# RBE 549 Project 4: Classical Visual Inertial Odometry

Amrit Krishna Dayanand, Venkata Sai Krishna Bodda
MS Robotics Engineering
WPI
Email: adayanand@wpi.edu, vbodda@wpi.edu

## I. Introduction

**This project primarily focuses on performing odometry by combining data from an inertial measurement unit and a stereo camera. In phase 1, this is accomplished through a classical approach, employing a filter-based stereo Vision-aided Odometry using a multi-state Constraint Kalman Filter (MSCKF). The primary objective is to precisely determine the robot's states—orientation, position, and velocity—and to visualize them. Eight methods were implemented in the starter code, and the results are visualized using the Pangolin visualizer.**

**Index: IMU, Stereo Camera, Multi-state Constraint Kalman Filter(MSCKF), Visual Inertial Odometry, Stereo Camera.**

## II. Classical Approach using MSCKF


Fig. 1. pipeline

### A. Initialize Gravity and Bias

The rotation and acceleration measurements were taken using a 6 degrees of freedom (DOF) inertial measurement unit (IMU). This IMU requires calibration to correct for biases. These biases are determined by averaging stationary readings, and then this bias is subtracted from the actual readings. Once the biases are subtracted, the ideal readings for the gyroscope and accelerometer, while stationary, should be [0, 0, 0] and [0, 0, -g] respectively, with minor fluctuations.

The function "initialize gravity and bias" is responsible for initializing the IMU's gravity and bias, as well as establishing the initial orientation of the robot based on the initial IMU readings. It accomplishes this by computing the gyro bias and estimating gravity within the IMU frame through the averaging of angular velocity and linear acceleration readings from the IMU messages. Ensuring alignment with the inertial frame, the initial orientation is set for consistency. This systematic approach guarantees a robust commencement for the Visual-Inertial Odometry system.

$$\mathbf{x}_I = \begin{pmatrix} {}^I_G\mathbf{q}^\top & \mathbf{b}_g^\top & {}^G\mathbf{v}_I^\top & \mathbf{b}_a^\top & {}^G\mathbf{p}_I^\top & {}^I_C\mathbf{q}^\top & {}^I\mathbf{p}_C^\top \end{pmatrix}^\top$$

Fig. 2. State Vector

### B. Batch IMU Processing

IMU messages from the sensor are read using Batch_IMU_Processing function and are processed until next batch of images are published from the stereo camera. The state vector employed for predicting the subsequent states encompasses various parameters associated with both the camera and IMU systems. These include quaternion representations for rotation, gyroscope and accelerometer biases, positional coordinates, and velocities.

Batch IMU processing is responsible for propagating the state of IMU by processing IMU measurements within a time bound before next batch is available. The function iterates through the IMU messages in the buffer, disregarding those already processed and halting at the specified time limit. For each unprocessed IMU message, the function applies the process model to adjust the IMU state using measurements of angular velocity and linear acceleration. It updates the timestamp and ID of the IMU state, removing processed messages from the buffer. This process guarantees precise state propagation and synchronization between the IMU and Visual Odometry components, playing a pivotal role in achieving dependable sensor fusion and localization.

### C. Process Model

The "process model" function propagates the system state and covariance using a 4th order Runge-Kutta integration method. It extracts pertinent information from the current system state, notably from the IMU (Inertial Measurement Unit), encompassing orientation, velocity, position, and gyroscope and accelerometer biases. Additionally, it calculates the time step based on the provided time and IMU state timestamp.

The code calculates the discrete transition matrix (F) and noise covariance matrix (G) to model system dynamics and

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

Fig. 3. DYnamics of an IMU

noise characteristics. It approximates the transition matrix using a 3rd order matrix exponential method, considering a small time step (dt). It predicts the new system state via a 4th order Runge-Kutta integration method, updating the state based on gyroscope and accelerometer measurements.

The transition matrix Phi is adjusted to accommodate the null space, while the state covariance matrix (Q) is updated using Phi, G, and the continuous noise covariance matrix. Covariance between IMU and camera states is also adjusted. Finally, symmetry is ensured in the state covariance matrix by averaging it with its transpose. The IMU state is then updated with current orientation, position, and velocity values, ready for the next iteration of state estimation.

