P4: Deep VIO(Phase 2)

1st Venkateshkrishna
Masters in Robotics
Worcester Polytechnic Institute
Worcester, MA 01609
vparsuram@wpi.edu

2nd Mayank, Bansal
Masters in Robotics
Worcester Polytechnic Institute
Worcester, MA 01609
mbansal1@wpi.edu

***Abstract*—Phase 2 of this project involves using deep learning to estimate odometry using Visual and Inertial data. The project is done in three parts: Inertial-only odometry, Vision-only odometry and finally, Visual-Inertial Odometry. The dataset was generated using MATLAB IMU framework and also, alternatively, using the OysterSim framework by Professor Sanket. The final predicted and ground-truth trajectory along with loss values are reported for each.**

I. Phase 2: Deep Learning Approach

A. Dataset Generation

The IMU dataset was generated using the MATLAB IMU framework which requires the waypoints, the sample rate of IMU, and the times at which each waypoint is arrived at. This then generates the corresponding position, orientation, acceleration and angular velocity values for each timestamp using interpolation. This data is then passed into the imu object to obtain the corresponding accelerometer and gyroscope values. The IMU dataset was generated at a 1000Hz sample rate and the trajectories are 10 seconds each. The position and orientation data of each trajectory were used to obtain image data from Blender at a 100Hz. A number of different types of trajectories were generated including the 'figure8', 'tiltedfigure8', '3Dfigure8', oval, and other random open-loop trajectories.

We also utilized Blender alongside the OysterSim simulation package to generate an alternative dataset for further validation of our networks. The process began with random sampling of points within the workspace. Assuming a maximum drone velocity of 2 m/s, we fitted a quintic polynomial trajectory through these sampled points. Subsequently, we sampled these trajectories to obtain points paired with timestamps. These data points were then input into oyster sim, which controlled the robot's movement in Blender. We set the IMU sample rate at 300 Hz and the camera frame rate at 30 Hz to simulate realistic drone behavior.

B. Loss Function

For position estimation, we commonly use the Root Mean Squared Error (RMSE) to quantify the discrepancy between predicted and actual values. The RMSE is defined as follows:

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

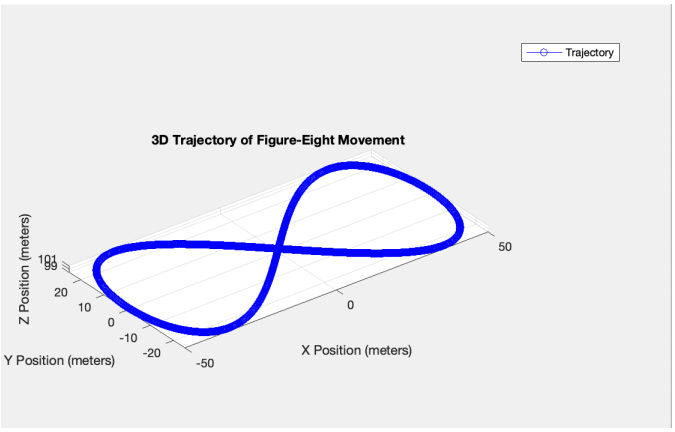

Fig. 1: Trajectory of figure 8

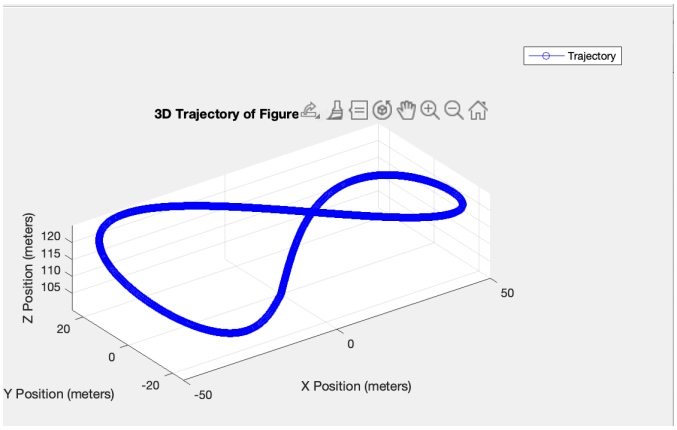

Fig. 2: Trajectory of 3D figure 8

where $\hat{y}_i$ represents the predicted values, y_i represents the actual values, and n is the number of samples.

