RBE 549 Project 4
Deep and Un-Deep Visual Inertial Odometry

Niranjan Kumar Ilampooranan
MS Robotics Graduate Student
Worcester Polytechnic Institute

Thanikai Adhithiyan Shanmugam
MS Robotics Graduate Student
Worcester Polytechnic Institute

I. Introduction

The task of estimating depth from a single camera in Computer Vision has persistently presented a formidable challenge. In response, researchers have proposed leveraging stereo cameras positioned at a distance, facilitating depth estimation through the correlation of features extracted from images captured by both cameras. However, this method is constrained by its applicability primarily to static images, thus proving inadequate for dynamic scenes or images characterized by motion blur—common occurrences in robotics. Despite the promise of stereo vision in capturing three-dimensional information, its limitations in scenarios featuring motion blur necessitate further exploration of alternative methodologies. Addressing these constraints is pivotal for advancing depth estimation capabilities.

In this project, our focus lies on the fusion of sensor data from an Inertial Measurement Unit (IMU) and a camera to ascertain the state and localization of the robot—a technique commonly referred to as Visual Inertial Odometry (VIO). While an IMU proves effective in capturing rapid movements and sudden accelerations, scenarios where a camera falters, it suffers from drift over time. Conversely, a camera excels in providing accurate localization information but struggles with dynamic motion. Our project employs a filter-based stereo VIO approach, employing the MultiState Constraint Kalman Filter (MSCKF) methodology. We plan to assess this implementation's efficacy by conducting tests on the Machine Hall 01 easy subset of the EuRoC dataset

II. Dataset

This project utilizes the "Machine Hall 01 easy" (MH 01 easy) subset from the EuRoC dataset. This data is of a quadrotor equipped with a 6-DoF sensor, following a predefined flight trajectory. This system accurately tracks the quadrotor's movements, providing essential reference data for evaluating the performance of our implementation.

III. Implementation

The functions implemented as part of the MSCKF package are listed below:

A. *initialize_gravity_and_bias*

In typical applications, an Inertial Measurement Unit (IMU) comprises 6 degrees of freedom (DOF), with three for the accelerometer and three for the gyroscope. Ideally, the accelerometer should register [0 0 -g] in its initial state, while the gyroscope should read [0 0 0]. Nevertheless, inherent uncertainties in mechanical systems often lead to errors. This function aids in mitigating such uncertainties and errors by compensating for them.

This function initializes the initial orientation and bias based on the first readings from the IMU. The average angular and linear velocities are calculated from the IMU message buffer's first few readings. The gyro bias is initialized using the average angular velocity, and gravity is calculated using the linear acceleration. The normalized gravity vector is used as the IMU state and the initial orientation are set consistently with the inertial frame. The quaternions represent the final vector, where GI q denotes the rotation from the inertial frame to the body frame, which in this case is the IMU frame. The vectors GvI and GpI represent the body frame's velocity and position in the inertial frame, and bG and ba are the biases of the measured angular and linear velocities from the IMU. The representation of the final vector is given by the following expression

$$X_I = ({}^I_G\mathbf{q}^T \ \ \mathbf{b}_g^T \ \ {}^G\mathbf{v}_I^T \ \ \mathbf{b}_a^T \ \ {}^G\mathbf{p}_I^T \ \ {}^I_C\mathbf{q}^T \ \ {}^I\mathbf{p}_c^T)^T$$

B. *batch_imu_processing*

The function operates on IMU messages stored in the imu_msg_buffer, adhering to a specified time limit. It iterates through each IMU input within this timeframe, executing the process model until reaching the constraint. Subsequently, it advances the current IMU ID to the next state. Any remaining unused IMU messages are then purged from the buffer.