### *D. Predict New State*

The "predict new state" function executes a prediction step employing the Extended Kalman Filter (EKF). This step involves forward integrating IMU measurements, gyroscope, and accelerometer, over a time step (dt) to estimate the system's new state, comprising IMU orientation, velocity, and position. Integration is conducted using a fourth-order Runge-Kutta method with adaptive time step. Additionally, it calculates intermediate variables (k1, k2, k3, k4) using gyroscope measurements and the current IMU state, subsequently updating IMU orientation, velocity, and position based on these intermediate variables.

### *E. State Augmentation*

The "state augmentation" function incorporates a new camera state into the state server, adjusting the covariance matrix and ensuring symmetry upon adding new images. It computes the rotation and translation from the IMU to the camera, updates the camera state, and adjusts the covariance matrix accordingly. This step is pivotal for upholding consistency between IMU and camera states in the INS implementation.

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

Fig. 4. State Augmentation Jacobian

### *F. Adding Feature Observation*

The "add feature observations" function integrates feature observations from a new image frame into the map server of a visual-inertial odometry system, generating new map features for unseen ones, updating observations for existing features, and calculating the tracking rate.

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

Fig. 5. Stereo Measurement Calculation

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

Fig. 6. Feature's Position in Left and right cameras

### *G. Measurement Update*

The "measurement update" function utilizes measurements from visual features and inertial sensors, decomposing the Jacobian matrix H using QR decomposition to reduce computational complexity when the number of rows in H exceeds the number of columns, resulting in a reduced-size H thin matrix and a transformed measurement vector r thin. It computes the Kalman gain using the reduced-size H thin matrix, the state covariance P, and the observation noise covariance to determine the weight of measurements in the update step. The error in the state, represented by delta x, is computed by multiplying the Kalman gain with the transformed measurement vector r thin, which is then divided into sub-vectors for updating IMU and camera states separately. IMU state is updated using small-angle quaternion operations, while camera states 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 ensure symmetry.

## III. Results

The Rotation and orientation values are sent to Pangolin visualizer allowing it to display them alongside the stereo camera output.

The RMSE absolute trajectory error (ATE) quantitatively measures the difference between the estimated and ground truth trajectories. Since the estimated and ground truth frames may not be the same, an SE(3) alignment is first performed, followed by an RMSE calculation as follows. We compare the two trajectories in terms of position and rotation, which

are summarized in table I. The RMSE ATE is useful because it makes it easy to compare trajectories, but the metric is sensitive to the time when the error occurs.

$$\epsilon_{ATE} = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\frac{1}{K}\sum_{k=1}^{K}\|\Delta x_{k,i}\|^2} \quad (1)$$

| Type | RMSE ATE |
|---|---|
| Position | 0.08548 |
| Rotation | 154.34993 |

TABLE I
RMSE ATE OF ESTIMATED AND GROUND TRUTH ODOMETRY FOR POSITION AND ROTATION

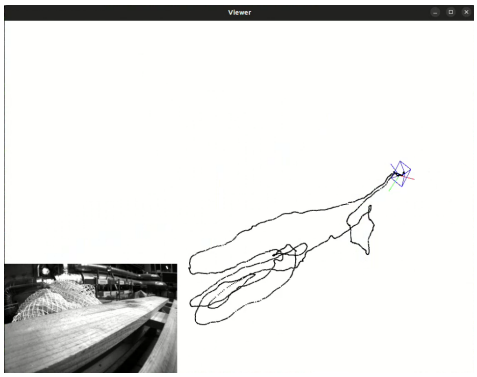


Fig. 7. Output from pangolin viewer

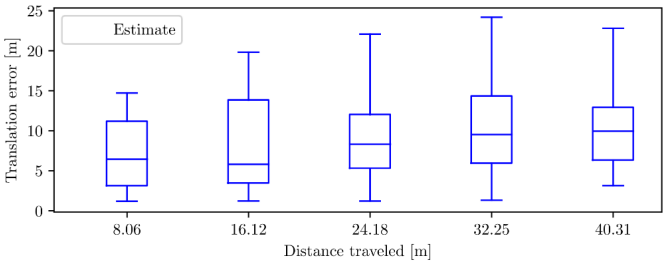


Fig. 8. The relative translation error of projected path with respect to ground truth

