P4 - Deep and Un-Deep Visual Inertial Odometry

Mihir Deshmukh
Robotics Engineering
Worcester Polytechnic Institute
Email: mpdeshmukh@wpi.edu

Ashwin Disa
Robotics Engineering
Worcester Polytechnic Institute
Email: amdisa@wpi.edu

***Abstract*—In this project, we implement stereo-visual inertial odometry that utilizes the Multi-State Constraint Kalman Filter [1]. It is a tightly coupled sensor fusion approach for VIO. The project involves sensor fusion from a stereo camera and an IMU (Inertial Measurement Unit). We implement seven functions of MSCKF.**

I. Phase I: Classical Visual-Inertial Odometry

A. Dataset

For this project, we utilized the Machine Hall 01 easy dataset. The dataset was gathered using a Micro Aerial Vehicle (MAV), and it includes stereo images along with synchronized Inertial Measurement Unit (IMU) data. Additionally, precise motion and structure ground truth information are available. This dataset is a subset of the EuRoC dataset [2], and the ground truth data is provided by a Vicon Motion Capture system with sub-millimeter accuracy.

B. Initialize gravity and bias

The robot's gravity and gyroscope bias are set during initialization using data from the first 200 messages received from the IMU while the robot is stationary. The gyroscope bias (b_g) is initialized by calculating the average of these 200 gyroscope readings. The gravity vector (g) is initialized as $[0, 0, -g]$, where g represents the magnitude of the first 200 accelerometer readings.

C. Batch IMU processing

Upon receiving a new feature, our procedure involves initially processing all IMU messages received before the timestamp of the new feature. For each IMU message stored within the IMU message buffer prior to the feature timestamp, we adjust our state estimation utilizing the process model.

D. Process Model

The continuous dynamics of the estimated IMU state is,

$$\begin{aligned} {}^I_G\dot{\hat{\mathbf{q}}} &= \frac{1}{2}\Omega(\hat{\boldsymbol{\omega}})\,{}^I_G\hat{\mathbf{q}}, \quad \dot{\hat{\mathbf{b}}}_g = \mathbf{0}_{3\times 1}, \\ {}^G\dot{\hat{\mathbf{v}}} &= C\left({}^I_G\hat{\mathbf{q}}\right)^\top \hat{\mathbf{a}} + {}^G\mathbf{g}, \\ \dot{\hat{b}}_a &= \mathbf{0}_{3\times 1}, \quad {}^G\dot{\hat{\mathbf{p}}}_I = {}^G\hat{\mathbf{v}}, \\ {}^I_C\dot{\hat{\mathbf{q}}} &= \mathbf{0}_{3\times 1}, \quad {}^I\dot{\hat{\mathbf{p}}}_C = \mathbf{0}_{3\times 1} \end{aligned}$$

where $\hat{\omega}$ and $\hat{a}$ are are the IMU measurements for angular velocity and acceleration respectively with biases removed, that is, $\hat{\omega} = \omega_m - b_g$ and $\hat{a} = m - b_g$.

The quarternion detivative Ω is given by

$$\Omega(\boldsymbol{\omega}) = \begin{bmatrix} -[\boldsymbol{\omega}\times] & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^T & 0 \end{bmatrix}$$

The linearised continuous-time model for IMU error-state is

$$\dot{\tilde{x}} = F\tilde{X} + Gn_I$$

The matrix F facilitates the derivation of the discrete-time state transition matrix. Similarly, the matrix G serves to ascertain the discrete-time noise covariance matrix. The term ϕ_K is approximated through Taylor's expansion up to the third order of F, whereas Q_K represents the discrete-time state covariance matrix, which is computed using continuous-time techniques based on the state covariance Q and the G matrix. The utilization of the state matrix involves its transformation into a symmetric matrix.

The F matrix is given by,

$$\mathbf{F} = \begin{pmatrix} -\lfloor\hat{\boldsymbol{\omega}}_\times\rfloor & -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -C\left({}^I_G\hat{\mathbf{q}}\right)^\top\lfloor\hat{\mathbf{a}}_\times\rfloor & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left({}^I_G\hat{\mathbf{q}}\right)^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix}$$

and the G matrix is given by,

$$\mathbf{G} = \begin{pmatrix} -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left({}^I_G\hat{\mathbf{q}}\right)^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix}$$

E. Predict new state

Upon receiving a new IMU measurement, the integration is performed using a fourth-order Runge-Kutta method with an adaptive time step. Further it calculates intermediate variables (k_1, k_2, k_3, k_4) using the gyroscope measurements and the

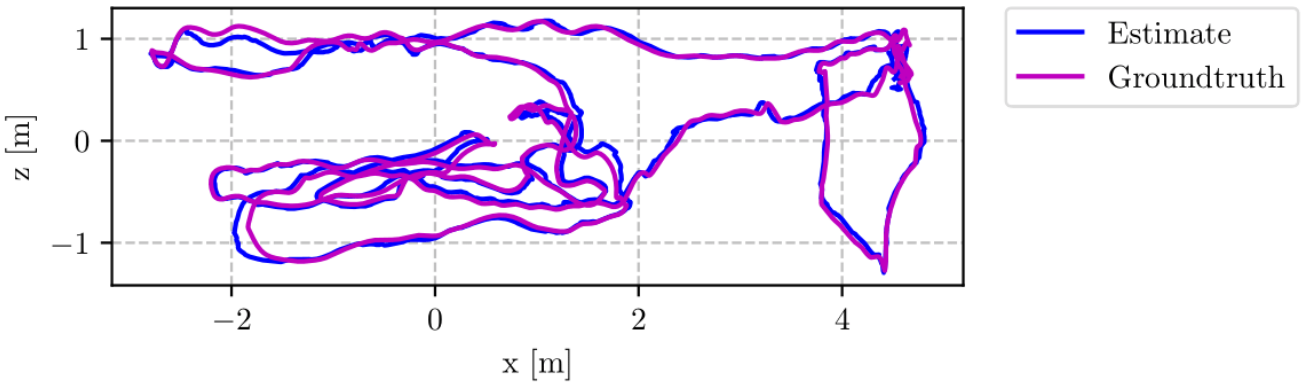

Fig. 1: Trajectory Error Side view

current state of the IMU, and then updates the orientation, velocity, and position of the IMU using these intermediate variables.

