# 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. 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.

## II. 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.

## III. 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.

## IV. Process Model

The continuous dynamics of the estimated IMU state is,

$$\begin{aligned} {}^I_G\dot{\hat{\mathbf{q}}} &= \frac{1}{2}\Omega(\hat{\boldsymbol{\omega}})\,{}^I_G\hat{\mathbf{q}}, \quad \dot{\mathbf{b}}_g = \mathbf{0}_{3\times1}, \\ {}^G\dot{\hat{\mathbf{v}}} &= C\left({}^I_G\hat{\mathbf{q}}\right)^\top \hat{\mathbf{a}} + {}^G\mathbf{g}, \\ \dot{\hat{b}}_a &= \mathbf{0}_{3\times1}, \quad {}^G\dot{\hat{\mathbf{p}}}_I = {}^G\hat{\mathbf{v}}, \\ {}^I_C\dot{\hat{\mathbf{q}}} &= \mathbf{0}_{3\times1}, \quad {}^I\dot{\hat{\mathbf{p}}}_C = \mathbf{0}_{3\times1} \end{aligned}$$

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 $\Omega$ 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 $\phi_K$ 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,

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

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

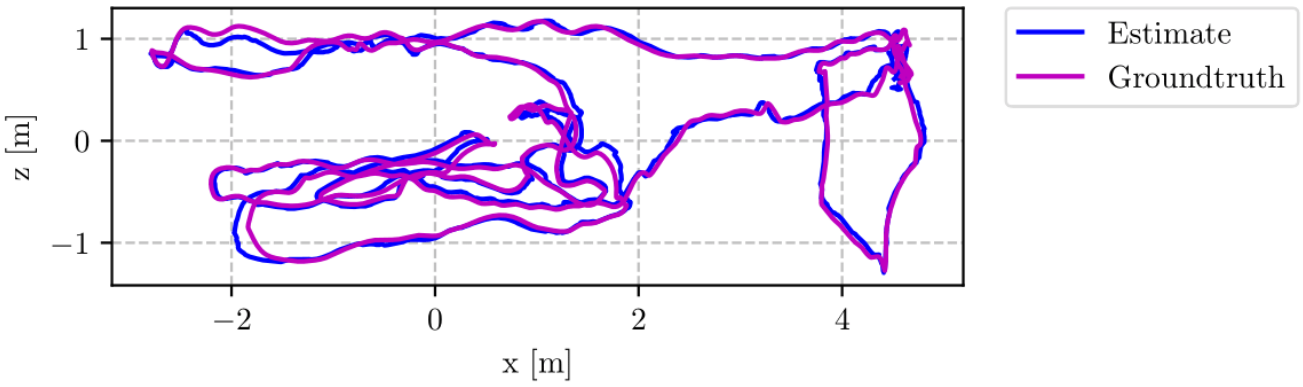


Fig. 1. Trajectory Error Side view

## VI. 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,

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

## VII. 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.

## VIII. 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

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

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

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

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

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

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

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

The Kalman gain K can be computed with the following equation

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

The state error $(\Delta X)$ 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.

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

## IX. 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:

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

## X. 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.

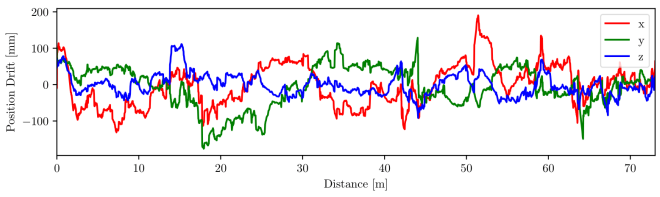


Fig. 2. Translation Error

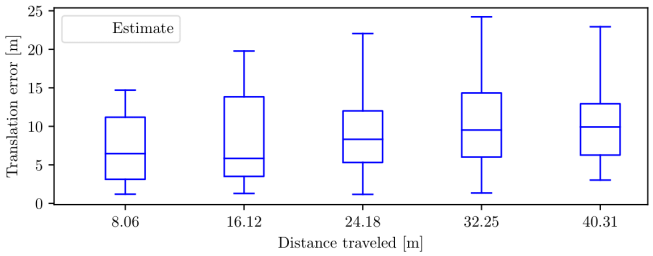


Fig. 3. Relative Translation Error

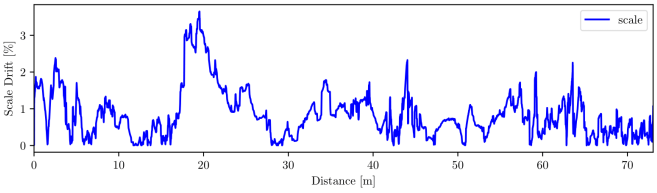


Fig. 4. Scale Error

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.

