RBE 549 Project 4 Phase 2: Deep Visual Inertial Odometry

Aabha Tamhankar
Masters in Robotics Engineering
Worcester Polytechnic Institute
astamhankar@wpi.edu

Miheer Diwan
Masters in Robotics Engineering
Worcester Polytechnic Institute
msdiwan@wpi.edu

***Abstract*—This paper presents a deep learning-based approach to compute visual and inertial odometry for a Micro Aerial Vehicle (MAV) using the publicly available Euroc dataset. We propose a novel architecture that combines convolutional neural networks (CNN) and long short-term memory (LSTM) networks to learn spatiotemporal features from the visual and inertial data streams, respectively. The CNN and LSTM networks are trained independently on the visual and inertial data streams, and the outputs can be fused using a dense neural network to estimate the MAV's pose. We evaluate the proposed approach on the Euroc dataset.**

I. Introduction

In recent years, there has been a significant increase in the use of unmanned aerial vehicles (UAVs) in various applications, such as mapping, surveillance, and search and rescue missions. Accurate and robust estimation of the UAV's position and orientation is crucial for the success of these applications. Visual and inertial odometry (VIO) is a popular technique for estimating the pose of a UAV using camera and inertial sensor measurements. However, VIO can suffer from drift and instability due to various factors such as sensor noise and calibration errors. To address these issues, deep learning methods have been applied to improve VIO performance by leveraging the power of neural networks to learn robust feature representations and model complex nonlinear relationships. Due to advances in robotics in the field of aerial vehicles, state estimation is crucial for gaining pose and achieving stability of the robot while in flight. The combination of visual information from cameras and measurements from an Inertial Measurement Unit (IMU) is referred to as Visual Inertial Odometry (VIO)[1]. This project presents an implementation of VIO using deep learning methods. Our approach combines visual and inertial measurements using a deep neural network architecture that takes advantage of the complementary information provided by these sensors. Specifically, we use a convolutional neural network (CNN) to process the visual measurements and a long short-term memory (LSTM) network to process the inertial measurements. The work demonstrates the potential of deep learning in improving VIO performance.

A. Dataset

The data was collected on board a Micro Aerial Vehicle (MAV). The datasets contain stereo images, synchronized IMU measurements, and accurate motion and structure ground truth. It is a subset of the EuRoC dataset and the ground truth is provided by a sub-mm accurate Vicon Motion capture system. The Machine Hall 02 easy (MH 02 easy), Machine Hall 03 medium (MH 03 medium), Machine Hall 04 difficult (MH 04 difficult), and Machine Hall 05 difficult (difficult) subsets of the EuRoC dataset for training the neural network. The Machine Hall 01 easy dataset was used as test data to predict the results.

II. Visual Odometry

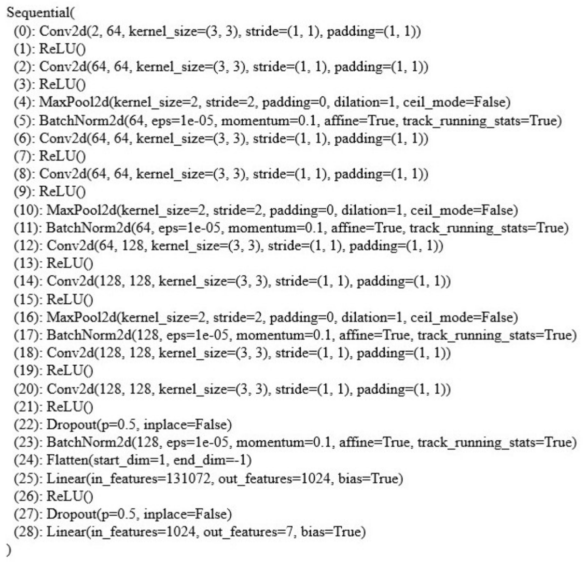
Visual Odometry is the process of determining the equivalent odometry information using sequential camera images. Our approach was to estimate the relative pose between two images using a simple CNN (Convolutional Neural Network). In this section, we will talk about the implementational details, the issues faced during implementation, and our brief analysis of said issues. The results and output trajectory comparison are illustrated in 8

