Project 4: Deep and Un-Deep Visual Inertial Odometry
RBE 549: Computer Vision

Riley Blair
School of Engineering
Worcester Polytechnic Institute
Email: rpblair@wpi.edu

Scott Pena
School of Engineering
Worcester Polytechnic Institute
Email: sbpena@wpi.edu

***Abstract*—Visual Inertial Odometry (VIO) aims to provide accurate odometry of a moving object (like a drone or car) in the wake of imperfect sensors. An Inertial measurement Unit (IMU) is noisy, and prone to drifting over long periods of integration, whereas a camera is data-hungry, and susceptible to short-term errors (motion blur, data-loss), when polled infrequently. By combing the two in a Multi-State Constraint Kalman Filter (MSCKF), we aim to create a high-precision robust tracking software stack for any mobile system.**

I. Introduction

A classic problem in computer vision and robotics is detecting scale and depth via onboard sensors. Using a single RGB camera, it is impossible to obtain depth without any prior knowledge of the environment without using a deep learning model. Even then, the model is highly constrained to the dataset it was trained on. An alternative solution is to use a stereo camera, but obtaining depth and scale using stereo is computationally expensive and can be very slow. The best option is to use an IMU onboard the robot. An IMU can be fused with the power of an RGB camera to estimate the camera's pose and depth within the camera's image. The process to accomplish localization using the combination of computer vision and an IMU sensor is known as Visual-Inertial Odometry, or VIO. This paper highlights our implementation of VIO on a drone flying in an unknown environment, using the implementation of a Multi-State Contraint Kalman Filter provided by a paper written by Anastasios I. Mourikis and Stergios I. Roumeliotis. [1]

II. Dataset

The dataset provided for this assignment was the Machine Hall 01 Easy, or `MH_01_easy` subset of the EuRoC dataset. The dataset was generated via a VI sensor integrated within a drone flying a trajectory in an environment. This recorded data regarding the IMU readings for linear acceleration and angular velocity, as well as captured a video onboard the drone. The ground truth of the drone's pose was recorded using the Vicon Motion capture system. The IMU readings and RGB images were used to perform our implementation of VIO, where we compared our output with the ground truth trajectory.

III. Implementation

A. Initialize Gravity and Bias

A strong disadvantage to using an IMU is that the readings for linear acceleration and angular velocity are prone to high amounts of error due to unpreventable mechanical inconsistencies. To combat the readings from drifting too far when the drone first launches, we initialize the IMU's gravity vector and the IMU's bias for both the linear acceleration and angular velocity.

We assume the drone is stationary at the beginning of the flight, and therefore can calculate the biases before any movement has occurred. For the first buffer of IMU messages (200 messages), we calculate the average linear acceleration and the average angular velocity. We set the gyroscope's bias to be equal to the average angular velocity over the 200 messages. To find the gravity vector, we calculate the norm of the average linear acceleration. Since the IMU detects the acceleration of gravity, we can create the gravity vector to be the calculated norm pointed in the negative z axis. Now that we have estimated the bias of the gyroscope and the gravity vector, we can also initialize the IMU's orientation with respect to the world.

B. Batch IMU Processing

During execution, this function processes the data recorded in IMU messages in batches. It iterates through each message in the batch, extracting the timestamp, angular velocity, and linear acceleration to be processed through the model. The function also has a time limit, preventing the function from lagging behind the sensor readings. If the current message's timestamp is too far behind the current IMU's timestamp, then the function moves onto the next batch. The function also keeps track of the current and next IMU IDs to keep track of the IMU state.

C. Process Model

This function calculates the dynamics of the error IMU state, and assists in predicting the new state of the IMU. We first calculate the error of the angular velocity and linear acceleration by subtracting the currently tracked velocity and acceleration by the new IMU message data. We also get a time difference subtracting the previous and current timestamps. The IMU state error can be calculated using Equation 1.

$$\dot{\tilde{X}}_I = F\tilde{X}_I + Gn_I \tag{1}$$

where F and G are defined as as per [1]:

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

$$G = \begin{bmatrix} -\boldsymbol{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \boldsymbol{I}_3 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & -C({}^I_G\hat{\boldsymbol{q}})^T & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \boldsymbol{I}_3 \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \end{bmatrix}$$

After computing F and G, we approximate the matrix exponentially to the third order using Equation 2 of Sun et. Al. [2].

$$\Phi = \boldsymbol{I}_{21} + F * dt + 0.5 * (F * dt)^2 + \frac{1}{6} * (F * dt)^3 \tag{2}$$

After this approximation, we predict the new state of the system before continuing this method (see Section 3D).

D. Predict New State

Using the IMU readings from the IMU message, we can predict the next state of the drone. Using the calculated state error estimated in Section 3C, we can propagate the state using 4th order Runge-Kutta via the MSCKF paper. We first calculate the Ω matrix, containing the time evolution of the IMU state. The Ω matrix is defined as:

$$\Omega = \begin{bmatrix} [-\hat{\omega}] & \omega \\ -\omega^T & 0 \end{bmatrix} \tag{3}$$

Afterwards, we get the current orientation, position, and velocity of the IMU state, we can re-estimate the IMU's angular velocity and linear acceleration using 4th order Runge-Kutta:

$$k_1 = f(t_n, y_n) \tag{4}$$

$$k_2 = f(t_n + 0.5dt, y_n + 0.5k_1dt) \tag{5}$$

$$k_3 = f(t_n + 0.5dt, y_n + 0.5k_2dt) \tag{6}$$

$$k_4 = f(t_n + 0.5dt, y_n + 0.5k_3dt) \tag{7}$$

We update the IMU state by converting the estimated orientation into a quaternion and assigned the estimated orientation, velocity, and acceleration to the current IMU state.

