# P4-Phase2
# Deep and Un-Deep Visual Inertial Odometry

Smit M Shah
Email: smshah1@wpi.edu
Worcester Polytechnic Institute

Rigved Sanku
Email: rsanku@wpi.edu
Worcester Polytechnic Institute

## I. Phase-1 : Traditional Approach

### A. Introduction

The main goal of this paper is to implement and recreate the Stereo Multi-State Contraint Kalman Filter(MSCKF). The following functions of the MSCKF python implementation were changed initialize_gravity_and_bias, batch_imu_processing, process_model, predict_new_state, state_augmentation, add_feature_observations, measurement_update and predict_new_state.

### B. Initialize Gravity and Bias

*1) Gyroscope Initialization:* The gyroscope measures the rate of rotation around the IMU's axes. However, even when the IMU is stationary, the gyroscope might show some readings due to bias. We average the angular velocity of initial 200 readings from the IMU while it is assumed to be stationary. This average gives you an estimate of the gyroscope bias.

*2) Accelerometer Bias and Gravity Initialization:* The accelerometer measures the acceleration in all three axes of the IMU. When stationary, the only acceleration an IMU should theoretically measure is the acceleration due to gravity pointing downwards. By averaging the accelerometer readings(initial 200) during a period when the IMU is static, you can estimate the direction and magnitude of gravity. This average should approximate the gravitational acceleration vector, typically around 9.81 m/s² pointing toward the earth.

*3) Calculating Initial Orientation:* The IMU needs to know its orientation relative to the world frame. Without this, you can't accurately translate the IMU's readings into movements in your SLAM map. With the estimated gravity vector from the accelerometer and the known gravity vector in the world frame, you can compute the initial orientation. This is done by finding the rotation that aligns the +z vector (from the IMU frame) to the measured gravity direction.

### C. Batch IMU Processing

This function processes the buffered IMU data up to a given time bound. The function is designed to propagate the IMU's state based on incoming IMU data up to a specified time. It updates the system's understanding of the IMU's current orientation, position, and velocity based on the angular velocities and linear accelerations measured by the IMU.

### D. Process Model

Method designed to update the IMU's state based on the current IMU readings (gyroscope and accelerometer), correct these readings for biases, and propagate these updates through the system using a mathematical model of IMU dynamics. The mathematical model in continuous time is described equation 1.

$$\begin{aligned} {}_G^I\dot{\bar{q}}(t) &= \frac{1}{2}\Omega(\omega(t))_G^I\bar{q}(t), \\ \dot{b}_g(t) &= n_{wg}(t), \\ {}^G\dot{v}_I(t) &= {}^G a(t), \\ \dot{b}_a(t) &= n_{wa}(t), \\ {}^G\dot{p}^I(t) &= {}^G v_I(t) \end{aligned} \tag{1}$$

Where ${}_G^I\bar{q}(t)$ is the unit quaternion for rotation from Global frame G to IMU frame I.

$$\Omega(\omega) = \begin{bmatrix} \omega & \hat{\omega} \\ \omega^T & 0 \end{bmatrix} \tag{2}$$

Here $\hat{\omega}$ is a skew-symmetric matrix of the $\omega$ vector.
The gyroscope measurements $w_m$ are written as

$$\omega_m = \omega + b_g + n_g \tag{3}$$

The filter propagation equations are derived by discretization of the continuous-time IMU system model. The time evolution of IMU state dynamics is given by

$$\begin{aligned} {}_G^I\dot{\hat{q}}(t) &= \frac{1}{2}\Omega(\hat{\omega}_G^I)\hat{q}, \\ \dot{\hat{b}}_g &= 0_{3\times1}, \\ {}^G\dot{\hat{v}}_I &= C_T^{\hat{q}}\hat{a} - 2\lfloor\omega_{G\times}\rfloor^G\hat{v}_I + \lfloor\omega_{G\times}\rfloor^{2G}\hat{p}_I +^G g, \\ \dot{\hat{b}}_a &= 0_{3\times1}, \\ {}^G\dot{\hat{p}}_I &= {}^G\hat{v}_I \end{aligned} \tag{4}$$