A. Training Dataset and Ground Truth

We referred to [6] for relative pose estimation from two images and trained multiple models with different training datasets and tested against the MH 01 Easy Dataset from the EuRoC dataset. The model was trained primarily on two datasets: (1) MH 02 Easy, (2) MH 02, 03, 04, and 05 combined. For the combined dataset, the model was trained on one dataset first, and the checkpoints were saved. The training was then continued on the remaining datasets by loading the checkpoints from the previous training session. Because we were predicting the relative camera pose, the ground truth labels needed to be modified and the images had to be resized. The network takes two consecutive images of size $256 \times 256 \times 1$ as input and the difference between the pose of the respective images as the ground truth labels. The pose has translation in p_x, p_y, p_z and the orientation denoted by q_w, q_x, q_y, q_z. One major issue during this process was that the data rates of the IMU and the camera were different. To fix this, we simply took the common ground truth data from the image timestamps available.

B. Network Architecture and Hyperparameters

The network used is a simple CNN with eight convolutional layers and the activation function used was ReLU. There are max pooling and dropout layers between the convolutional layers and two fully connected layers at the very end of the model. The model outputs 7 values which are the predicted pose ($\hat{p}$). These values are extracted and stored as they will be used for the visual-inertial fusion part of this project. The input was two stacked consecutive images of size $256 \times 256 \times 1$ and their ground truth values. The batch size used was 32 and the learning rate was 1e-4.



```
Sequential(
  (0): Conv2d(2, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU()
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU()
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (6): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (7): ReLU()
  (8): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (9): ReLU()
  (10): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (11): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (12): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (13): ReLU()
  (14): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (15): ReLU()
  (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (17): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (18): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (19): ReLU()
  (20): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (21): ReLU()
  (22): Dropout(p=0.5, inplace=False)
  (23): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (24): Flatten(start_dim=1, end_dim=-1)
  (25): Linear(in_features=131072, out_features=1024, bias=True)
  (26): ReLU()
  (27): Dropout(p=0.5, inplace=False)
  (28): Linear(in_features=1024, out_features=7, bias=True)
)
```

1) Loss Function: Initially, we used a simple MSE loss to regress the pose. However, after referring to [6], we updated our loss function to account for the orientation too. The updated loss function used is:

$$loss(I) = \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \left\| \hat{\mathbf{q}} - \frac{\mathbf{q}}{\|\mathbf{q}\|} \right\|_2$$

C. *Output and Inference*

The model loss converged after it was trained for 20 epochs as seen in 1. The average training loss after 20 epochs was found to be $2.4 \times 1e - 4$. Despite the loss converging, the model produced very inaccurate results on the validation and test dataset. We believe that this was primarily because of three reasons:

- The model was overfitting during training. This is highly plausible as the loss converged within 10 training epochs and there was minimal improvement over the next 10 epochs.
- Another possible reason for the inaccuracies could be that the orientation biases were not scaled properly.

To plot the trajectories of the predicted camera pose, we also overfit the model on the test data itself. The results of this can be seen in Fig. 8. The major assumption of this approach was that the neural network is able to learn to predict the relative poses and biases.

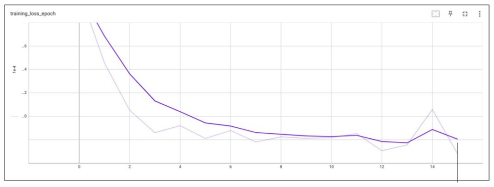


Fig. 1: Training Loss Over Epochs

III. Inertial Odometry

Inertial Odometry (IO) is a technique used in robotics and navigation systems to estimate the position and orientation of a moving vehicle or object using measurements from an inertial measurement unit (IMU), which includes accelerometers and gyroscopes. Deep learning has been applied to IO to improve its accuracy and robustness in challenging environments.

