Left-Invariant EKF for Robot Localization: A Mobile Robotics Project

Sean Boylan, Cheng Jiang, Glen Haggin, Juhan Wang, Xuran Zhao
University of Michigan
{sboylan, chengjia, ghaggin, juhanw, xuranzh}@umich.edu

Github: https://github.com/ghaggin/invariant-ekf
YouTube: https://www.youtube.com/watch?v=SUFUShReSNg

***Abstract*—In this project we derive and implement a Left-Invariant Extended Kalman Filter (LI-EKF) for quadrotor localization in MATLAB and C++. We applied our filter to simulated data that we generated and also to the Zurich Urban Micro Aerial Vehicle Dataset Dataset [1]. The filter takes in IMU data for prediction and GPS data for correction. A traditional EKF is implemented as a baseline for comparison. For the simulated data, the tracking of the left-invariant EKF is accurate and has outperformed that of the tradition EKF in both orientation and position. It also achieved decent tracking result with Zurich Urban Dataset, though the estimation is highly dependent on GPS.**

I. Introduction

The process model of a 6 degree of freedom vehicle is nonlinear, and there is no general method to design a globally stable observer. Historically, methods have relied on linearizing the process model by using first order approximations (EKF) or performing an unscented transform (UKF) [2]. These methods are practical for applications that operate in a small region where linear approximations are valid, but for aerial vehicles undergoing large rotations there is a limited envelope for which they are effective. Therefore, in order to provide an accurate state estimation over the full flight envelope, we seek an observe that is capable of handling such maneuvers without relying on the traditional techniques.

For nonlinear process that are group affine, it is possible to design an invariant observer with guaranteed global convergence [3]. In this paper, we present an Invariant EKF that uses an inertial measurement unit (IMU) and a gyroscope for prediction and GPS for an update. This filter measures the full state of the rigid body, including the position, velocity, and orientation. The filter is tested using simulated data we generated and the Zurich Urban Dataset.

II. bias-corrected Left-Invariant EKF

A. State Representation in $SE_2(3)$

Our goal is to track the robots orientation, velocity, and position over time. We define this state and the estimation error in $SE_2(3)$ because the error dynamics have the properties of logarithm-linearity and autonomy. Thus, the state variable $\mathbf{X}_t$, containing the orientation $\mathbf{R}(t)$, velocity $\mathbf{v}(t)$ in body frame, and the position $\mathbf{p}(t)$ in world frame, is defined as:

$$\mathbf{X}_t = \begin{bmatrix} \mathbf{R}(t) & \mathbf{v}(t) & \mathbf{p}(t) \\ \mathbf{0}_{1\times 3} & 1 & 0 \\ \mathbf{0}_{1\times 3} & 0 & 1 \end{bmatrix}$$

B. IMU and Gyro System Dynamics

Firstly, without taking IMU bias into account, we only add a white Gaussian noise:

$$\tilde{\boldsymbol{\omega}}_t = \boldsymbol{\omega}_t + \boldsymbol{w}_t^g \qquad \boldsymbol{w}_t^g \sim \mathcal{GP}(\mathbf{0}_{3,1}, \Sigma^g) \tag{1}$$

$$\tilde{\boldsymbol{a}}_t = \boldsymbol{a}_t + \boldsymbol{w}_t^a \qquad \boldsymbol{w}_t^a \sim \mathcal{GP}(\mathbf{0}_{3,1}, \Sigma^a) \tag{2}$$

where $\tilde{\boldsymbol{\omega}}_t$ and $\tilde{\boldsymbol{a}}_t$ are the measured angular velocity from the gyroscope and measured linear acceleration from the accelerometer.

Using IMU measurements, the continuous-time system dynamics can be represented as:

$$\frac{d}{dt}\boldsymbol{R}_t = \boldsymbol{R}_t[\tilde{\boldsymbol{\omega}}_t - \boldsymbol{w}_t^g]_\times \tag{3}$$

$$\frac{d}{dt}\boldsymbol{v}_t = \boldsymbol{R}_t(\tilde{\boldsymbol{a}}_t - \boldsymbol{w}_t^a) + \boldsymbol{g} \tag{4}$$

$$\frac{d}{dt}\boldsymbol{p}_t = \boldsymbol{v}_t \tag{5}$$

Collecting terms in matrix form and writing the process model with respect to current state X_t yields:

$$\frac{d}{dt}\boldsymbol{X}_t = \begin{bmatrix} \boldsymbol{R}_t[\tilde{\boldsymbol{\omega}}_t]_\times & \boldsymbol{R}_t\tilde{\boldsymbol{a}}_t + g & \boldsymbol{v}_t \\ \mathbf{0}_{1\times 3} & 0 & 0 \\ \mathbf{0}_{1\times 3} & 0 & 0 \end{bmatrix} - \begin{bmatrix} \boldsymbol{R}_t & \boldsymbol{v}_t & \boldsymbol{p}_t \\ \mathbf{0}_{1\times 3} & 1 & 0 \\ \mathbf{0}_{1\times 3} & 0 & 1 \end{bmatrix} \begin{bmatrix} [\boldsymbol{w}_t^g]_\times & \boldsymbol{w}_t^a & \mathbf{0}_{3,1} \\ \mathbf{0}_{1\times 3} & 0 & 0 \\ \mathbf{0}_{1\times 3} & 0 & 0 \end{bmatrix} = f_{u_t}(\boldsymbol{X}_t) - \boldsymbol{X}_t\boldsymbol{w}_t^\wedge \tag{6}$$

where $\boldsymbol{w}_t \triangleq [\boldsymbol{w}_t^g, \boldsymbol{w}_t^a, \mathbf{0}_{1x3}]^T$.