And the error dynamics of IMU error state are given by

$$\dot{\tilde{X}}_I = F\tilde{X} + Gn_I \tag{5}$$

Where $F$ and $G$ are

$$F = \begin{bmatrix} \lfloor\hat{\omega}\times\rfloor & -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C(^I_G\hat{q})^T\lfloor\hat{a}_\times\rfloor & 0_{3\times3} & 0_{3\times3} & -C(^I_G\hat{q})^T & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \quad (6)$$

$$G = \begin{bmatrix} -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -C(^I_G\hat{q})^T & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} & I_3 \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \quad (7)$$

## E. Predict State

This function manages the transition and covariance matrices. It begins by calculating the norm of the gyroscope value and retrieving the current orientation, velocity, and position from the IMU. Next, it propagates the state using the 4th order Runge-Kutta method. The updated values for orientation, velocity, and position for the subsequent state are then computed, leading to an updated state for the IMU.

## F. State Augmentation

The state_augmentation function executes the state augmentation process by incorporating a new camera state into the state server, updating the covariance matrix, and maintaining symmetry when new images are added. It computes the rotation and translation from the IMU to the camera, refreshes the camera state, and adjusts the covariance matrix according to the new state. This step is essential for preserving alignment between the IMU and camera states in the INS implementation. The augmented $J$ and $J_1$ matrix are

$$J = \begin{bmatrix} J_1 & O_{6\times6N} \end{bmatrix} \quad (8)$$

$$J1 = \begin{bmatrix} C(^I_Cq) & 0_{3\times9} & 0_{3\times3} & I_3 & 0_{3\times3} \\ \lfloor C(^I_Gq)^{T\,I}p_{C\times}\rfloor & 0_{3\times9} & I_3 & 0_{3\times3} & I_3 \end{bmatrix} \quad (9)$$

$$P_{k|k} = \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix} P_{K|K} \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix}^T \quad (10)$$

## G. Feature Observation

The add_feature_observations function incorporates feature observations from a new image frame into the map server of a visual-inertial odometry system. It generates new map features for previously unobserved features, updates observations for existing features, and determines the tracking rate.

## H. Measurement Update

The measurement_update function performs the update based on measurements from visual features and inertial sensors. To reduce computational complexity, the Jacobian matrix $H$ is initially decomposed using QR decomposition when the number of rows in $H$ exceeds the number of columns. This results in a reduced-size matrix, $H_{\text{thin}}$, and a transformed measurement vector, $r_{\text{thin}}$. The Kalman gain, which is crucial for weighing measurements during the update step, is computed using $H_{\text{thin}}$, the state covariance $P$, and the observation noise covariance.

The state error, denoted as $\delta x$, is determined by multiplying the Kalman gain with $r_{\text{thin}}$. $\delta x$ is then divided into subvectors for separate updates of the IMU and camera states. For the IMU, small-angle quaternion operations are applied to update the orientation, gyro bias, velocity, accelerometer bias, and position. Additionally, the extrinsic rotation and translation between the IMU and camera are updated.

For the camera states, updates to the orientation and position are executed using small-angle quaternion operations based on the sub-vector $\delta x_{\text{cam}}$.

Finally, the state covariance is updated using the Kalman gain and $H_{\text{thin}}$ to compute the $I - KH$ matrix, which is then used to update the state covariance. Adjustments are made to ensure that the updated state covariance remains symmetric.