A. *Reference Network*

The reference network, and data loader functions were taken from [4]. This paper proposes a deep neural network for estimating the pose (position and orientation) of a mobile device based on inertial measurements from its internal sensors (IMU). The proposed network architecture comprises three main components: feature extraction, pose estimation, and recurrent refinement.

- Feature Extraction: The input to the network is a sequence of raw IMU measurements over a fixed time interval. The first component of the network consists of two parallel convolutional layers that extract spatial and temporal features from the input data. The spatial features are extracted using 1D convolutions, while the temporal features are extracted using a 2D convolutional layer that operates on the entire sequence of measurements.
- Pose Estimation: The second component of the network takes the features extracted by the first component and uses them to estimate the device's pose. This component consists of several fully connected layers, followed by two parallel branches that predict the position and orientation separately. The position branch outputs a 3D position vector, while the orientation branch outputs a quaternion.
- Recurrent Refinement: The final component of the network refines the pose estimates using a recurrent neural network (RNN). The RNN takes the predicted pose from the second component and the raw IMU measurements as input and outputs an updated pose estimate. The RNN is designed to capture the temporal dynamics of the IMU data and to refine the pose estimate over time.

The network proposed in the paper is illustrated in 2

The network is trained end-to-end using a combination of supervised and unsupervised learning. The supervised learning component uses ground truth pose data to train the network to predict accurate pose estimates. The unsupervised learning component uses a sequence-to-sequence autoencoder to learn a compressed representation of the IMU data, which is used as an additional input to the RNN during training. The network

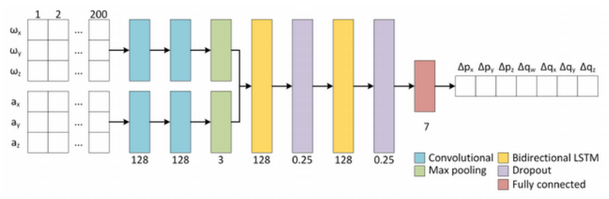


Fig. 2: Reference Network

is evaluated on a dataset of real-world IMU measurements, and the results show that it outperforms several state-of-the-art methods for IMU-based pose estimation.

B. Network 1

The first network used was very basic, as illustrated in 3

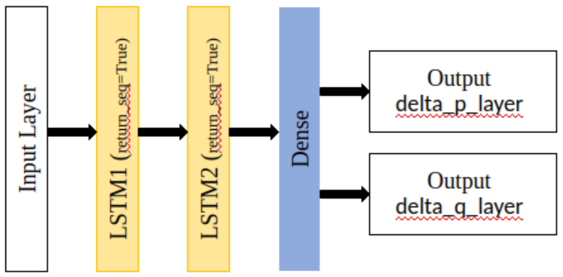


Fig. 3: Trial Network 1

The input layer is defined with the specified shape, and the first LSTM layer is defined with 64 units and returnsequences=True, meaning that it outputs a sequence of vectors that correspond to the input sequence. The second LSTM layer also has 64 units but returnsequences=False, meaning that it outputs a single vector that summarizes the information in the input sequence.

A dense layer with 64 units is added on top of the LSTM layers, which allows the network to learn non-linear relationships between the LSTM outputs and the output labels. Finally, there are two output layers: one for deltap with 3 units, and one for deltaq with 4 units. These output layers are connected to the dense layer, which provides the final predictions for the task.

The results and output trajectory comparison are illustrated in 9

C. Network 2

There are various techniques that can be used to improve the accuracy of the network for getting pose from IMU data. Regularization techniques like dropout or normalization can help prevent over-fitting of the network. This can help the network generalize better to unseen data, which can improve accuracy. The second network implements these techniques to get better output results. It is illustrated in 4

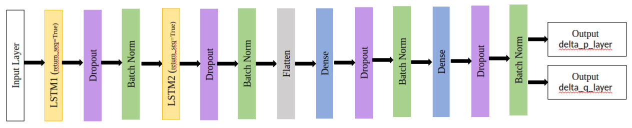


Fig. 4: Trial Network 2

The input layer is defined with the shape of the input data, and the first LSTM layer is defined with 128 units and set to return sequences. A dropout layer with a rate of 0.2 is added to the output of the first LSTM layer to prevent overfitting, followed by a batch normalization layer.