C. *process_model*

The process_model function aims to derive the camera module's pose dynamics from the latest IMU state update. It takes arguments such as time, gyro (current angular velocity), and acc(current linear acceleration). Subsequently, it computes the error for each IMU state using the formula given below

$$\tilde{X}_I = ({}^I_G\tilde{\theta}^T \ \tilde{\mathbf{b}}_g^T \ {}^G\tilde{\mathbf{v}}_I^T \ \tilde{\mathbf{b}}_a^T \ {}^G\tilde{\mathbf{p}}_I^T \ {}^I_C\tilde{\theta}^T \ {}^I\tilde{\mathbf{p}}_c^T)^T$$

The IMU state error can be calculated from the linearized continuous dynamics using the equation:

$$\dot{\tilde{X}}_I = F\tilde{X}_I + Gn_I$$

From the MSCKF paper, we refer to the F and G matrices:

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

he

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

The matrix is then approximated to its 3rd order as follows:

$$\phi = I_{21\times21} + F(\tau).d\tau + \frac{1}{2} * (F(\tau).d\tau)^2 + \frac{1.0}{6.0} * (F(\tau).d\tau)^3$$

D. predict_new_state

Upon acquiring the current system state, the process employs the fourth-order Runge-Kutta method to project the state forward and anticipate its new configuration. The predict_new_state function accepts parameters such as the time increment (d), gyroscope readings, and acceleration data for the current state. Initially, the method calculates the normalized error state of the angular velocity dataset. Subsequently, it proceeds to generate the Ω matrix through a specific set of steps, tailored to the requirements of the task. The Ω matrix is written as:

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

We acquire the present orientation, velocity and orientation from the IMU sever state function. With the current state and Ω matrix, we can compute the angular velocity and acceleration using the Runga-Kutta approximation method of 4th order given below:

$$k1 = f(t_n, y_n)$$

$$k2 = f(t_n + \frac{d\tau}{2}, y_n + k1 * \frac{d\tau}{2})$$

$$k3 = f(t_n + \frac{d\tau}{2}, y_n + k2 * \frac{d\tau}{2})$$

$$k4 = f(t_n + d\tau, y_n + k3 * d\tau)$$

We then convert the estimated orientation to quaternions after the approximation and update the velocity and position back to the current state. This is then used to estimate the next state in the sequence.

E. state_augmentation

This function computes the state covariance matrix, which aids in understanding and quantifying the uncertainty associated with the current state. Initially, we isolate the IMU and camera state parameters relevant to the rotation from the IMU to the camera and the translation vector from the camera to the IMU. Following this, we integrate a new camera state into the state server, leveraging the initial IMU and camera states. The approximated Jocabian matrix is given as:

$$J_I = \begin{bmatrix} C({}^I_G\hat{\mathbf{q}}) & 0_{3\times9} & 0_{3\times3} & I_3 & 0_{3\times3} \\ -C({}^I_G\hat{\mathbf{q}})^T \lfloor {}^I\hat{\mathbf{p}}_{c\times} \rfloor & 0_{3\times9} & I_3 & 0_{3\times3} & I_3 \end{bmatrix}$$

The state covariance matrix is resized and given as

$$P_{k+1|k} = \begin{bmatrix} P_{II_{k+1|k}} & \phi_k P_{IC_{k|k}} \\ P^T_{IC_{k|k}} \phi_k^T & P_{CC_{k|k}} \end{bmatrix}$$

$$P_{k+1|k} = \begin{bmatrix} P_{II_{k+1|k}} & \phi_k P_{IC_{k|k}} \\ P^T_{IC_{k|k}} \phi_k^T & P_{CC_{k|k}} \end{bmatrix}$$

F. add_feature_observations

This function determines the current IMU state ID and assesses the number of features present. Following this, we

sequentially add each feature from the feature message to the map server if it hasn't already been included. Additionally, we keep track of the count of tracked features. With each state update, the map server is refreshed to ensure continuous tracking of all features. The tracking rate is calculated as the ratio of tracked features to the total number of available features.

