RBE549 Project 4 - Deep and Un-Deep Visual Inertial Odometry

Yaşar İdikut
yidikut@wpi.edu

Harshal Bhat
hbhat@wpi.edu

I. Introduction

In this project, we complete seven functions in the given starter code that implements the work in [2]. We only test the code on the "Machine Hall 01" data from the EuRoC MAV Dataset [4]. In this episode, drone starts stationary, then it flies around before landing back to the same spot.

II. Initialize Gravity and Bias

Since, the start of the data includes a brief period where the drone is stationary, we take the first 200 readings, average them, and use this to offset the gyroscope bias and gravity vector for the accelerometer sensor.

Gyroscope bias is set to the average values of the gyroscope readings as $[b_{avg-gx}, b_{avg-gy}, b_{avg-gz}]$. Accelerometer gravity vector's z axis is set to the norm of the average accelerometer readings as $[0, 0, -gravity_{avg-norm}]$. Finally, the initial orientation of the IMU is initialized to be the vector between average accelerometer readings and the gravity vector.

III. Batch IMU Processing

In this function, we read the the IMU readings(gyroscope and accelerometer) from the buffer, and send do processing on them using the `process_model` function.

IV. Process Model

In this function, we call the `predict_new_state` function to estimate the next step of the IMU state (this can be thought of as step 1 of a Kalman Filter), then update the error covariance for the next step (this can be thought of as step 2 of a Kalman Filter).

Below is the dynamics model of the IMU followed.

$$ {}^{I}_{G}\dot{\bar{q}}(t) = \frac{1}{2}\Omega(\omega(t)) {}^{I}_{G}\bar{q}(t) \tag{1} $$

$$ \dot{b}_g(t) = n_{wg}(t) \tag{2} $$

$$ {}^{G}\dot{v}_I(t) = {}^{G}a(t) \tag{3} $$

$$ \dot{b}_a(t) = n_{wa}(t) \tag{4} $$

$$ {}^{G}\dot{p}_I(t) = {}^{G}v_I(t) \tag{5} $$

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

$$ [\omega\times] = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix} $$

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

$$ a_m = {}^{I}R_G \left(a - {}^{G}g + 2\left[\omega_{G\times}\right] {}^{G}v_I + 2\left[\omega_{G\times}\right]^2 {}^{G}p_I \right) \tag{7} $$

$$ {}^{I}_{G}\dot{\hat{q}}(t) = \frac{1}{2}\Omega(\hat{\omega}_G)\hat{q}, \tag{8} $$

$$ \dot{\hat{b}}_g = 0_{3\times1}, \tag{9} $$

$$ {}^{G}\dot{\hat{v}}_I = C^T_{\hat{q}}\hat{a} - 2[\hat{\omega}_{G\times}]^{G}\hat{v}_I + [\hat{\omega}_{G\times}]^{2G}\hat{p}_I + {}^{G}\hat{g}, \tag{10} $$

$$ \dot{\hat{b}}_a = 0_{3\times1}, \tag{11} $$

$$ {}^{G}\dot{\hat{p}}_I = {}^{G}\dot{\hat{v}}_I \tag{12} $$

Error dynamics of the IMU model is as follows. This model is used to update the state covariance matrix. In implementation, we use the 3^{rd} order approximation. $\tilde{\mathbf{x}}_I$ is the IMU state and $\mathbf{n}_I$ is the Gaussian noise.

$$ \dot{\tilde{\mathbf{x}}}_I = \mathbf{F}\tilde{\mathbf{x}}_I + \mathbf{G}\mathbf{n}_I \tag{13} $$

$$ F = \begin{bmatrix} [\hat{\omega}_\times] & -I_3 & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ -C({}^{I}_{G}\hat{q})[\hat{a}_\times] & 0_{3\times3} & -C({}^{I}_{G}\hat{q})^T & 0_{3\times3} & -C(\hat{q})^T \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & I_3 & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \end{bmatrix} $$

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

Calculate discrete time noise covariance matrix, $\mathbf{\Phi}_k$, and continuous time noise covariance matrix, $\mathbf{Q}_k$.

$$ \mathbf{\Phi}_k = \mathbf{\Phi}\left(t_{k+1}, t_k\right) = \exp\left(\int_{t_k}^{t_{k+1}} \mathbf{F}(\tau) d\tau \right) \tag{14} $$

$$ \mathbf{Q}_k = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}\left(t_{k+1}, \tau\right) \mathbf{G}\mathbf{Q}\mathbf{G}\mathbf{\Phi}\left(t_{k+1}, \tau\right)^\top d\tau \tag{15} $$

Propagated covariance of the IMU state is updated as follows.

$$\mathbf{P}_{II_{k+1|k}} = \mathbf{\Phi}_k \mathbf{P}_{II_{k|k}} \mathbf{\Phi}_k^\top + \mathbf{Q}_k \tag{16}$$

V. Predict New State

This function is called by the previous `process_model` to estimate the next IMU state given the bias-adjusted angular velocity and linear acceleration measurements. This function approximates the next state by integrating the model for a time step dt using 4^{th} order Runge-Kutta integration, however, in the original MSCKF paper, authors use the 5^{th} order Runge-Kutta integration [1].