F. State Augmentation

In this step, we calculate the state covariance matrix, which will help us in disseminating the ambiguity of the given state. Initially, we extract the IMU and camera state values that correspond to the rotation from the IMU to the camera and the translation vector from the camera to the IMU. Subsequently, we incorporate a fresh camera state into the state server, utilizing the initial IMU and camera state. The augmentation Jacobian is calculated as follows,

$$J = \begin{pmatrix} J_I & 0_{6\times 6N} \end{pmatrix}$$

where J_I is given by

$$\mathbf{J}_I = \begin{pmatrix} C\left({}^I_G\hat{\mathbf{q}}\right) & \mathbf{0}_{3\times 9} & \mathbf{0}_{3\times 3} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} \\ -C\left({}^I_G\hat{\mathbf{q}}\right)^\top \lfloor {}^I\hat{\mathbf{p}}_{C\times} \rfloor & \mathbf{0}_{3\times 9} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} & \mathbf{I}_3 \end{pmatrix}$$

the full uncertainty propagation is represented as,

$$P_{k+1|k} = \begin{bmatrix} P_{II_{k+1|k}} & \Phi_k P_{IC_{k|k}} \\ P^\top_{IC_{k|k}} \Phi^\top_k & P_{CC_{k|k}} \end{bmatrix}$$

G. Add feature observations

This function retrieves the present IMU state identifier and the count of active features. Every feature contained in the 'feature message' is subsequently incorporated into the 'self.map server'. Utilizing both the previously known features and the newly tracked ones, we compute and refresh the rate at which tracking occurs.

H. Measurement update

This function performs the update based on measurements from visual features and inertial sensors. The state vector only contains the pose of the left camera, the pose of the right camera can be easily obtained using the extrinsic parameters from the calibration. Stereo measurements are given by

$$\mathbf{z}_i^j = \begin{pmatrix} u^j_{i,1} \\ v^j_{i,1} \\ u^j_{i,2} \\ v^j_{i,2} \end{pmatrix} = \begin{pmatrix} \frac{1}{{}^{C_{i,1}}Z_j} & \mathbf{0}_{2\times 2} \\ \mathbf{0}_{2\times 2} & \frac{1}{{}^{C_{i,2}}Z_j} \end{pmatrix} \begin{pmatrix} {}^{C_{i,1}}X_j \\ {}^{C_{i,1}}Y_j \\ {}^{C_{i,2}}X_j \\ {}^{C_{i,2}}Y_j \end{pmatrix}$$

The positions of the features in the left and right camera frames are given by,

$$\begin{aligned} {}^{C_{i,1}}\mathbf{p}_j &= \begin{pmatrix} {}^{C_{i,1}}X_j \\ {}^{C_{i,1}}Y_j \\ {}^{C_{i,1}}Z_j \end{pmatrix} = C\left({}^{C_{i,1}}_{G}\mathbf{q}\right)\left({}^G\mathbf{p}_j - {}^G\mathbf{p}_{C_{i,1}}\right) \\ {}^{C_{i,2}}\mathbf{p}_j &= \begin{pmatrix} {}^{C_{i,2}}X_j \\ {}^{C_{i,2}}Y_j \\ {}^{C_{i,2}}Z_j \end{pmatrix} = C\left({}^{C_{i,2}}_{G}\mathbf{q}\right)\left({}^G\mathbf{p}_j - {}^G\mathbf{p}_{C_{i,2}}\right) \\ &= C\left({}^{C_{i,2}}_{C_{i,1}}\mathbf{q}\right)\left({}^{C_{i,1}}\mathbf{p}_j - {}^{C_{i,1}}\mathbf{p}_{C_{i,2}}\right) \end{aligned}$$

We compute the residuals by stacking multiple observations of the same feature.

$$r^j = H^j_x \tilde{x} + H^j G_f \tilde{p}_j + n^j$$

The measurement update function takes measurement matrix H and residual matrix r as input to compute the Kalman gain K, and then updates the IMU state X_{IMU}, camera state, and state covariance matrix P.

The measurement matrix H is a matrix with block rows $H(j)$, $j = 1$ to L. L is the number of all detected features. If the number of rows (feature) is larger than the number of state X components, we employ QR decomposition for the matrix H. With the reduced mode of $numpy.linalg.qr$ function, we can directly get Q and T_H.

$$H = \begin{bmatrix} Q & Q_2 \end{bmatrix} \begin{bmatrix} T_H \\ O \end{bmatrix}$$

The matrix Q can then be used to compute residual r_n as

The Kalman gain K can be computed with the following equation

$$r_n = Q^T r = T_H \widetilde{X} + n_n$$

$$\mathbf{K} = \mathbf{P}\mathbf{T}_H^T \left(\mathbf{T}_H \mathbf{P}\mathbf{T}_H^T + \mathbf{R}_n\right)^{-1}$$

The state error (ΔX) is determined by the product of the Kalman gain K and r_{thin}.

$$\Delta X = K r_n$$

Finally, the state covariance matrix P is updated.

$$\mathbf{P}_{k+1|k+1} = \left(\mathbf{I}_\xi - \mathbf{K}\mathbf{T}_H\right)\mathbf{P}_{k+1|k}\left(\mathbf{I}_\xi - \mathbf{K}\mathbf{T}_H\right)^T + \mathbf{K}\mathbf{R}_n\mathbf{K}^T$$

I. Problems faced

While initializing the gravity and biases, we weren't aware of the quaternions being in the JPL convention which caused errors with the state exploding. There was an inconsistency in the measurement update function between the CPP code [3] and the paper. The equations are different, the covariance equation from the CPP code works as expected, but the equation from the paper wasn't working in our case. The simplified update equation we used following the CPP implementation is:

$$\mathbf{P}_{k+1|k+1} = \left(\mathbf{I}_\xi - \mathbf{K}\mathbf{T}_H\right)\mathbf{P}_{k+1|k}$$

J. Evaluation

To evaluate the performance of the multi-constraint filter, we calculate the Absolute trajectory error(ATE) using the *rpg_trajectory_evaluation* [4] package. We first convert the ground truth CSV to the required format and plot both trajectories utilizing this package. We configured the package to use SE(3) alignment before error computation.

