RBE 549: Project 3
P4: Classical Visual-Inertial Odometry

Deepak Harshal Nagle
Robotics Engineering
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: dnagle@wpi.edu
Telephone: (774) 519-8335

Irakli Grigolia
Computer Science
Worcester Polytechnic Institute
Worcester, Massachusetts 01609
Email: igrigolia@wpi.edu
Telephone: (508) 373-3402

***Abstract*—This report presents the implementation of a stereo visual-inertial odometry algorithm that utilizes the Multi-State Constraint Kalman Filter (MSCKF). The implementation is based on a pre-existing codebase provided as starter code. The paper details the mathematical concepts underlying the stereo-MSCKF algorithm and the implementation of the algorithm within the codebase. The results and observations for each step of the implementation are documented in this report.**

INTRODUCTION

The main objective of this project is to estimate depth from images by obtaining scale, which is a challenging task as depth cannot be directly obtained from a single camera without prior knowledge about the environment. A stereo camera with a known pose can be used to estimate depth by matching features, but this approach has limitations such as being computationally expensive and not working well with motion blur. To overcome these limitations, an Inertial Measurement Unit (IMU) can be used, which measures linear and angular acceleration. An IMU works well with fast movements and jerks where a camera fails but drifts over time, which is where a camera excels. Therefore, this complementary nature of the two systems can be used for accurate pose estimation and backtracking depth. The specific approach used in this project involves implementing a filter-based stereo visual inertial odometry using the MultiState Constraint Kalman Filter (MSCKF) and testing it on the Machine Hall 01 easy subset of the EuRoC dataset. The starter code provided by the authors of the S-MSCKF paper in Python will be modified to implement key functions for this approach.

DATA

Machine Hall 01 easy subset of the EuRoC dataset is used to test the implementation. The data is collected using a VI sensor carried by a quadrotor flying a trajectory. The ground truth is provided by a sub-mm accurate Vicon Motion capture system.

FUNCTIONS

We implement Multi-State Constraint Kalman Filter (MSCKF), some of the functions implemented are described below.

initialize_gravity_and_bias:

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 is set consistently with the inertial frame. The quaternions represent the final vector, where ${}^{\mathrm{G}}_{\mathrm{I}}\mathbf{q}$ denotes the rotation from the inertial frame to the body frame, which in this case is the IMU frame. The vectors ${}^{\mathrm{G}}\mathbf{v}_{\mathrm{I}}$ and ${}^{\mathrm{G}}\mathbf{p}_{\mathrm{I}}$ represent the body frame's velocity and position in the inertial frame, and $\mathbf{b}_{\mathrm{G}}$ and $\mathbf{b}_{\mathrm{a}}$ 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

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

batch_imu_processing:

The function deals with processing IMU messages from a buffer, taking into account a specified time range. It operates by running the process model for each IMU input that falls within the time range, repeating this process until the end of the range is reached. Once completed, the current IMU ID is updated to the next state ID, and any unused IMU messages are removed from the buffer.

process_model:

The aim of this function is to determine the camera module's pose (dynamics) based on the most recent update of the IMU state. To achieve this, the function takes in the time, m_gyro (current angular velocity), and m_acc (current linear acceleration) as arguments. Following this, the function calculates the error for each IMU state, as represented by the following formula.

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

The linearized continuous dynamics for the error IMU state is calculated as follows:

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

F and Q (discrete transition matrices) are calculated as follows:

$$F = \begin{bmatrix} -\lfloor\hat{\omega}_\times\rfloor & -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C({}^I_G\mathbf{q}^T)\lfloor\hat{a}_\times\rfloor & 0_{3\times3} & 0_{3\times3} & -C({}^I_G\mathbf{q}^T) & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix}$$

$$G = \begin{bmatrix} -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & -C({}^I_G\mathbf{q}^T) & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & I_3 \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix}$$

The matrix exponential is approximated to third order as follows:

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

predict_new_state:

After obtaining the current state, we apply the fourth-order Runge-Kutta method to propagate the state and predict its new value. Specifically, we use a function called "predict new state," which takes as input the time step (dτ), the gyroscope data, and the acceleration information for the current state. To begin, we calculate the normalized error state of the angular velocity data. Next, we compute the Ω matrix by following a specific procedure.

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

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

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

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

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

We retrieve the present orientation, velocity, and position data from the IMU state server. Based on the current state and the Ω values, we compute the angular velocity and acceleration, which we then approximate using the Runge-Kutta method.

Once we have computed the estimated orientation, we convert it into quaternions and use this information to update the velocity and position data for the current IMU state. These updated values are then used as the current state information to determine the next state in the sequence.

State_Augmentation:

In this step, our aim is to calculate the state covariance matrix, which will help us in disseminating the ambiguity of the given state. Initially, we extract the IMU and camera state values that correspond to the rotation from the IMU to the camera and the translation vector from the camera to the IMU. Subsequently, we incorporate a fresh camera state into the state server, utilizing the initial IMU and camera state. The augmentation Jacobian is calculated as follows:

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

The state covariance matrix is resized and the IMU state is propagated:

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

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

This is the Augmented covariance matrix. It is regularly updated through this function.

F. Incorporating feature observations

Firstly, the feature message is acquired as input for this function. We then determine the current IMU state ID and

evaluate the number of features. Following that, we successively add each feature from the feature message to the map server if it isn't already present. Additionally, we maintain a count of the tracked features. For every given state, the map server is updated, and all features are tracked. The tracking rate is computed as the ratio of the number of tracked features to the total number of available features.

G. Updating measurements

A measurement model is utilized to update state estimates. A residual, denoted as r, is linearly dependent on state errors, represented by the following relation [3]:

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

In the equation above, H represents the measurement Jacobian matrix, and the noise term denotes a zero-mean, white, uncorrelated state error. The estimated Kalman filter framework is implemented. Initially, we examine whether the existing H and r values are zero. We then attempt to reduce the complexity of the Jacobian matrix using QR decomposition to minimize computation requirements:

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

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

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

In this equation, K represents the Kalman gain, P is the state covariance matrix,T_H^T is the upper triangular matrix, and R_n is the noise covariance matrix. After calculating the Kalman gain, the state error is computed as:

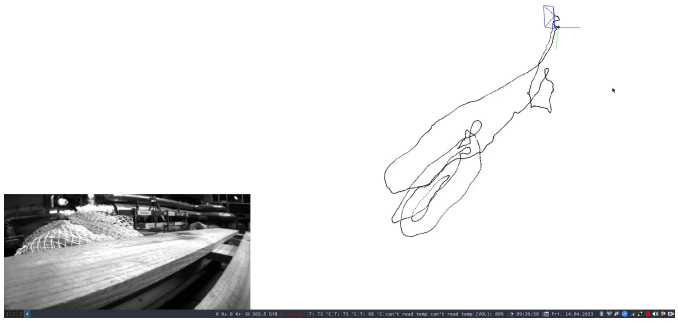
ΔX = K r_n

Using this state error, the IMU state is updated first, followed by the camera states. Lastly, the state covariance is updated, and the covariance matrix is adjusted to be symmetric.

IV. Results

The input data employed for this project is sourced from the Machine Hall 01 easy (MH 01 easy) subset of the EuRoC dataset. Figure 1 displays the trajectory output for this data, which aligns with the anticipated outcome. The output video is included with the code files.

Final plot:



REFERENCES

[1] https://www-users.cse.umn.edu/ stergios/papers/ICRA07-MSCKF.pdf
[2] https://rbe549.github.io/spring2023/proj/p4/
[3] https://arxiv.org/pdf/1712.00036.pdf