For the hybrid robot estimation problem with continuous-time motion model and discrete-time observation model, we discretized the above continuous-time system dynamics Eqns. (3, 4, 5). With the assumption that the angular velocity and the linear acceleration remain unchanged over a short

period Δt, we can write the discrete process model from step k to $k+1$ as the following:

$$\boldsymbol{R}_{k+1} = \boldsymbol{R}_k \boldsymbol{\Gamma}_0(\boldsymbol{\omega}_k \Delta t) \tag{7}$$

$$\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \boldsymbol{R}_k \boldsymbol{\Gamma}_1(\boldsymbol{\omega}_k \Delta t)\boldsymbol{a}_k \Delta t + \boldsymbol{g}\Delta t \tag{8}$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \boldsymbol{v}_k \Delta t + \boldsymbol{R}_k \boldsymbol{\Gamma}_2(\boldsymbol{\omega}_k \Delta t)\boldsymbol{a}_k \Delta t^2 + \frac{1}{2}\boldsymbol{g}\Delta t^2 \tag{9}$$

where $\boldsymbol{\omega}_k = \tilde{\boldsymbol{\omega}}_k - \boldsymbol{w}_k^g$ and $\boldsymbol{a}_k = \tilde{\boldsymbol{a}}_k - \boldsymbol{w}_k^a$. $\boldsymbol{w}_k^g$ and $\boldsymbol{w}_k^a$ are the white noise of motion model for the angular velocity and linear acceleration respectively. The derivation of Γ_i functions are explained in detail in Appendix A and used here to get the state transition matrices:

$$\boldsymbol{\Gamma}_0(\phi) = \boldsymbol{I}_3 + \frac{\sin(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|}[\boldsymbol{\phi}]_\times + \frac{1-\cos(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^2}[\boldsymbol{\phi}]_\times^2 \tag{10}$$

$$\boldsymbol{\Gamma}_1(\boldsymbol{\phi}) = \boldsymbol{I}_3 + \frac{1-\cos(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^2}[\boldsymbol{\phi}]_\times + \frac{\|\boldsymbol{\phi}\| - \sin(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^3}[\phi]_\times^2 \tag{11}$$

$$\boldsymbol{\Gamma}_2(\boldsymbol{\phi}) = \frac{1}{2}\boldsymbol{I}_3 + \frac{\|\boldsymbol{\phi}\| - \sin(\|\boldsymbol{\phi}\|)}{\|\boldsymbol{\phi}\|^3}[\boldsymbol{\phi}]_\times + \frac{\|\boldsymbol{\phi}\|^2 + 2\cos(\|\boldsymbol{\phi}\|) - 2}{2\|\boldsymbol{\phi}\|^4}[\boldsymbol{\phi}]_\times^2 \tag{12}$$

Where $\boldsymbol{\Gamma}_0(\boldsymbol{\phi})$ is the exponential map of SO(3), and $\boldsymbol{\Gamma}_1(\boldsymbol{\phi})$ is the left Jacobian of SO(3).

C. Error Dynamics

With measurements coming from GPS in global frame, We choose the left invariant error η_t^l:

$$\boldsymbol{\eta}_t^l = \mathbf{X}_t^{-1}\bar{\mathbf{X}}_t = (\mathbf{L}\bar{\mathbf{X}}_t)^{-1}(\mathbf{L}\mathbf{X}_t)$$

With Lie group action, we get corresponding error in Lie algebra:

$$\boldsymbol{\eta}_t^l = \exp([\boldsymbol{\xi}_t^l]_\times) \tag{13}$$

where $\exp([\boldsymbol{\xi}_t^l]_\times) \approx I + [\boldsymbol{\xi}_t^l]_\times$, and the $\approx$ should be $=$ from the logarithm-linear property of error in Lie group.

The skew operator $[\cdot]_\times : \mathbb{R}^9 \to \mathfrak{se}_2(3)$ is a mapping from vector basis to Lie algebra matrix basis. It is defined as:

$$[\boldsymbol{\xi}]_\times = \begin{bmatrix} [\boldsymbol{\xi}^R]_\times & \boldsymbol{\xi}^v & \boldsymbol{\xi}^p \\ \mathbf{0}_{1\times 3} & 0 & 0 \\ \mathbf{0}_{1\times 3} & 0 & 0 \end{bmatrix} \tag{14}$$

Where the $[\boldsymbol{a}]_\times$ operator for $\boldsymbol{\xi}^R$ denotes the transformation $\mathbb{R}^3 \to so(3)$:

$$\boldsymbol{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \implies [\boldsymbol{a}]_\times = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{15}$$

In order to find the error dynamics equation in Lie algebra as the following form with $\boldsymbol{w}_t$ as the motion noise and its covariance defined as $\boldsymbol{Q}_t$:

$$\frac{d}{dt}[\boldsymbol{\xi}_t^l]_\times = A_t^l[\boldsymbol{\xi}_t^l]_\times + \boldsymbol{w_t} \tag{16}$$

we substitute the above elegant linear equation into system dynamics equations and obtain the error Jacobian matrix $\boldsymbol{A}_t$:

$$\boldsymbol{A}_t^l = \begin{bmatrix} -[\hat{\boldsymbol{\omega}}_t]_\times & \mathbf{0} & \mathbf{0} \\ -[\hat{\boldsymbol{a}}_t]_\times & -[\hat{\boldsymbol{\omega}}_t]_\times & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_{3\times 3} & -[\hat{\boldsymbol{\omega}}_t]_\times \end{bmatrix} \tag{17}$$

Therefore the covariance of the left invariant error could be computed through Riccati equation:

$$\frac{d}{dt}\boldsymbol{P}_t = \boldsymbol{A}_t\boldsymbol{P}_t + \boldsymbol{P}_t\boldsymbol{A}_t^T + \boldsymbol{Q}_t \tag{18}$$

D. Left-invariant observation model

In order to get a invariant observation model in Lie group, we rearrange every measurement from GPS as:

$$\mathbf{Y}_k = \mathbf{X}_k\boldsymbol{b} + \mathbf{V}_k \tag{19}$$

$$\begin{bmatrix} x_{gps} \\ y_{gps} \\ z_{gps} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_k & \boldsymbol{v}_k & \boldsymbol{p}_k \\ \mathbf{0} & 1 & 0 \\ \mathbf{0} & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \begin{bmatrix} \boldsymbol{q}_k \\ 0 \\ 0 \end{bmatrix} \tag{20}$$

where $\boldsymbol{q}_k$ is measurement white noise with covariance $\boldsymbol{\Sigma}^q$.