- RMSE ATE: 0.0860 m
- Absolute Median Translation Error: 0.0724 m
- Scale Error RMSE: 1.0833 %
- Rotation error RMSE: 154.33 degrees

Fig. 1 shows the trajectory plotted with the ground truth and the estimated trajectory superimposed. Fig. 2 shows the plot of the translation error in the trajectory. Fig. 3 shows a box graph for the translation error along sub-segments of the trajectory.

Fig. 5 shows the plot of the rotation error. Fig. 6 shows the box graph for the relative error in yaw along sub-segments of the trajectory.

Fig. 4 shows the scale drift along the trajectory. As a stereo camera is used, the scale drift is minimal as evidenced by the graph in Fig. 4.

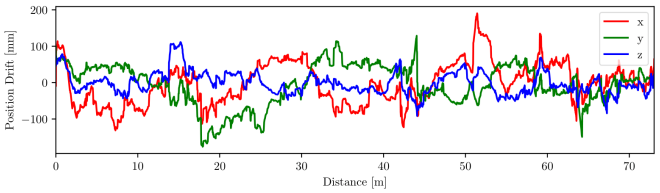

Fig. 2: Translation Error

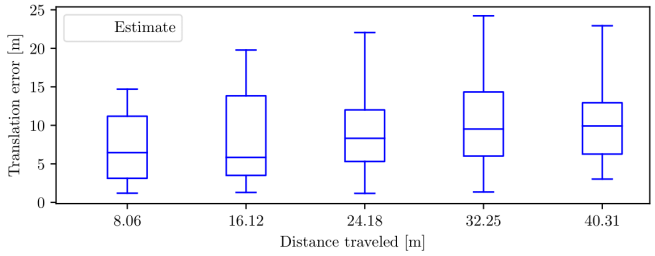

Fig. 3: Relative Translation Error

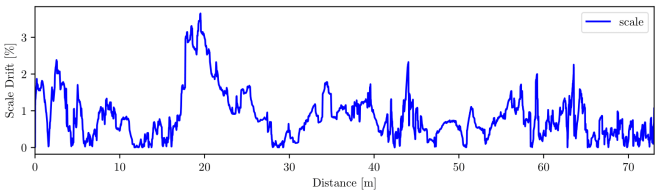

Fig. 4: Scale Error

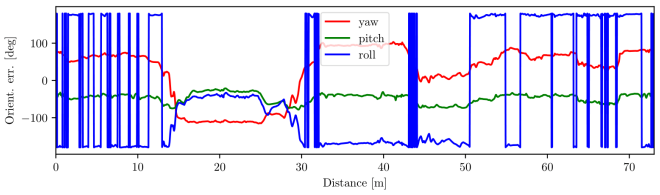

Fig. 5: Rotation Error

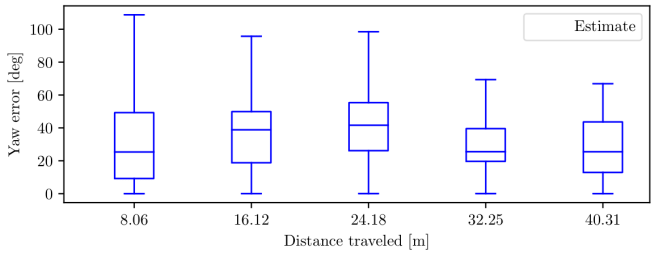

Fig. 6: Relative Yaw Error

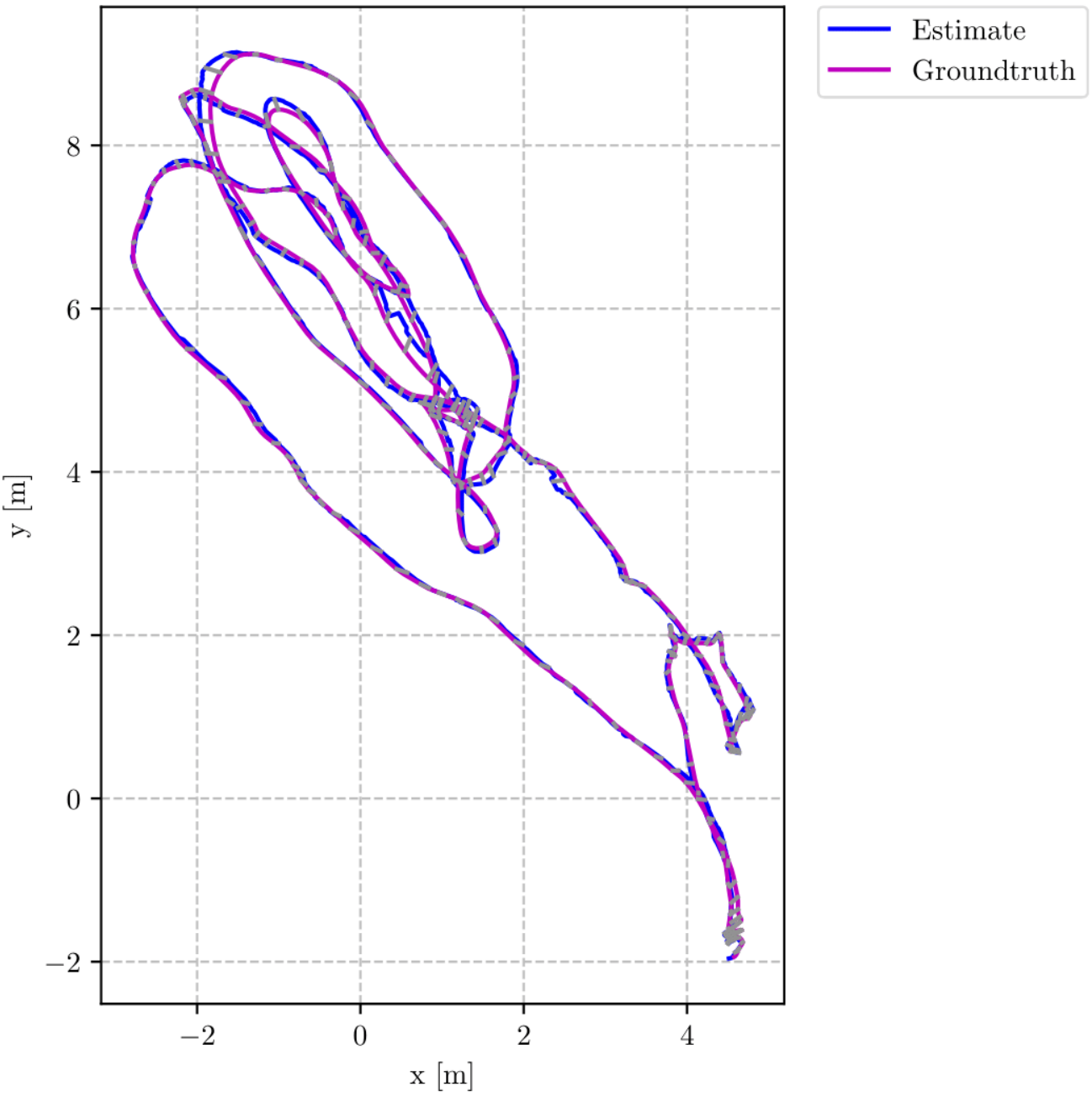

Fig. 7: Trajectory Error Top view

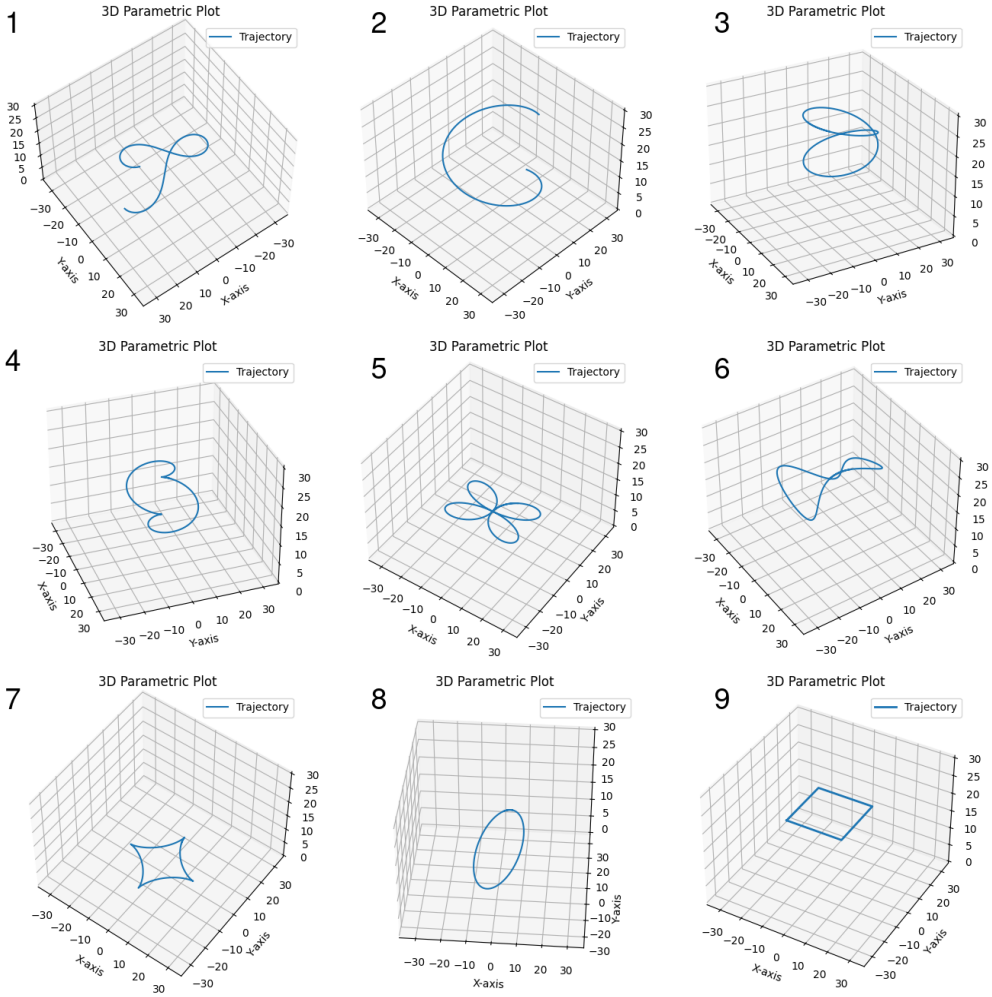

Fig. 8: Train and Test trajectories.

II. Phase II: Deep Visual-Inertial Odometry

Intro something..

A. Dataset Generation

