RBE/CS549: Computer Vision
Project 4 - Deep and Un-Deep Visual Inertial Odometry

Shreya Bang
M.S. Robotics Engineering
Email: srbang@wpi.edu

Rajus Nagwekar
M.S. Robotics Engineering
Email: rmnagwekar@wpi.edu

***Abstract*—Phase 1 focuses on the implementation of a filter-based stereo Vision-aided Odometry using Multi-state Constraint Kalman Filter (MSCKF). This project involves the approach that utilizes sensor fusion from a stereo camera and an IMU. We have implemented eight functions from MSCKF. With the data from these two sensors, we aim to accurately determine the state and localization of the robot. We also have evaluated output of S-MSCKF with respect to ground truth for EuRoC dataset.**

Index: Multi-state Constraint Kalman Filter, Stereo Camera, IMU

I. PHASE 1: CLASSICAL APPROACH

A. Initialize Gravity and Bias

The 6-DOF IMU sensor used for rotation (gyroscope) and acceleration (accelerometer) measurements requires calibration to account for biases. This involves taking the mean of stationary readings to determine the bias, which is then subtracted from subsequent readings. The gyroscope should ideally read [0, 0, 0] but may have small fluctuations. Similarly, the accelerometer should ideally read [0, 0, -g] in the world frame, but may also have fluctuations due to noise and bias. Calibration is performed before the start of flight to remove biases in both gyroscope and accelerometer readings.

The function "initialize_gravity_and_bias" initializes the gravity and bias of the IMU, as well as the initial orientation of the robot, based on the first few IMU readings. It calculates the gyro bias and estimates the gravity in the IMU frame by averaging the angular velocity and linear acceleration readings from the IMU messages respectively. The initial orientation is set to be consistent with the inertial frame. This process ensures a reliable start for the Visual-Inertial Odometry system.

B. Batch IMU Processing

The IMU batch processing function is used to read IMU messages until the next set of images is available from the stereo camera. The state vector used for estimating the next states consists of states related to camera and IMU, including quaternion for rotation, biases for gyroscope and accelerometer, positions, and velocities.

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

Fig. 1: State Vector

The "batch_imu_processing" function is a crucial part of the implemented Visual-Inertial Odometry system. It propagates the state of the IMU by processing the IMU measurements within a specified time bound. The function iterates through the IMU messages in the buffer, discarding already processed messages and stopping at the time bound. For each unprocessed IMU message, the function applies the process model to update the IMU state based on angular velocity and linear acceleration measurements. The IMU state's timestamp and ID are updated, and the processed IMU messages are removed from the buffer. This function ensures accurate state propagation and synchronization between the IMU and Visual Odometry components of the system, making it a crucial step in achieving reliable sensor fusion and localization.

C. Process Model

The function "process_model" propagates the system state and covariance using a 4th order Runge-Kutta integration method. It first extracts relevant information from the current system state, including the IMU (Inertial Measurement Unit) state, which includes orientation, velocity, position, and gyroscope and accelerometer biases. It then calculates the time step based on the provided time and IMU state timestamp.

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

Fig. 2: Dynamics of IMU

Next, the code computes the discrete transition matrix (F) and noise covariance matrix (G) to describe the system dynamics and noise characteristics. The transition matrix is approximated using a 3rd order matrix exponential method, assuming a small time step (dt). Intermediate matrices Fdt, Fdt_square, and Fdt_cube are calculated for this purpose.

The code then predicts the new system state using the 4th order Runge-Kutta integration method, calling the predict_new_state function. This likely updates the system state

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

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

based on gyroscope and accelerometer measurements and current state estimates. The transition matrix Phi is also modified to account for the null space, which is the space of states that are not directly observable from the sensor measurements.

The state covariance matrix (Q) is updated using Phi, G, and the continuous noise covariance matrix, which represents uncertainties or errors in the system model. The covariance between IMU states and camera states (if present) is also updated accordingly. Finally, the state covariance matrix is fixed to be symmetric by averaging it with its transpose.

