Error-state Kalman Filter in Pose Estimation

Teng Ma

***Abstract*—This report is my study note on Error-state Kalman Filter (ESKF). I am trying to make three points clear: 1. What is ESKF? 2. Why do we choose ESKF? 3. How to use ESKF for pose estimation in systems with IMU?**

***Index Terms*—Error-state KF, pose estimation, IMU**

I. Introduction

When I tried to break down the whole SLAM problem into small parts, I saw error state a lot in filtering-based VIO/LIO system [2] [3] . I am curious about this "new term" ESKF, so I did some study on it and tried to implement ESKF in LiDAR-inertial odometry. is Multiplicative Extended Kalman Filter, MEKF

II. Theory Part

A. What is Error-state Kalman Filter?

Error state is the difference between True state and Nominal state. To be specific, high-frequency IMU data u_m is integrated into a nominal-state x. This nominal state does not take into account the noise terms w and other possible model imperfections. As a consequence, it will accumulate errors. These errors are collected in the error-state δx and estimated with the Error-State Kalman Filter, this time incorporating all the noise and perturbations. In parallel with integration of the nominal state, the ESKF predicts a Gaussian estimate of the error-state. It only predicts, because by now no other measurement is available to correct these estimates. The filter correction is performed at the arrival of information other than IMU (e.g. GPS, vision, etc.), which is able to render the errors observable and which happens generally at a much lower rate than the integration phase. This correction provides a posterior Gaussian estimate of the error-state. After this, the error-state's mean is injected into the nominal-state, then reset to zero.
The idea is to consider the nominal state as large-signal (integrable in non-linear fashion) and the error-state as small signal (thus linearly integrable and suitable for linear-Gaussian filtering) [1].

B. Why ESKF?

In most modern IMU systems, people often use Error state Kalman filter (ESKF) instead of the original state Kalman filter. The reasons [1] are as follows:

- The orientation error-state is minimal (i.e., it has the same number of parameters as degrees of freedom), avoiding issues related to over-parametrization (or redundancy) and the consequent risk of singularity of the involved covariances matrices, resulting typically from enforcing constraints.
- The error-state system is always operating close to the origin, and therefore far from possible parameter singularities, gimbal lock issues, or the like, providing a guarantee that the linearization validity holds at all times.
- The error-state is always small, meaning that all second-order products are negligible. This makes the computation of Jacobians very easy and fast. Some Jacobians may even be constant or equal to available state magnitudes.
- The error dynamics are slow because all the large-signal dynamics have been integrated in the nominal-state. This means that we can apply KF corrections (which are the only means to observe the errors) at a lower rate than the predictions.

C. Equation of State

First of all, here is the table contains all the variables used below from [1]:

Magnitude	True	Nominal	Error	Composition	Measured	Noise
Full state ([1])	$\mathbf{x}_t$	$\mathbf{x}$	$\delta\mathbf{x}$	$\mathbf{x}_t = \mathbf{x} \oplus \delta\mathbf{x}$		
Position	$\mathbf{p}_t$	$\mathbf{p}$	$\delta\mathbf{p}$	$\mathbf{p}_t = \mathbf{p} + \delta\mathbf{p}$		
Velocity	$\mathbf{v}_t$	$\mathbf{v}$	$\delta\mathbf{v}$	$\mathbf{v}_t = \mathbf{v} + \delta\mathbf{v}$		
Quaternion ([2,3])	$\mathbf{q}_t$	$\mathbf{q}$	$\delta\mathbf{q}$	$\mathbf{q}_t = \mathbf{q} \otimes \delta\mathbf{q}$		
Rotation matrix ([2,3])	$\mathbf{R}_t$	$\mathbf{R}$	$\delta\mathbf{R}$	$\mathbf{R}_t = \mathbf{R}\,\delta\mathbf{R}$		
Angles vector ([4])			$\delta\boldsymbol{\theta}$	$\delta\mathbf{q} = e^{\delta\boldsymbol{\theta}/2}$ $\delta\mathbf{R} = e^{[\delta\boldsymbol{\theta}]_\times}$		
Accelerometer bias	$\mathbf{a}_{bt}$	$\mathbf{a}_b$	$\delta\mathbf{a}_b$	$\mathbf{a}_{bt} = \mathbf{a}_b + \delta\mathbf{a}_b$		$\mathbf{a}_w$
Gyrometer bias	$\boldsymbol{\omega}_{bt}$	$\boldsymbol{\omega}_b$	$\delta\boldsymbol{\omega}_b$	$\boldsymbol{\omega}_{bt} = \boldsymbol{\omega}_b + \delta\boldsymbol{\omega}_b$		$\boldsymbol{\omega}_w$
Gravity vector	$\mathbf{g}_t$	$\mathbf{g}$	$\delta\mathbf{g}$	$\mathbf{g}_t = \mathbf{g} + \delta\mathbf{g}$		
Acceleration	$\mathbf{a}_t$				$\mathbf{a}_m$	$\mathbf{a}_n$
Angular rate	$\boldsymbol{\omega}_t$				$\boldsymbol{\omega}_m$	$\boldsymbol{\omega}_n$

