Extended Kalman Filter Localization, Mapping, and Visual-Inertial SLAM using landmarks and an Autonomous Vehicle

Kwok Hung Ho
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
khh019@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, U.S.A.
natanasov@eng.ucsd.edu

I. Introduction

With rapid developments in the autonomous vehicles and robotics space, the problem of Simultaneous Localization and Mapping (SLAM) has become as popular as ever in the mobile robotics literature.

This paper focuses on the implementation of the Extended Kalman Filter (EKF) on data collected from an autonomous car. There will be three parts to the implementation, Self Localization via EKF Prediction, Landmark Mapping via EKF Update, and combining the two to achieve Visual-Inertial SLAM. By using the EKF approach, real world non-linearities of the system models can be approximated. This implementation is also optimized to run in an efficient manner due to dynamic resizing of matrices. As we see more and more autonomous vehicles in the modern world, reliable and fast SLAM systems are needed to ensure safety of passengers and civilians.

II. Problem Formulation

A. Problem Formulation: Data and Objective

Understanding that there can be a myriad of approaches to solve the problem, only the input, output, and associated models that cannot be changed and are inherent to the problem will be presented here. The technical approach, where solutions to the problems that will be described, will be presented in the next section.

The 2 equally formatted datasets were provided. Each includes a list of timestamps, optical frame pixel coordinates of features in the left and right stereo cameras, linear velocity and angular velocity provided by an Internal Measuring Unit (IMU), IMU to camera pose matrix, and camera Intrinsic parameters and baseline. All the data are given as numpy objects to be loaded.

The given IMU linear and angular velocities and Stereo pixel coordinates, all have an axis which aligns with the length of the timestamps. The features are also assumed to be data outputted from some prior feature detection model unbeknownst to this implementation. There is also 2 image datasets to verify trajectories and landmarks while implementing the system.

To be be more specific about the data, the features are given as 3 dimensional numpy arrays, the first being size 4 signifying the left and right stereo coordinates $[ul, vl, ur, vr]$, the second being the amount of features/landmarks and the third being for each timestep. The angular and linear velocities are given as 3×timesteps arrays signifying velocities in $[x, y, z]$ and angular velocities in $[yaw, pitch, roll]$. If the features $[ul, vl, ur, vr]$ for a given time step and landmark is equal to $[-1, -1, -1, -1]$, then the landmark can no longer be seen by the stereo camera.

With the data provided, the goal is to implement EKF SLAM, predicting where the car is at each timestamp and mapping the environment all the while updating our predictions based on new observations. As a result, a 2D top down map of the world showing the robot's path, and the location of each landmarks is desired.

B. Problem Formulation: Localization and Mapping

The goal of localization is to estimate the pose in $SE(3)$ of the car. Although the goal is to estimate the $[x, y, z]$ of the car as well as the $[yaw, pitch, roll]$, the estimate of the z position is mostly ignored since the sensors do not move sufficiently in the z direction. Furthermore, 2d top-down mapping is desired rather than a 3d point cloud. By similar argument, the pitch and roll rotational directions should be mostly ignored too in the output.

C. Problem Formulation: Motion and Observation Models

A motion model is defined as a function relating the current state **x** and control input **u** of a robot with its state change subject to motion noise **w**. It's parameters are as follows:

$$\dot{x}(t) = f(x(t), u(t), w(t)) \tag{1}$$

$$x_{t+1} = f(x_t, u_t, w_t) \tag{2}$$

for continuous and discrete time respectively. Due to non-linearities, these will be redefined and parameterized in the

technical approach section. An observation model is a function relating the robot state x_t and the environment m_t with the sensor observation z_t subject to measurement noise v_t:

$$z_t = h(x_t, m_t, v_t) \tag{3}$$

with the following pdf due to the presence of measurement noise:

$$p_h(\cdot|x_t, m_t) \tag{4}$$

D. Problem Formulation: Conclusion

With the problem formulated, the technical approach which describes how to solve the problem will be presented. Modifications and implementations of the formulation above to achieve the goal will be presented. The realization of mapping, localization and SLAM via the EKF will be shown in detail.

III. Technical Approach

A. Technical Approach: Introduction

Although there could be other ways to solve the problem formulated in the above section, the section: technical approach, will attempt to solve the problem formulated in the previous section via the Extended Kalman Filter (EKF), which is the centerpiece of this paper.

B. Technical Approach: Non-linear Models

Similar to the Kalman Filter, the EKF assumes a Gaussian prior PDF $p_{t|t}$, a Gaussian motion model noise w_t and observation noise v_t. As well as independence of w_t and v_t with each other, the state x_t and across time.