A huge plane of dimensions 75×75 meters is placed on the ground and an image of 4K resolution is added to it. The image has uniformly distributed highly detailed features. The trajectories in the dataset are generated using a quadcopter model to get more realistic data considering the actual dynamics of a flying vehicle. A downward facing camera is attached to the quadcopter for capturing images of the ground plane. This is shown in Fig. 9

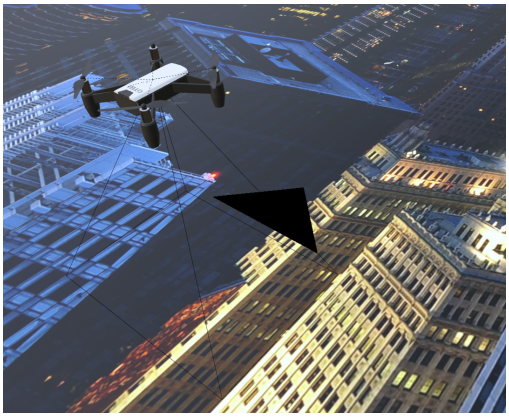

Fig. 9: DJI Tello with the FOV of the downward facing camera and the Plane.

Each simulation/trajectory is 100 seconds long with a time step of 0.01 seconds, totalling 10000 frames. The IMU data consists of 10000 readings, each per 1 frame. Camera data consists of 1000 images, each per 10 frames. Therefore, between every 2 images we get 10 IMU samples.

9 trajectories, consisting of 6 train and 3 test trajectories are chosen with both 2D and 3D shapes as shown in Fig. 8. The controller of the quadcopter takes in waypoints along the x, y, z axes as input. Each trajectory is a function of simulation time. The parametric equations of all trajectories give the position in x, y, z, which is sent to the controller. The dynamics of the quadcopter are updated each iteration.