G. *measurement_update*

A measurement model is employed for refining state estimations. A residual, labelled as r, demonstrates a linear dependency on state errors, as indicated by the following expression.

$$\mathbf{r} = \mathbf{H}\tilde{\mathbf{X}} + noise$$

Here, H stands for the measurement Jacobian matrix, while the noise term signifies a zero-mean, white, uncorrelated state error. The estimated Kalman filter framework is applied. Initially, we check if the current values of H and r are both zero. Following this, we endeavour to simplify the Jacobian matrix's complexity using QR decomposition, aiming to reduce computational demands.

$$H_x = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} T_h \\ 0 \end{bmatrix}$$

Here, Q1 and Q2 are unitary matrices with columns that form bases for the range and null space of Hx., respectively. This is an upper triangular matrix. Next, we calculate the Kalman gain according to the equation:

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

After calculating the Kalman gain, we can find the state error using this equation

$$\Delta X = K r_n \tag{2}$$

The equation is updated by adjusting the IMU state using this state error, then proceeds to update the camera states. Finally, the state covariance undergoes an update, ensuring that the covariance matrix is adjusted to maintain symmetry.

IV. Results

The project utilizes data from the Machine Hall 01 easy (MH 01 easy) subset of the EuRoC dataset. The trajectory output depicted in Figure 1 corresponds well with the expected results. Additionally, the code files include a video showcasing the output. Below is the table that shows RMSE ATE (absolute trajectory error).