E. State Augmentation

This function calculates the state covariance matrix using Equation 3 in the MSCKF paper. First, it gets the current IMU state, as well as the rotation and translation from the IMU frame to the camera frame. We then build the Jacobian matrix J to estimate the covariance matrix P of the EKF.

$$J = \begin{bmatrix} C({}^I_G\hat{\boldsymbol{q}}) & 0_{3\times9} & 0_{3\times3} & I_3 & 0_{3\times3} \\ -C({}^I_G\hat{\boldsymbol{q}})^T[\hat{\boldsymbol{p}}_{c\times}] & 0_{3\times9} & I_3 & 0_{3\times3} & I_3 \end{bmatrix} \tag{8}$$

$$P_{k+1|k} = \begin{bmatrix} P_{II_{k+1|k}} & \phi_k P_{IC_{k|k}} \\ P^T_{IC_{k|k}}\phi^T_k & P_{CC_{k|k}} \end{bmatrix} \tag{9}$$

F. Add Feature Observations

The camera produces a set of messages based on the features it is tracking within an image. That message gets sent to this function, where for each feature, we append its position to the set with the same feature id. The set contains every frame for which this feature is visible, and is used in the measurement step to calculate the residuals in the measurement step.

G. Measurement Update

The measurement step reduces uncertainty in the system by using the tracked features as a reference for a duration of uncertain movement. Update steps happen once a feature is no longer tracked (falls outside of the image frame), or when there is enough tracked features in the map server for which we can prune them.

We use there parameters for the measurement model: the residual r, the measurement Jacobian H and the state error $\tilde{X}$. They are related via this equations

$$r = H * \tilde{X} + noise \tag{10}$$

Where the noise is a zero-mean gaussian. We reduce the size of the Jacobian to increase speed via QR decomposition, to find the parameters Q_1, Q_2, and T_H.

$$H\tilde{X} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} T_H \\ 0 \end{bmatrix} \tag{11}$$

We then compute the Kalman gain K with the covariance of the system P, covariance of the noise $R_N = Q_1^T R_o Q_1$, and the upper triangle matrix from the previous equation T_H.

$$K = PT_H^T(T_H P T_H^T + R_N)^{-1} \tag{12}$$

Then using the residual and our Kalman gain, we update the current state $\Delta X = Kr_n$. Finally we update our state covariance based on the Kalman gain, and fix the covariance to be symmetric.

$$P_{k+1|k+1} = (I - KH)P_{k|k} + KR_N K^T \tag{13}$$

IV. Results

We tested our methods on the EuRoC dataset, and used the rpg trajectory evaluation package [3] for generating our metric.

The RMSE error from the Machine Hall 01 easy flight are shown below.

TABLE I: RMSE error

Relative Type	Value
Relative Translation	38.43
Relative Rotation	7.68
Relative Rotational Velocity	15.37

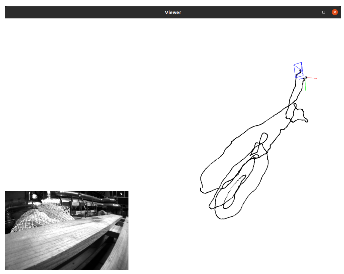

Fig. 1: Final Path graphed in Pangolin window

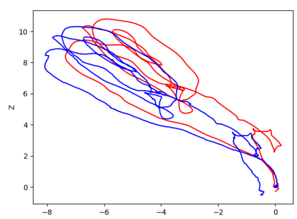

Fig. 2: Estimated vs Ground-Truth xy coordinates over full flight

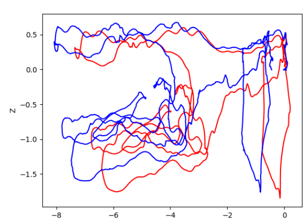

Fig. 3: Estimated vs Ground-Truth xz coordinates over full flight

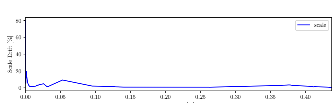

Fig. 4: Scale Error over distance. Note the beginning has extremely high percentages because of division of very small deltas in position.

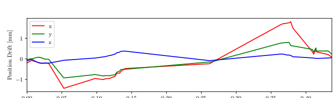

Fig. 5: Translation error over distance

V. Conclusion

In this phase, we implemented a method for performing state estimation via the combination of two ordinary sensors, a camera and an IMU. Although the IMU is great for short term - high frequency values, its propensity to drift makes it unstable long term. And while a camera provides robust features usable for long-term tracking, it struggles with sudden movements where features cannot be easily tracked. By fusing thw two, you attain stable positional state estimation in real time, and can correct for the biases generated during a flight.

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, 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, 2018.

[3] Zichao Zhang and Davide Scaramuzza. A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry. In *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, 2018.