To fit in the Kalman filter correction model, we calculate the Jacobian H by:

$$\boldsymbol{H}\boldsymbol{\xi}_t^r = \boldsymbol{\xi}_t^r\boldsymbol{b} = \begin{bmatrix} \boldsymbol{\xi}_t^R & \boldsymbol{\xi}_t^v & \boldsymbol{\xi}_t^p \\ \mathbf{0} & 0 & 0 \\ \mathbf{0} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{0} \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \mathbf{0}_{3\times 3} & \boldsymbol{I}_{3\times 3} \\ \mathbf{0}_{1\times 3} & \mathbf{0}_{1\times 3} & \mathbf{0}_{1\times 3} \\ \mathbf{0}_{1\times 3} & \mathbf{0}_{1\times 3} & \mathbf{0}_{1\times 3} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_t^R \\ \boldsymbol{\xi}_t^v \\ \boldsymbol{\xi}_t^p \end{bmatrix} \tag{21}$$

Considering computational efficiency, we choose H in its reduced form (Eq. 22) and the corresponding compensation is made later when calculating the Kalman gain.

$$\boldsymbol{H} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \boldsymbol{I}_{3\times 3} \end{bmatrix} \tag{22}$$

In the correction step, an innovation vector is found to quantify new information:

$$\boldsymbol{\nu}_k = \bar{\boldsymbol{X}}_k^{-1}\boldsymbol{Y}_k - b \tag{23}$$

After substituting the relationship between state and error in Lie algebra, we got the covariance of innovation as:

$$\boldsymbol{S}_k = \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T + \bar{\boldsymbol{N}}_k \tag{24}$$

where $\bar{\boldsymbol{N}}_k$ is the reduced form of $\boldsymbol{N}_k$:

$$\boldsymbol{N}_k = \bar{\boldsymbol{X}}_k^{-1}\begin{bmatrix} \boldsymbol{\Sigma}^q & & \\ & \mathbf{0} & \\ & & \mathbf{0} \end{bmatrix}\bar{\boldsymbol{X}}_k \tag{25}$$

$$\bar{\boldsymbol{N}}_k = \bar{\boldsymbol{R}}_k^T\boldsymbol{\Sigma}^q\bar{\boldsymbol{R}}_k \tag{26}$$

E. Including IMU Biases

In actual application, IMU usually has biases which corrupt the measurements progressively. Hence we modify the input of IMU system dynamics model as below:

$$\tilde{\boldsymbol{\omega}}_t = \boldsymbol{\omega}_t + \boldsymbol{b}_t^g + \boldsymbol{w}_t^g \qquad \boldsymbol{w}_t^g \sim \mathcal{GP}(\mathbf{0}_{3,1}, \Sigma^g) \tag{27}$$

$$\tilde{\boldsymbol{a}}_t = \boldsymbol{a}_t + \boldsymbol{b}_t^a + \boldsymbol{w}_t^a \qquad \boldsymbol{w}_t^a \sim \mathcal{GP}(\mathbf{0}_{3,1}, \Sigma^a) \tag{28}$$

Since each bias term remains almost unchanged between every time step, we choose the Brownian motion model for bias dynamics:

$$\frac{d}{dt}\boldsymbol{b}_t^g = \boldsymbol{w}_t^{bg} \qquad \boldsymbol{w}_t^{bg} \sim \mathcal{GP}(\mathbf{0}_{3,1}, \Sigma^{bg}) \tag{29}$$

$$\frac{d}{dt}\boldsymbol{b}_t^a = \boldsymbol{w}_t^{ba} \qquad \boldsymbol{w}_t^{ba} \sim \mathcal{GP}(\mathbf{0}_{3,1}, \Sigma^{ba}) \tag{30}$$

We combine all biases in a single vector (Eq. 31), and augment the error vector with the error in lie algebra (Eq. 32) instead of the state directly to avoid dimensional inconsistency. Then we found the new error Jacobian matrix $\boldsymbol{A}_t^l$, and compared the result with that in the paper of Hartley [4].

$$\boldsymbol{\theta}_k \triangleq \begin{bmatrix} b_t^g \\ b_t^a \end{bmatrix} \in \mathbb{R}^6 \tag{31}$$

$$\frac{d}{dt}\begin{bmatrix} \boldsymbol{\xi}_t \\ \boldsymbol{\zeta}_t \end{bmatrix} = \boldsymbol{A}_t \begin{bmatrix} \boldsymbol{\xi}_t \\ \boldsymbol{\zeta}_t \end{bmatrix} + \boldsymbol{w}_t \tag{32}$$

$$\boldsymbol{A}_t^l = \begin{bmatrix} -[\hat{\boldsymbol{\omega}}_t]_\times & \mathbf{0} & \mathbf{0} & -\boldsymbol{I}_{3\times3} & \mathbf{0} \\ -[\hat{\boldsymbol{a}}_t]_\times & -[\hat{\boldsymbol{\omega}}_t]_\times & \mathbf{0} & \mathbf{0} & -\boldsymbol{I}_{3\times3} \\ \mathbf{0} & \boldsymbol{I}_{3\times3} & -[\hat{\boldsymbol{\omega}}_t]_\times & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{33}$$

where $\boldsymbol{w}_t \triangleq [\boldsymbol{w}_t^g, \boldsymbol{w}_t^a, \mathbf{0}_{1\times3}, \boldsymbol{w}_t^{bg}, \boldsymbol{w}_t^{ba}]^T$ and its covariance is defined as $\boldsymbol{Q}_t$