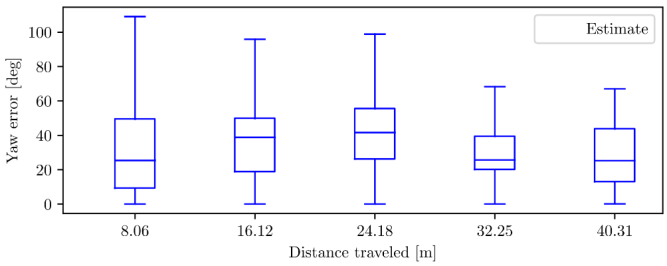


Fig. 9. The relative yaw error of projected path with respect to ground truth

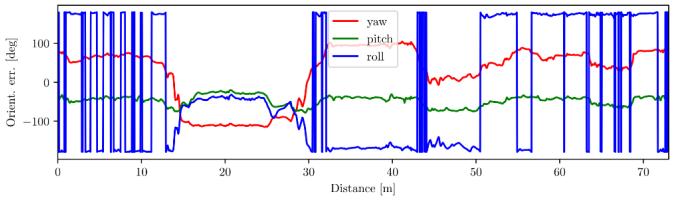


Fig. 10. Error in rotation of projected path w.r.t. ground truth

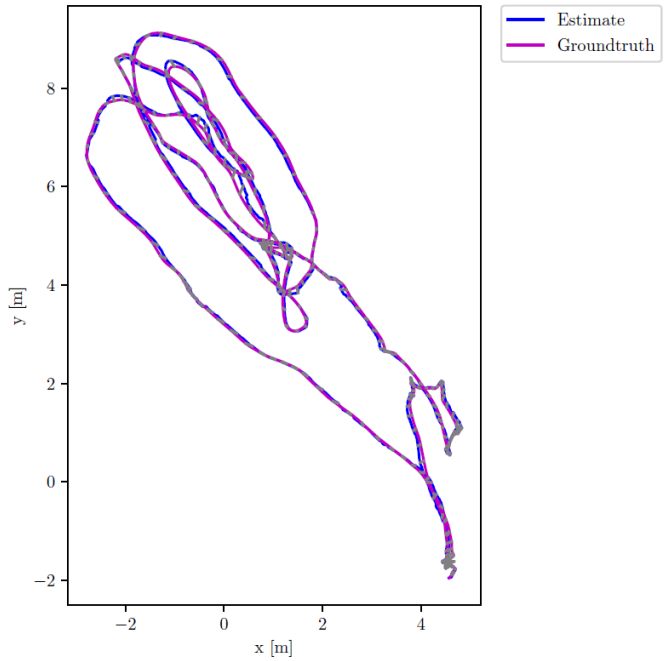


Fig. 11. Estimated and ground truth trajectory (top)

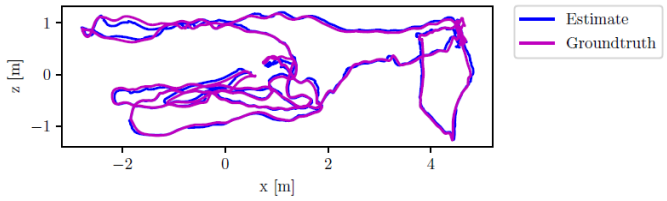


Fig. 12. Estimated and ground truth trajectory (side)

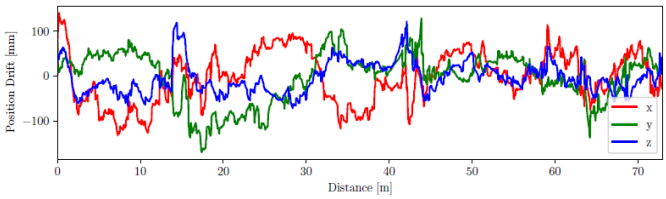


Fig. 13. Error between estimated and ground truth trajectories