# RBE 549 Project 4: Deep and Un-Deep VIO (Traditional Visual Odometry Pipeline)

UdayGirish Maradana
Robotics Engineering (MS)
Worcester Polytechnic Institute
Worcester, MA 01609
Email: umaradana@wpi.edu

Pradnya Sushil Shinde
Robotics Engineering (MS)
Worcester Polytechnic Institute
Worcester, MA 01609
Email: pshinde1@wpi.edu

***Abstract*—The following report consists of a detailed analysis of the implementation of vision-aided inertial odometry for state estimation. We implemented a filter-based stereo visual-inertial odometry that uses the Multi-State Constraint Kalman Filter (MSCKF).**
**Keywords: Visual Inertial Odometry, Inertial measurement Unit, Kalman Filter, State estimation**

## I. Phase I: Classical Approach

The goal in this phase is to implement the paper attached here [1]. We have also referred to the seminal VIO paper using MSCKF [2] to understand the mathematical model.

**Algorithm 1** Multi-State Constraint Filter

**Propagation**: For each IMU measurement received, propagate the filter state and covariance (cf. Section III-B).

**Image registration**: Every time a new image is recorded,

- augment the state and covariance matrix with a copy of the current camera pose estimate (cf. Section III-C).
- image processing module begins operation.

**Update**: When the feature measurements of a given image become available, perform an EKF update (cf. Sections III-D and III-E).

Fig. 1: MSCKF Algorithm

## II. Methodology

Fig. 1 gives an overview of the steps performed in the MSCKF algorithm. We will begin by defining the IMU state vector as:

$$x = ({}^{I}_{G}q, b_g, {}^{G}v_I, b_a, {}^{G}p_I, {}^{I}_{G}q, {}^{I}p_C)$$

where ${}^{I}_{G}q$ represents the rotation from the inertial frame to the body frame. The body frame is set to be the IMU frame. The vectors ${}^{G}v_I \in \mathbb{R}^3$ and ${}^{G}p_I \in \mathbb{R}^3$ represent the velocity and position of the body frame in the inertial frame. The vectors $b_g\mathbb{R}^3$ and $b_a \in \mathbb{R}^3$ are the biases of the measured angular velocity and linear acceleration from the IMU. Finally, the quaternion, ${}^{I}p_C$ and ${}^{G}p_I \in \mathbb{R}^3$ represents the relative transformation between the camera frame and the body frame.

For most of the code implementation, we took the concept from paper and also followed the implementation similar to the original C++ version given in the references.

### A. Initialize Gravity and Bias

To initialize gravity and bias, we utilize the first 300 messages received from the IMU in static state of the robot. Each message contains information about the accelerometer and gyroscope readings which are used to calculate $g$ as $[0, 0, g_{\text{norm}}]$ where $g_{\text{norm}}$ is the norm of the accelerometer readings and gyroscope bias $b_g$ as the average of the gyroscope readings respectively.

### B. Batch IMU Processing

The IMU message buffer contains a queue of messages. For every message within this buffer, that's received prior to the feature time stamp, we process and update our state estimation using the process model.

### C. Process Model

The continuous dynamics of the estimated IMU state is:

$$\begin{aligned} {}^{I}_{G}\dot{\hat{q}} = \frac{1}{2}\Omega(\hat{\omega})\,{}^{I}_{G}\hat{q}, \quad \dot{\hat{b}}_g = 0_{3\times1}, \\ {}^{G}\dot{\hat{v}} = C({}^{I}_{G}\hat{q})\hat{a} + {}^{G}g, \\ \dot{\hat{b}}_a = 0_{3\times1}, \quad {}^{G}\dot{\hat{p}}_I = {}^{G}\hat{v}, \\ \dot{\hat{b}}_a = 0_{3\times1}, \quad {}^{I}\dot{\hat{p}}_C = 0_{3\times3} \end{aligned} \tag{1}$$

$$\Omega(\hat{\omega} = \begin{bmatrix} -[\omega_\times] & \omega \\ -\omega^T & 0 \end{bmatrix} \tag{2}$$

$$[\omega_\times] = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{3}$$

$$\omega_m = \omega + b_g + n_g \tag{4}$$

$$a_m = {}^{I}_{G}R({}^{G}a - {}^{G}g + 2[\omega_{G_\times}]\,{}^{G}v_I + 2[\omega_{G_\times}]^2\,{}^{G}p_I) + b_a + n_a \tag{5}$$

