Deep and Un-Deep Visual-Inertial Odometry

Computer Vision (RBE549) Project 4

Hrishikesh Pawar
MS Robotics Engineering
Worcester Polytechnic Institute
Email: hpawar@wpi.edu

Tejas Rane
MS Robotics Engineering
Worcester Polytechnic Institute
Email: turane@wpi.edu

***Abstract*—In this project, we implemented seven key functions of Stereo Multi-State Constraint Kalman Filter (MSCKF) (1), building upon the seminal work presented in the seminal MSCKF (2) research.**

I. Phase 1 - Classical VIO

A. Introduction

In the realm of micro aerial vehicles (MAVs), accurate and robust state estimation is paramount for stable flight and effective operation in GPS-denied environments. The paper introduces a tightly-coupled sensor fusion approach for Visual Inertial Odometry (VIO) that synergizes visual data from cameras with inertial measurements from an IMU.

Following the convention in (2), IMU state is defined as

$$\mathbf{x}_I = \left({}^I_G\mathbf{q}^T, \mathbf{b}_g^T, {}^G\mathbf{v}_I^T, \mathbf{b}_a^T, {}^G\mathbf{p}_I^T, {}^I_C\mathbf{q}^T, {}^I\mathbf{p}_C^T\right)^T \tag{1}$$

Here, ${}^I_G\mathbf{q}$ is the quaternion denoting rotation from the inertial frame to the body frame, ${}^G\mathbf{v}_I$ and ${}^G\mathbf{p}_I$ are the velocity and position of the body in the inertial frame, and $\mathbf{b}_g$ and $\mathbf{b}_a$ are the biases of the measured angular velocity and linear acceleration from the IMU. The quaternion ${}^I_C\mathbf{q}$ and vector ${}^I\mathbf{p}_C$ represent the transformation between the camera frame and the body frame.

B. Initialization of Gravity and Gyroscope Bias

The initial alignment and calibration of the IMU are accomplished through the `initialize_gravity_and_bias` function. This process leverages the first 200 IMU messages to estimate the gyroscope bias and the gravity vector in the IMU frame. The gyroscope bias, $\mathbf{b}_g$, is determined as the average of the angular velocities from these initial samples:

$$\mathbf{b}_g = \frac{1}{N}\sum_{i=1}^{N}\boldsymbol{\omega}_i \tag{2}$$

where $N = 200$ and $\boldsymbol{\omega}_i$ is the angular velocity vector from the i-th IMU message.

Simultaneously, the gravity vector, $\mathbf{g}$, is initialized by averaging the linear acceleration readings, and adjusting its orientation to align with the world frame, thus:

$$\mathbf{g} = [0, 0, -\|\mathbf{g}_{\text{avg}}\|]^T \tag{3}$$

where $\mathbf{g}_{\text{avg}}$ is the averaged acceleration vector from the initial samples. This averaged vector's magnitude provides the norm of the gravity vector experienced by the IMU.

Lastly, the initial orientation of the IMU with respect to the world frame is computed to align the estimated gravity direction in the IMU frame with the actual direction of gravity. This orientation is represented by a quaternion.

C. Batch IMU Processing for State Update

The `batch_imu_processing` function sequentially process messages from the IMU message buffer within a specified time bound. The core operation of this function is to iteratively apply the process model to each IMU message, thereby updating the state information of the IMU based on the readings of angular velocities and linear accelerations. The procedure emcompasses the following key operation:

- **Time Filtering**: Messages are selectively processed based on their timestamps to maintain the temporal integrity of the data streams, ensuring that only messages preceding the *time_bound* contribute to state updates.
- **State Propagation**: Utilizing the angular velocity ω and linear acceleration $\mathbf{a}$ from each IMU message, the system's state is advanced through the `process_model` function:

$$\text{State}_{\text{new}} = \text{process\_model}(\text{State}_{\text{current}}, \omega, \mathbf{a}, \Delta t) \tag{4}$$

  where Δt denotes the time increment since the last state update.
- **Synchronization and Buffer Management**: Following each message's processing, the IMU state's timestamp is synchronized to the most recent message time. Subsequently, processed messages are purged from the `imu_msg_buffer`, streamlining the buffer for future operations.

