P4 - Visual Inertial Odometry

Uday Sankar
usankar@wpi.edu

Gowri Shankar Sai Manikandan
gmanikandan@wpi.edu

Shaurya Parashar
sparashar@wpi.edu

***Abstract*—In this project, we aim to build a stereo visual inertial odometry system employing a filter-based approach, specifically the MultiState Constraint Kalman Filter (MSCKF). The fundamental mathematical principles of the stereo-MSCKF have been integrated into a provided base-code framework. The document systematically captures findings and insights obtained throughout each phase of the project.**

I. Introduction

The primary goal of this project is to determine scale from an image, which allows for depth evaluation. It is well-known that obtaining depth information from a single camera is challenging without any prior knowledge of the surroundings. A more straightforward alternative involves using a stereo camera with a known pose, where depth can be estimated by matching features. However, there are some limitations to this approach, such as the computational complexity and difficulty of matching, as well as its ineffectiveness in dealing with motion blur commonly found in robotic applications.

To address these challenges, an Inertial Measurement Unit (IMU) can be employed. A typical 6-DoF (Degrees of Freedom) IMU measures both linear and angular acceleration. IMUs excel in handling rapid movements and sudden jolts, where cameras often struggle, but they tend to drift over time, which is a weakness that cameras can compensate for. This complementary nature creates an opportunity for a multi-modal fusion problem, enabling accurate camera pose estimation and subsequent depth determination.

In this phase of the project, we implement [1] and refer to [2] for the mathematical model.

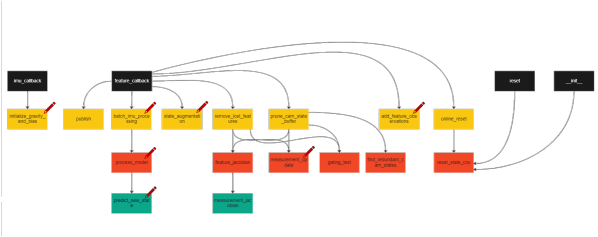

Fig. 1: VIO Pipeline

The pipeline involved in this process is shown in figure 1. In the given baseline starter code, there were seven functions that had to be written by us using the concepts discussed in the paper [ ].

II. Data

The data utilized for testing our implementation is the "Machine Hall 01 easy" (MH 01 easy), a subset of the larger EuRoC dataset. This data was gathered using a 6-DoF sensor mounted on a quadrotor, which followed a specific flight path. In order to obtain the ground truth for the system, a highly accurate Vicon Motion capture system with sub-millimeter precision was employed.

III. Implementation

An starter code has been supplied for the development of the Multi-State Constraint Kalman Filter (MSCKF). To accomplish the implementation of the model, we made modifications to specific functions within the msckf.py Python file. From the pipeline in figure 1, it is clear that there are seven functions to implemented in the msckf.py file. They are as follows:

A. *initialize_gravity_and_bias*

In this function, the initial IMU readings are used to establish the bias and initial orientation. The first few readings from the IMU message buffer are averaged to get the angular and linear velocities. The gyro bias is set using the average angular velocity, while gravity is determined by the linear velocity. Subsequently, the normalized gravity vector is incorporated as the IMU state.

By utilizing these two vectors, the initial orientation is set up to ensure consistency with the inertial frame. The quaternions are then incorporated as the IMU's orientation state. The resulting vector is expressed as follows [2]:

$$X_I = \begin{pmatrix} \hat{q}_{IG}^T \\ \hat{b}_g^T \\ \hat{v}_I^T \\ \hat{b}_a^T \\ \hat{p}_I^T \\ \hat{q}_C^T \\ \hat{p}_c^T \end{pmatrix} \tag{1}$$

B. *batch_imu_processing*

This function processes the IMU messages within the IMU message buffer based on a specified time constraint. The process model is executed for each IMU input within the given time frame, continuing until the time constraint is reached. Following this, the current IMU ID is updated to the subsequent state IMU ID. All remaining unused IMU messages are then removed from the IMU message buffer.

