RBE/CS-549: Project 4 - Classical approach for Visual Inertial Odometery

Dhrumil Sandeep Kotadia, Dhiraj Kumar Rouniyar, Krunal M. Bhatt

I. Introduction

In the project 4, we aim to learn and implement Visual Inertial Odometry stack from scratch. As we know a single camera struggles to estimate the scale of the environment. We combining visual data with inertial , can estimate the scale, providing a more accurate understanding of the environment. Cameras also struggle with motion blur when the camera is in motion. IMU performs well under these conditions. Hence, fusing the data from both the IMU and the camera can facilitate wider range of applications.

In the first phase we implement a traditional approach to fusing visual and inertial data to obtain Odometry. We implemented [1] from Dr. Vijay Kumar's lab which is a phenomenal work on VIO attaining 20 m/s autonomous flight using the VIO approach. We also refer to [2] seminal MSCKF paper for a robust understanding of the mathematical model and the fundamentals. Fig. 1 shows the structure of the code base that is used. Boxes marked with a pencil icon are implemented from scratch.

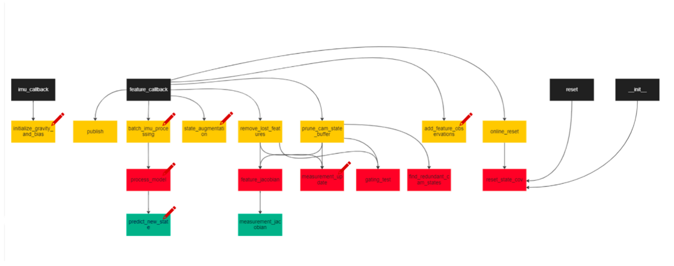

Fig. 1: Pencil icon indicates the functions implemented from scratch

II. Phase1: Classical Approach

We are using the Machine Hall 01 dataset, a subset of the EuRoC dataset, to test our implementation. Data is collected using a VI Sensor carried by a quadrotor flying a trajectory. The ground truth is a sub-mm accurate Vicon Motion Capture system data. In this section the following sub-sections define the process of estimating a trajectory using VIO. The final subsection talks about the error analysis between the ground truth and the estimated trajectory. We perform an SE(3) alignment before computing the error.

The authors are with the Robotics Engineering Department of Worcester Polytechnic Institute, Worcester, MA 01609 USA(email: dkotadia@wpi.edu; dkrouniyar@wpi.edu; kmbhatt@wpi.edu)

A. Initialization of Gravity and Bias

Calibration is needed to account for the bias in a 6-DoF IMU used for gyroscope and accelerometer measurements. We calculate the bias by taking mean of the stationary readings and subtract it from subsequent readings. $[0, 0, 0]$ should be the gyroscope reading, ideally, but it might have negligible fluctuations. We initialize the gravity g as the norm of the first 200 readings, as $[0, 0, g_{norm}]$.

The function "$initialize\_gyro\_and\_bias$" initializes parameters for IMU. It also initializes the initial orientation for the robot based on few initial readings. It averages the angular velocity and linear acceleration values from the IMU. Initial Orientation is set to be consistent with the inertial frame.

B. Batch IMU Processing

We want to first process all the IMU messages received prior to the new feature's time stamp, when we receive a new feature. We use the following state vector for estimating the next states consisting of states related to camera and IMU, including quaternion for rotation, bias for gyroscope and accelerometer, position and velocity. The "$batch\_imu\_processing$" function propagates the state of IMU within specified time bound. It iterates through the IMU buffer. For each unprocessed IMU, this function applies the process model to update the state based on linear acceleration and angular velocities.

$$x_I = [{}_G^I q^T \ b_g^T \ {}^G v_I^T \ b_a^T \ {}^G p_I^T \ {}_C^I q^T \ {}^I p_C^T]^T \tag{1}$$

IMU state's timestamp and ID are updated, and the processed messages are removed.