D. Process Model

The estimated IMU state is modeled with continuous dynamics as:

$$\begin{aligned} {}^I_G\dot{\hat{\mathbf{q}}} &= \frac{1}{2}\Omega(\hat{\boldsymbol{\omega}}){}^I_G\hat{\mathbf{q}}, \quad \dot{\hat{\mathbf{b}}}_g = \mathbf{0}_{3\times1}, \\ {}^G\dot{\hat{\mathbf{v}}} &= C\left({}^I_G\hat{\mathbf{q}}\right)^\top\hat{\mathbf{a}} + {}^G\mathbf{g}, \\ \dot{\hat{b}}_a &= \mathbf{0}_{3\times1}, \quad {}^G\dot{\hat{\mathbf{p}}}_I = {}^G\hat{\mathbf{v}}, \\ {}^I_C\dot{\hat{\mathbf{q}}} &= \mathbf{0}_{3\times1}, \quad {}^I\dot{\hat{\mathbf{p}}}_C = \mathbf{0}_{3\times1} \end{aligned} \tag{5}$$

where $\hat{\boldsymbol{\omega}}$ and $\hat{\mathbf{a}}$ are the IMU measurements for angular velocity and linear acceleration respectively.

$$\hat{\boldsymbol{\omega}} = \boldsymbol{\omega}_m - \hat{\mathbf{b}}_g, \quad \hat{\mathbf{a}} = \mathbf{a}_m - \hat{\mathbf{b}}_a \tag{6}$$

$$\Omega\left(\hat{\boldsymbol{\omega}}\right) = \begin{pmatrix} -[\hat{\boldsymbol{\omega}}_\times] & \boldsymbol{\omega} \\ -\boldsymbol{\omega}^\top & 0 \end{pmatrix} \tag{7}$$

The continuous dynamics of the IMU state are linearized as:

$$\dot{\tilde{\mathbf{x}}}_I = \mathbf{F}\tilde{\mathbf{x}}_I + \mathbf{G}\mathbf{n}_I \tag{8}$$

where $\mathbf{n}_I^\top = \begin{pmatrix} \mathbf{n}_g^\top & \mathbf{n}_{wg}^\top & \mathbf{n}_a^\top & \mathbf{n}_{wa}^\top \end{pmatrix}^\top$. The vectors $\mathbf{n}_g$ and $\mathbf{n}_a$ represent the Gaussian noise of the gyroscope and accelerometer measurement, while $\mathbf{n}_{wg}$ and $\mathbf{n}_{wa}$ are the random walk rate of the gyroscope and accelerometer measurement biases.

Here the discrete transition matrix $\mathbf{F}$ and noise covariance matrix $\mathbf{G}$ are given by:

$$\mathbf{F} = \begin{pmatrix} -\lfloor\hat{\boldsymbol{\omega}}_\times\rfloor & -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -C\left({}^I_G\hat{\mathbf{q}}\right)^\top\lfloor\hat{\mathbf{a}}_\times\rfloor & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left({}^I_G\hat{\mathbf{q}}\right)^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \tag{9}$$

$$\mathbf{G} = \begin{pmatrix} -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left({}^I_G\hat{\mathbf{q}}\right)^\top & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{pmatrix} \tag{10}$$

The discrete time state transition matrix is given as:

$$\boldsymbol{\Phi}_k = \boldsymbol{\Phi}(t_{k+1}, t_k) = \exp\left(\int_{t_k}^{t_{k+1}} \mathbf{F}(\tau)d\tau\right) \tag{11}$$