With the bias term corrected, the system dynamics and the prediction model now become:

$$\bar{\boldsymbol{R}}_{k+1} = \boldsymbol{R}_k \boldsymbol{\Gamma}_0(\bar{\boldsymbol{\omega}}_k \Delta t) \tag{34}$$

$$\bar{\boldsymbol{v}}_{k+1} = \boldsymbol{v}_k + \boldsymbol{R}_k \boldsymbol{\Gamma}_1(\bar{\boldsymbol{\omega}}_k \Delta t)\bar{\boldsymbol{a}}_k \Delta t + \boldsymbol{g}\Delta t \tag{35}$$

$$\bar{\boldsymbol{p}}_{k+1} = \boldsymbol{p}_k + \boldsymbol{v}_k \Delta t + \boldsymbol{R}_k \boldsymbol{\Gamma}_2(\bar{\boldsymbol{\omega}}_k \Delta t)\bar{\boldsymbol{a}}_k \Delta t^2 + \frac{1}{2}\boldsymbol{g}\Delta t^2 \tag{36}$$

$$\bar{\boldsymbol{\theta}}_k = \boldsymbol{\theta}_{k-1} \tag{37}$$

$$\bar{\boldsymbol{P}}_k = \boldsymbol{\Phi}_k \boldsymbol{P}_{k-1} \boldsymbol{\Phi}_k^T + \boldsymbol{\Phi}_k \boldsymbol{Q}_t \boldsymbol{\Phi}_k^T \Delta t \tag{38}$$

where $\boldsymbol{\Phi}_k = \exp(\boldsymbol{A}_t \Delta t)$ is the state transition matrix, $\bar{\boldsymbol{\omega}}_k = \tilde{\boldsymbol{\omega}}_k - \boldsymbol{w}_k^g$ and $\bar{\boldsymbol{a}}_k = \tilde{\boldsymbol{a}}_k - \boldsymbol{w}_k^a$ are the bias-corrected input, $\boldsymbol{w}_k^g$ and $\boldsymbol{w}_k^a$ are the bias terms for the angular velocity and linear acceleration respectively.

After biases appended, we manually add two zero components to the Jacobian H to account for the bias term and update them simultaneously with the state by partitioning the Kalman gain in two parts:

$$\boldsymbol{H}_t = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \boldsymbol{I}_{3\times3} & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{39}$$

$$\boldsymbol{S}_k = \boldsymbol{H}_k \bar{\boldsymbol{P}}_k \boldsymbol{H}_k^T + \bar{\boldsymbol{N}}_k \tag{40}$$

$$\boldsymbol{K}_k = \begin{bmatrix} \boldsymbol{K}_k^\xi \\ \boldsymbol{K}_k^\zeta \end{bmatrix} = \bar{\boldsymbol{P}}_k \boldsymbol{H}_k^T \boldsymbol{S}_k^{-1} \tag{41}$$

Then the final bias-corrected correction model is:

$$\boldsymbol{X}_k = \bar{\boldsymbol{X}}_k \exp[\boldsymbol{K}_k^\xi \Pi(\bar{\boldsymbol{X}}_k^{-1}\boldsymbol{Y}_k)] \tag{42}$$

$$\boldsymbol{\theta}_k = \bar{\boldsymbol{\theta}}_k + \boldsymbol{K}_k^\zeta \Pi(\bar{\boldsymbol{X}}_k^{-1}\boldsymbol{Y}_k) \tag{43}$$

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)\bar{\boldsymbol{P}}_k(\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k)^T + \boldsymbol{K}_k \bar{\boldsymbol{N}}_k \boldsymbol{K}_k^T \tag{44}$$

where $\boldsymbol{\Pi} = [\boldsymbol{I}_{3\times3}, \mathbf{0}_{3\times2}]$ is a constant matrix to eliminate effects from truncating zero rows in $\boldsymbol{H}$.

What also should be noted here is the estimated covariance is in the Lie algebra, and we need to transfer it to the Cartesian space to plot the uncertainty, for which one way is to use the unscented transform and get the covariance in Cartesian space by propagating the corresponded vector in Lie algebra of estimated state.

III. Simulation Results

A. Data Preprocessing

The LI-EKF is tested with two types of data, i.e., simulated data generated by the group and Zurich Urban Micro Aerial Vehicle Dataset [1].

Onboard Pose data from PX4 autopilot board and onboard GPS data are utilized. GPS data is in international WGS 84 (GPS) coordinate system, which are transformed into world frame and aligned with pose coordination system. The noise covariance matrix of the GPS is calculated by taking the covariant of error between GPS and the ground truth data included in the Dataset.

B. Verification with Simulated Data

We ran a series of tests using a generated set of data to test the accuracy of the filter and test its robustness to input noise. The set of tests was run on the MATLAB version of the filter in order to use the built in plotting feature to provide a visual verification. We also used the simulated data tests to generate test cases for the C++ version of the filter, which are added as additional tests in its testing suite.

The simulated data consists of a set of state variables and their derivatives, created from a set of random polynomials which are functions of time. First, we generate six polynomials, three for the position (x, y, z) in the world frame and three for the angle axis rotation of the body frame with respect to the world frame. From the position polynomials, we take the derivative once to get the velocity as a function of time and once more to get the acceleration. Using the time data input as a parameter, we sample the polynomials at the desired time points and save the data. We similarly generate rotation matrices from the angle axis polynomials for each time point, and take the discrete derivative between time points to find the angular rates. The world frame acceleration is then rotated into the body frame, after having the gravity bias that an accelerometer would measure added to it. At the end of the procedure, we provide ourselves with simulated IMU measurements along with the exact ground truth data from which it was generated.