The linearized continuous dynamics for the error IMU state is:

$$\dot{\tilde{x}} = F\tilde{x}_I + Gn_I \tag{6}$$

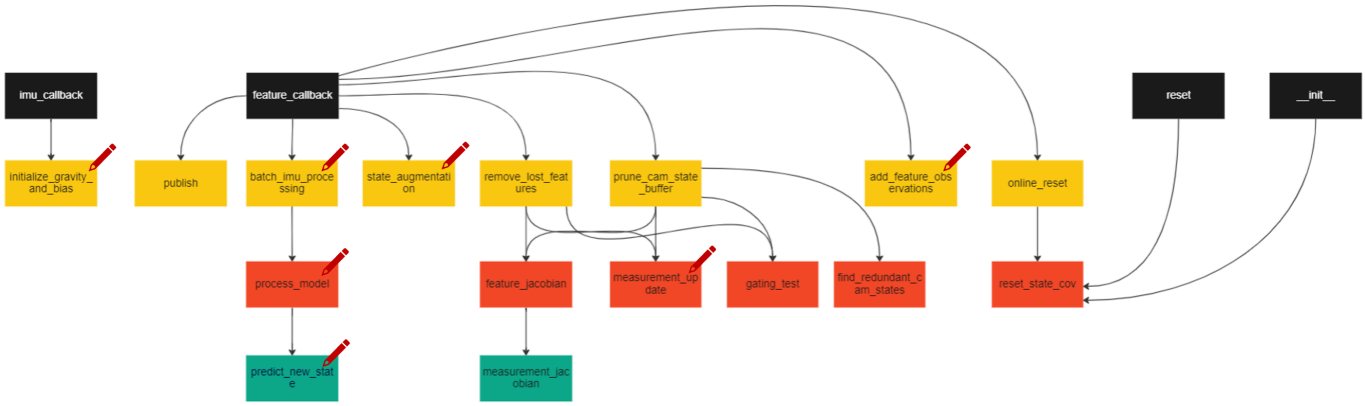


Fig. 2: Overview of Multi-State Constraint Kalman Filter

F and G in the above equation can be defined as:

$$F = \begin{pmatrix} -[\hat{\omega}_\times] & -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C({}^I_G\hat{q})[\hat{a}_\times] & 0_{3\times3} & 0_{3\times3} & -C({}^I_G\hat{q})[\hat{a}_\times] & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{pmatrix}$$

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

### D. Predict New State

For every new IMU measurement received, we use the 4th order Runge-Kutta numerical integration to propagate our estimation for next state as mentioned in the Sample code provided. Usually the update is:

- We get the k1,k2,k3, k4 from $dq/dt$ and $dq/dt2$.
- Once we get these we predict new state (Orientation, Velocity , Position)

### E. State Augmentation

State Augmentation includes updating the Camera Pose defined as $({}^G_{C_i}q, {}^Gp_{C_i})$ where ${}^{C_i}_Gq$ is the orientation and ${}^Gp_{C_i}$ is the position. This pose is calculated as:

$${}^Gp_C = {}^Gp_I + C({}^{C_i}_Gq)^T\, {}^Ip_C \tag{7}$$

$${}^{C_i}_Gq = {}^C_Gq \otimes {}^C_Gq \tag{8}$$

$$J = [J_1 O_{6\times6N}] \tag{9}$$

$$J = \begin{bmatrix} C({}^I_Cq) & 0_{3\times9} & 0_{3\times3} & I_3 & 0_{3\times3} \\ [C({}^I_Gq)^T\, {}^Ip_{C_\times}] & 0_{3\times9} & I_3 & 0_{3\times3} & I_3 \end{bmatrix} \tag{10}$$

$$P_{k|k} = \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix} P_{K|K} \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix}^T \tag{11}$$

### F. Adding Feature Observation

This step/function updates the latest feature detected to the total feature map if it does not exist. Feature map addition is done with the help of feature ID and current state ID keys.

$$Z_i^j = [u_{i,1}^j v_{i,1}^j u_{i,2}^j v_{i,2}^j]^T \tag{12}$$

### G. Measurement Update

A single feature $f_i$, as observed by the stereo cameras with pose $({}^{C_i}_Gq, {}^Gp_{C_i})$. As stereo cameras have different poses, this feature can be represented as $({}^{C_{i,1}}_Gq, {}^Gp_{C_{i,1}})$ and $({}^{C_{i,2}}_Gq, {}^Gp_{C_{i,2}})$ for the left and right cameras respectively. The measurement matrix, $H$ is represented as:

$$H = [QQ_2] \begin{bmatrix} T_H \\ O \end{bmatrix} \tag{13}$$

$$r_n = Q^T r = T_H \tilde{X} + n_n \tag{14}$$

where $r_n$ is the residual.