Here, the matrix exponential is approximated by the first three terms of the Power Series expansion as:

$$\begin{gathered} \exp\left(\int_{t_k}^{t_{k+1}} \mathbf{F}(\tau)d\tau\right) = \\ I + \mathbf{F}(\tau)d\tau + \frac{1}{2}(\mathbf{F}(\tau)d\tau)^2 + \frac{1}{6}(\mathbf{F}(\tau)d\tau)^3 \end{gathered} \tag{12}$$

The discrete time noise covariance matrix is given as:

$$\mathbf{Q}_k = \int_{t_k}^{t_{k+1}} \boldsymbol{\Phi}(t_{k+1}, \tau)\mathbf{G}\mathbf{Q}\mathbf{G}\boldsymbol{\Phi}(t_{k+1}, \tau)^\top d\tau \tag{13}$$

Here, Q is the continuous time noise covariance matrix. The propagated covariance of the IMU state is,

$$\mathbf{P}_{II_{k+1|k}} = \boldsymbol{\Phi}_k\mathbf{P}_{II_{k|k}}\boldsymbol{\Phi}_k^\top + \mathbf{Q}_k \tag{14}$$

By partioning the covariance of the whole state as,

$$\mathbf{P}_{k|k} = \begin{pmatrix} \mathbf{P}_{II_{k|k}} & \mathbf{P}_{IC_{k|k}} \\ \mathbf{P}_{IC_{k|k}}^\top & \mathbf{P}_{CC_{k|k}} \end{pmatrix} \tag{15}$$

the full uncertainty propagation can be represented as,

$$\mathbf{P}_{k+1|k} = \begin{pmatrix} \mathbf{P}_{II_{k+1|k}} & \boldsymbol{\Phi}_k\mathbf{P}_{IC_{k|k}} \\ \mathbf{P}_{IC_{k|k}}^\top\boldsymbol{\Phi}_k^\top & \mathbf{P}_{CC_{k|k}} \end{pmatrix} \tag{16}$$

All these computations happen in the `process_model` function.

E. Predict New State

Since we approximate the matrices are approximated in discrete time, we propagate the state using 4th order Runge Kutta method for numerical integration in the `predict_new_states` function.

Applying the 4th order Runge Kutta method, we get the coefficients:

$$\begin{aligned} k_1 &= f(t_n, y_n) \\ k_2 &= f(t_n + \frac{\Delta t}{2}, y_n + k_1\frac{\Delta t}{2}) \\ k_3 &= f(t_n + \frac{\Delta t}{2}, y_n + k_2\frac{\Delta t}{2}) \\ k_4 &= f(t_n + \Delta t, y_n + k_2\Delta t) \end{aligned} \tag{17}$$

The state is then propogated using the equation:

$$I + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{18}$$

F. State Augmentation

In the `state_augmentation` function, we update the camera state when the new images are received. The pose of the new camera state can be computed from the latest IMU state as:

$${}^C_G\hat{\mathbf{q}} = {}^C_I\hat{\mathbf{q}} \otimes {}^I_G\hat{\mathbf{q}}, \quad {}^G\hat{\mathbf{p}}_C = {}^G\hat{\mathbf{p}}_C + C\left({}^I_G\hat{\mathbf{q}}\right)^\top {}^I\hat{\mathbf{p}}_C \tag{19}$$

The state covariance matrix is also augmented as:

$$\mathbf{P}_{k|k} = \begin{pmatrix} \mathbf{I}_{21+6N} \\ \mathbf{J} \end{pmatrix} \mathbf{P}_{k|k} \begin{pmatrix} \mathbf{I}_{21+6N} \\ \mathbf{J} \end{pmatrix}^\top \tag{20}$$

