Computer Vision - Project 4: Deep and Un-Deep Visual Inertial Odometry (Phase 1)

Jesdin Raphael
Worcester Polytechnic Institute
Worcester, MA, USA
Computer Science
Email: jraphael@wpi.edu

Harsh Verma
Worcester Polytechnic Institute
Worcester, MA, USA
Robotics Engineering
Email: hverma@wpi.edu

Muhammad Sultan
Worcester Polytechnic Institute
Worcester, MA, USA
Robotics Engineering
Email: msultan@wpi.edu

***Abstract*—Phase 1: Using a Multi-state Constraint Kalman Filter (MSCKF), a filter-based stereo Vision-aided Odometry is implemented. The method used in this research combines sensor data from an IMU and a stereo camera. The goal is to precisely ascertain the robot's location and state by utilizing the data gathered from these two sensors. Additionally, we have assessed the SMSCKF output for the EuRoC dataset to the ground truth.**

I. Phase 1

A. Introduction

In phase 1 we implement the following functions: initialize_gravity_and_bias (estimates gravity and bias using initial few measurements of IMU), batch_imu_processing (Processes the messages in the imu_msg_buffer, executes the process model and updates the state), process_model (Dynamics of the IMU error state), predict_new_state (Handles the transition and covariance matrices), state_augmentation (Adds the new camera to state and updates), add_feature_observations (Adds the image features to the state), measurement_update (Updates the state using the measurement model) and predict_new_state (Propogates the state using 4th order Runge-Kutta). These functions are implemented based on the paper "Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight" by Sun Et. al. [1].

B. Initialize Gravity and Bias

When the robot is positioned statically, the first 200 messages from the IMU are used to initialize the gyroscope bias and gravity. The average of these 200 gyroscope readings serves as the initial value for the gyroscope bias b_g. The initial value of the gravity g is [0, 0, gnorm], where gnorm represents the average of the first 200 accelerometer data. The quaternion from $-g$ to $gnorm$ is used for the initialization of the orientation. This is implemented in the function "initialize_gravity_and_bias".

C. Batch IMU Processing

This function is used to propagate the state of the IMU by processing IMU measurements in a given time bound. This can be done in the following steps

1) Process the imu messages in the imu_msg_buffer, while discarding already processed messages.
2) Execute process model for every unprocessed message.
3) Update the state info.
4) Repeat steps 1-3 until the $time\_bound$ is reached.
5) Once done processing remove all the used IMU messages from the buffer.

D. Process Model

The IMU system model is given as a continuous-time model form equation 1. The equation is obtained from the 'A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation' paper [2].

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

Here, $\omega(t) = [\omega_x, \omega_y, \omega_z]^T$ is the rotational velocity in the IMU frame, and ${}^{I}_{G}\bar{q}(t)$ is the unit quaternion that applies rotation from the global frame G to the local IMU frame I. Furthermore, $\Omega(\omega(t))$ is,

$$\Omega(\omega) = \begin{bmatrix} -\lfloor\omega_\times\rfloor & \omega \\ -\omega^T & 0 \end{bmatrix} \tag{2}$$

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

ω_m and a_m are gyroscope and accelerometer measurements, respectively, and can be expressed as

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

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

Here, the rotation matrix, ${}^{I}_{G}R$, has been calculated from the quaternion, ${}^{I}_{G}\bar{q}$. The expectation operator is then applied on equation 1 to get the equations for propagating IMU state estimates:

$$\begin{aligned} {}^{I}_{G}\dot{\hat{\bar{q}}}(t) &= \frac{1}{2}\Omega(\hat{\omega}_G^I)\hat{\bar{q}}, \\ \dot{\hat{b}}_g &= 0_{3\times 1}, \\ {}^{G}\dot{\hat{v}}_I &= C_{\hat{q}}^T\hat{a} - 2\lfloor\omega_{G\times}\rfloor\, {}^{G}\hat{v}_I + \lfloor\omega_{G\times}\rfloor^2\, {}^{G}\hat{p}_I + {}^{G}g, \\ \dot{\hat{b}}_a &= 0_{3\times 1}, \\ {}^{G}\dot{\hat{p}}_I &= {}^{G}\hat{v}_I \end{aligned} \tag{6}$$

The linearized model for continuous-time IMU error-state is presented as follows:

$$\dot{\tilde{X}}_{IMU} = F\tilde{X} + Gn_{IMU} \tag{7}$$

Here, F and G are

$$F = \begin{bmatrix} \lfloor\hat{\omega}\times\rfloor & -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C({}^{I}_{G}\hat{q})^T\lfloor\hat{a}_\times\rfloor & 0_{3\times3} & 0_{3\times3} & -C({}^{I}_{G}\hat{q})^T & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{8}$$

$$G = \begin{bmatrix} -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -C({}^{I}_{G}\hat{q})^T & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} \tag{9}$$

E. Predict New State

A prediction step is implemented using the Extended Kalman Filter. New states are predicted using the Runge-Kutta 4th-order equations. During the prediction stage, the accelerometer, gyroscope, and IMU measurements are forward integrated over a time step (dt) to estimate the new state of the system, which includes the IMU's position, orientation, and velocity. Using an adaptive time step and fourth-order Runge-Kutta method, the integration is carried out. Using the gyroscope measurements and the IMU's current state, it also computes intermediate variables (k1, k2, k3, and k4). Using these intermediate variables, it then updates the IMU's orientation, velocity, and location.