$$R_n = \sigma_{im}^2 I_{q\times q} \tag{15}$$

The Kalman gain K can be computed as:

$$K = PT_H^T(T_H P T_H^T + R_n)^{-1} \tag{16}$$

$$SK^T = T_H P \tag{17}$$

$$\Delta X = K r_n \tag{18}$$

$$P_{k+1|k+1} = (I_{k\times k} - KT_H)P_{k|k} \tag{19}$$

## III. Results

### A. Final Output from the Video- VIO Implementation

1) We have added functionality to grab the frame buffer from OpenGL and write to a video using CV2 Video write.
2) We have also added screen recording.
3) Further we have added one function which saves the resultant poses with timestamp to a text file. The format is "timestamp, tx,ty,tz, qx,qy,qz,qw".

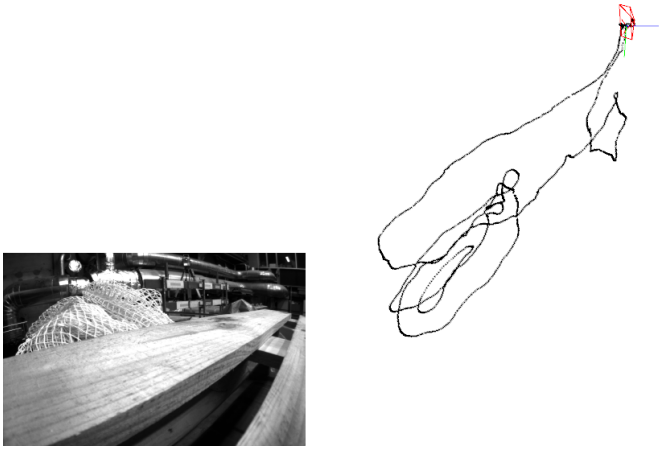

Fig. 3: Final Output of the Simulation - Pangolin Viewer

### B. Outputs from the RPG Toolbox

1) As mentioned in the instructions, we have used RPG toolbox for generating plots.
2) We initially converted the ASL ground truth to ".txt" format using the ASL Data converter in RPG Toolbox.
3) From the function we have implemented above , we take the results.txt and calculate the Absolute Trajectory Error (ATE) and RMSE from the RPG toolbox.
4) *Absolute Median Trajectory Error (ATE):* 0.06344428
5) *Root Mean Square Rotation Error (RMSE):* 127.42737
6) *Root Mean Square Translation Error (RMSE):* 0.08326015
7) *Root Mean Square Scale Error (RMSE):* 1.1683506

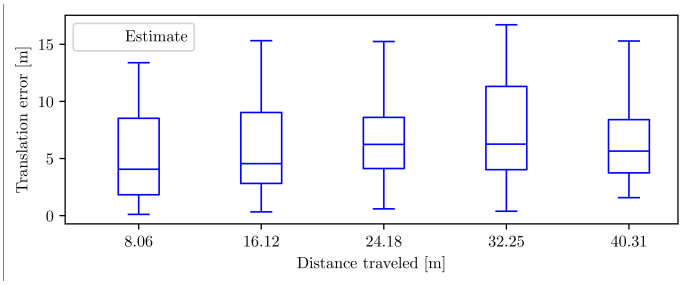


Fig. 4: Translation Error

### C. Discussion

- One issue we have noticed is that there is no sync between the image rendering and the trajectory rendering. Have to debug this.

## References

[1] K. Sun et al., 'Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight', arXiv [cs.RO]. 2018.