The second LSTM layer is defined with 64 units and also set to return sequences. The same process of adding a dropout layer and a batch normalization layer is repeated after the second LSTM layer.

After flattening the output of the second LSTM layer, two dense layers with 128 and 64 units, respectively, are defined with ReLU activation. Dropout and batch normalization layers are added to both dense layers to prevent overfitting.

Finally, two output layers are defined with 3 and 4 units, respectively, for deltap and deltaq. These output layers take the output of the last batch normalization layer as input.

Overall, this model uses the power of LSTM layers to capture the temporal dependencies in the input data and produces accurate predictions of deltap and deltaq.

The results and output trajectory comparison are illustrated in 10

IV. Visual Inertial Odometry

In this section, we combined the results from the Visual and the Inertial Odometry methods. The predicted poses from the Visual Odometry and the Inertial Odometry were stored in CSV files as translation and orientation (in quaternions). Only the common timestamps for the camera and IMU data were used. For this task, we used two main ideas:

- Weighted average Method: We matched the predicted poses based on the time stamps of the images and the IMU readings. We then proceeded to take a simple weighted average of the data to get the output of Visual Inertial Odometry.
- Deep Learning Method: The second approach that we used incorporated a fully connected neural network to fuse the predicted poses. The input for this model was just the predicted poses. However, some state-of-the-art implementations to solve VI Odometry also use the original image data and the ground truth data for this problem.

V. Results

The trajectory computed by the classical approach was plotted against a video of the aerial vehicle in flight using the given in Machine Hall 01 easy dataset which is a subset of the EuRoC dataset. The trajectory was generated using OpenGL and Pangolin as demonstrated in [3], and can be seen in Fig.5. This trajectory is close to accurate to the ground truth trajectory.

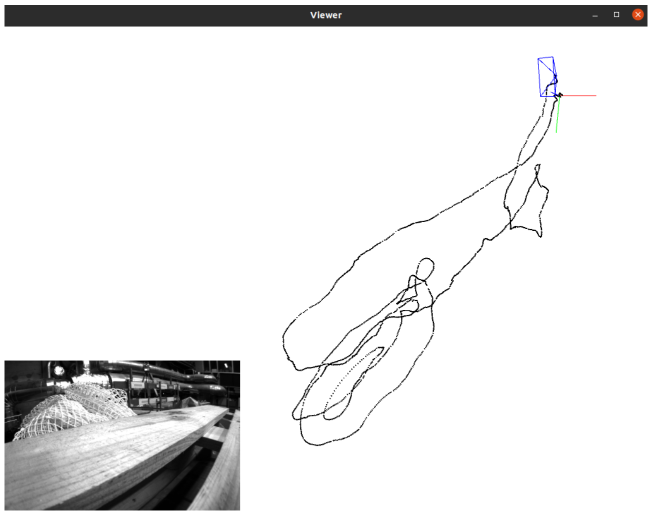


Fig. 5: Output of S-MSCKF for EuRoC MH 01Easy Using Pangolin

This trajectories can be easily compared with the ground truth and estimated errors can be calculated. The "data.csv" file from the given dataset contains the transformations and quaternions for each timestamp in the flight. This file can be used as the data for the ground truth trajectory. A similar "estimated.txt" file is generated through the given starter code, which is all the estimated values of transformations and quaternions. Using these two files in the format required by [4], the plots for estimated against ground-truth can be generated.
The compared trajectories for classical approach are illustrated in Fig.6 and Fig.7. They are close to the trajectory of ground truth. This method also provides rotations and translation errors in the form of graphs.

The resulting trajectory evaluation using Visual Odometry and Inertial Odometry (for both network 1 and network 2) are given below.