F. State Augmentation

In this part, a new camera state is added to the state server, and the state covariance matrix is augmented. The rotation and translation from the IMU to the camera is calculated and camera state is updated, while also computing the state covariance matrix based on the new state. We compute the augmented state covariance matrix using the below Equation.

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

where J is given by

$$J = \begin{bmatrix} J_I & \mathbf{0}_{6\times 6N} \end{bmatrix} \tag{11}$$

and J_I is given by

$$J_I = \begin{bmatrix} C({}^{I}_{G}\hat{q}) & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} \\ -C({}^{I}_{G}\hat{q})^T\lfloor{}^{I}\hat{p}_{C\times}\rfloor & \mathbf{0}_{3\times9} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{I}_3 \end{bmatrix} \tag{12}$$

G. Add feature Observations

Here we add feature observations from a new picture frame to the map server. For features that were not seen previously, it generates new map features, updates observations for features that were previously visible, and determines the tracking rate.

H. Measurement Update

This function computes the Kalman gain K by taking the measurement and residual matrices H and r, respectively, as input. The function then proceeds to update the IMU state X_{IMU}, camera state, and state covariance matrix P.

When the number of rows in H is greater than the number of columns, matrix H is decompsed using QR decomposition to reduce computational comlpexity. This gives the followng Q and T_H.

$$H = \begin{bmatrix} Q & Q_2 \end{bmatrix} \begin{bmatrix} T_H \\ O \end{bmatrix}$$

The covariance matrix R_n of the noise vector n_n is given below.

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

Here, σ_{im}^2 is the noise in the observation, and q is the number of rows of matrix Q.

Now, the Kalman gain K can be calculated using the previous expressions in the following equation.

$$K = PT_H^T(T_H PT_H^T + R_n)^{-1}$$

After obtaining the Kalman gain, we use it to calculate the error in the state which is represented by ΔX, as follows

$$\Delta X = Kr_n$$

Where, $r_n = Q^T r = T_H\tilde{X} + n_n$. This ΔX is then divided into sub-vectors to update the camera and IMU states, separately.

Finally, the state covariance matrix P is updated as shown below

$$P_{k+1|k+1} = (I_{k\times k} - KTH)P_{k|k}$$

I. Result

For comparison we have plotted the relative error between Ground Truth and Estimated Trajectory using the RPG trajectory evaluation toolbox [3]. Various other errors are also depctied in this seciotn. The absolute median trajectory error (ATE) is **0.083261** m and the root mean square translation error (RMSE) is **0.102648** m.

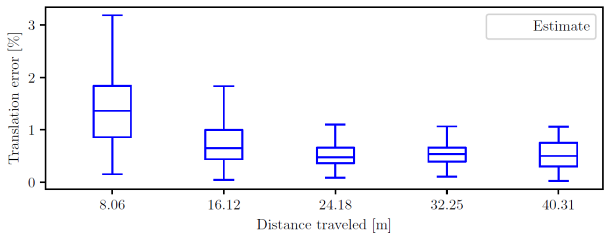

Fig. 1: Relative Translation Error (percentage)

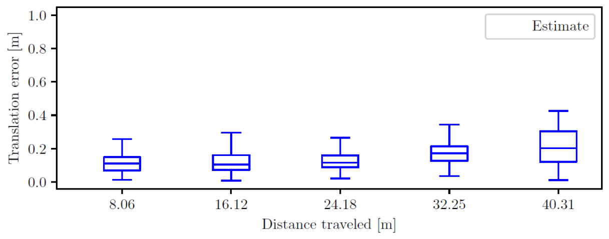

Fig. 2: Relative Translation Error (meters)