While simulation on "perfect" simulated data can be a good first check for the filter, we also verify the filter after corrupting the simulated data with random noise. In order to more closely

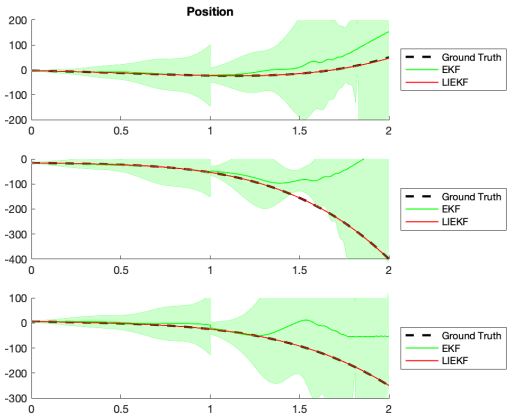

Fig. 1. (Simulated data, measurements without noise) EKF (solid green) and LIEKF (solid red) closely follow and converge to the ground truth (dashed black) from perturbed initial conditions. The covariance of each filter is shown as the shaded zone

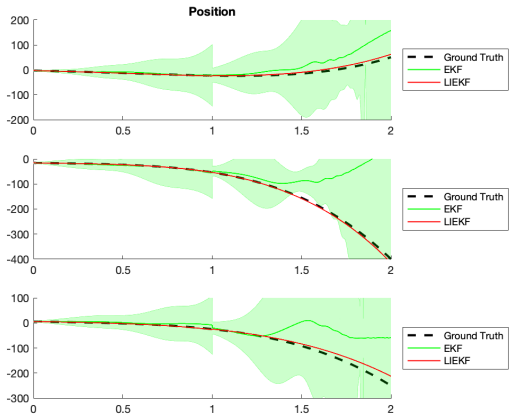

Fig. 3. (Simulated data, measurements with noise) EKF (solid green) and LIEKF (solid red) closely follows ground truth position despite noisy accelerometer reading. The covariance of each filter is shown as the shaded zone

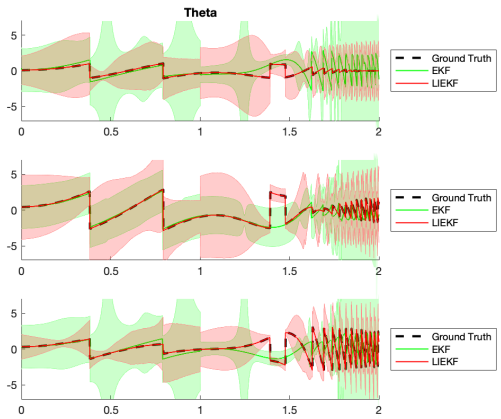

Fig. 2. (Simulated data, measurements without noise) EKF (solid green) and LIEKF (solid red) closely follow and converge to the rotational state of the body. Quantity plotted is $\theta = \text{Log}(R)$, where R is the rotation of the body. The covariance of each filter is shown as the shaded zone

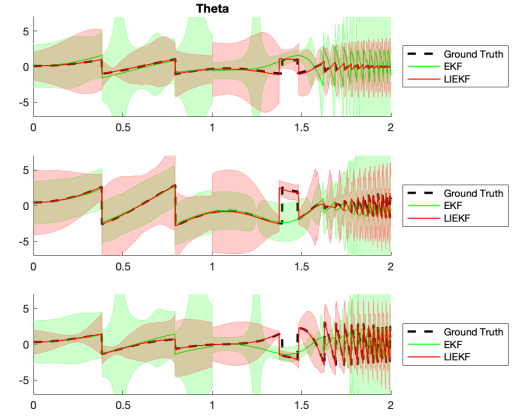

Fig. 4. (Simulated data, measurements with noise) EKF (solid green) and LIEKF (solid red) closely follows ground truth rotation despite noisy acclerometer readings. Quantity plotted is $\theta = \text{Log}(R)$, where R is the rotation of the body. The covariance of each filter is shown as the shaded zone

simulate the measurements obtained by a real IMU, we add zero mean Gaussian noise to the angular rate and body frame acceleration measurements. The noise has a tunable covariance parameter to achieve the desired noise level.

The plots in Figs. (1, 2) show a simulated run on the filter with no noise, but with position initial conditions perturbed from the ground truth. The polynomial functions used for the position and angle are listed in Appendix (B). The plots show that the filter converges in both the position and the orientation, and continues to follow the ground truth after converging. The variance of estimation from LI-EKF is bounded well along the trajectory, demonstrating the benefits of autonomous error dynamics.

The plots in Figs. (3,4) show another simulated run but with some large noise added to the measurements. In this test, we add normally disturbed, zero mean random noise with a standard deviation of $6g$'s (58.86 m/s^2) to the accelerometer measurements. Additionally we added similarly distributed noise with a standard deviation of 1 rad (57 deg) to the gyro measurements. Initial conditions were set the ground truth and the filter was run with an update frequency of 10 Hz. Despite the noisy conditions and the slow update rate, the filter tracked closely in position and orientation. However, there is visible disparity between the LI-EKF estimation and ground truth for velocity, especially along z-axis. The orientation also shows less accuracy, and there is a small but noticeable steady state error in the estimation. On another hand, the variance of the estimation is still bounded well and is not influenced much by the addition of noise.

C. Tests on Zurich Urban

We selected the Zurich Urban data set [1] to test our Left-invariant EKF implementation. The data set includes GPS, accelerometer, and gyroscope measurements for a quad-rotor traversing the streets of Zurich. Additionally, the data set

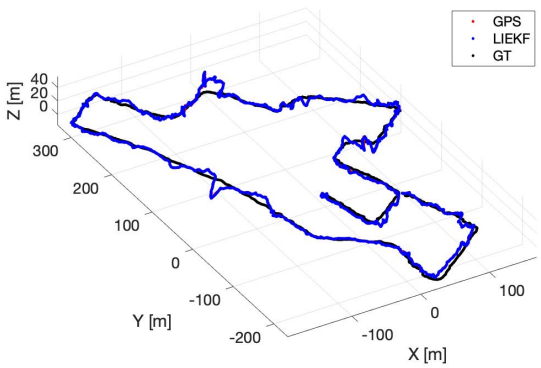

Fig. 5. The filter's state estimation on the Zurich Urban data set (blue) vs the ground truth (black). The filter follows the XY position very well, but the Z position is very noisy. The GPS readings (red) are also plotted, but are difficult to see as the estimate overlaps it in most places.

includes "ground-truth" data obtained using image processing. In our tests we use this estimate as our ground-truth. Using the accelerometer and gyroscope readings as our prediction step and the GPS measurements to update our prediction. The accelerometer and gyroscope update at roughly the same time step and so we assume these happen synchronously. The IMU and gyro update roughly 10 times per second. However, the GPS updates asynchronously, updating roughly once per second. To handles this, our filter runs a prediction step when it receives new IMU/gyro data and runs the correction step whenever GPS readings are received. Figures 5 and 6 show the results of applying our filter to the data set.