C. Process Model

The function "$process\_model$" sends the system state and covariance using a 4^{th} order Runge-Kutta integration method. It takes the current system state, and then calculates the time step based on the provided time and IMU state time. The following equation models the IMU system in continuous time.

$$\begin{aligned} {}_I^G\dot{\bar{q}} &= \frac{1}{2}\Omega(\omega(t)){}_I^G\bar{q}(t), \\ \dot{b}_g(t) &= n_{wg}(t), \\ {}^G\dot{v}_I(t) &= {}^G a(t), \\ \dot{b}_a(t) &= n_{wa}(t), \\ {}^G\dot{p}_I(t) &= {}^G v_I(t) \end{aligned} \tag{2}$$

Here, ${}_I^G\dot{\bar{q}}$ is the quaternion describing the rotation from global frame G to IMU frame I. We also have the $\omega(t) = [\omega_x, \omega_y, \omega_z]^T$, rotational velocity in IMU frame. Also,

$$\Omega(\omega) = \begin{bmatrix} -\lfloor \omega_x \rfloor & \omega \\ -\omega^T & 0 \end{bmatrix} \tag{3}$$

$$\lfloor \omega_x \rfloor = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} \tag{4}$$

We can write accelerometer and gyroscope measurements, ω_m and a_m as

$$a_m = {}^I_G R({}^G a - {}^G g + 2\lfloor \omega_{Gx} \rfloor {}^G v_I + 2\lfloor \omega_{Gx} \rfloor^{2G} p_I) + b_a + n_a \tag{5}$$

$$\omega_m = \omega + b_g + n_g \tag{6}$$

Here, ${}^I_G R$ is the rotation matrix calculated from quaternion ${}^G_I \bar{q}$. Now, applying expectation operator to equation 1, we get the values of next state of IMU as:

$$\begin{gathered} {}^G_I \dot{\hat{\bar{q}}} = \frac{1}{2}\Omega(\hat{\omega}(t)^G_I)\hat{\bar{q}}, \\ \dot{\hat{b}}_g = 0_{3\text{x}1}, \\ {}^G\dot{\hat{v}}_I = C^T_{\hat{q}} \hat{a} - 2\lfloor \omega_{Gx} \rfloor {}^G \hat{v}_I + \lfloor \omega_{Gx} \rfloor^{2G} \hat{p}_I + {}^G g, \\ \dot{\hat{b}}_a = 0_{3\text{x}1}, \\ {}^G\dot{\hat{p}}_I = {}^G \hat{v}_I \end{gathered} \tag{7}$$

We now have the linearized continuous-time model for IMU error-state is:

$$\dot{\tilde{X}}_{IMU} = F\tilde{X} + Gn_{IMU} \tag{8}$$

Here we have discrete transition matrix (F) and noise covariance matrix (G) as follows to describe the system dynamics and noise characteristics:

$$F = \begin{bmatrix} \lfloor \hat{\omega_X} \rfloor & -I_3 & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} \\ 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} \\ -C({}^I_G\hat{q})^T \lfloor \hat{a}_X \rfloor & 0_{3\text{x}3} & 0_{3\text{x}3} & -C({}^I_G\hat{q})^T & 0_{3\text{x}3} \\ 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} \\ 0_{3\text{x}3} & 0_{3\text{x}3} & I_3 & 0_{3\text{x}3} & 0_{3\text{x}3} \\ 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} \\ 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} \end{bmatrix} \tag{9}$$

$$G = \begin{bmatrix} -I_3 & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} \\ 0_{3\text{x}3} & I_3 & 0_{3\text{x}3} & 0_{3\text{x}3} \\ 0_{3\text{x}3} & 0_{3\text{x}3} & -C({}^I_G\hat{q})^T & 0_{3\text{x}3} \\ 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & I_3 \\ 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} & 0_{3\text{x}3} \end{bmatrix} \tag{10}$$

The F matrix is approximated using a 3rd order matrix exponential method, assuming a really small time step (dt). Code predicts the new system state using "*dict_new_state*" which does a 4^{th} order Range-Kutta integration method. Transition matrix (ϕ) is modified by taking into account the space of states that are not directly observable from the sensor measurements.

The state covariance matrix (Q), which represents the uncertainties or errors in the system model, is updated with the help of ϕ and G. Finally, state covariance matrix is fixed to be symmetric by averaging it with its transpose. IMU state is updated and it serves as the null space values for the next state estimation.

D. Prediction of new state

On every new IMU measurement, the "$predict_new_state$" function uses the 4^{th} order Range-Kutta numerical integration to update the estimation.

E. State Augmentation

The function "*state_augmentation*" augments the state by adding a new camera state to the state vector, updates the covariance matrix, and ensures the symmetry on addition of new images. We update the camera position ${}^G p_C$ and orientation ${}^C q_G$ with last IMU state and augment the state covariance matrix P.