VI. State Augmentation

In this function, we augment the state covariance matrix (calculated in `process_model`) with incoming camera states.

$$\mathbf{P}_{k|k} = \begin{pmatrix} \mathbf{P}_{II_{k|k}} & \mathbf{P}_{IC_{k|k}} \\ \mathbf{P}_{IC_{k|k}}^\top & \mathbf{P}_{CC_{k|k}} \end{pmatrix} \tag{17}$$

$${}^C_G\hat{\mathbf{q}} = {}^C_I\hat{\mathbf{q}} \otimes {}^I_G\hat{\mathbf{q}}, \quad {}^G\hat{\mathbf{p}}_C = {}^G\hat{\mathbf{p}}_C + C\left({}^I_G\hat{\mathbf{q}}\right)^\top {}^I\hat{\mathbf{p}}_C \tag{18}$$

$$\mathbf{J}_I = \begin{pmatrix} C\left({}^I_G\hat{\mathbf{q}}\right) & \mathbf{0}_{3\times 9} & \mathbf{0}_{3\times 3} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} \\ -C\left({}^I_G\hat{\mathbf{q}}\right)^\top \lfloor {}^I\hat{\mathbf{p}}_{C\times} \rfloor & \mathbf{0}_{3\times 9} & \mathbf{I}_3 & \mathbf{0}_{3\times 3} & \mathbf{I}_3 \end{pmatrix} \tag{19}$$

$$\mathbf{P}_{k|k} = \begin{pmatrix} \mathbf{I}_{21+6N} \\ \mathbf{J} \end{pmatrix} \mathbf{P}_{k|k} \begin{pmatrix} \mathbf{I}_{21+6N} \\ \mathbf{J} \end{pmatrix}^\top \tag{20}$$

VII. Add Feature Observations

In this function, we iterate over the tracked and new features. Each feature has its own ID and is associated to a state ID. If a feature existed for earlier frames, they are used to enhance tracking. If not, it is added to the map to be looked up in later frames.

VIII. Measurement Update

This function takes measurement matrix, H, and residual matrix, r. It computes the update part of the Kalman Filter (calculating the gain, updating the state estimation, and updating the state covariance matrix).

Kalman Gain is computed as follows (Kalman filter step 3).

$$\mathbf{K} = \mathbf{P}\mathbf{T}_H^T \left(\mathbf{T}_H \mathbf{P}\mathbf{T}_H^T + \mathbf{R}_n\right)^{-1} \tag{21}$$

Correction to the state estimation is computed as follows (Kalman filter step 4).

$$\Delta\mathbf{X} = \mathbf{K}\mathbf{r}_n \tag{22}$$

State covariance matrix is updated as follows (Kalman filter step 5).

$$\mathbf{P}_{k+1|k+1} = \left(\mathbf{I}_\xi - \mathbf{K}\mathbf{T}_H\right)\mathbf{P}_{k+1|k}\left(\mathbf{I}_\xi - \mathbf{K}\mathbf{T}_H\right)^T + \mathbf{K}\mathbf{R}_n\mathbf{K}^T \tag{23}$$

IX. Results

The calculated trajectory length is 80.62621184 meters. Following figures and tables show the trajectory following performance. All of these results are generated using `rpg_trajectory_evaluation` tool as mentioned in [3].

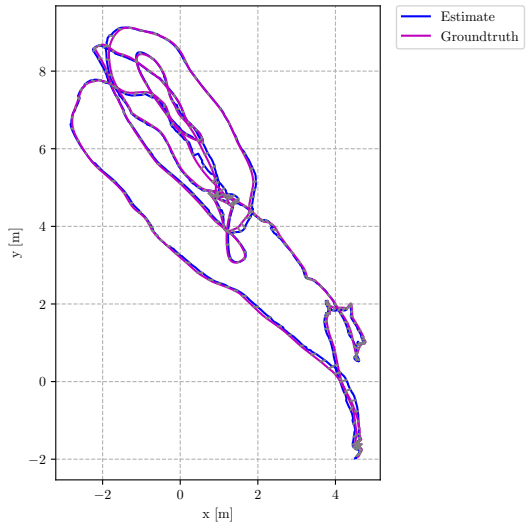

Fig. 1. MSCKF VIO position estimate vs. ground truth in the XY plane (top/birds eye view)