From the plots 5 and 6 we can see that the filter tracks the state well. However, the estimte is highly dependent upon the GPS measurements. In fact, in this case we fail to perform significantly better than just using pure GPS data. We can see in figure 7 that there is some error present, but that it is nearly identical at all points to the GPS error. Even though we use a bias correction on the IMU and gyro, we fail to gain any information from their inclusion.

The reason underneath that lies in the the quality of the measurements rather than the quality of our filter, since we have tested our filter on simulated data. Therefore, we could potentially improve the estimation results by including additional measurements such as incorporating magnetometer readings or using the gravity vector to correct the heading. Alternatively, we could add a bias correction in the measurement step which could help correct the noisy GPS data.

D. Comparison to Traditional EKF

As a baseline for comparison, we have developed traditional EKF algorithm in Matlab. The estimated states of the vehicle are position $\mathbf{p}(t)$, velocity $\mathbf{v}(t)$ and orientation in Euler angles $\phi(t)$.

It could be observed from Figure 1 to Figure 4, with or without noise in the measurement, LI-EKF would be able to track both the position and orientation of the body accurately. In comparison, EKF magnifies local oscillation for orientation tracking, and fails to follow the ground truth position at the end when the gradient becomes steep and the nonlinearity is accumulated more, since the approximation of linearization is breached. However, the variance of LI-EKF is bounded along the trajectory because of the autonomous error dynamic of Lie algebra; however, that of EKF would grow unbounded after a short while, which also demonstrate the failing in its tracking. The inclusion of noise seems to have a small but non-critical impact on the performance of both filters, with slightly increased values of error in position and orientation tracking.

For the Zurich Urban Dataset, the advantage of LI-EKF is not obvious through visual inspection. For both of the filers, the estimation results closely follow GPS data along the trajectory, indicating the prediction from IMU is not included in the estimation process. In Figure 6, it could be observed that though the estimation of the position for the two filters overlapped, the variance for LI-EKF is better bounded compared with EKF, suggesting a more reliable estimation statistically.

IV. C++ Implementation

In addition to the Matlab scripts, we provide a C++ implementation of the filter that is capable of running on a real system. The implementation uses the Eigen library, which is a header only library for linear algebra. Dependency and build information is listed on our Github[1], along with usage instructions.

We verify the correctness of this implementation using an extensive testing suite. Aside from simple checks on assumptions and basic input output, we compare the output of the filter in a number of conditions against the Matlab implementation. For each of these test, we generate a combination of IMU and GPS measurements, and create an expected output from our Matlab filter. Then, we run the C++ filter and compare the output to what was generated in Matlab.

We also provide an example driver for the Zurich urban dataset, where red-black trees are used to order asynchronous measurements.

V. Conclusion

In this project, we have developed a LI-EKF for drone localization in MATLAB and C++, taking in IMU measurement for prediction and GPS data for an update. The estimation is verified with self-generated simulated data and Zurich Urban Dataset. EKF algorithm is implemented as a baseline. For the simulated data, the LI-EKF has outperformed EKF in both tracking and boundness of variance; however, the performance of the filters are similar for the Zurich Urban Dataset, since the estimation are following GPS mostly due to the corruption of IMU data. In the future, the LI-EKF might be tested with the inclusion of more sensors or switch to another Dataset.

[1] https://github.com/ghaggin/invariant-ekf

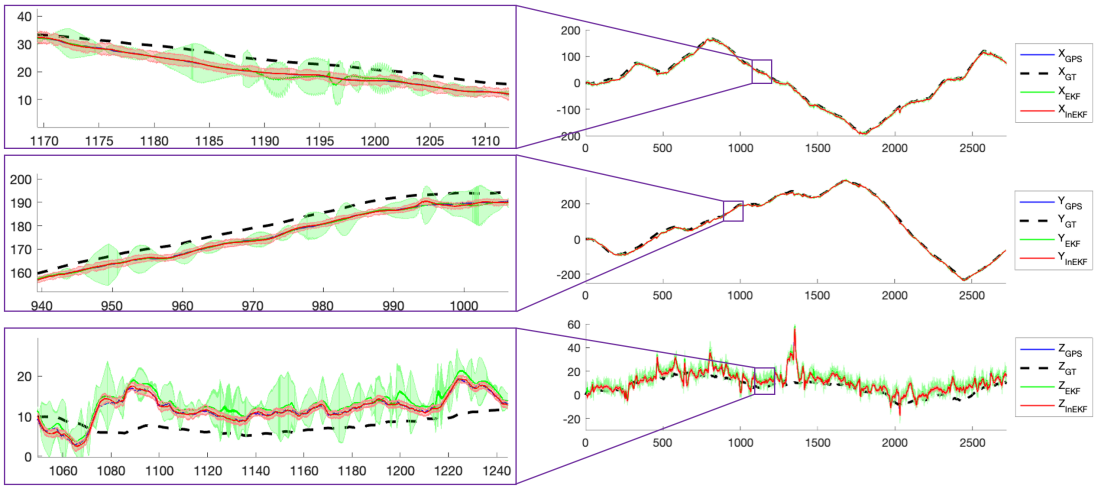

Fig. 6. The x (top), y (middle), and z (bottom) positions and standard deviation of the drone over time. The dashed black line is the ground truth position, and the blue line (covered by green and red) is the GPS position. The red line is the LI-EKF position and the green line is the EKF position. The shaded region is 3σ covariance. Although both trajectory follow the GPS trajectory closely, we can see that in the zoomed in version, the variance for LI-EKF is lower than the standard deviation for the EKF. This means that there is less error in the LI-EKF position