Here, the Jacobian matrix J is given as:

$$\mathbf{J} = \begin{pmatrix} \mathbf{J}_I & \mathbf{0}_{6\times6N} \end{pmatrix} \tag{21}$$

$$\mathbf{J}_I = \begin{pmatrix} C\left({}^I_G\hat{\mathbf{q}}\right) & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} \\ -C\left({}^I_G\hat{\mathbf{q}}\right)^\top\lfloor{}^I\hat{\mathbf{p}}_{C\times}\rfloor & \mathbf{0}_{3\times9} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{I}_3 \end{pmatrix} \tag{22}$$

G. Adding Feature Observation

The `add_feature_observation` function adds the detected features from the latest frame to the feature map. The function first detects whether the feature already exists in the feature map based on the feature ID. If the feature already exists, then the observations are updated in the feature map. Else, the feature map is updated with the unseen features. The feature collected at state ID i and feature ID j is represented as shown in Eq. 23

$$Z_i^j = [u_{i,1}^j, v_{i,1}^j, u_{i,2}^j, v_{i,2}^j] \tag{23}$$

Here, 1 represents the left camera, and 2 represents the right camera.

H. Measurement Update

In the `measurement_update` function, the Kalman Gain K is calculated from the Jacobian matrix H and the residual matrix r_o. To reduce computational complexity, we use QR Decomposition to H, to get:

$$H = [Q_1 Q_2] \begin{bmatrix} T_H \\ 0 \end{bmatrix} \tag{24}$$

We then compute the residual matrix r_n from the matrix Q_1 as:

$$r_n = Q_1^T r_o = T_H \tilde{X} + noise \tag{25}$$

The Kalman Gain is now calculated as:

$$K = P T_H^T (T_H P T_H^T + R_n)^{-1} \tag{26}$$

Here, P is the state covariance matrix and R_n is the noise covariance matrix. The Kalman Gain is used to find the correction in the state as:

$$\Delta X = K r_n \tag{27}$$

The state covariance matrix P is also updated as:

$$P_{k+1|k+1} = (I - K T_H) P_{k+1|k} \tag{28}$$

I. Results

The final results of our implementation are shown in this section. We use the `rpg_trajectory_evaluation`[1] toolbox to generate the following plots and results. The final 3D trajectory as plotted in `pangolin` is shown in Fig. 1. The Top view and the side view of the trajectory is plotted in `matplotlib`, and shown in Fig. 2 and Fig. 3. Fig. 4 and Fig. 5 show the rotation and translation error of the trajectory as compared to the ground truth trajectory. Finally, Fig. 6 shows the Relative Translation Error and Fig. 7 shows the Relative Yaw Error.

The absolute median trajectory error (ATE) is $\mathbf{0.08387606868046654m}$ and the root mean square translation error (RMSE) is $\mathbf{0.10087849118423484m}$.

J. Discussion and Conclusion

In this project, we implemented the classical version of Visual Inertial Odometry. We integrated and implemented the stereo Multi-State Constrained Kalman Filter to ensure accurate and computationally efficient pose estimation. This implementation effectively addresses challenges related to data fusion and system scalability, showcasing the successful utilization of advanced mathematical models and algorithms in real-time navigation scenarios. In addition to reducing computational demands, this research lays a solid foundation for future advancements in optical-inertial odometry systems.

This work can extend into multiple research directions. One challenge that can be addressed is enhancing the robustness and real-time performance of the filtering process. This may involve optimizing algorithms and implementations for efficiency, reducing computational complexity, and leveraging