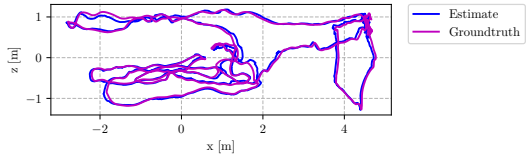

Fig. 2. MSCKF VIO position estimate vs. ground truth in the ZX plane (side view)

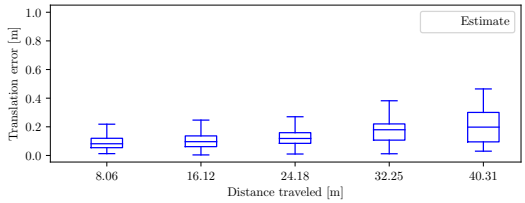

Fig. 3. Position tracking error (in meters) over distance travelled (in meters)

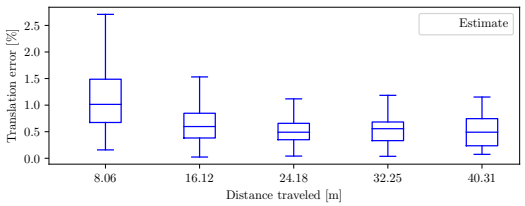

Fig. 4. Position tracking error (in percentage) over distance travelled (in meters)

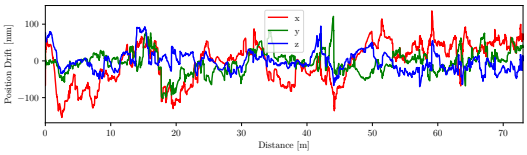

Fig. 5. Position tracking drift error (in millimeters) over distance travelled (in meters)

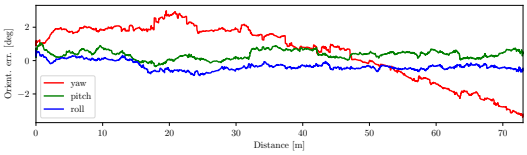

Fig. 6. Rotational tracking error (in degrees) over distance travelled (in meters)

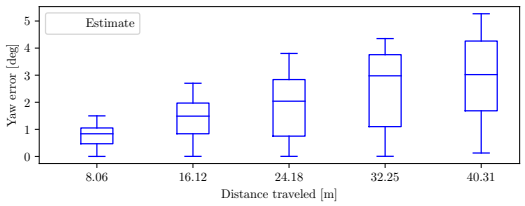

Fig. 7. Yaw tracking error (in degrees) over distance travelled (in meters)

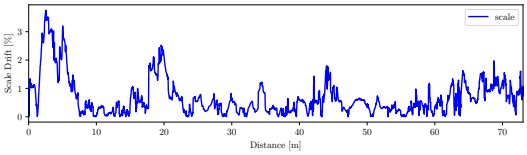

Fig. 8. Scale drift error (in percentage) over distance travelled (in meters)

Absolute Error Statistics		
Statistic	Translational (Meters)	Rotational (Degrees)
Maximum	0.16513240m	3.48515797°
Mean	0.06423876m	1.74583618°
Median	0.06085964m	1.83840256°
Minimum	0.00592799m	0.48334253°
RMSE	0.07075978m	1.86783709°
Standard Deviation	0.02967033m	0.66398147°

X. Discussion and Conclusion

Our results show close tracking of pose (both position and orientation) throughout the episode. In saving the results to text files for generating graphs, we used the `Rotation` class from `scipy.spatial.transform` package for converting from rotation matrix to quaternions. This gave use the rotation on the correct format as opposed to `to_rotation` function from `utils.py`.

One possible research area for improving upon the classical VIO methods would be to combine RGB and event cameras together to obtain jello-free frames, thereby improving tracking of features across frames in fast moving bases.

References

[1] A. I. Mourikis and S. I. Roumeliotis, "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation," Proceedings 2007 IEEE International Conference on Robotics and Automation, Rome, Italy, 2007, pp. 3565-3572, doi: 10.1109/ROBOT.2007.364024.

[2] Sun, Ke Mohta, Kartik Pfrommer, Bernd Watterson, Michael Liu, Sikang Mulgaonkar, Yash Taylor, Camillo Kumar, Vijay. (2017). Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight. IEEE Robotics and Automation Letters. PP. 10.1109/LRA.2018.2793349.

[3] Zichao Zhang, Davide Scaramuzza: A Tutorial on Quantitative Trajectory Evaluation for Visual(-Inertial) Odometry, IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS), 2018.

[4] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. Achtelik and R. Siegwart, The EuRoC micro aerial vehicle datasets, International Journal of Robotic Research, DOI: 10.1177/0278364915620033, early 2016.