The IMU state is updated with the current values of orientation, position, and velocity, which serve as the null space values for the next iteration of the state estimation process.

D. *Predict New State*

The function "predict_new_state" implements a prediction step using Extended Kalman Filter (EKF). The prediction step involves forward integrating the IMU measurements, gyroscope and accelerometer, over a time step (dt) to estimate the new state of the system, which includes the orientation, velocity, and position of the IMU. The integration is performed using a fourth-order Runge-Kutta method with adaptive time step. Further it calculates intermediate variables (k1, k2, k3, k4) 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.

E. *State Augmentation*

The function "state_augmentation" implements the state augmentation step by adding a new camera state to the state server, updating the covariance matrix, and ensuring symmetry on the addition of new images. It calculates the rotation and translation from the IMU to the camera, updates the camera state, and modifies the covariance matrix based on the new state. This step is crucial for maintaining consistency between IMU and camera states in the INS implementation.

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

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

Fig. 3: The State Augmentation Jacobian

F. *Adding Feature Observation*

The function "add_feature_observations" adds feature observations from a new image frame to the map server in a visual-inertial odometry system. It creates new map features for unseen features, updates observations for existing features, and calculates the tracking rate.

G. *Measurement Update*

The "measurement_update" function performs the update based on measurements from visual features and inertial sensors. Measurements of Stereo and posisions of the features are given in Fig. 04 and 05.

$$\mathbf{z}_i^j = \begin{pmatrix} u_{i,1}^j \\ v_{i,1}^j \\ u_{i,2}^j \\ v_{i,2}^j \end{pmatrix} = \begin{pmatrix} \frac{1}{{}^{C_{i,1}}Z_j} & \mathbf{0}_{2\times2} \\ \mathbf{0}_{2\times2} & \frac{1}{{}^{C_{i,2}}Z_j} \end{pmatrix} \begin{pmatrix} {}^{C_{i,1}}X_j \\ {}^{C_{i,1}}Y_j \\ {}^{C_{i,2}}X_j \\ {}^{C_{i,2}}Y_j \end{pmatrix}$$

Fig. 4: Measurements of Stereo

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

Fig. 5: Position of the features in left and right Camera Frame

To reduce computational complexity, it first decomposes the Jacobian matrix H using QR decomposition when the number of rows in H is greater than the number of columns. This results in a reduced-size H_thin matrix and a transformed measurement vector r_thin.

The Kalman gain, which determines the weight of the measurements in the update step, is computed using the reduced-size H_thin matrix, the state covariance P, and the observation noise covariance.

The error in the state, represented by delta_x, is computed by multiplying the Kalman gain with the transformed measurement vector r_thin. The delta_x is then divided into subvectors for updating the IMU and camera states separately.

Further, the IMU state is updated by applying small-angle quaternion operations to update the orientation, gyro bias, velocity, accelerometer bias, and position of the IMU state. Additionally, the extrinsic rotation and translation between the IMU and camera are also updated. The camera states, including the orientation and position, are updated using small-angle quaternion operations based on the sub-vector delta_x_cam.

The state covariance is updated using the Kalman gain and the reduced-size H_thin matrix to obtain the I_KH matrix, which is then used to update the state covariance. To ensure symmetry, the updated state covariance is fixed to be symmetric.

H. Results

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.