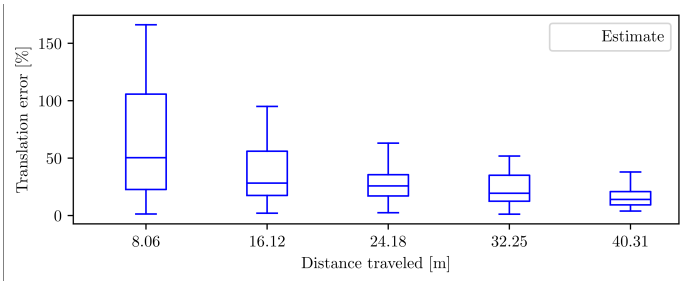


Fig. 5: Relative Translation Error

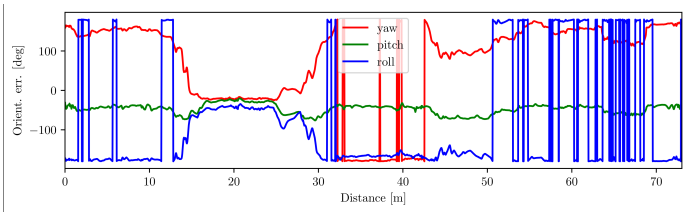


Fig. 6: Rotation Error

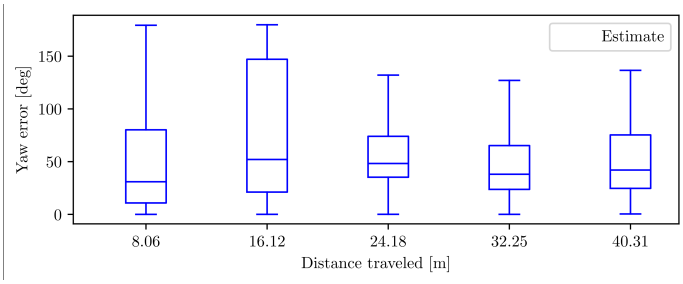


Fig. 7: Relative Yaw Error

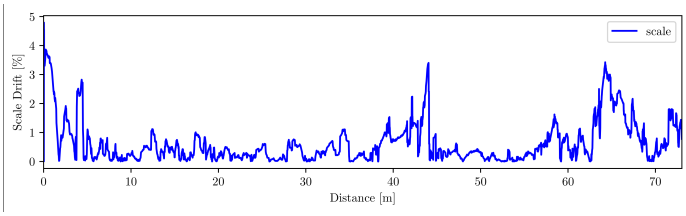


Fig. 8: Scale Error

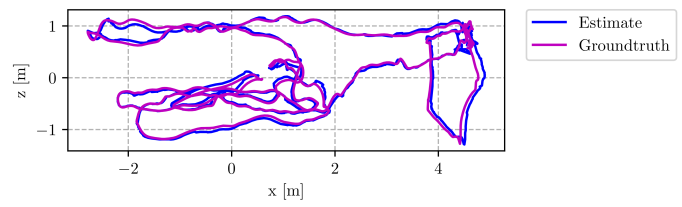


Fig. 9: Trajectory Side View

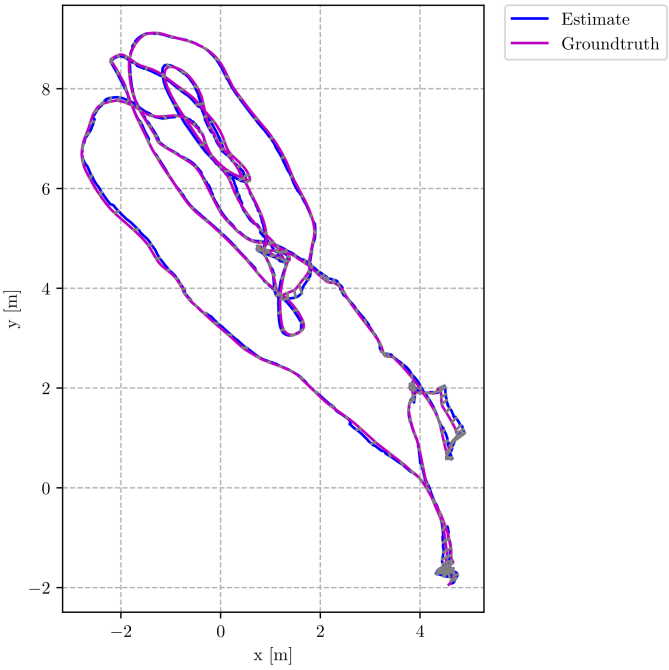


Fig. 10: Trajectory Top View

[2] A. I. Mourikis and S. I. Roumeliotis, 'A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation', in Proceedings 2007 IEEE International Conference on Robotics and Automation, 2007, pp. 3565–3572.

[3] Zichao Zhang, Davide Scaramuzza: A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS), 2018.

[4] Github Code - RPG Toolbox: https://github.com/uzh-rpg/rpg_trajectory_evaluation

[5] Github Code - MSCKF Original C++ Version: https://github.com/KumarRobotics/msckf_vio/tree/master