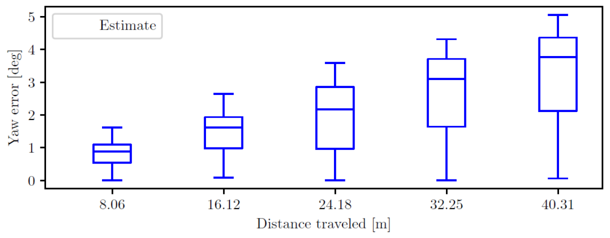

Fig. 3: Relative Yaw Error

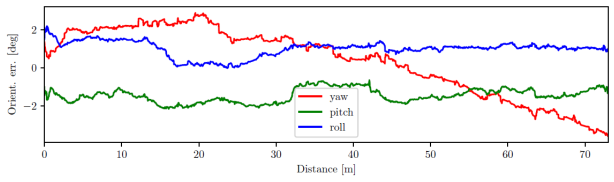

Fig. 4: Rotational Error

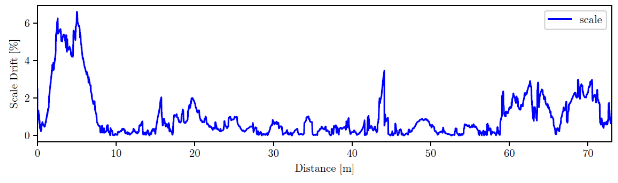

Fig. 5: Scale Error

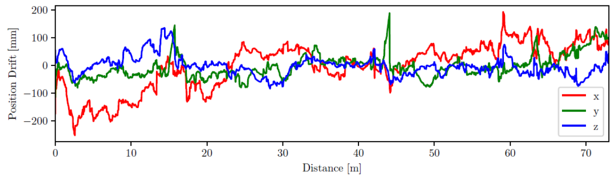

Fig. 6: Position Error

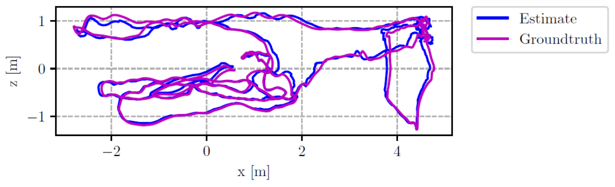

Fig. 7: Estimated vs Ground Truth Trajectory (Side view)

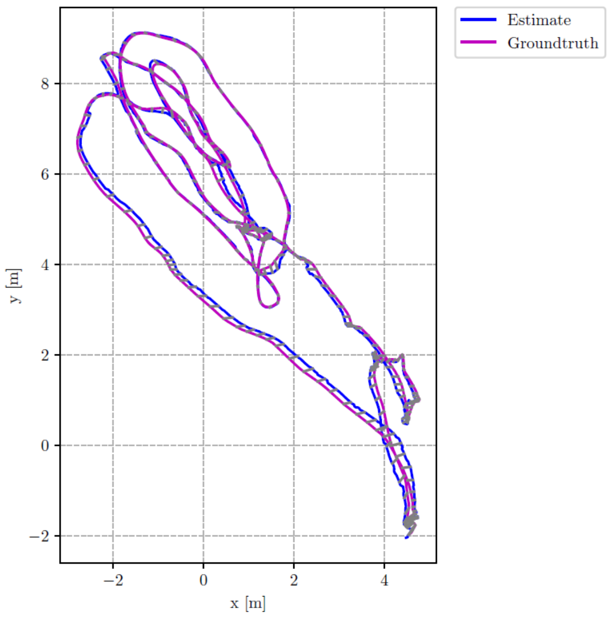

Fig. 8: Estimated vs Ground Truth Trajectory (Top view)

J. Classical Visual Inertial Odometry Research Areas

The following are a few examples of the areas in the classical approach to Visual-Inertial Odometry (VIO) where research can be done.

- **Feature Management and Selection:** It is imperative to develop effective algorithms for tracking, matching, and feature recognition across image sequences. The goal of adaptive feature selection strategies is to preserve performance under a variety of environmental circumstances.
- **State Estimation and Filtering Techniques:** It is crucial to refine state estimate techniques like particle filters, Unscented Kalman Filters, and Extended Kalman Filters. Enhancing robustness against noise and outliers, decreasing computational overhead, and improving convergence qualities are the goals.
- **Handling Degenerate Cases and Failure Detection:** In situations where optical or inertial information is unreliable, robust procedures are needed. The main goal of research is to create algorithms that can handle degenerate scenarios and identify and mitigate failure modes while maintaining system state awareness.
- **Scale Consistency and Drift Correction:** In monocular VIO, maintaining consistent scale and correcting drift

over long sequences are significant challenges. Research explores methodologies for scale estimation and drift correction, incorporating techniques such as loop closure and SLAM.

- **Multi-Sensor Fusion:** Two major issues in monocular VIO are correcting drift over lengthy sequences and keeping consistent size. Scale estimation and drift correction approaches, including loop closure and SLAM, are investigated in research.
- **Real-Time Performance and Optimization:** It is crucial to guarantee that VIO algorithms can function in real-time on platforms with limited resources. Algorithms are optimized in an attempt to minimize computational demands without compromising accuracy.
- **Robust Calibration Techniques:** It is essential to precisely calibrate the relative position and timing between cameras and IMUs. The goal of research is to create field-applicable calibration processes and adaptively adjust calibration parameters while the system is in use.

References

[1] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," 2018.

[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] Z. Zhang, J. Huber, J. Delmerico, and C. Forster, "rpg_trajectory_evaluation," https://github.com/uzh-rpg/rpg_trajectory_evaluation, 2024, version 0.0.0.