Fig. 1. All variables in the error-state Kalman filter.

True state x_t in ESKF: $x_t = [p_t, v_t, R_t, a_{bt}, \omega_{bt}, g_t]^T$, x_t change over time and the subscript t denotes true state. In continuous time, we record the IMU reading as a_m, ω_m, then we can write the relationship between the derivative of the state variable with respect to the observed measurement:

$$\dot{p_t} = v_t$$
$$\dot{v_t} = R_t(a_m - a_{bt} - a_n) + g_t$$
$$\dot{R_t} = R_t(\omega_m - \omega_{bt} - \omega_n)^\wedge$$
$$\dot{a_{bt}} = a_\omega$$
$$\dot{\omega_{bt}} = \omega_\omega$$
$$\dot{g_t} = 0$$

All the symbols remain the same as what [1] use, more details about the meaning of variables can be found in [1]. The only difference is that in [1] the author use Quaternion to represent rotation, I use SO3 rotation matrix which is familiar

to me, so during the derivation I could make some mistakes.
Nominal state x kinematics corresponds to the modeled system without noise or perturbations,
$\dot{p} = v$
$\dot{v} = R(a_m - a_b) + g$
$\dot{R} = R(\omega_m - \omega_b)^\wedge$
$\dot{a_b} = 0$
$\dot{\omega_b} = 0$
$\dot{g_t} = 0$
Then we have error state δx kinematics:
$\dot{\delta p} = \delta v$
$\dot{\delta v} = -R(a_m - a_b)^\wedge \delta\theta - R\delta a_b + \delta g - Ra_n$
$\dot{\delta\theta} = -(\omega_m - \omega_b)^\wedge \delta\theta - \delta\omega_b - \omega_n$
$\dot{\delta a_b} = a_\omega$
$\dot{\delta\omega_b} = \omega_\omega$
$\dot{\delta g} = 0$
The discrete form of error state kinematics:
$\delta p = \delta p + \delta v \Delta t$
$\delta v = \delta v + (-R(a_m - a_b)^\wedge \delta\theta - R\delta a_b + \delta g)\Delta t + v_i$
$\delta\theta = -R^T((\omega_m - \omega_b)\Delta t)\delta\theta - \delta\omega_b \Delta t + \theta_i$
$\delta a_b = \delta a_b + a_i$
$\delta\omega_b = \delta\omega_b + \omega_i$
$\delta g = \delta g$
Here, $v_i, \theta_i, a_i, \omega_i$ are the random impulses applied to the velocity, orientation and bias estimates, modeled by white Gaussian processes. Their mean is zero, and their covariances matrices are obtained by integrating the covariances of $a_n, \omega_n, a_\omega, \omega_\omega$ over the step time Δt.