For quaternions, which represent orientations, the loss is calculated using the mean squared error of the angle between the predicted quaternion $\hat{q}$ and the true quaternion q. This loss aims to minimize the angular difference between them and is defined as:

$$L(q, \hat{q}) = 1 - \langle q, \hat{q} \rangle^2 \tag{2}$$

Here, $\langle q, \hat{q} \rangle$ denotes the dot product between the two quaternions, ensuring that the loss is minimized when the orientation predicted by the model aligns closely with the true orientation.

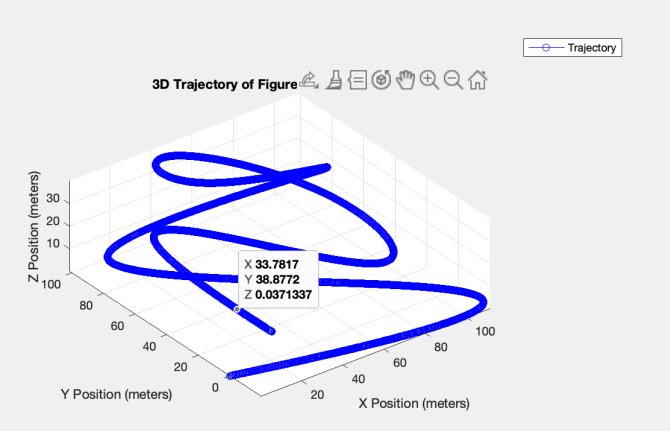

Fig. 3: Random Trajectory

C. Deep IO

Our neural network architecture consists of two fully connected layers followed by two LSTM layers, which are designed to handle the temporal sequences inherent in our dataset. The output from the LSTM layers is then processed through one additional fully connected layers. The network accepts input in the form of a 10x6 matrix, where '10' represents the sequence or window size, and '6' corresponds to IMU data—specifically, accelerometer and gyroscope readings across three axes.

The network's output is a 1x7 vector. The first three elements of this vector represent the position increment relative to the previous position, while the next four elements reflect changes in orientation, expressed as a quaternion, from the previous orientation. The use of LSTM layers is crucial for our model as it needs to learn from sequences of data, capturing temporal dynamics effectively. This setup allows the network to predict changes in both position and orientation based on the sequential IMU input.

D. Deep VO

In our Visual Odometry system, we concatenate two consecutive images channel-wise to form the input for the network. The output structure mirrors that of the Inertial Odometry (IO) network, producing a 1x7 vector that captures the relative position and orientation changes.

The image input undergoes initial processing through two convolutional neural network (CNN) layers. Following each CNN layer, we apply a ReLU activation function to introduce non-linearity, perform max-pooling to reduce spatial dimensions while retaining important features, and conduct batch normalization to stabilize and accelerate training.

After processing through the CNN layers, the data passes through two fully connected layers. Internally, the network estimates the homography between the two consecutive images. This homography is then utilized to calculate the odometry, effectively estimating the movement and orientation changes based on visual inputs. This method allows for precise tracking

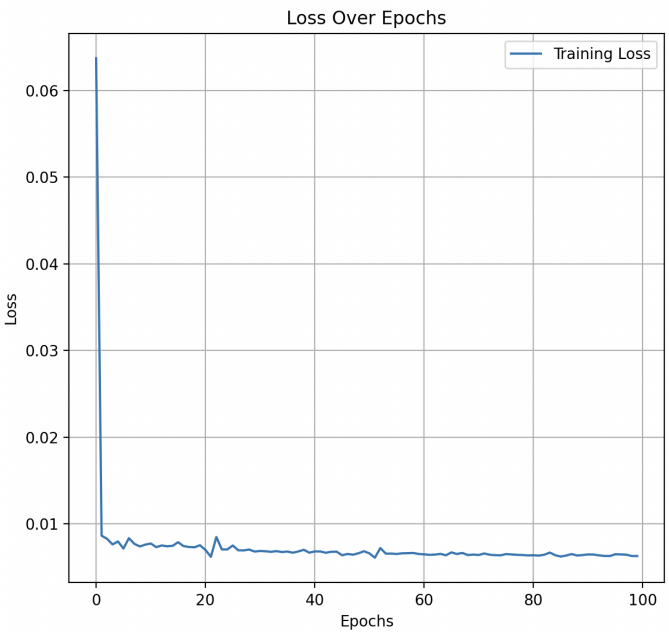

Fig. 4: Training loss for Deep IO

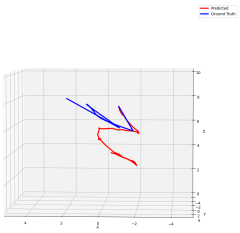

Fig. 5: Predicted vs Ground-truth IO output

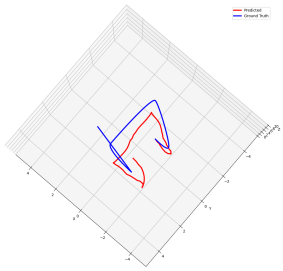

Fig. 6: Predicted vs Ground-truth IO output(top view)

