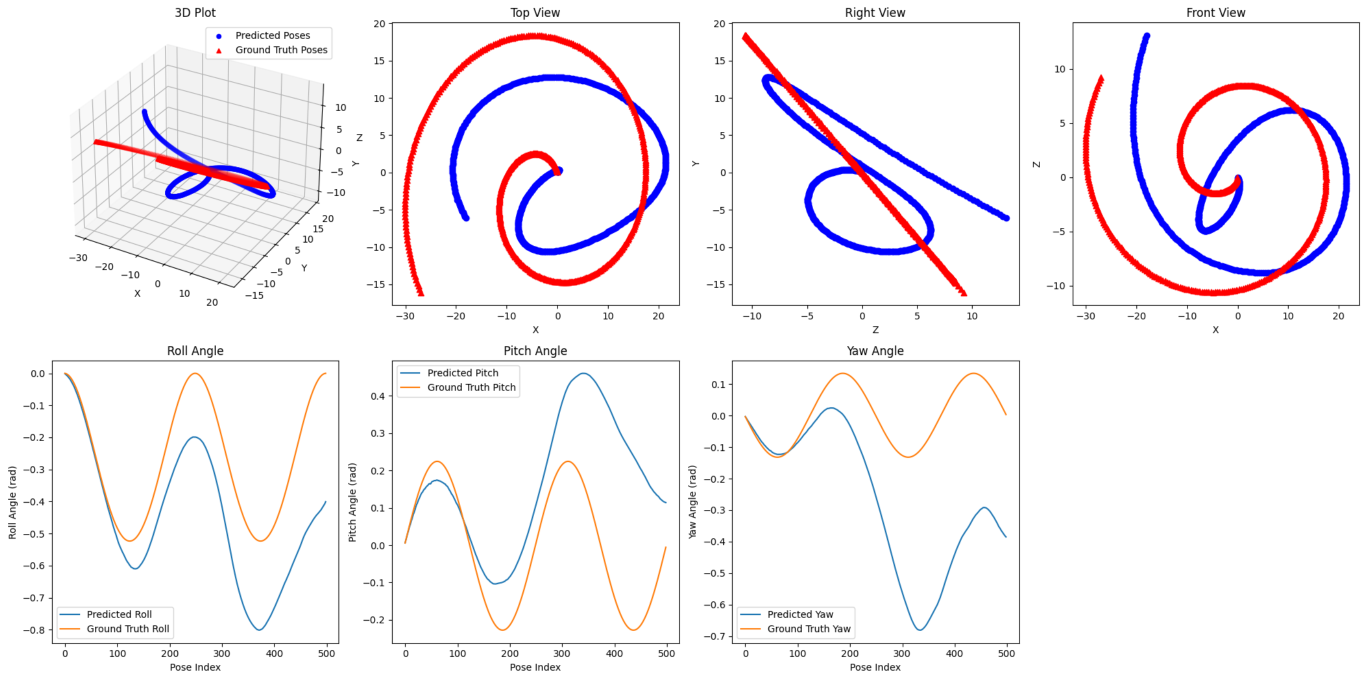

Fig. 10: Testing results of Inertial Odometry model, trained on the Rotated Spiral Trajectory and tested on the Flat Spiral Trajectory. Top row: Plots showing different views of the positions of ground truth (red) and the predicted (blue) trajectories. Bottom row: Plots showing the angles (roll, pitch, yaw) of the ground truth (orange) and the predicted (blue) trajectories.