To model the IMU sensor, we extract the linear and angular

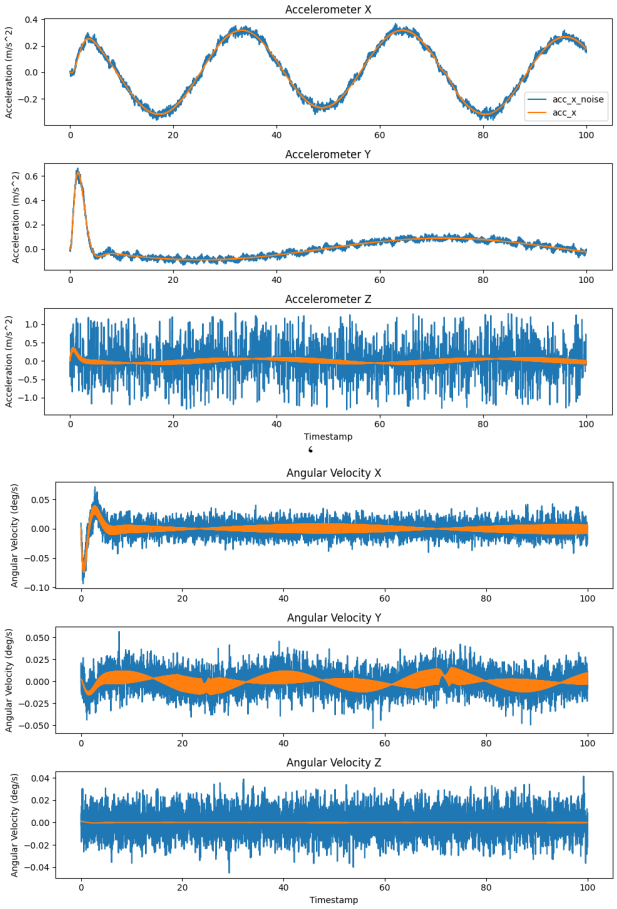

Fig. 10: Gaussian noise added to accelerations and angular velocities.

velocities from the state vector. To estimate the current linear accelerations, we integrate the linear velocities. The angular velocities are in quaternion form. We add Gaussian noise with zero mean and 0.01 as the standard deviation to model close to realistic data. This is shown in Fig. 10.

The input to the model are the relative translation and rotations between two consecutive image frames. Hence, we estimate the relative pose and render an image at the same exact iteration. The relative translation are estimated by simply subtracting the relative position and multiplying with the inverse of the rotation matrix of the previous orientation. The relative rotation is calculated by multiplying the inverse of the previous rotation matrix by the current rotation matrix. The rotation matrices are calculated using the in-built *scipy* functions from the quaternion values extracted from the state vector. The IMU, relative pose data and current timestamp at each iteration are appended to their respective *.csv* files.

B. Loss Function

In the initial phase of our project, we experimented with various loss functions to optimize the training of our models. Initially, we employed the L2 loss to measure the discrepancy in position predictions, paired with a cosine similarity loss for the quaternion orientations. However, we observed that the cosine similarity loss did not impose a stringent constraint necessary for precise orientation predictions.

To address this, we transitioned to a geodesic loss function, which is defined by the following formula:

$$\text{loss}(R_S, R_T) = \arccos\left(\frac{\text{tr}(R_S R_T^T) - 1}{2}\right)$$

Fig. 11: Geodesic loss function.

where R_T represents the true quaternion and R_S denotes the predicted quaternion. This loss function effectively captures the shortest path between two orientations on a unit sphere, making it more suitable for our application.

The geodesic loss yielded significant improvements in model performance, and as such, it was adopted across all our models. Geodesic loss measures the distance between points along the shortest path on the surface or manifold where the data points exist, rather than through the ambient (possibly higher-dimensional) space. This is crucial for data that inherently has a non-Euclidean structure, such as angles, rotations, or other manifold-valued data.

During later stages, we also explored the Bingham loss [5], specifically designed for smooth learning of quaternion orientations. This entailed modifying the output layer of the orientation head to accommodate 10 outputs. This is transformed into a matrix which can then be decomposed to get the quaternion. The Bingham loss function is formulated as follows:

$$\text{Bingham Loss} = 2d_i^2(4 - d_i^2)$$

where d_i is the unit quaternion difference. Although the Bingham loss performed well, it did not provide improvements over the geodesic loss in our specific use cases as we didn't have abrupt changes to the orientation which is where it is especially useful. Consequently, we continued utilizing the geodesic loss for the remainder of our project. Also, as the relative positions were incremental we found the L1 loss performed better hence we switched from L2 to L1 loss for the position. The losses will remain the same for all three networks.

C. Inertial Odometry

Inertial Odometry (IO) does not rely on visual inputs, making it suitable for environments where visual data is unreliable or unavailable, such as in smoky, foggy, or completely dark conditions. IO systems can be simpler to implement in terms of data processing compared to VO and VIO, as they only require the integration of inertial sensor data. Unlike VO and VIO, IO can continue to provide positional estimates even when the cameras are occluded or subjected to high-dynamic-range scenarios where camera sensors might fail.

For inertial odometry for tracking and navigation, we adopted a neural network architecture utilizing two bidirectional Long Short-Term Memory (LSTM) layers followed by

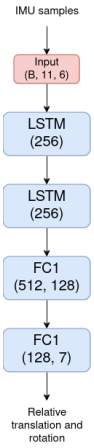

Fig. 12: Inertial odometry network architecture

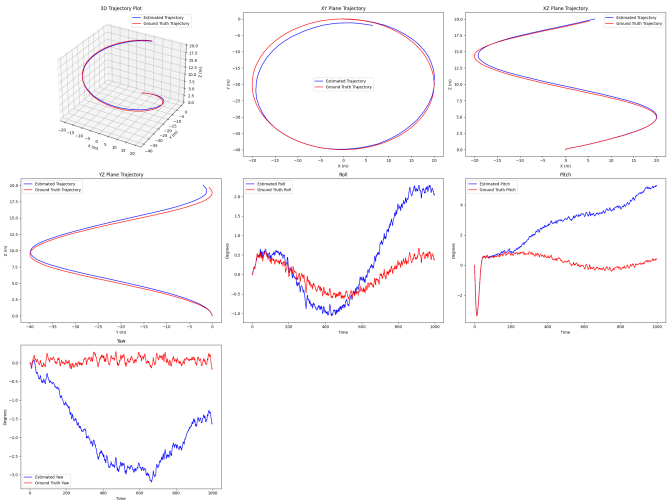

Fig. 13: IO network outputs on training

two linear layers [6]. Bidirectional LSTMs are particularly effective in this context as they process data points from both forward and backward directions, enhancing the learning of temporal dependencies within the sensor data.