## I. Results

The absolute median trajectory error (ATE) is 0.08699091908516823 m and the root mean square translation error (RMSE) is 0.09346765838541407 m. The outcomes of our implementation are depicted in the following results. Additionally, we have visualized the errors relative to the ground truth using the MH 01 easy EuROC dataset. Refer Fig 1 to Fig 9

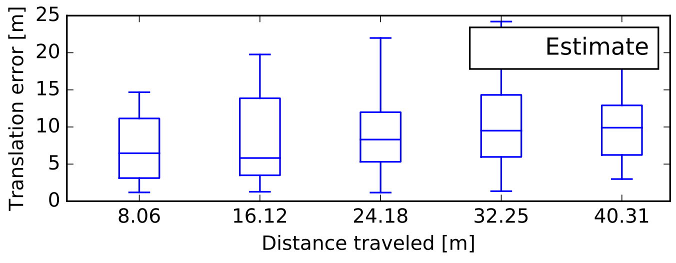

Fig. 1. Relative Translation Error

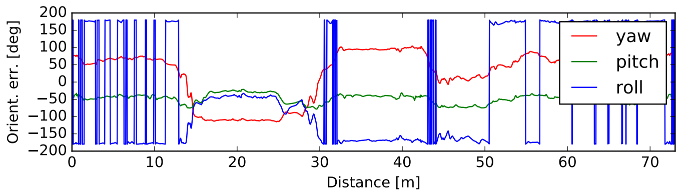

Fig. 2. Rotation Error

## II. Deep Learning Approach

***Abstract*—In this module, various deep learning models and techniques are employed to determine the odometry (translation and rotation) of a drone simulated in Blender. Three distinct methods are utilized: visual-only, inertial-only, and visual-inertial**

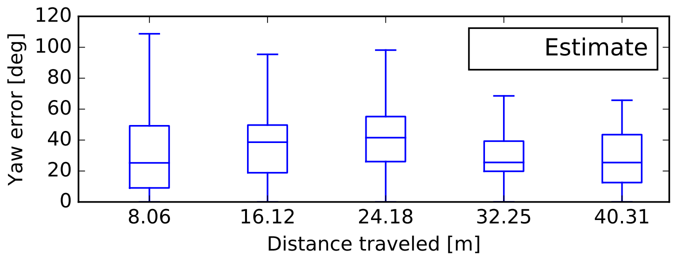

Fig. 3. Relative Yaw Error

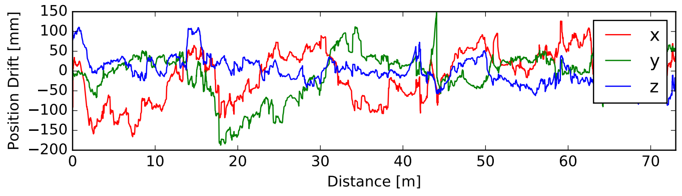

Fig. 4. Translation Error

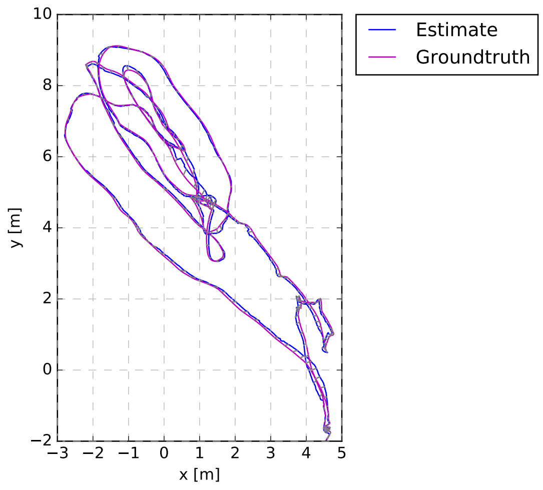

Fig. 5. Ground truth vs Estimated Trajectory (Side View)

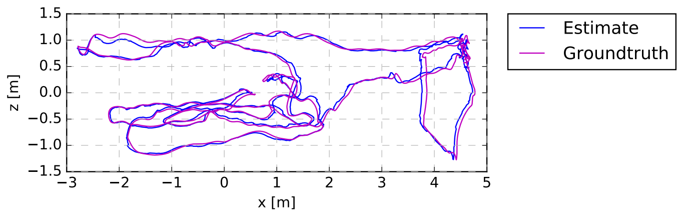

Fig. 6. Ground Truth vs Estimated Trajectory (Top View)

