Visual Inertial Odommetry RBE549 Project 4

Shiva Kumar Tekumatla
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, U.S.A.
stekumatla@wpi.edu

Ajith Kumar Jayamoorthy
Robotics Engineering Department
Worcester Polytechnic Institute
Worcester, MA, U.S.A.
ajayamoorthy@wpi.edu

***Abstract*—This document consist of project implementation of a filter-based stereo visual inertial odometry that uses the Multi-State Constraint Kalman Filter (MSCKF). The mathematical concepts of a stereo-MSCKF are implemented within a framework of given starter-code. The results and observations for each step has been recorded in this document.**

I. Introduction

The main objective of the project is to obtain scale from a image and thus evaluated depth. As we have understood before it is not possible to obtain depth from a single camera without any prior information about the environment. A simpler alternative solution would be to utilize a stereo camera with a known pose, where we can directly estimate depth by matching features. However, there are first a few drawbacks to just matching a image. First of all, matching is expensive and hard, and secondly, this does not work when there is motion blur which is common on robots. We can solve this by using an Inertial Measurement Unit (IMU). A common 6-DoF (Degree of Freedom) IMU measures linear acceleration and angular acceleration. The best part about an IMU is that it works well with fast movement and jerks where camera fails but drifts over time in which camera excels. Hence, this complementary nature lends to a beautiful multi-modal fusion problem which can be used to estimate accurate pose of the camera to backtrack depth. [1]

II. DATA

The data we would be using is Machine Hall 01 easy (**MH_01_easy**), a subset of the **EuRoC** dataset to test our implementation. The data has been collected using a 6-DoF sensor carried by a quad-rotor flying in a trajectory. The ground truth for the system has been obtained from a sub-mm accurate Vicon Motion capture system.

III. IMPLEMENTATION

A started code has been provided for the implementation of the Multi-State Constraint Kalman Filter (MSCKF). We updated the following functions in a msckf.py python file to complete the implementation of the model.

A. *Initialize_gravity_and_bias*

In this function the bias and initial orientation are initialized based on the starting IMU reading. The angular and linear velocity are obtained by averaging the imu_msg_buffer first few readings. They gyro bias is initialized with the average angular velocity and the gravity is obtained by linear velocity. Then the normalized gravity vector is passed as IMU state. Then from these two vectors we initialize the initial orientation, so that the estimation is consistent with the inertial frame. Then the quaternions are passed as orientation state of the IMU. The final vector is represented as follows [2]:

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

where the quaternions ${}^I_G\mathbf{q}$ represents the rotation from the inertial frame to the body frame. The body frame in our case is considered to be the IMU frame. The vectors ${}^G\mathbf{v}_I$ and ${}^G\mathbf{p}_I$ represent the velocity and position of the body frame in Inertial frame. $\mathbf{b}_g$ and $\mathbf{b}_a$ are the biases of measure angular and linear velocity from the IMU.

B. *batch_imu_processing*

In this function we process the imu messages in the imu_msg_buffer given the time bound. We first run the process model for every imu input in each time bound. This is repeated until the time bound is reached. After that the current imu id is update to the next state imu id. All the unused imu messages are removed from the imu_msg_buffer.

C. *process_model*

The purpose of the function is to calculated the dynamics (pose) of the camera module state computed from the latest IMU state update. First the IMU state update are obtained arguments to the function in the form of time, m_gyro (current angular velocity) and m_acc (current linear acceleration). Next the error for each IMU state is calculated as and is represented as follows [2]:

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

We evaluate the linearized continuous dynamics for the error IMU state as [2]:

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

We calculate the discrete transition matrices F and Q as follows [3]:

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

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

Next we approximate the matrix exponential to the 3rd order as follows:

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

Next we propagate the state and predict new state using the 4^{th} order Runge-Kutta method by using the **predict_new_state function**.

D. *predict_new_state*

In this function the state is propogated using 4^{th} order Runge-Kutta method [2]. The input for this function is the time step dτ, gyro and acceleration for that given state. First we start by computing the normalized value of the error state of angular velocity (gyro). Then we work on getting the Ω matrix as follows:

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

We then obtain the current state of orientation, velocity and position from the imu_state server. Using the current values and the Ω values, the angular velocity and the angular acceleration are calculated and the values are further approximated using the Runge-Kutta method.

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

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

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

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

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

E. *state_augmentation*

In this function we are going to compute the state covariance matrix to propagate the uncertainty of the state. We first obtain the IMU and camera state values corresponding to the rotation from the IMU to camera and the translation vector from the camera to the IMU. Next we add a new camera state to the state server by using the initial imu and camera state.

Next we update the state augmentation Jacobian $\mathbf{J_I}$ given as follows:

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

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

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

Finally The Augmented covariance matrix $\mathrm{P_{k|k}}$ is given as follows:

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

Then we update the state covariance in the server.

F. *add_feature_observations*

We first obtain the feature_msg as the input to the for this function. The current imu state id is obtained and the number of features is evaluated. Then we append each feature one by one in the feature_msg to the map_server, if it is not already present in the map_server.We also have a count of the number of features tracked. for every given state, the map_server is updated and all the features are tracked. The tracking rate is calculated as the ratio of number of tracked features to the number of current features available.