D. Prediction and updating equations

Now we can get write the motion equation in discrete time domain,
$\delta x = f(\delta x) + \omega, \omega \sim N(0, Q)$
ω is noise, which is composed by $v_i, \theta_i, a_i, \omega_i$ mentioned above, so Q matrix should be:
$Q = diag(cov(v_i), cov(\theta_i), cov(a_i), cov(\omega_i))$
The prediction equations are written:
$\delta x_{pred} = F\delta x$
$P_{pred} = FPF^T + Q$
where F is the Jaccobian of the error state function f, the expression is detailed below:

$$\boldsymbol{F} = \begin{bmatrix} \boldsymbol{I} & \boldsymbol{I}\Delta t & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} & -\boldsymbol{R}(\tilde{\boldsymbol{a}} - \boldsymbol{b}_a)^\wedge \Delta t & -\boldsymbol{R}\Delta t & \boldsymbol{0} & \boldsymbol{I}\Delta t \\ \boldsymbol{0} & \boldsymbol{0} & \operatorname{Exp}(-(\tilde{\boldsymbol{\omega}} - \boldsymbol{b}_g)\Delta t) & \boldsymbol{0} & -\boldsymbol{I}\Delta t & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{I} \end{bmatrix}$$

Fig. 2. Jaccobian matrix

Suppose an abstract sensor can produce observations of state variables, and its observation equation is written as:
$z = h(x) + v, v \sim N(0, V)$,
where h is a general nonlinear function of the system state (the true state), and v is measurement noise, a white Gaussian noise with covariance V. The updating equations are:
$K = P_{pred}H^T(HP_{pred}H^T + V)^{-1}$
$\delta x = K(z - h(x_t))$
$P = (I - KH)P_{pred}$
Where K is Kalman gain, P_{pred} is prediction covariance matrix, P is covariance matrix after updating and H is defined as the Jacobian matrix of measurement equation of error state,
$H = \frac{\partial h}{\partial \delta x}$
According to chain rule,
$H = \frac{\partial h}{\partial x}\frac{\partial x}{\partial \delta x}$
First part $\frac{\partial h}{\partial x}$ can easily obtained by linearizing the measurement equation, the second part $\frac{\partial x}{\partial \delta x}$ is the Jacobian of the true state with respect to the error state, which is the combination of 3x3 identity matrix (for example, $\frac{\partial(p+\delta p)}{\partial \delta p} = I_3$), expect for the rotation part, in quaternion form it is $\frac{\partial(q \otimes \delta q)}{\partial \delta\theta}$, the deduction is in [1], here in the form of rotation matrix in $SO3$, it is $\frac{\partial log(RExp(\delta\theta))}{\partial \delta\theta}$, where $Exp(\delta\theta)$ is the Lie algebra of rotation δR, H can be obtained according to Baker–Campbell–Hausdorff (BCH) formula.
After updating, we have
$x_{k+1} = x_k \oplus \delta x_k$
$\delta x_k = 0$
Note: I only wrote the equations that define angular error in the local reference, the global situation is shown in [1].

III. Experiments

A. Description

I generated the trajectory data from "2011-09-26-drive-0001-sync/oxts", KITTI, which contains GPS and IMU data, and found the LiDAR odometry and ground truth on github, which should save my time if things go as planned. What's not so good is that ground truth only contains position data. All the codes are implemented in pyhton 3.8.5, ubuntu 20.04. The first experiment is fusing GPS data (Latitude, longitude, altitude) and IMU data (acceleration and angular velocity) to estimate the trajectory. I use toolbox that KITTI dataset offers to get transformation matrix from GPS/IMU data, and logged the position data calculated from GPS data (according to the source code in MATLAB) in a txt file "pos.txt", the same form as ground truth in file "gt.txt". The results are in the figure 3, 4.

The second experiment is fusing LiDAR data with IMU data, which didn't go well. The LiDAR data didn't align with ground truth, and bad data yields bad result. Then I had 2 options: 1. Extracting LiDAR odometry from raw data from beginning, which I tried to get the transformation matrix between 2 neighbour frames of point cloud data, I have to implement some feature descriptor, otherwise no matter ICP or NDT cannot generate good results; 2. Finding what is wrong in the given data. After I checking the position of sensors on the car that KITTI employed, I found that the mismatch could lie in the coordinate system of LiDAR data, I am trying to reach out the author by issuing on github, but for now let's see how is the data after transformation. The parameters remain the same as experiment 1, I only changed the covarience matrix of measurement. The results is in figure 5, 6 and 7.