During our experimental phase, we also evaluated the use of Gated Recurrent Units (GRUs) as an alternative to LSTMs. Although GRUs are computationally more efficient due to their simpler structure, our experiments demonstrated that LSTMs provided superior performance in terms of accuracy and stability in predicting complex motion patterns. Consequently, we decided to proceed with LSTMs for our final model implementation.

The detailed architecture of our inertial-only network is depicted in Figure 12. This diagram illustrates the sequential arrangement of the bidirectional LSTM layers, the following linear layer, and the data flow through the network. A comprehensive discussion of the performance metrics and results from testing of LSTM models is provided in Section **??**.

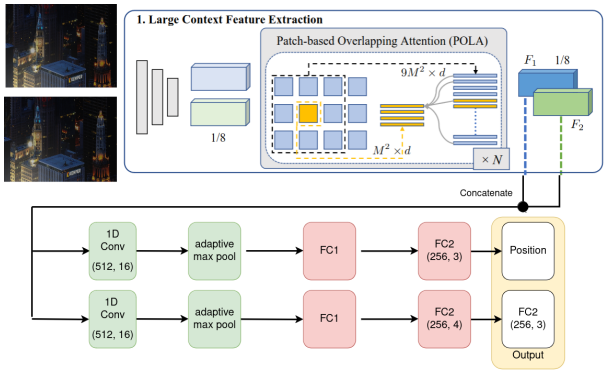

Fig. 14: Visual odometry network architecture

1) Training

The LSTM network is configured with the following key hyperparameters:

- **Optimizer:** The network uses the Adam optimizer with the following settings:
  Learning Rate (lr) = 0.001
  Epochs = 200
  Batch Size = 32
- **Input Data:** Each input sequence to the network consists of 11 frames of IMU data.
- **Output Prediction:** The network predicts the relative pose between the initial and final reading of the input sequence.

D. Visual Odometry

Visual Odometry works well in feature-rich environments. In environments where there are plenty of visual features to track, VO can provide high positional accuracy without the drift typically associated with inertial sensors over time.

Following the examples set by [7] and [8], we employ the encoder from an optical flow prediction network as our feature extractor to encode the information and differences between two images. Instead of opting for the conventional FlowNet 2.0 [9], we chose the feature extractor from GMFlowNet [10], noted for its robustness. During training, this feature extractor remains frozen, as it has been pre-trained on a substantially larger dataset. We focus our efforts on fine-tuning the other components of the network. Figure 14 illustrates the architecture of our visual odometry (VO) model, which is elaborated upon in the subsequent detailed summary.

Network Components

The network comprises several key components:

1) **Feature Network (FNet):** This component uses a `BasicConvEncoder` followed by a `POLAUpdate` block. The encoder outputs a 256-dimensional feature space, refined by the POLA Update mechanism [10], which employs overlapping attention for enhanced feature integration across the image space.
2) **Convolutional Layers:** Separate convolutional layers are used for initial feature reduction from the combined

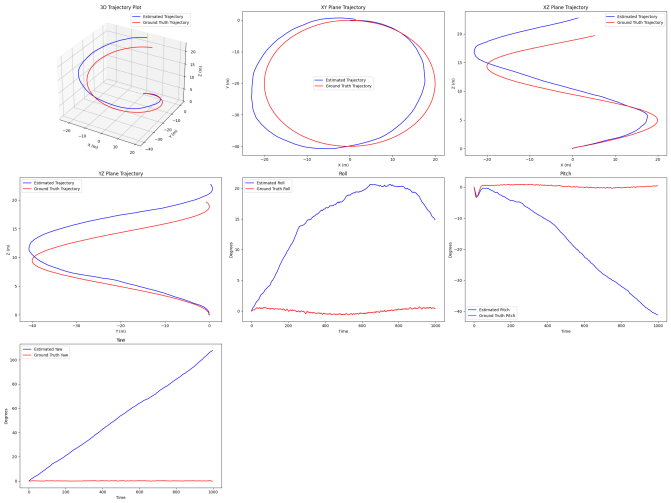

Fig. 15: VO network outputs on training

feature maps of two consecutive images, focusing independently on position and orientation data.

3) **Adaptive Max Pooling:** Post convolution, an adaptive max pooling layer condenses the spatial dimensions, preparing the data for the fully connected layers.
4) **Fully Connected Layers:** Flattened feature maps are processed through two stages of fully connected layers for each of position and orientation, finalizing the regression to 3D coordinates and orientation quaternions.
5) **Forward Pass** During the forward pass, the model takes pairs of images, normalizes them, and processes them through the FNet to generate feature maps. These maps are then concatenated and filtered through separate pathways for position and orientation, culminating in a prediction of relative pose between the two images.

1) Training

The training of our visual odometry (VO) network is strategically structured with several critical hyperparameters optimized to enhance performance:

- **Optimizer:** The network utilizes the Adam optimizer. The configuration specifics include:
  - Learning Rate (lr) = 0.001
  - Epochs = 50
  - Batch Size = 16
- **Input Data:** Input to the network consists of sequences of two consecutive image frames of the drone in a trajectory chosen randomly, deliberately chosen so that the model doesn't overfit the dynamic motion in a specific trajectory.
- **Output Prediction:** The network is trained to predict the relative pose between the initial and final frame of the pair of images.

This methodical training approach ensures that the VO network is robust and highly accurate in pose prediction, which is vital for effective navigation and mapping.