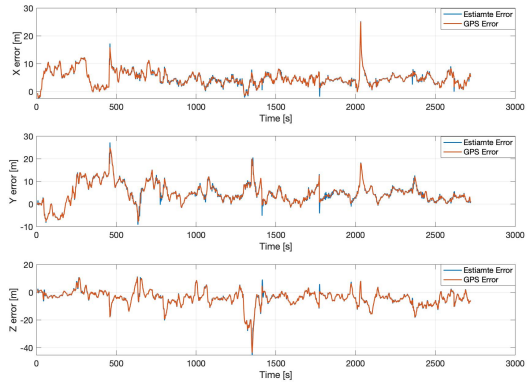

Fig. 7. The X (top), Y (middle), and Z (bottom) error of the drone position over time. The LI-EKF filter error is in blue and the GPS error is in orange dashed lines. In most places these lines overlap.

References

[1] András L Majdik, Charles Till, and Davide Scaramuzza. The zurich urban micro aerial vehicle dataset. *The International Journal of Robotics Research*, 36(3):269–273, 2017.

[2] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[3] Silvère Bonnabel Axel Barrau. The invariant extended kalman filter as a stable observer. *IEEE Transactions on Automatic Control (Volume: 62 , Issue: 4, April 2017)*.

[4] Ross Hartley, Maani Ghaffari, Ryan M Eustice, and Jessy W Grizzle. Contact-aided invariant extended kalman filtering for robot state estimation. *The International Journal of Robotics Research*, 39(4):402–430, 3/2020.

Appendix A
Derivation of Discrete Dynamics

The continuous time differential equations which govern the motion of the rigid body are

$$\frac{d}{dt}\boldsymbol{R} = \boldsymbol{R}[\boldsymbol{\omega}]_\times \tag{45}$$

$$\frac{d}{dt}\boldsymbol{v} = \boldsymbol{R}\boldsymbol{a} + \boldsymbol{g} \tag{46}$$

$$\frac{d}{dt}\boldsymbol{p} = \boldsymbol{v} \tag{47}$$

where $\boldsymbol{R}$ is the rotation of the body with respect to the world frame on SO(3), $\boldsymbol{\omega}$ is the angular velocity of the body frame with respect to the world frame, $\boldsymbol{v}$ is the linear velocity, $\boldsymbol{a}$ is the acceleration measured in the body frame, $\boldsymbol{g}$ is the local gravity, and $\boldsymbol{p}$ is the position of the vehicle.

The imu measurements occur in discrete increments and the time between measurements is denoted Δt. Since we have no knowledge of the acceleration and the angular rate between measurements, we are unable to exactly integrate the continuous time dynamics. To make the problem tractable, we assume that the acceleration and rates are constant between measurements, and take the last measured value throughout the time period Δt. Under this assumption, we can derive a exact state transition equations for $\boldsymbol{R}$, $\boldsymbol{p}$, and $\boldsymbol{v}$ between measurements using Eqns. (45)-(47).

One might be tempted to directly integrate Eqn. (46) under the assumption of a constant $\boldsymbol{R}$. This assumption is not valid since from Eqn. (45), we can see that $\boldsymbol{R}$ is in fact time varying with a constant $\boldsymbol{\omega}$. The following derivation takes the time varying nature of $\boldsymbol{R}$ into account when integrating Eqn. (46).

We can first note that the solution to Eqn. (45) is simply the matrix exponential function, since $[\boldsymbol{\omega}]_\times$ is a constant matrix.

$$\boldsymbol{R}(t) = \boldsymbol{R}_0 \exp([\boldsymbol{\omega}]_\times t)) \tag{48}$$

Using the Rodriguez rotation formula, we can rewrite Eqn. (48) in a format that will be easier to integrate in the other dynamics equations.

$$\begin{aligned}\boldsymbol{R}(t) &= \boldsymbol{R}_0\left[\boldsymbol{I} + \frac{\sin(\|\boldsymbol{\omega}t\|)}{\|\boldsymbol{\omega}\|}[\boldsymbol{\omega}]_\times + \frac{1-\cos(\|\boldsymbol{\omega}t\|)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times^2\right] \\ &= R_0\boldsymbol{\Gamma}_0(t) \end{aligned} \tag{49}$$

Here we denote the bracketed expression as $\boldsymbol{\Gamma}_0(t)$ and notice that all matrices are constants multiplied by scalar trigonometric functions.

Now, writing the velocity dynamics in terms of $R(t)$, we get

$$\frac{d}{dt}v = \boldsymbol{R}(t)\boldsymbol{a} + \boldsymbol{g} \tag{50}$$

$$\frac{d}{dt}v = \boldsymbol{R}_0\boldsymbol{\Gamma}_0(t)\boldsymbol{a} + \boldsymbol{g} \tag{51}$$

Integrating this with respect to time, we find

$$\boldsymbol{v}(t) = R_0\left(\int \boldsymbol{\Gamma}_0(t)dt\right) + \boldsymbol{g}t + C_1 \tag{52}$$

where

$$\begin{aligned}&\int \boldsymbol{\Gamma}_0(t)dt = \\ &\boldsymbol{I}t + \frac{-\cos(\|\boldsymbol{\omega}t\|)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times + \frac{\|\boldsymbol{\omega}t\| - \sin(\|\boldsymbol{\omega}t\|)}{\|\boldsymbol{\omega}\|^3}[\boldsymbol{\omega}]_\times{}^2\end{aligned} \tag{53}$$