[1]RPG Trajectory Evaluation Toolbox

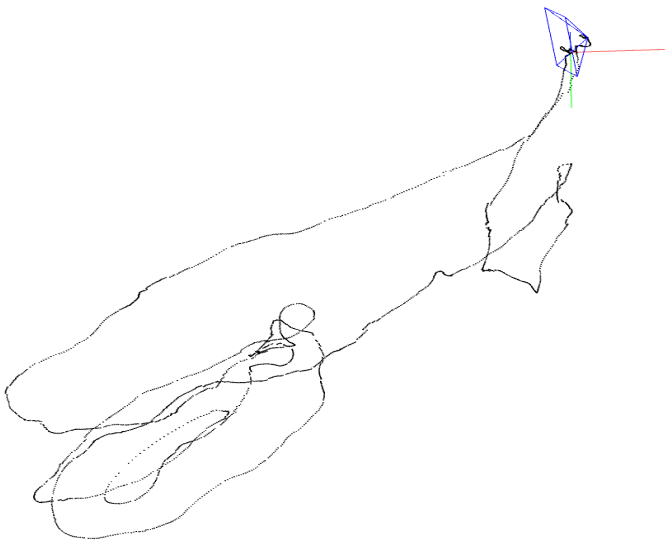

Fig. 1: Visualization of Final 3D trajectory on `pangolin`.

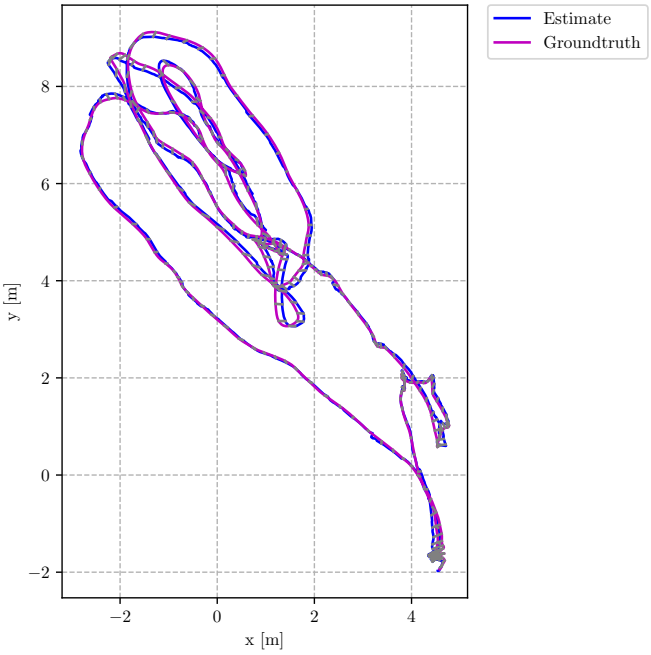

Fig. 2: Visualization of Final trajectory (top view), compared with the ground truth trajectory, on `matplotlib`.

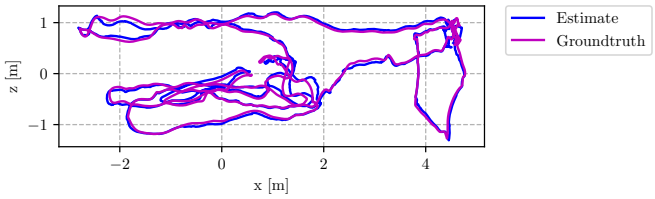

Fig. 3: Visualization of Final trajectory (side view), compared with the ground truth trajectory, on `matplotlib`.

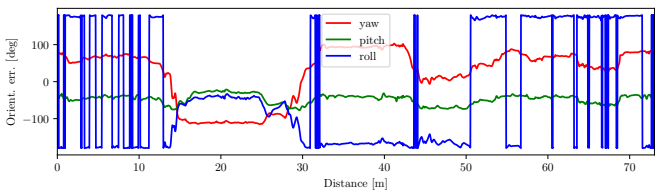

Fig. 4: Rotation Error

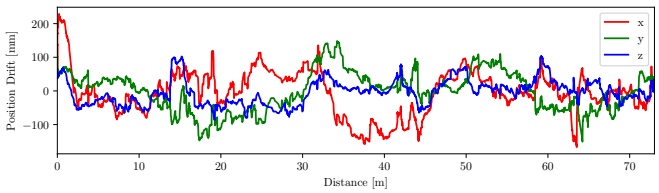

Fig. 5: Translation Error

hardware accelerators such as GPUs or specialized processing units. A faster visual-inertial odometry pipeline can facilitate robust localization for high-speed robots and micro aerial vehicles (MAVs). Another potential research direction involves extending VIO capabilities to support long-term localization and mapping. This includes tackling issues such as map management and scalability, loop closure detection, and maintaining consistency over prolonged periods or in large-scale environments.