The pose of the camera can be calculated as,

$${}^G p_C = {}^G p_I + C({}^C_G q)^{TI} p_C \tag{11}$$

$${}^C_G q = {}^C_I q \otimes {}^I_G q \tag{12}$$

$$J = [J_1 \; O_{6\text{x}6N}] \tag{13}$$

$$J_1 = \begin{bmatrix} C({}^I_C q) & 0_{3\text{x}9} & 0_{3\text{x}3} & I_3 & & 0_{3\text{x}3} \\ \lfloor C({}^I_G q)^{TI} p_C \rfloor & 0_{3\text{x}9} & I_3 & 0_{3\text{x}3} & I_3 & \end{bmatrix} \tag{14}$$

$$P_{k|k} = \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix} P_{k|k} \begin{bmatrix} I_{21+6N} \\ J \end{bmatrix}^T \tag{15}$$

F. Add Feature Observations

The function "*add_feature_observations*" adds feature observations from a new image frame to the map server in VIO. It creates a new map that shows the unobserved features.

G. Updating the Measurements

Here, the "*measurement_update*" function updates from visual features. Measurements of stereo and position features in left and right camera are given as below:

$$z^j_i = \begin{bmatrix} u^j_{i,1} \\ v^j_{i,1} \\ u^j_{i,2} \\ v^j_{i,2} \end{bmatrix} = \begin{bmatrix} \frac{1}{{}^{C_{i,1}} z_j} & 0_{2\text{x}2} \\ 0_{2\text{x}2} & \frac{1}{{}^{C_{i,2}} z_j} \end{bmatrix} \begin{bmatrix} {}^{C_{i,1}} X_j \\ {}^{C_{i,1}} Y_j \\ {}^{C_{i,2}} X_j \\ {}^{C_{i,2}} Y_j \end{bmatrix} \tag{16}$$

and

$${}^{C_{i,1}} p_j = \begin{bmatrix} {}^{C_{i,1}} X_j \\ {}^{C_{i,1}} Y_j \\ {}^{C_{i,1}} Z_j \end{bmatrix} = C({}^{C_{i,1}}_G q)({}^G p_j - {}^G p_{c_{i,1}}) \tag{17}$$

$${}^{C_{i,2}} p_j = \begin{bmatrix} {}^{C_{i,2}} X_j \\ {}^{C_{i,2}} Y_j \\ {}^{C_{i,2}} Z_j \end{bmatrix} = C({}^{C_{i,2}}_G q)({}^G p_j - {}^G p_{c_{i,2}}) \tag{18}$$

QR decomposition is used to decompose the H matrix, when the rows are more than the columns. This results in a H_thin matrix. Kalman gain, is used to determine the weight of measurement and is computed with the help of the aforementioned H_thin matrix, covariance P, and noise covariance.

$$K = PT_H^T(T_H PT_H^T + R_n) \tag{19}$$

Taking the inverse of this matrix is computationally unstable, so we can use the Ax=b trick to solve the problem. We finally update the covariance matrix P as:

$$P_{k+1|k+1} = (I_{k\text{x}k} - KT_H)P_{k|k} \tag{20}$$

H. Trajectory Error Evaluation

This subsection shows the plots for error between Ground Truth(pink) and Estimated Trajectory(blue) with the $rpg\_trajectory\_evaluation$ toolbox [3]. The absolute trajectory error (ATE) and the root mean square translation error (RMSE) is shown in the below table I, II and III for Rotation, Scale and Translation respectively. Further, the output are represented in the Fig. 2, 3, 4, 5, 6 and 7.

TABLE I: Absolute Error_Rotation

Metric	Value
Maximum	179.9886173933859
Mean	170.59123433130742
Median	172.9962809779764
Minimum	143.294222905647
Number of Samples	2921
RMSE	170.7202200317232
Standard Deviation	6.635080783301496

TABLE II: Absolute Error_Scale

Metric	Value
Maximum	11.298875556865084
Mean	1.8612449174449517
Median	1.4386692855281602
Minimum	0.00048575769031611316
Number of Samples	2921
RMSE	2.480465707298732
Standard Deviation	1.6396577332999516

TABLE III: Absolute Error_Translation

Metric	Value
Maximum	0.9077707681773487
Mean	0.16196289916673598
Median	0.13821373270450427
Minimum	0.0033654260529236814
Number of Samples	2921
RMSE	0.18829498416429352
Standard Deviation	0.0960365573879929