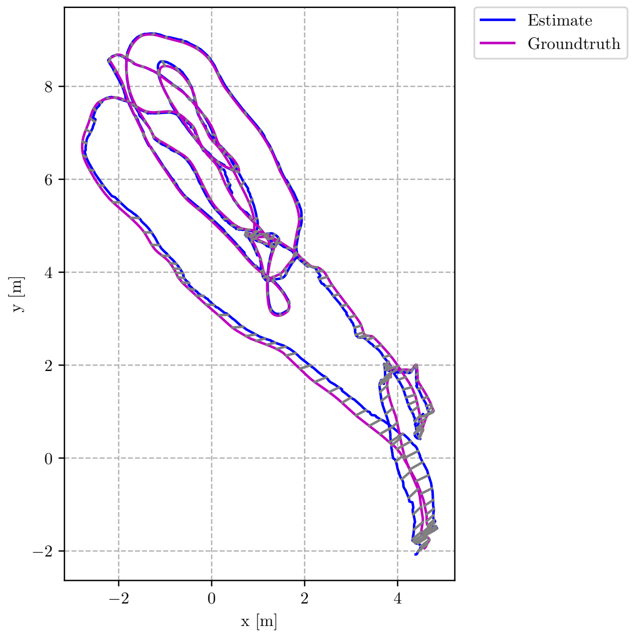


Fig. 6: Evaluation of output of S-MSCKF with respect to ground truth for EuRoC MH01Easy

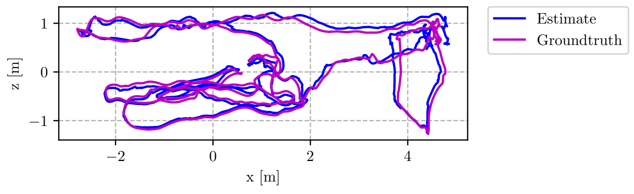


Fig. 7: Evaluation of output of S-MSCKF with respect to ground truth for EuRoC MH01Easy

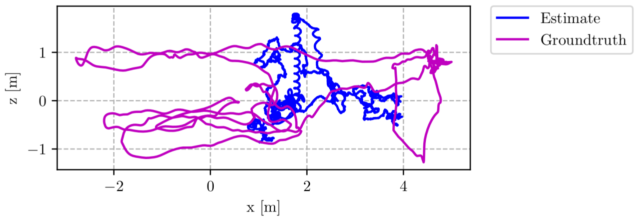


Fig. 8: Evaluation of output of with respect to ground truth for EuRoC MH 01 Easy using Visual Odometry

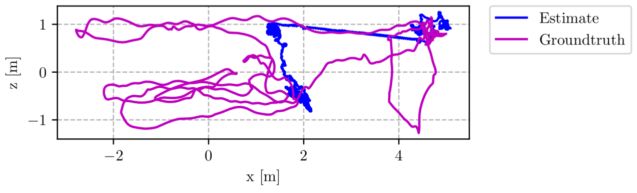


Fig. 9: Evaluation of output of with respect to ground truth for EuRoC MH01Easy using Inertial Odometry Network 1

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."
[2] Anastasios I. Mourikis and Stergios I. Roumeliotis, "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation"
[3] https://github.com/uoip/stereo$_msckf$
[4] https://github.com/uzh-rpg/rpg$_trajectory_evaluation$
[5] João Paulo Silva do Monte Lima, Hideaki Uchiyama 3and Rin-ichiro Taniguchi, "End-to-End Learning Framework for IMU-Based 6-DOF Odometry", 31 August 2019.

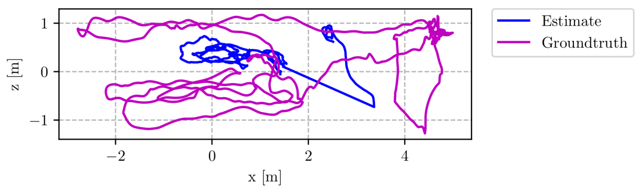


Fig. 10: Evaluation of output of with respect to ground truth for EuRoC MH01Easy using Inertial Odometry Network 2

[6] PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization

[7] DiffPoseNet: Direct Differentiable Camera Pose Estimation