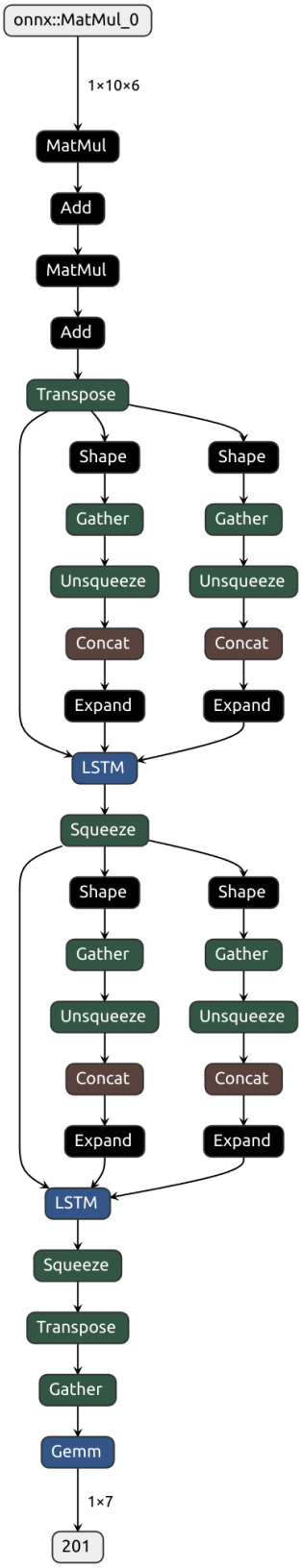

Fig. 7: Network architecture for Deep IO

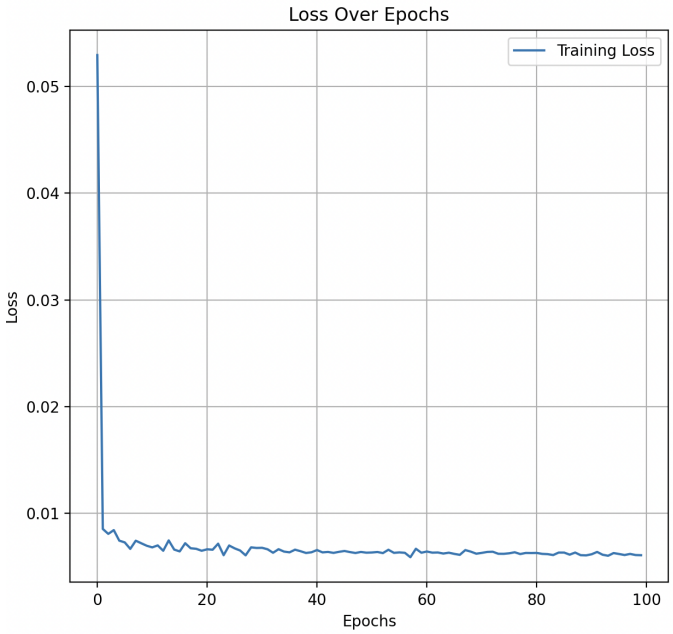

Fig. 8: Training loss for Deep VO

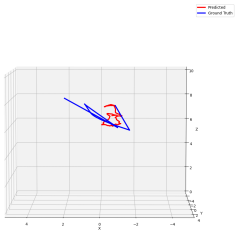

Fig. 9: Predicted vs Ground-truth VO output

of the device's trajectory in environments where other sensory data might be limited or unreliable.

E. Deep VIO

In our Visual-Inertial Odometry (VIO) system, we integrate the methodologies of the Visual Odometry (VO) and Inertial Odometry (IO) networks to enhance the accuracy and robustness of our trajectory estimation. The input consists of two consecutive images alongside the IMU data recorded between these images. The output, like in the individual systems, is a 1x7 vector detailing relative position and orientation changes.

The processing begins by feeding the images into the VO network and the IMU data into the IO network separately. Each network processes its respective inputs and produces its own 1x7 output vector. These vectors from both networks are then concatenated to form a preliminary 1x14 vector.