Solving for C in Eqn. (52),

$$\boldsymbol{v}(0) = \boldsymbol{v}_0 = -\frac{[\boldsymbol{\omega}]_\times}{\|\boldsymbol{\omega}\|^2} + C_1 \tag{54}$$

And plugging this into (52),

$$\boldsymbol{v}(t) = \boldsymbol{R}_0\boldsymbol{\Gamma}_1(t)\boldsymbol{a}t + \boldsymbol{g}t + \boldsymbol{v}_0 \tag{55}$$

with

$$\boldsymbol{\Gamma}_1(t)t = \boldsymbol{I}t + \frac{1-\cos(\|\boldsymbol{\omega}\boldsymbol{t}\|)}{\|\boldsymbol{\omega}\|^2}[\boldsymbol{\omega}]_\times + \frac{\|\boldsymbol{\omega}\| - \sin(\|\boldsymbol{\omega}\boldsymbol{t}\|)}{\|\boldsymbol{\omega}\|^3}[\boldsymbol{\omega}]_\times{}^2 \tag{56}$$

Note that we specify $\boldsymbol{\Gamma}_1(t)t$, which is $\boldsymbol{\Gamma}_1$ as a function of t times the variable t. We use this expression so that the next integral is clearer. In the paper, we remove the factor of t from the function to make the equation clearer and in terms of $\boldsymbol{\omega}t$ as a unit.

Now we can integrate the position dynamics.

$$\frac{d}{dt}\boldsymbol{p} = \boldsymbol{R}_0\boldsymbol{\Gamma}_1(t)\boldsymbol{a}t + \boldsymbol{g}t + \boldsymbol{v}_0 \tag{57}$$

$$\boldsymbol{p} = \boldsymbol{R}_0\left(\int \boldsymbol{\Gamma}_1(t)t\ dt\right)\boldsymbol{a} + \frac{1}{2}\boldsymbol{g}t^2 + \boldsymbol{v}_0t + C_2 \tag{58}$$

where

$$\begin{aligned}\int \boldsymbol{\Gamma}_1(t)t\ dt = \frac{1}{2}\boldsymbol{I}t^2 + \frac{\|\boldsymbol{\omega}t\| - \sin(\|\boldsymbol{\omega}t\|)}{\|\boldsymbol{\omega}\|^3}[\boldsymbol{\omega}]_\times + \\ \frac{\|\boldsymbol{\omega}t\|^2 + 2\cos(\|\boldsymbol{\omega}t\|)}{2\|\boldsymbol{\omega}\|^4}[\boldsymbol{\omega}]_\times{}^2\end{aligned} \tag{59}$$

Solving for C_2 in (58),

$$\boldsymbol{p}(0) = \boldsymbol{p}_0 = \frac{[w]_\times^2}{\|w\|^4} + C_2 \tag{60}$$

and plugging this back into (58),

$$\boldsymbol{p}(t) = \boldsymbol{R}_0\boldsymbol{\Gamma}_2(t)\boldsymbol{a}t^2 + \frac{1}{2}\boldsymbol{g}t^2 + \boldsymbol{v}_0t + \boldsymbol{p}_0 \tag{61}$$

with

$$\begin{aligned}\boldsymbol{\Gamma}_2(t) = \frac{1}{2}\boldsymbol{I} + \frac{\|\boldsymbol{\omega}t\| - \sin(\|\boldsymbol{\omega}t\|)}{\|\boldsymbol{\omega}t\|^3}[\boldsymbol{\omega}t]_\times + \\ \frac{\|\boldsymbol{\omega}t\|^2 + 2\cos(\|\boldsymbol{\omega}t\|) - 2}{2\|\boldsymbol{\omega}t\|^4}[\boldsymbol{\omega}t]_\times{}^2\end{aligned} \tag{62}$$

Thus, if we take the initial conditions to be at timestep k and the value at $k+1$ to be the quantity of the state transition over time Δt, the discrete dynamical equations become

$$\boldsymbol{R}_{k+1} = \boldsymbol{R}_k\exp([\boldsymbol{\omega}]_\times\Delta t) \tag{63}$$

$$\boldsymbol{v}_{k+1} = \boldsymbol{R}_k\boldsymbol{\Gamma}_1(\boldsymbol{\omega}\Delta t)\boldsymbol{a}\Delta t + \boldsymbol{g}\Delta t + \boldsymbol{v}_k \tag{64}$$

$$\boldsymbol{p}_{k+1} = \boldsymbol{R}_k\boldsymbol{\Gamma}_2(\boldsymbol{\omega}\Delta t)\boldsymbol{a}\Delta t^2 + \frac{1}{2}\boldsymbol{g}\Delta t^2 + \boldsymbol{v}_k\Delta t + \boldsymbol{p}_k \tag{65}$$

APPENDIX B
SIMULATED DATA

The simulated data used for the tests shown in Sec. (III) is listed here. Each dimension of the angle or positional data is created from a randomly generated polynomial, where each coefficient is sampled from some random distribution. Each polynomial is a 4th order polynomial and can be evaluated at any point in time. The coefficients are listed in Tabs. (I, II), with the first coefficient listed being the scalar multiplied by x^4 and the last being the constant (x^0 term). This the default ordering that the Matlab function `polyval` expects.

TABLE I
POSITION GROUND TRUTH RANDOM POLYNOMIAL COEFFICIENTS.

x:	[13.862459 -16.814181 0.209844 -17.388540 -2.875107]
y:	[-16.138763 -14.913601 3.869812 -10.959520 -15.722173]
z:	[-11.187752 -6.006949 -1.288501 -11.930271 5.616269]

TABLE II
ANGLE GROUND TRUTH RANDOM POLYNOMIAL COEFFICIENTS.

x:	[-0.677207 0.209469 -4.524294 11.745498 3.200167]
y:	[-13.508056 8.030094 18.582043 0.000334 15.580803]
z:	[-6.335454 2.685765 -2.898161 -2.530109 11.062367]