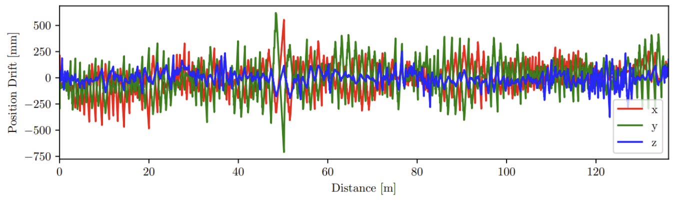

Fig. 2: Translation Error

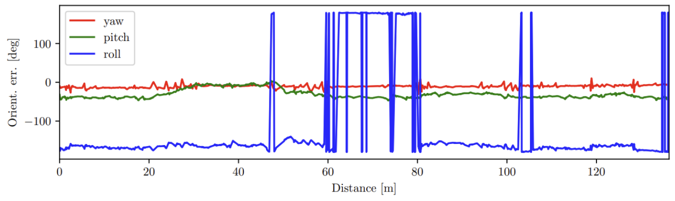

Fig. 3: Rotation Error

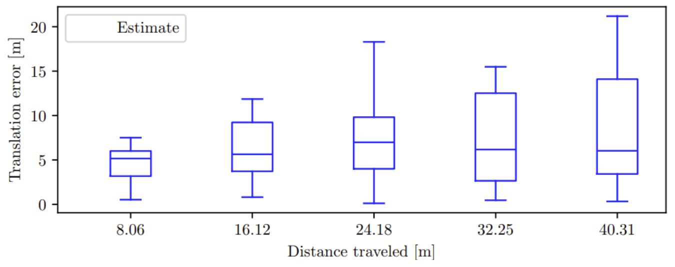

Fig. 4: Relative Translation Error

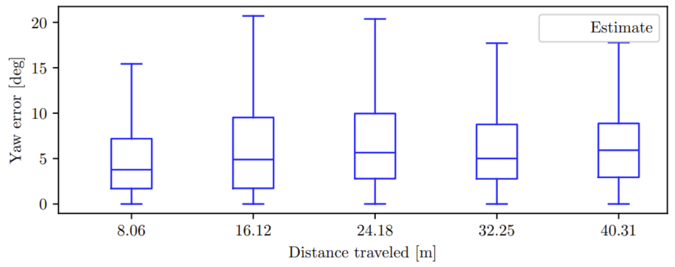

Fig. 5: Relative Yaw Error

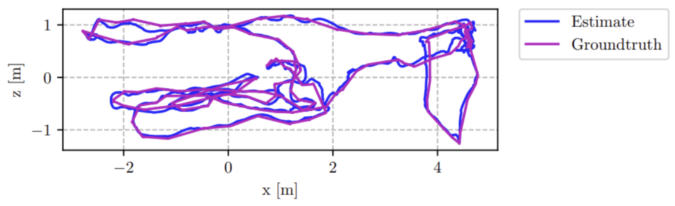

Fig. 6: Trajectory Side View

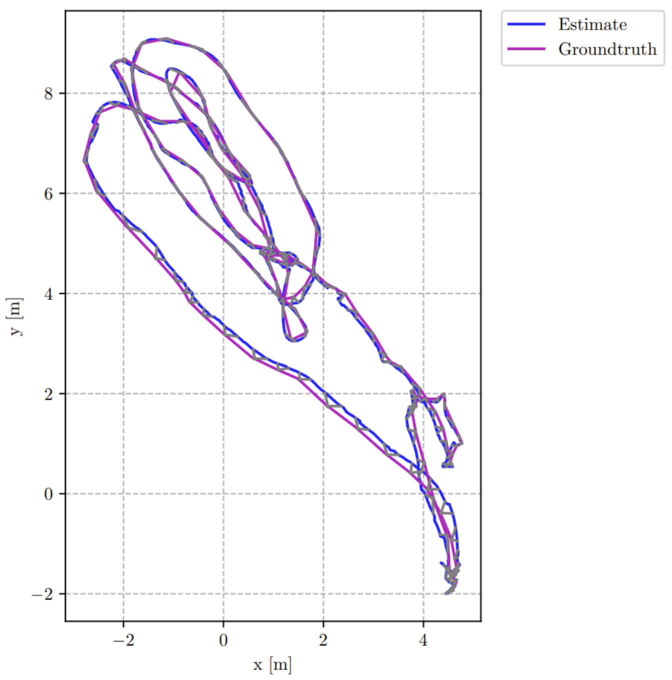

Fig. 7: Trajectory Top View

References

[1] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," 2018.

[2] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3565–3572.

[3] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 2502–2509.