G. *measurement_update*

A measurement model has been employed for updating the state estimates. A residual **r** is defined that depends linearly on the state errors, , by the given relation [3]:

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

In the above equation, **H** is the measurement Jacobian matrix and the noise term is a zero-mean, white, uncorrelated state error. Estimated Kalman filter framework has been implemented. First we try to check if the existing **H** and **r** are zero. After that we try to reduce the complexity of the Jacobian matrix by using **QR decomposition**, to reduce the computation requirements as follows:

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

where, $\mathrm{Q_1}$ and $\mathrm{Q_2}$ are unitary matrices whose columns form bases for range and nullspace of $\mathrm{H_x}$, respectively and $\mathrm{T_H}$ is an upper triangular matrix. Next, we compute the Kalman gain according to the equation:

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

where, K is the Kalman gain, P is the state covariance matrix, T_H^T is the upper triangular matrix and R_n is the covariance matrix of noise. Following the calculation of Kalman gain, the state error is computed as :

$$\Delta X = K r_n$$

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

IV. Results

The input data used for this project is from Machine Hall 01 easy or (MH_01_easy) subset of the **EuRoC** dataset. The figure 1 shows the trajectory output for this data and it is similar to the expected output. The video of this output is attached along with the code files.

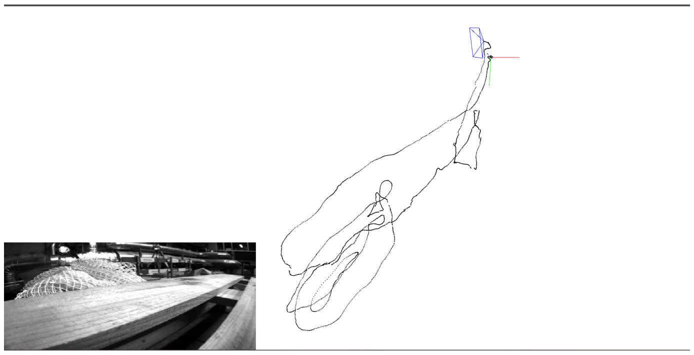

Fig. 1. Trajectory output for the MH_01_easy data

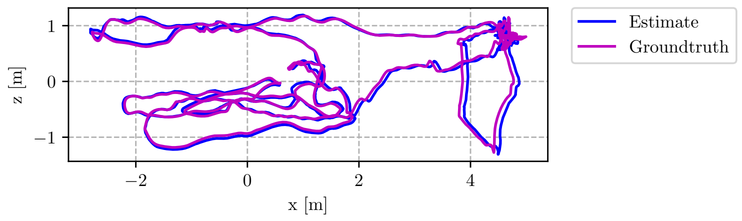

Fig. 2. Trajectory output for the MH_01_easy data

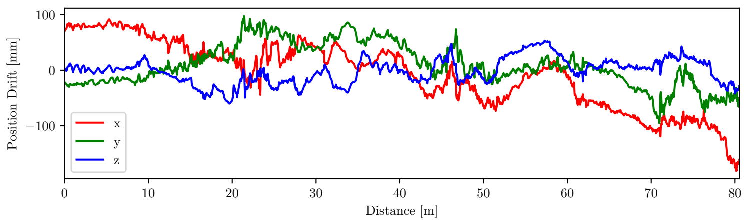

Fig. 3. Trajectory output for the MH_01_easy data

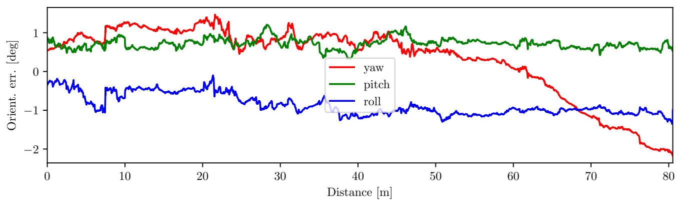

Fig. 4. Trajectory output for the MH_01_easy data

The following are the error statistics:

1) Rotation:
   a) max: 2.5960746020383834
   b) mean: 1.4915827054175759
   c) median: 1.4271887897129485
   d) min: 0.9230520272754749
   e) num_samples: 3639
   f) rmse: 1.5202436938405717
   g) std: 0.29380592499337316
2) scale:
   a) max: 3.278213410981534
   b) mean: 1.1981694701227577
   c) median: 1.2653058044715615
   d) min: 0.000625509235674393
   e) num_samples: 3639
   f) rmse: 1.4313479030454652
   g) std: 0.7830368704080289
3) translation:
   a) max: 0.18863288477670725
   b) mean: 0.07658838915096859
   c) median: 0.07894468624273385
   d) min: 0.0157409411189315
   e) num_samples: 3639
   f) rmse: 0.08137476450369072
   g) std: 0.027496744267111357

References

[1] https://rbe549.github.io/fall2022/proj/p4/
[2] https://arxiv.org/pdf/1712.00036.pdf
[3] https://www-users.cse.umn.edu/ stergios/papers/ICRA07-MSCKF.pdf
[4] https://github.com/KumarRobotics/msckf_vio