C. *process_model*

This function calculates the camera module's pose based on the latest IMU state update. It takes the current time, angular velocity (m_gyro), and linear acceleration (m_acc) as input

arguments. The error for each IMU state is computed and represented as follows:

$$\tilde{X}_I = \begin{pmatrix} \tilde{\theta}_{IG}^T \\ \tilde{b}_g^T \\ \tilde{v}_I^T \\ \tilde{p}_I^T \\ \tilde{\theta}_C^T \\ \tilde{p}_c^T \\ \tilde{b}_a^T \end{pmatrix} \tag{2}$$

The linearized continuous dynamics for the error IMU state are evaluated as:

$$\dot{\tilde{X}}_I = F_I \tilde{X}_I + G_I n \tag{3}$$

The discrete transition matrices F and Q are calculated as shown:

$$F = \begin{bmatrix} -[\hat{\omega}\times] & -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ -C\left(\hat{q}_I^T\right)[\hat{a}\times] & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left(\hat{q}_I^T\right) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{4}$$

$$G = \begin{bmatrix} -\mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C\left(\hat{q}_I^T\right) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_3 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix} \tag{5}$$

The matrix exponential is approximated up to the 3rd order as follows:

$$\phi = \mathbf{I}_{21\times21} + F(\tau)\cdot d\tau + \frac{1}{2}\cdot(F(\tau)\cdot d\tau)^2 + \frac{1}{6}\cdot(F(\tau)\cdot d\tau)^3 \tag{6}$$

Lastly, the state is propagated, and the new state is predicted using the 4th order Runge-Kutta method by calling the "predict_new_state" function.

In simpler terms, this function computes the camera module's pose based on the latest IMU state update. The error for each IMU state and the linearized continuous dynamics for the error IMU state are evaluated. Discrete transition matrices are calculated, and the matrix exponential is approximated. Finally, the state is propagated, and the new state is predicted using the 4th order Runge-Kutta method.

*D. **predict_new_state***

This function describes the process of propagating the state of an Inertial Measurement Unit (IMU) using a 4th order Runge-Kutta method. The inputs for this function are the time step $(d\tau)$, gyroscopic data (gyro), and acceleration data for the given state. In simpler terms, the function updates the orientation, velocity, and position of an IMU based on the provided inputs.

First, the error state of the angular velocity (gyro) is normalized. Next, the $(d\omega)$.matrix is computed using the normalized angular velocity:

$$\Omega(\hat{\omega}) = \begin{bmatrix} -[\hat{\omega}\times] & \omega \\ -\omega^T & 0 \end{bmatrix} \tag{7}$$

The current orientation, velocity, and position are obtained from the IMU state server. With these values, the angular velocity and acceleration are calculated. The Runge-Kutta method is then used to approximate the updated state. The method consists of four steps (k_1, k_2, k_3, k_4):

$$k_1 = f(t_n, y_n) \tag{8}$$

$$k_2 = f\left(t_n + \frac{d\tau}{2}, y_n + k1\cdot\frac{d\tau}{2}\right) \tag{9}$$

$$k_3 = f\left(t_n + \frac{d\tau}{2}, y_n + k2\cdot\frac{d\tau}{2}\right) \tag{10}$$

$$k_4 = f\left(t_n + d\tau, y_n + k_3\cdot d\tau\right) \tag{11}$$

After calculating the approximate orientation, it is converted to quaternions. The velocity and position of the current IMU state are updated based on the new approximation. These updated values are then used as the current state values for evaluating the next state.

*E. **state_augmentation***

In this function, we compute the state covariance matrix to propagate the uncertainty of the IMU and camera states. First, we obtain the rotation and translation values between the IMU and the camera. Then, we add a new camera state to the state server using the initial IMU and camera states.

Next, we update the state augmentation Jacobian, J_I, as follows:

$$J_I = \begin{bmatrix} C(\hat{q}_{I_G}) & \mathbf{0}_{3\times9} & \mathbf{0}_{3\times3} \\ -\left(C(\hat{q}_{I_G})\right)^T[\hat{p}_c\times] & \mathbf{0}_{3\times9} & \mathbf{I}_3 \\ \mathbf{0}_{3\times3} & \mathbf{I}_3 & \end{bmatrix} \tag{12}$$

We then resize the state covariance matrix and propagate the covariance of the IMU state. The full propagation of the uncertainty is represented as:

$$P_{k+1|k} = \begin{bmatrix} P_{II_{k+1|k}} & \phi_k P_{IC_{k|k}} \\ \left(\phi_k P_{IC_{k|k}}\right)^T & P_{CC_{k|k}} \end{bmatrix} \tag{13}$$

Finally, the augmented covariance matrix, $P_{k|k}$, is given as:

$$P_{k|k} = \begin{bmatrix} J_{21+6N} \\ J \end{bmatrix}^T \begin{bmatrix} P_{k|k}^{21+6N} \\ J \end{bmatrix} \tag{14}$$

We then update the state covariance in the server.