E. Visual Inertial Odometry

By combining visual data with inertial data, VIO systems can be more robust to different types of environmental conditions, such as low-light or feature-poor scenes where VO might struggle. The integration of visual data helps to reduce the drift common in standalone inertial systems, especially over longer periods or in faster motions. Inertial sensors provide immediate information about acceleration and orientation, which help in quickly initializing the system and maintaining robustness during temporary losses of visual information. VIO can handle dynamic motions better than VO because inertial sensors can rapidly sense changes in velocity and orientation.

Our Visual Inertial Odometry (VIO) model is structured to leverage attention mechanisms for enhancing the fusion of visual and inertial data, providing a robust framework for precise navigation and mapping. The model employs a self-attention mechanism, specifically designed to process concatenated visual and inertial data. This is similar to the approach utilized in [8].

Network Architecture

The architecture incorporates several components that are optimized for feature extraction and the effective integration of multimodal data:

- **Feature Network (FNet):** This remains the same as in the VO model to extract the visual features about the relevant information in the pair of images.
- **LSTM for Inertial Data:** Sequential inertial measurements are processed through an LSTM network, capturing temporal dynamics and providing critical temporal context to the inertial features.
- **Self-Attention Module:** A key element of our architecture, the `SelfAttentionModule`, applies multi-head attention to the concatenated feature set of visual and inertial data. This approach allows the model to dynamically prioritize and integrate features within the multimodal data, enhancing accuracy and robustness.
- **Convolutional and Pooling Layers:** Post-feature extraction, these layers reduce dimensionality and prepare the data for detailed processing in the fully connected layers.
- **Fully Connected Layers:** Positioned to process the refined and attention-enhanced features, these layers output the final predictions for position and orientation, ensuring high precision.

Self-Attention Mechanism

The aim of implementing self-attention in our VIO network is it should enhance its ability to handle the intricacies of visual and inertial data fusion:

- **Contextual Understanding:** Self-attention refines the network's grasp of spatial and temporal contexts, crucial for accurately predicting device orientation and trajectory by emphasizing key movement and orientation features.
- **Adaptive Integration:** This mechanism adaptively prioritizes visual or inertial data based on environmental conditions, ensuring robust navigation even when visual data is compromised (e.g., in low-light or cluttered spaces).

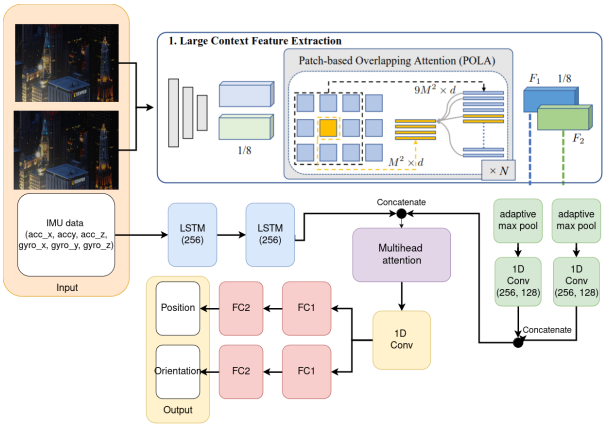

Fig. 16: Visual Inertial Odometry network architecture

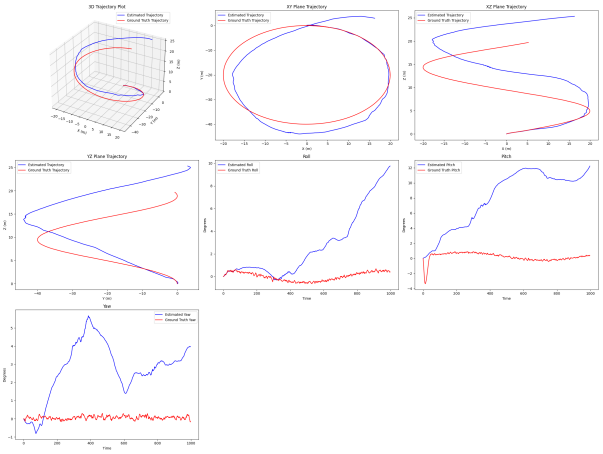

Fig. 17: VIO network outputs on training

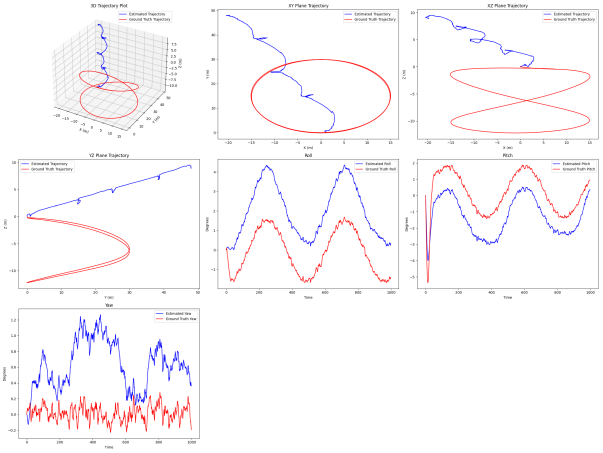

Fig. 18: IO network outputs on Test Set

- **Error Correction:** Self-attention aids in correcting inertial measurement drifts by leveraging visual cues, enhancing long-term navigation precision, and reducing cumulative errors.

Training

We utilized the following training recipe for the VIO model. The FNet in this case is also kept frozen.

- **Optimizer:** The network utilizes the Adam optimizer. The configuration specifics include:
  - Learning Rate (lr) = 0.001
  - Epochs = 50
  - Batch Size = 16
- **Input Data:** The network is trained on concatenated pairs of image frames and the corresponding sequences of IMU data to ensure that it learns to interpret both these data and weigh each one correctly to predict the camera pose.

This refined architecture not only supports robust feature extraction and data integration but also ensures that the VIO network can reliably predict the motion and orientation in complex and dynamically changing environments.

F. Experiments & Results

Train and test on the same trajectory: To first test, if the networks were capable of learning from the data, we first trained it on 70% of the trajectory and tested it on 30% of the trajectory. All three networks were able to learn on this. After this sanity check we move to train the models on multiple trajectories.

Train on multiple trajectories and test on unseen trajectories:

We choose 5 trajectories from the dataset specifically: 1,2,4,7,8. The model is able to learn all the training trajectories as seen from Fig. 13, 17, and 15. We test the trained model on the four test trajectories 3,5,6,9.