**odometry. Synthetic or simulated data, including downward-facing images, position, orientation (represented by quaternions), translational velocity, and rotational velocity, are used to train several architectures. This approach facilitates the exploration and validation of different computational models under controlled conditions.**

## A. Data Collection

Simulation base of the drone is taken from Phase-2 assignment 0 Alohomora! of Hands-On Autonomous Aerial Robotics course [1]. The codebase and simulation is tweaked to fit the aim of the paper. Images from the drones camera, its position, translational velocity, orientation (quaternion) and Angular Velocity is recorded and stored. To simulate an on-board IMU model, MATALB's IMU model [2] is used. For every 10 values of IMU there is 1 Image. The drone in the simulation is made to follow different trajectories (3D) above a textured plane. Trajectories were created using parametric equations of certain shapes like Circle, Helix, Flower, Hourglass and Frog.

## B. Calculating Relative pose

Quaternion representations of the drone's orientation at discrete time intervals t and t+10 were converted into corresponding rotation matrices. The relative rotational transformation between these two matrices was computed, yielding a matrix that characterizes the rotation from the initial to the final orientation. Additionally, the positional displacements between the two time points were determined. To contextualize these displacements relative to the initial orientation, they were transformed using the rotation matrix from time t.

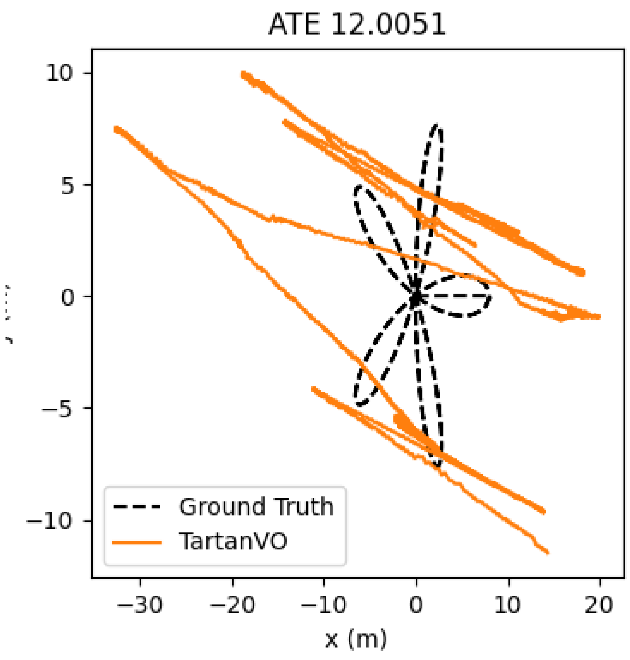

Fig. 7. Initial Output Vision Only

## C. *Visual Only Odometry*

3 Different models were trained and tested for Visual Only Odometry. One is a basic Deep Neural Network that takes two consequent images stacked horizontally, mimicing a stereo camera input. It has 4 layers of Convluted Neural Networks (CNN) with ReLU as activation function, Batch normalization and Max Pooling. The Loss function used is Pose Loss. It is a combination of cosine similarity loss and MSE loss. The bias of the contribution of both the loss is controlled through a variable $\alpha$. This control helps as MSE is more suited for attaining exact pose in terms of magnitude whereas Cosine Loss is scale Invariant and hence focuses more on the orientation of the drone. The second model employs a ResNet architecture, inspired by UnDeepVO [3]. This model is distinctively designed to output position and orientation separately, acknowledging their fundamentally different behaviors. This differentiation allows for independent tuning of the model weights for position and orientation, enhancing the accuracy and relevance of the output in practical applications. We have changed the output of orientation to output Quaternions. The third model utilizes a PWC-Net based matching network to compute optical flow maps [4]. These maps are subsequently inputted into a pose decoder network, from which the pose is derived. This approach leverages the strengths of optical flow for motion estimation to enhance the accuracy of pose determination.