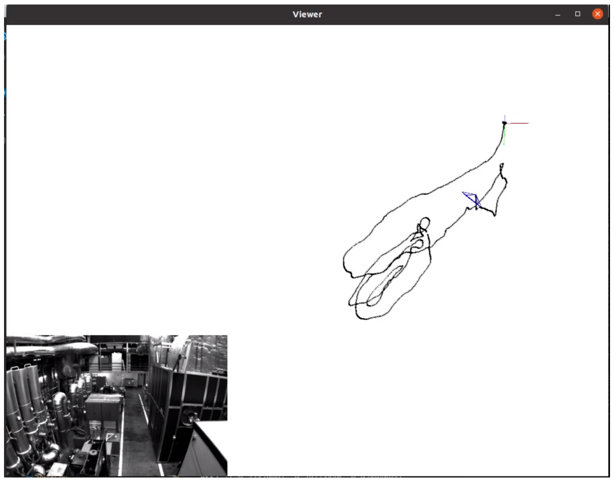


Fig. 6: Visualization

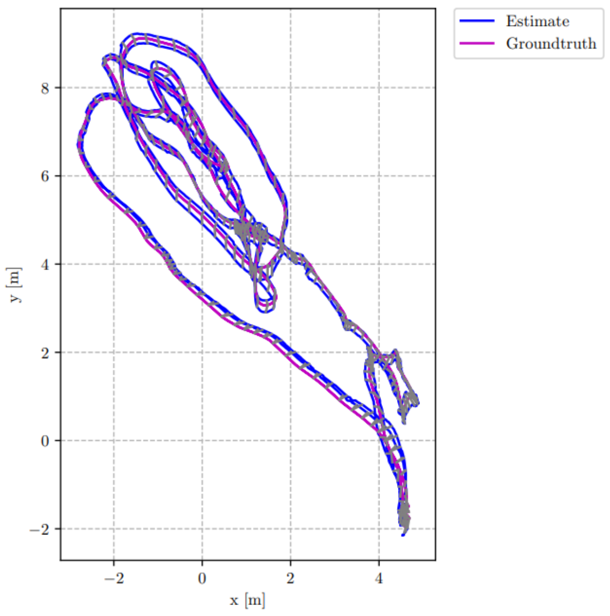


Fig. 7: Ground Truth vs Estimated Trajectory (Top View)

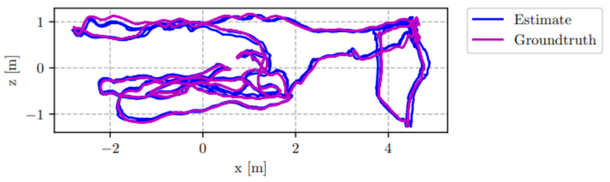
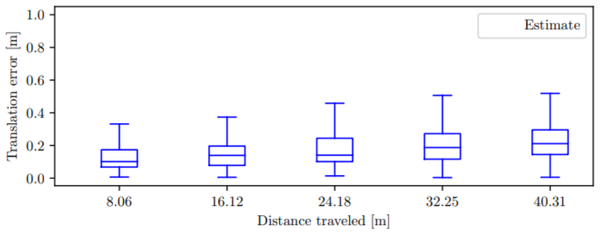
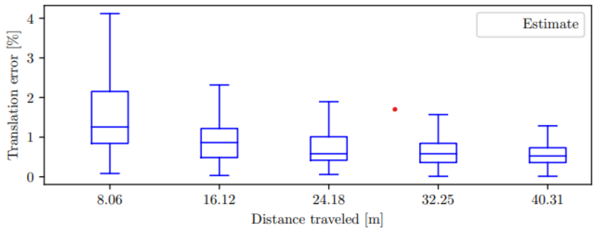


Fig. 8: Ground truth vs Estimated Trajectory (Side View)





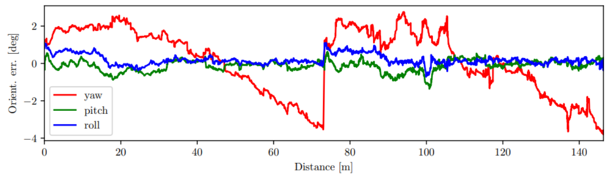


Fig. 9: Rotational Error

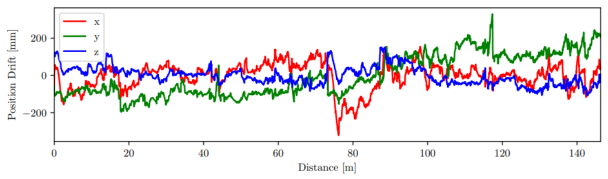


Fig. 10: Position Error