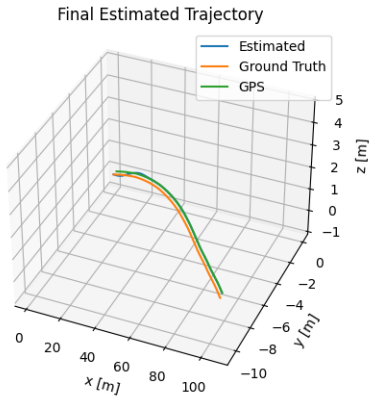

Fig. 3. Trajectories of estimation, GPS and ground truth

```
MSE x 2.399633404443814
MSE y 3.461294130366321
MSE z 0.09374177330455512
```

Fig. 4. Mean square error

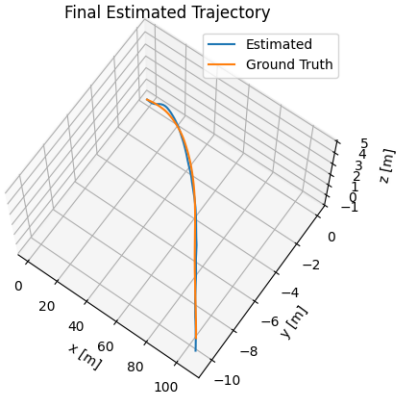

Fig. 5. Trajectories top view

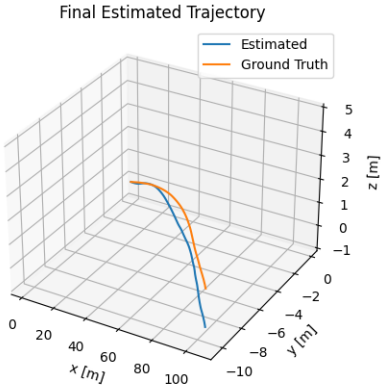

Fig. 6. Trajectories front view

```
MSE x 6.62261580376972
MSE y 1.660136617199816
MSE z 12.09818543973924
```

Fig. 7. Mean square error

B. results analysis

The first experiment worked well, since the data is aligned the result is good. The second one is not as good as I expected, after data correction the LiDAR odometry is still not close to the ground truth, but the trend is the aligned (not going the opposite direction as the original version). The error mainly lies in z axis direction, the result tend to diverge in z axis as time goes. The reason I thought is: 1, garbage in, garbage out. The acceleration given by IMU and position given by LiDAR odometry are both in the wrong direction. 2. the equations implemented in code is not totally right. The acceleration data integrates gravity so the formula changes compared to the book [1]. As far as I can tell, removing the gravity is not a big deal. Moreover, I found that the reason why we estimate gravity in ESKF is for IMU initialization. If we do not write the gravity variable in the equation of state, then the IMU orientation at the initial moment must be the in the direction of $R(0)$. At this time, the posture of the IMU is described relative to the initial horizontal plane. If the gravity is written, the initial attitude of the IMU can be set as the identity matrix, and the direction of gravity can be used as a measurement of the current attitude of the IMU compared to the horizontal plane. Both methods are feasible, but expressing the direction of gravity alone will make the initial posture expression easier, and it can also increase some linearity [4].

IV. Summary

ESKF is a Kalman Filter(or EKF) that suits IMU, I spent one-third of my time studying KITTI dataset, one-third for theory deduction and one third implementation. I think I should do more about datasets searching, every time that I search the data already packed well online and thought it could save me time, turns out I will do more research on it to use the data for my experiment. What's important is that I implemented a filtering algorithm in python on my own, and I feel good after finishing it.

References

[1] Sola J. Quaternion kinematics for the error-state Kalman filter[J]. arXiv preprint arXiv:1711.02508, 2017.

[2] Xu W, Zhang F. Fast-lio: A fast, robust lidar-inertial odometry package by tightly-coupled iterated kalman filter[J]. IEEE Robotics and Automation Letters, 2021, 6(2): 3317-3324.

[3] Xu W, Cai Y, He D, et al. Fast-lio2: Fast direct lidar-inertial odometry[J]. arXiv preprint arXiv:2107.06829, 2021.

[4] Lupton T, Sukkarieh S. Efficient integration of inertial observations into visual SLAM without initialization[C]//2009 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2009: 1547-1552.