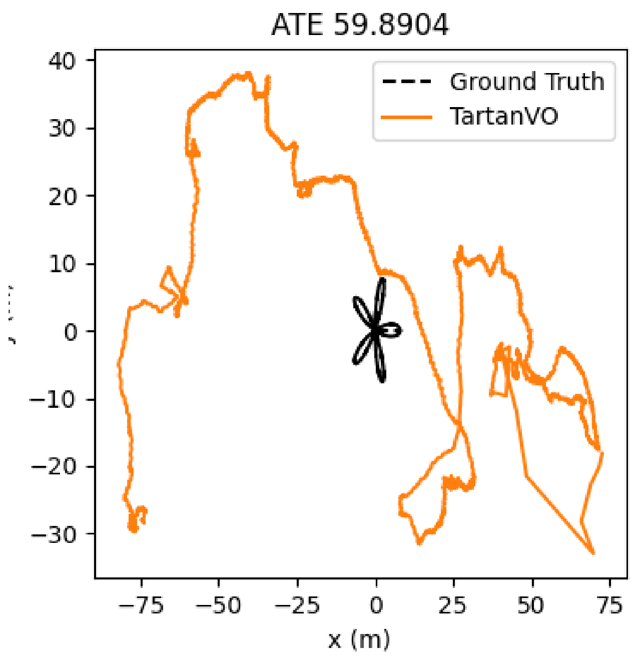

Fig. 8. Initial Output Vision Only

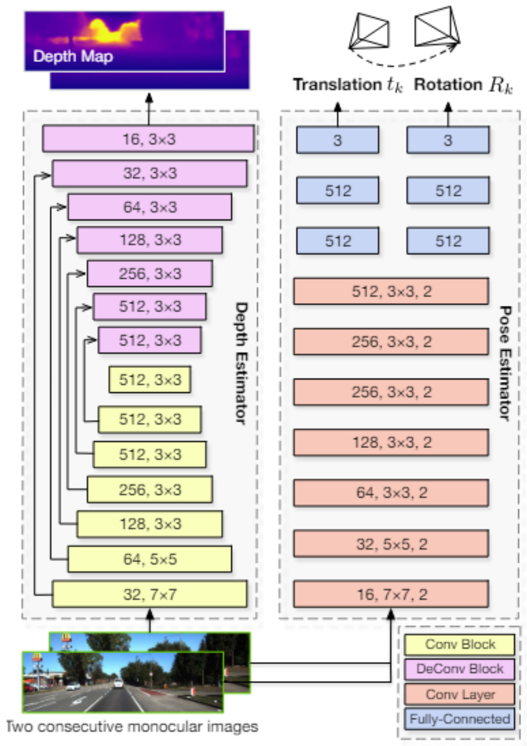

Fig. 10. UnDeepVO Architecture

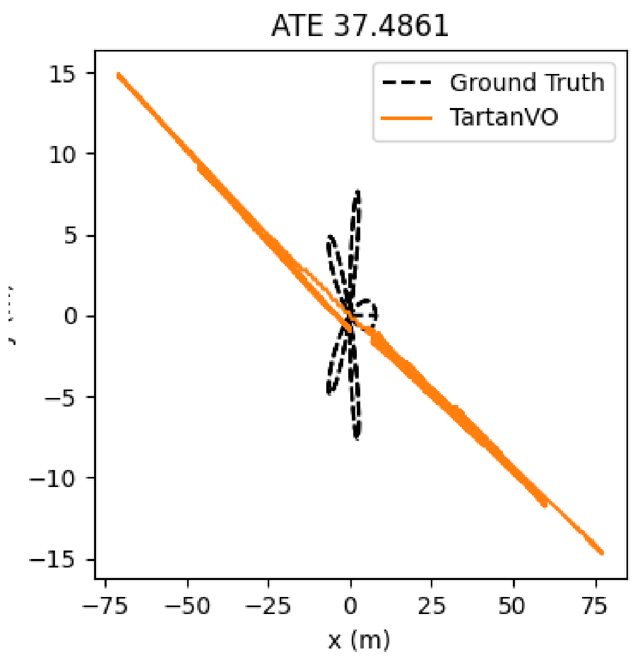

Fig. 9. Initial Output Vision Only

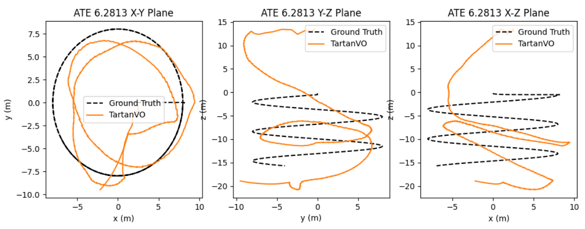

Fig. 11. Helical Trajectory - VIO

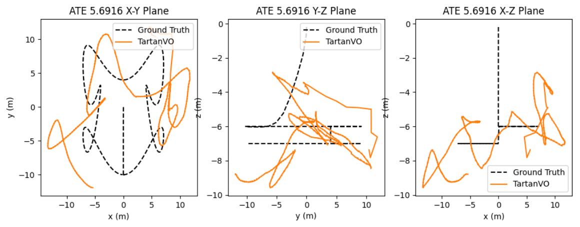

Fig. 12. Frog shaped Trajectory - VO

## D. Inertial Only

Two distinct models were developed and evaluated for Inertial Only Odometry, both utilizing Bi-directional LSTM architectures. The second model, however, integrates an attention mechanism with LSTM, enhancing its capability to focus on relevant features dynamically during sequence processing. This modification is aimed at improving the precision and effectiveness of the odometry predictions by leveraging the strengths of both LSTM and attention-based models. The Loss function used in first module is Pose Loss.

## E. Visual Inertial Ododmetry

Concatenation of Bi-directional LSTM with attention (Inertial) and RESNET (UnDeepVO). The architecture was then connected with 2 Fully Connected Layers and a linear output giving 3 position and 4 orientation (quaternion) values.