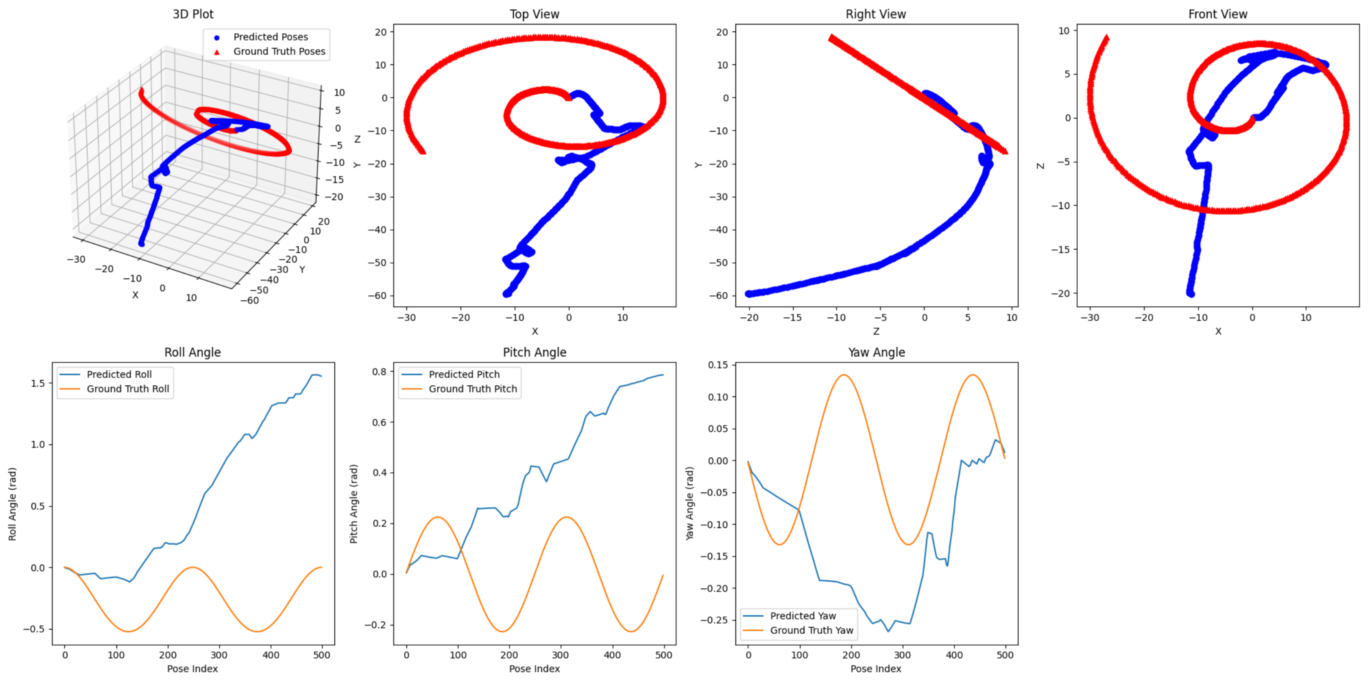

Fig. 11: Testing results of Visual Odometry model, trained on the Rotated Spiral Trajectory and tested on the Flat Spiral Trajectory. Top row: Plots showing different views of the positions of ground truth (red) and the predicted (blue) trajectories. Bottom row: Plots showing the angles (roll, pitch, yaw) of the ground truth (orange) and the predicted (blue) trajectories.

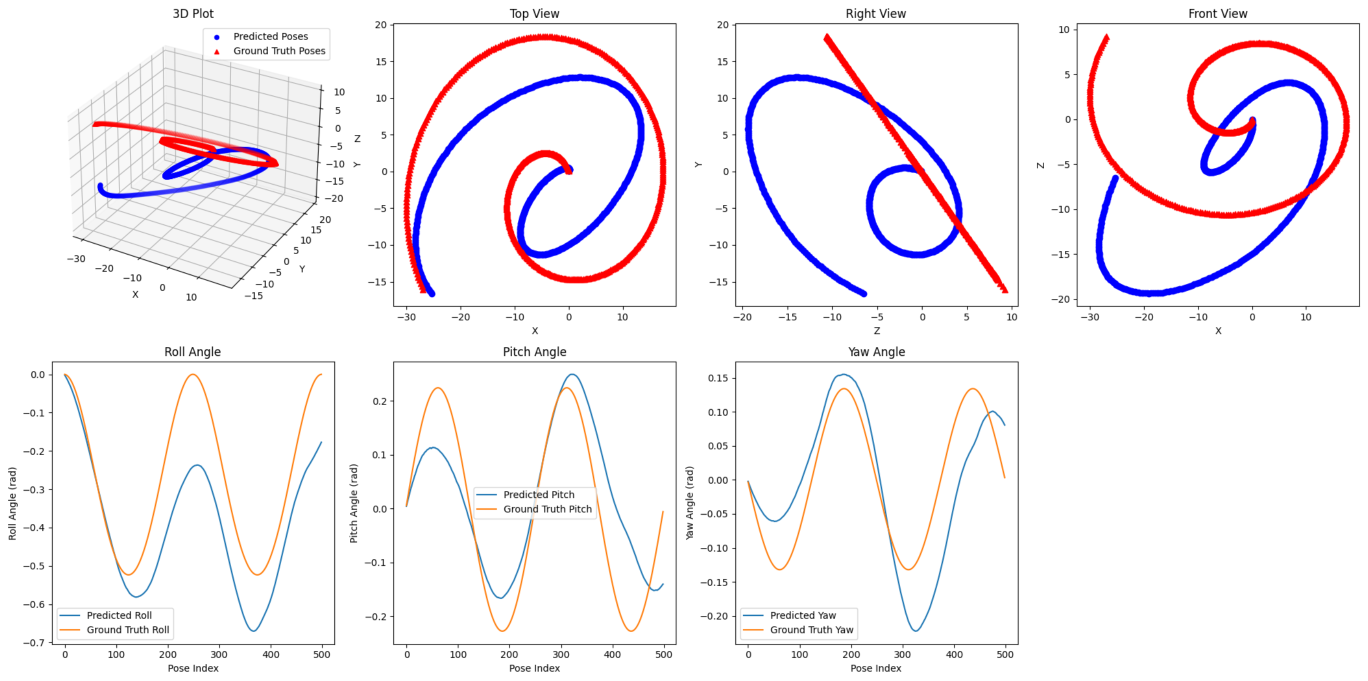

Fig. 12: Testing results of Visual-Inertial Odometry model, trained on the Rotated Spiral Trajectory and tested on the Flat Spiral Trajectory. Top row: Plots showing different views of the positions of ground truth (red) and the predicted (blue) trajectories. Bottom row: Plots showing the angles (roll, pitch, yaw) of the ground truth (orange) and the predicted (blue) trajectories.