K. *Discrepancies*

While implementing the above described functions, we found a difference in the way the augmentation of state

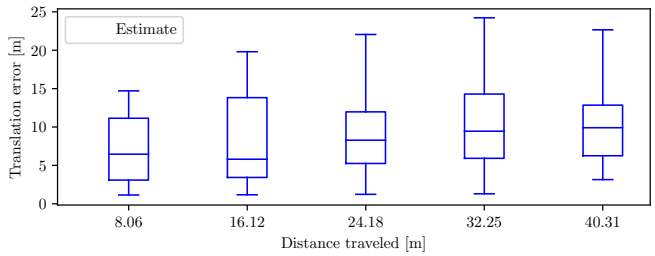

Fig. 6: Relative Translation Error

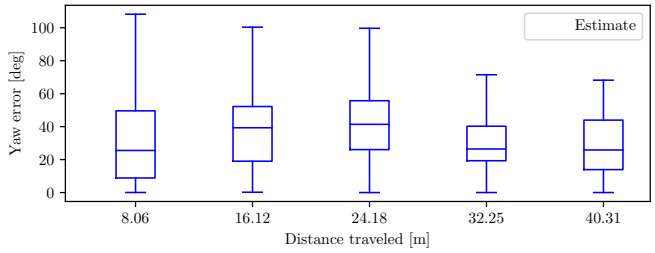

Fig. 7: Relative Yaw Error

covariance matrix (Eq. 26) is defined in (2) and in (3). (2) defines the equation as:

$$P_{k+1|k+1} = (I - KT_H)P_{k+1|k}(I - KT_H)^T + KR_nK^T \quad (29)$$

But, the implementation of (3) only writes the first part of the equation, that is:

$$P_{k+1|k+1} = (I - KT_H)P_{k+1|k} \quad (30)$$

Another discrepancy is in the definition of the Jacobian matrix J. (1) defines the matrix as:

$$\mathbf{J} = \begin{pmatrix} \mathbf{J}_I & \mathbf{0}_{6\times 6N} \end{pmatrix} \quad (31)$$

$$\mathbf{J}_I = \begin{pmatrix} C\left({}^I_G\hat{\mathbf{q}}\right) & \mathbf{0}_{3\times 9} & \mathbf{0}_{3\times 3} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} \\ -C\left({}^I_G\hat{\mathbf{q}}\right)^\top \lfloor {}^I\hat{\mathbf{p}}_{C\times} \rfloor & \mathbf{0}_{3\times 9} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} & \mathbf{I}_3 \end{pmatrix} \quad (32)$$

But the implementation of (3) augments this matrix as:

$$\mathbf{J} = \begin{pmatrix} \mathbf{J}_I & \mathbf{0}_{6\times 6N} \end{pmatrix} \quad (33)$$

$$\mathbf{J}_I = \begin{pmatrix} C\left({}^I_G\hat{\mathbf{q}}\right) & \mathbf{0}_{3\times 9} & \mathbf{0}_{3\times 3} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} \\ \lfloor C\left({}^I_G\hat{\mathbf{q}}\right)^\top {}^I\hat{\mathbf{p}}_{C\times} \rfloor & \mathbf{0}_{3\times 9} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} & C\left({}^I_G\hat{\mathbf{q}}\right)^\top \end{pmatrix} \quad (34)$$

We use the matrix as defined in the paper in out implementation.

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] K. Robotics, "MSCKF_VIO: Multi-State Constraint Kalman Filter for Visual-Inertial Odometry."