RMSE	Value
Rotation	1.449
Scale	1.051
Translation	0.081

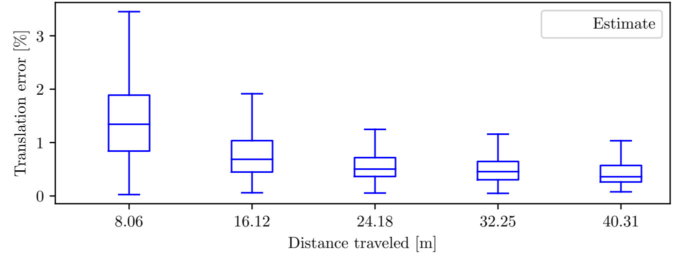

Fig. 1. Relative Translation Error Perception

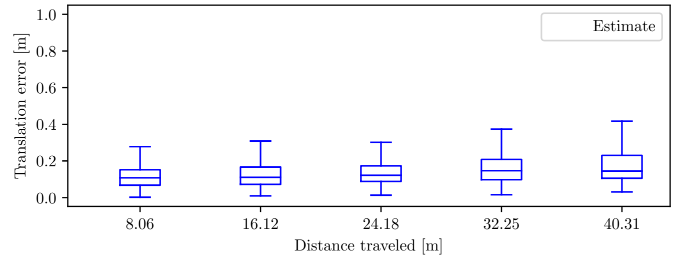

Fig. 2. Relative Translation Error

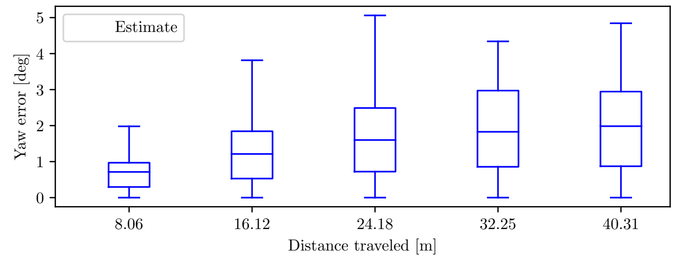

Fig. 3. Relative Yaw Error

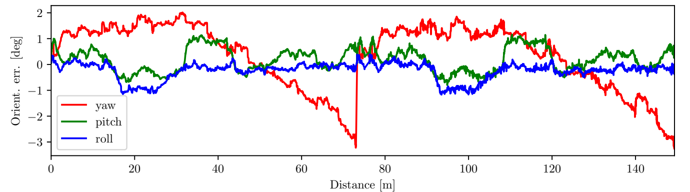

Fig. 4. Rotation Error

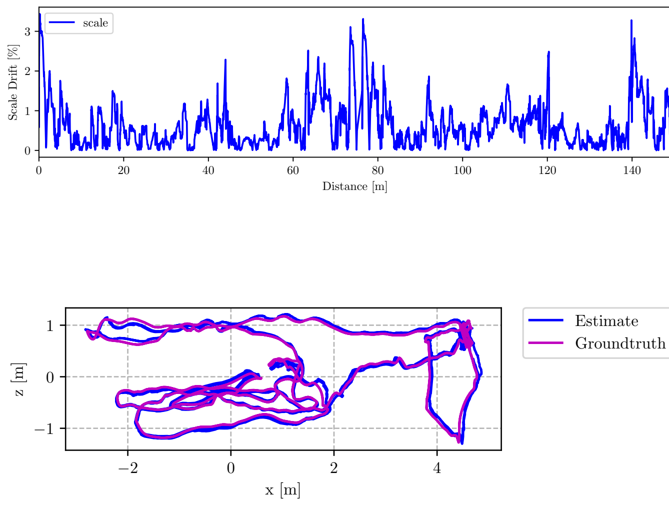

Fig. 5. Scale Error

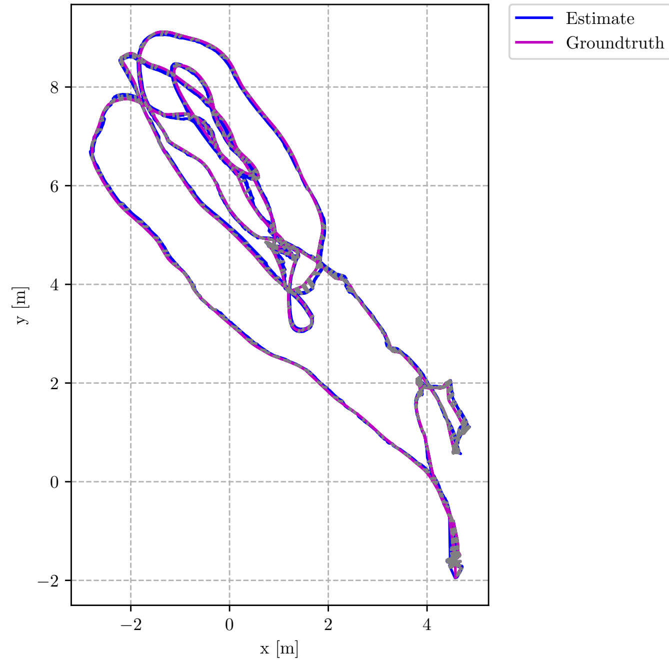

Fig. 6. Trajectory Top View

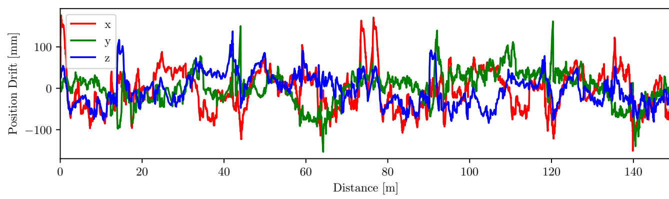

Fig. 7. Translation Error

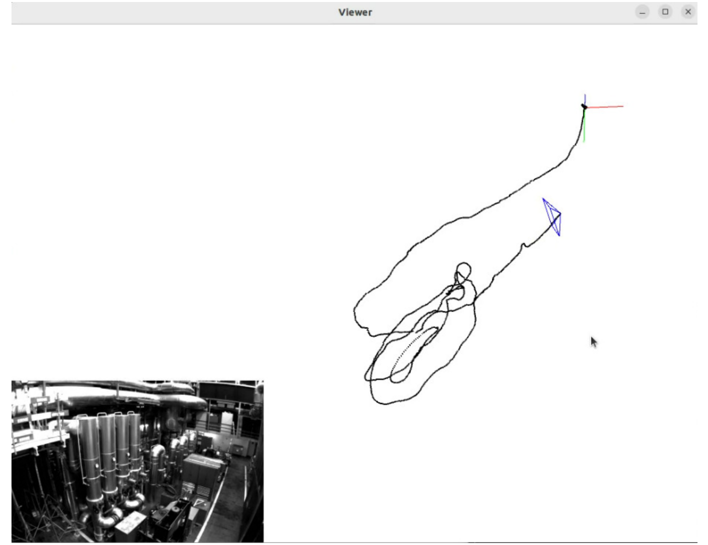

Fig. 8. Output Sample