This concatenated vector is subsequently processed through two fully connected layers. These layers integrate the visual and inertial information, refining the fusion to produce a final,

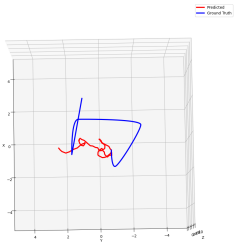

Fig. 10: Predicted vs Ground-truth VO(top view)

precise 1x7 output vector that encapsulates both positional and orientational estimates. This fusion approach leverages the strengths of both sensory modalities, ensuring more reliable and accurate odometry under varying conditions.

F. Trajectory Error Evaluation

The Absolute Median Trajectory Error (ATE) was measured at 56.863 meters, and the Root Mean Square Translation Error (RMSE) was recorded at 60.235 meters.

G. Conclusion

This project explored our efforts to perform deep learning based Inertial odometry, visual odometry and finally visual-inertial odometry. The results were good, especially for IO. We identified a few problems that we might work on

REFERENCES

[1] Deep VIO home page: link
[2] MATLAB IMU model: link
[3] OysterSim: link

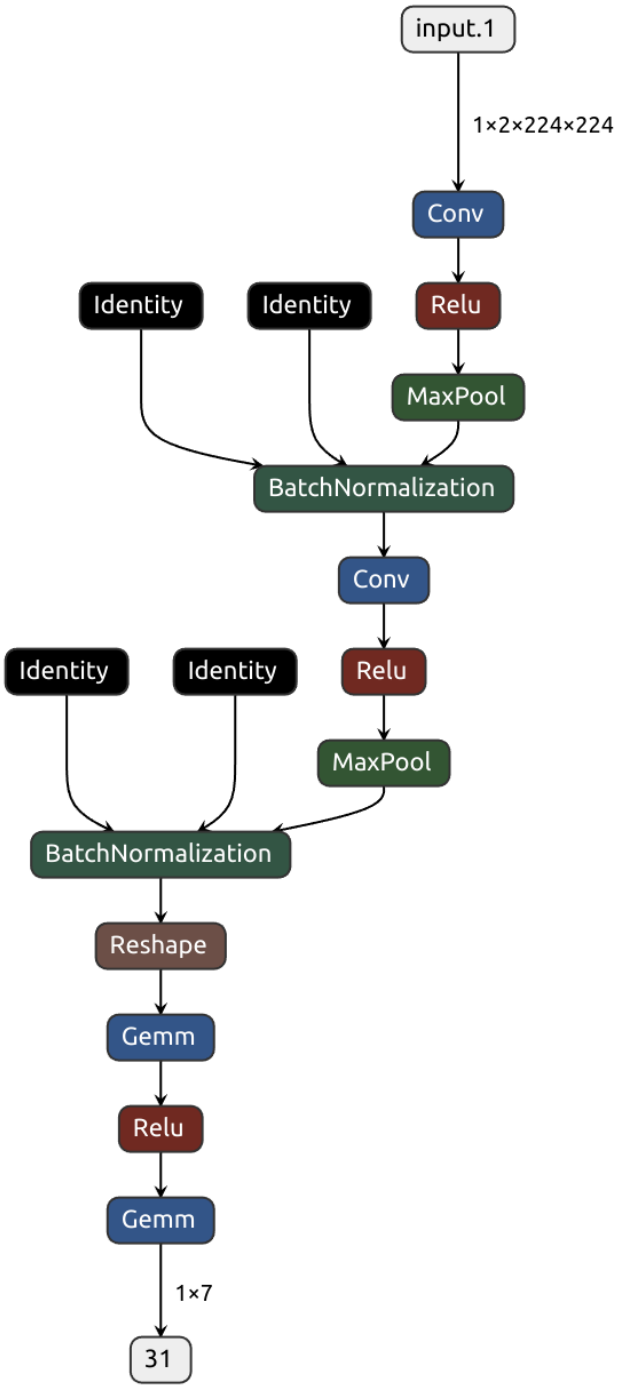

Fig. 11: Network architecture for Deep VO

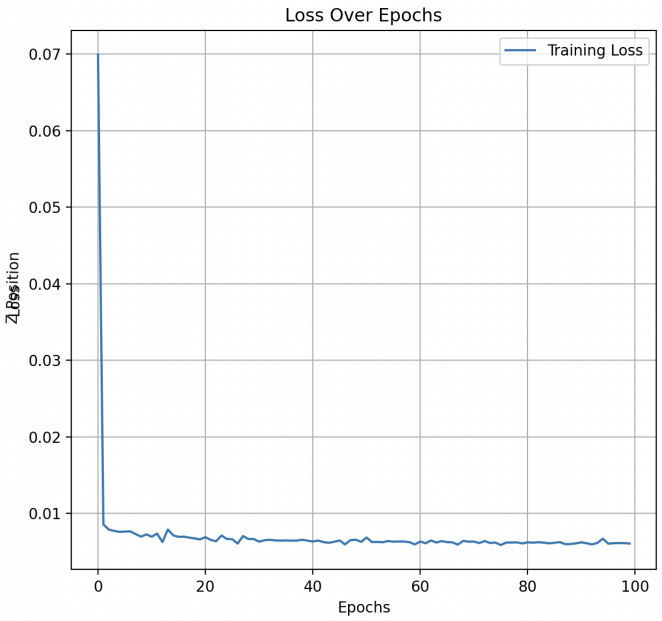

Fig. 12: Training loss for Deep VIO

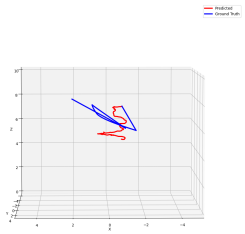

Fig. 13: Predicted vs Ground-truth VIO(top view)

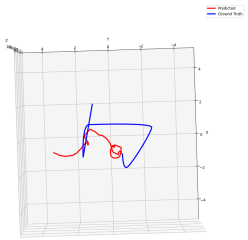

Fig. 14: Predicted vs Ground-truth VIO(top view)

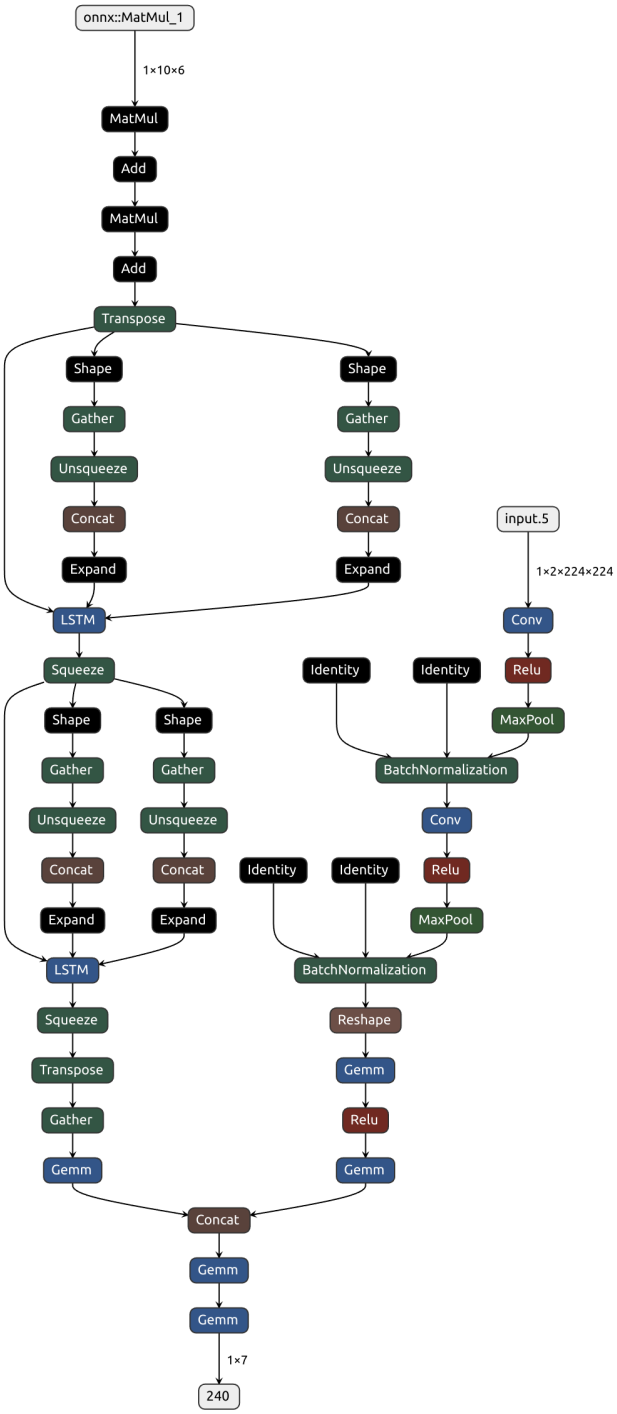

Fig. 15: Network architecture for Deep VIO