1) IO

Following are the mean RMSE ATE numbers for the test set. The plot for one such instance can be seen in Fig. 18

- RMSE ATE: 23.12 m
- Absolute Median Translation Error: 20.21 m

2) VO

Following are the mean RMSE ATE numbers for the test set. The plot for one such instance can be seen in Fig. 19.

- RMSE ATE: 24.56 m
- Absolute Median Translation Error: 26 m

3) VIO

Following are the mean RMSE ATE numbers for the test set. The plot for one such instance can be seen in Fig. 20.

- RMSE ATE: 24.72 m
- Absolute Median Translation Error: 29.11 m

G. Discussions

The models are not able to generalize to the test set. This might be attributed to the smaller size of the dataset.

H. Classical and deep VIO research problems.

1) Robustness to Dynamic Environments

Problem: Many VIO systems struggle in dynamic environments where objects move independently of the static background.

Research Idea: Develop methods that can distinguish between

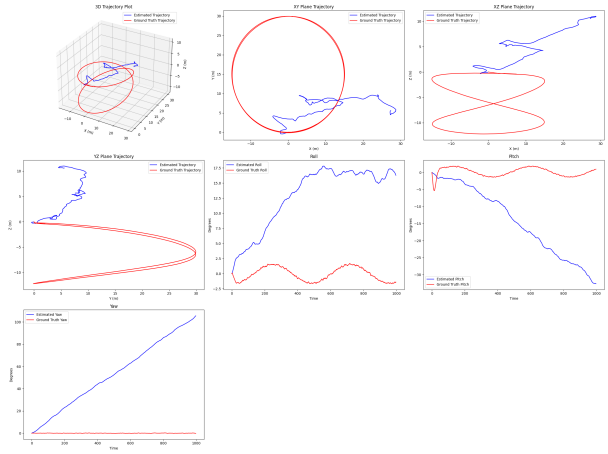

Fig. 19: VO network outputs on Test Set

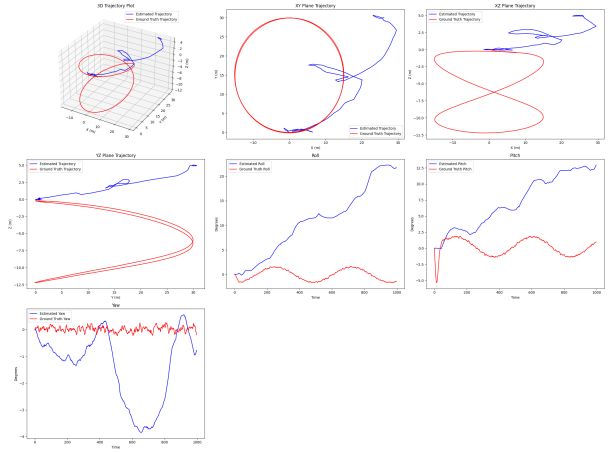

Fig. 20: VIO network outputs on Test Set

static and dynamic objects and adjust the odometry calculations accordingly. For classical approaches, this might involve more sophisticated scene segmentation and motion detection techniques. For deep learning approaches, training models on datasets with dynamic objects could help in automatically recognizing and categorizing elements in the scene.

2) Low-light and Nighttime Navigation

Problem: Visual sensors often perform poorly in low-light conditions, leading to decreased accuracy in VIO systems as discussed earlier.

Research Idea: Enhance sensor capabilities in low-light conditions using advanced image processing techniques or by integrating thermal or infrared cameras into the VIO system. Deep learning models could be trained on nighttime data or could use generative models to simulate well-lit conditions from low-light images.

3) Handling Rapid Motion and High-Speed Dynamics

Problem: Rapid motion can cause motion blur in cameras and rapid changes in inertial sensor readings, which can degrade the performance of VIO systems.

Research Idea: Investigate new methods for motion blur compensation in images and techniques for filtering or adapting to rapid changes in IMU data. Machine learning approaches might include training on blurred images or using recurrent neural networks to better handle temporal dynamics.

4) Scale Drift Correction

Problem: Over time, VIO systems can suffer from scale drift, where the estimated trajectory gradually diverges from the true scale of the environment.

Research Idea: For classical approaches, improved loop closure techniques could be investigated. For deep learning methods, training on longer sequences or incorporating scale-invariant features might help correct or mitigate scale drift.

5) Multi-modal Sensor Fusion

Problem: Combining data from multiple types of sensors can enhance the robustness and accuracy of VIO systems but also introduces complexity in data synchronization and integration.

Research Idea: Explore advanced fusion techniques that can effectively combine visual data with data from multiple IMUs, GPS, or other sensors. Deep learning methods could involve multi-input neural networks that learn to integrate and weigh information from various sources dynamically.

REFERENCES

[1] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J. Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight, 2018.

[2] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.

[3] Kumar Robotics. MSCKF_VIO: Multi-State Constraint Kalman Filter for Visual-Inertial Odometry.

[4] Robotics and Perception Group. RPG Trajectory Evaluation. https://github.com/uzh-rpg/rpg$_t rajectory_e valuation$, 2023. $Accessed$: $2023-04-13$.

[5] Valentin Peretroukhin, Matthew Giamou, David M. Rosen, W. Nicholas Greene, Nicholas Roy, and Jonathan Kelly. A smooth representation of belief over SO(3) for deep rotation learning with uncertainty. *CoRR*, abs/2006.01031, 2020.

[6] Changhao Chen, Peijun Zhao, Chris Xiaoxuan Lu, Wei Wang, Andrew Markham, and Niki Trigoni. Oxiod: The dataset for deep inertial odometry, 2018.

[7] Li Liu, Ge Li, and Thomas H Li. Atvio: Attention guided visual-inertial odometry. In *ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4125–4129, 2021.

[8] Zheming Tu, Changhao Chen, Xianfei Pan, Ruochen Liu, Jiarui Cui, and Jun Mao. Ema-vio: Deep visual–inertial odometry with external memory attention. *IEEE Sensors Journal*, 22(21):20877–20885, 2022.

[9] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *CoRR*, abs/1612.01925, 2016.

[10] Shiyu Zhao, Long Zhao, Zhixing Zhang, Enyu Zhou, and Dimitris Metaxas. Global matching with overlapping attention for optical flow estimation, 2022.