## F. Loss Function Used

1) Pose Loss
2) Geodesic Loss
3) Normalized Loss
4) Cosine Similarity Loss

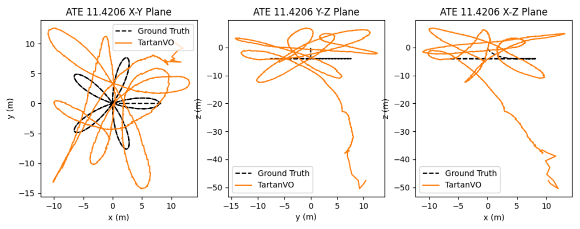

Fig. 13. Flower Trajectory - VO

## G. Result

TABLE I
PERFORMANCE METRICS OF DIFFERENT MODELS

| Model | Train | Val | Test | ATE | Scale |
|---|---|---|---|---|---|
| DCNN -Pose Loss | 0.144 | 0.5577 | 0.215 | 12.0051 | 0.3236 |
| DCNN - Normalized Loss | 0.4116 | 1.2976 | 0.574 | 37.4861 | 0.0106 |
| DCNN - Cosine) | 0.266 | 1.059 | 0.410 28 | ATE: 16.0 | |
| RESNET -Pose Loss | 0.08 | 0.553 | 0.1698 | ATE: 9.5255 | 0.6857 |
| RESNET -Normalized | 6.26 | 5.88 | 6.157 | ATE: 48.4786 | |
| RESNET - Cosine | 0.346 | 1.339 | 0.5722 | | |
| RESNET - Geodesic | 0.293 | 0.971 | 0.415 85 | | |

## REFERENCES

[1] "Project 0: Description," https://rbe549.github.io/rbe595/fall2023/proj/p0/, RBE 549/595, 2023, accessed on 2024-04-28.
[2] MathWorks, "imusensor system object - matlab," https://www.mathworks.com/help/nav/ref/imusensor-system-object.html, 2023, accessed: 2023-04-28.
[3] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning," 2018.
[4] W. Wang, Y. Hu, and S. Scherer, "Tartanvo: A generalizable learning-based vo," 2020.