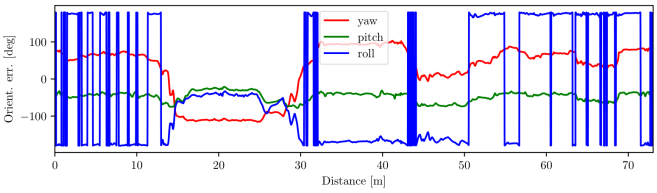


Fig. 5. Rotation Error

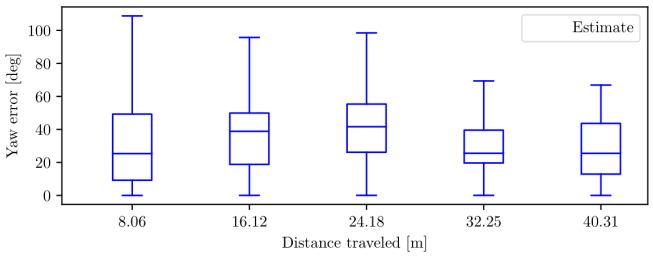


Fig. 6. Relative Yaw Error

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.

## 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.

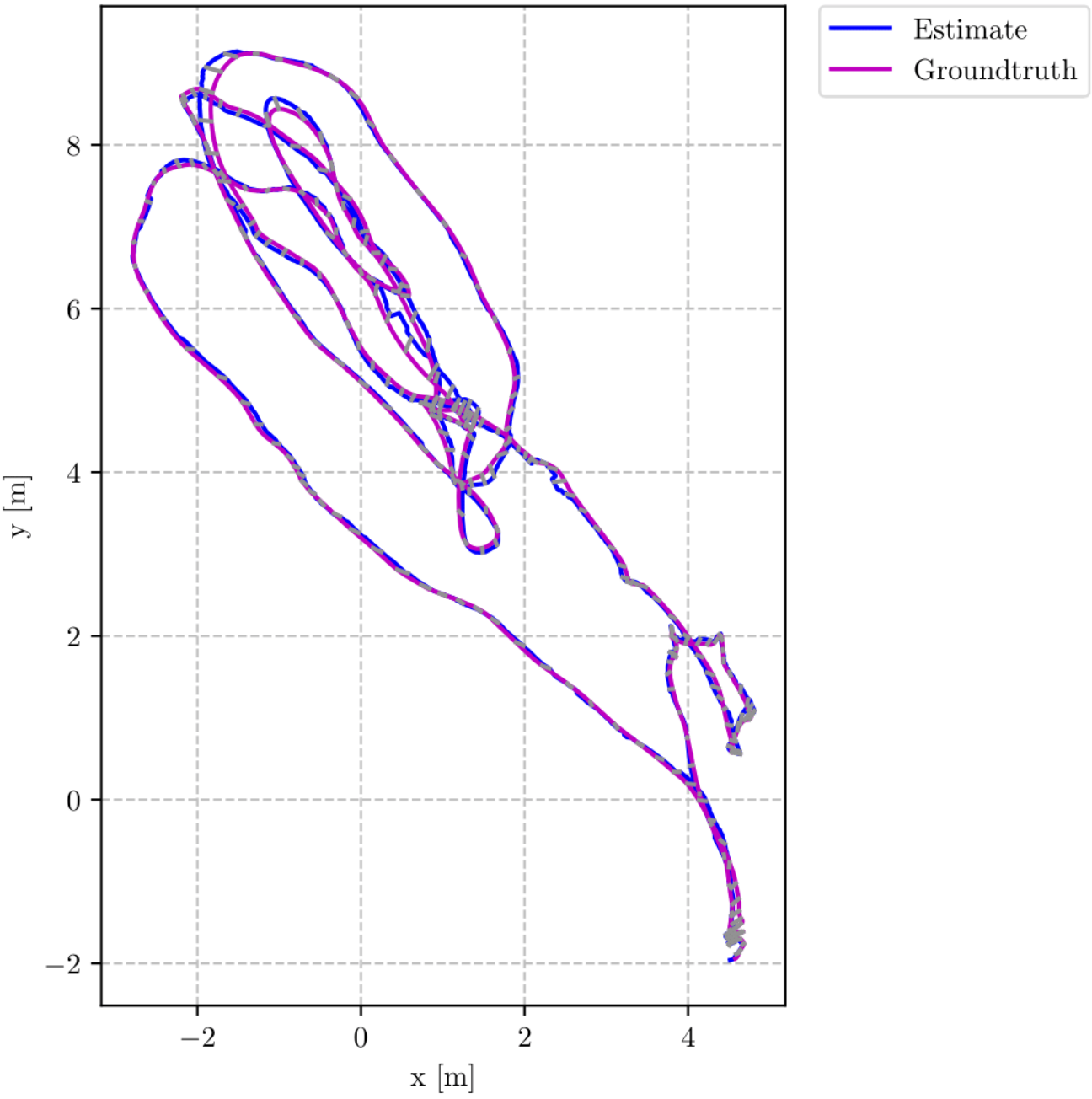


Fig. 7. Trajectory Error Top view

[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. $Accessed$ : $2023 - 04 - 13$.