In simpler terms, this function calculates the uncertainty of the IMU and camera states using the state covariance matrix.

The state augmentation Jacobian is updated, and the covariance of the IMU state is propagated. Finally, the augmented covariance matrix is computed and used to update the state covariance in the server.

F. add_feature_observations

The following points describe

1) Obtain the feature msg as the input for this function.
2) Get the current IMU state ID.
3) Evaluate the number of features.
4) Append each feature one by one in the feature msg to the map server, if it is not already present in the map server.
5) Maintain a count of the number of features tracked.
6) Update the map server for every given state, tracking all the features.
7) Calculate the tracking rate as the ratio of the number of tracked features to the number of current features available.

G. measurement_update

In order to update the state estimates, a measurement model has been implemented. A residual **r** that depends linearly on the state errors is defined as follows:

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

In the aforementioned equation, H represents the measurement Jacobian matrix, while the noise component signifies a zero-mean, white, uncorrelated state error. We have employed an estimated Kalman filter structure in this context. Initially, we verify whether the existing H and r values are zero.

Subsequently, we attempt to minimize the complexity of the Jacobian matrix through the application of QR decomposition, which serves to decrease the computational demands as outlined below:

$$H_x = \begin{pmatrix} Q1 & Q2 \end{pmatrix} \begin{pmatrix} T_h \\ 0 \end{pmatrix}$$

In this case, Q1 and Q2 represent unitary matrices, with their columns constituting bases for the range and nullspace of H_x, respectively. Moreover, T_H denotes an upper triangular matrix. Following this, we determine the Kalman gain by employing the subsequent formula:

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

Here, K denotes the Kalman gain, P symbolizes the state covariance matrix, T_H^T represents the upper triangular matrix, and R_n stands for the covariance matrix of noise. After determining the Kalman gain, we proceed to compute the state error as follows:

$$\Delta X = Kr_n$$

Utilizing the calculated state error, the IMU state is updated initially, followed by adjustments to the camera states. Ultimately, the state covariance undergoes an update, and the covariance matrix is adjusted to achieve symmetry.

IV. Results

Since we faced difficulties regarding the visualization using Pangolin module, we got the visualization video from Irakli Grigolia's personal computer.

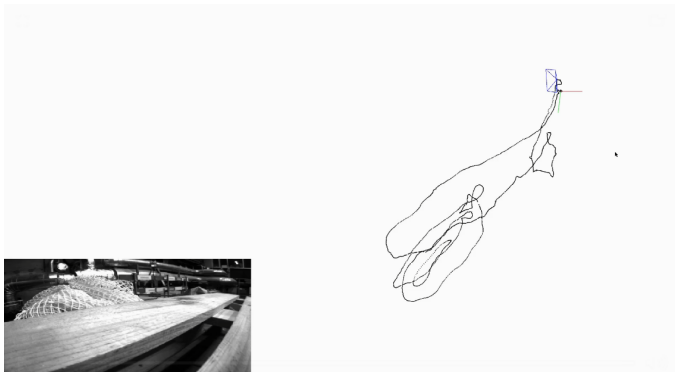

Fig. 2: Trajectory for EuRoC MH_01_easy

References

[1] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE international conference on robotics and automation*, pages 3565–3572. IEEE, 2007.

[2] Ke Sun, Kartik Mohta, Bernd Pfrommer, Michael Watterson, Sikang Liu, Yash Mulgaonkar, Camillo J Taylor, and Vijay Kumar. Robust stereo visual inertial odometry for fast autonomous flight. *IEEE Robotics and Automation Letters*, 3(2):965–972, 2018.