Contrary to the regular Kalman Filter, the Extended Kalman Filter does not assume the motion and observation model is linear. The motion model and observation model are as described in (2) and (3) respectively, except w_t and v_t are modeled as zero mean Gaussian noise:

$$w_t \sim \mathcal{N}(0, W) \tag{5}$$

$$v_t \sim \mathcal{N}(0, V) \tag{6}$$

and having state with prior:

$$x_t|z_{0:t}, u_{0:t-1} \sim \mathcal{N}(\mu_{t|t}, \Sigma_{t|t}) \tag{7}$$

Where z is the observation and u is the control input.

C. Technical Approach: IMU Localization

In EKF Localization, only the IMU data is used to perform localization. Trusting the sensor, only EKF prediction is performed via the motion model. Using $SE(3)$ kinematics, the prediction step is as follows:

$$\mu t + 1|t = \mu_{t|t} \exp(\tau_t \hat{u_t}) \tag{8}$$

$$\exp(-\tau \overset{\curlywedge}{u_t}) \Sigma t|t \exp(-\tau \overset{\curlywedge}{u_t})^T + W \tag{9}$$

Where μ is the mean pose of the car in $SE(3)$, τ is the time difference, u is the control input, and the covariance calculation can be ignored since SLAM is not involved. The $\wedge$ in (8) maps the vector into the space of skew symmetric matrices.

D. Technical Approach: Landmark Mapping

In Landmark mapping, the EKF update step is used. Firstly, the IMU trajectory from localization is assumed to be correct as opposed to updating that too in SLAM. The landmark positions $m \in \mathbb{R}^{3M}$ will be estimated, but only the x and y values will be mapped. Formally defining the EKF update for this observation model will require building the Jacobian H:

$$H_{t+1,i,j} = K_s \frac{d\pi}{dq} ({}_oT_i T_{t+1}^{-1} \underline{\mu_{t,j}}) {}_oT_i T_{t+1}^{-1} P^T \tag{10}$$

if the new observation landmark i corresponds with initiated landmarks j, and 0 otherwise. Ks is the intrinsic calibration as a $4x4$ matrix, the π function is the canonical projection function, ${}_oT_i$ is pose matrix that transforms points from IMU frame to optical frame, T is the pose matrix that takes points from IMU to the world frame, which is essentially the pose of the car in $SE(3)$ and $\underline{\mu_{t,j}}$ is the landmark in homogenous coordinates. The Kalman gain K would also be computed as:

$$K_{t+1} = \Sigma_T H_{t+1}^T (H_{t+1} \Sigma_t H_{t+1}^T + I \otimes V)^{-1} \tag{11}$$

Where $I \otimes V$ can be realized via numpy's np.kron function. To actually update the landmark poses and covariances:

$$\mu_{t+1} = \mu_t + K_{t+1}(z_{t+1} - \tilde{z}_{t+1}) \tag{12}$$

Where $z_{t+1} - \tilde{z}_{t+1}$ is the innovation, with z being the features being observed, and $\tilde{z}$ being the predicted pose of the landmarks mapped back onto the camera. The covariance of the landmarks are updated as so:

$$\Sigma_{t+1} = (I - K_{t+1} H_{t+1}) \Sigma_t \tag{13}$$

E. Technical Approach: Visual-Inertial SLAM

In Visual-Inertial SLAM, the IMU prediction step and landmark update step are combined, and as a bonus, the IMU update step is performed for a true realization of SLAM. In the prediction step, the car will use IMU data to predict the mean and covariance of the pose for the next time step, while in the update step, the observations will tweak the pose of the car and the pose and covariance of the landmarks.

EKF SLAM will be very similar to EKF mapping in that both will use the same prediction step, and landmark updating steps. SLAM differs in that it will have an extra updating IMU pose step, and will combine IMU covariance and Landmark covariance into a single $3M + 6 \times 3m + 6$ for better results. To update the pose:

$$\mu_{t+1|t+1} = \mu_{t+1|t} \exp((K_{t+1}(z_{t+1} - \tilde{z}_{t+1}))^\wedge) \tag{14}$$

Where the $+$ symbol is now replaced by multiplication with $\exp$ of the $\wedge$ of the kalman gain $\times$ innovation. This again maps the input into the space of symmetric matrices before taking the matrix exponential.

F. Technical Approach: Optimization and Conclusion

To optimize the accuracy, the covariances of the landmarks and IMU are combined. This will increase the accuracy of the localization and mapping because the mapping and pose are correlated.

To optimize the speed of the EKF SLAM, several things were implemented. Firstly, the option to use less features will increase the speed but sacrifice accuracy of the localization and amount of points being mapped. Instead of skipping just a flat amount of landmarks, the features will be skipping every k features. This allows better spread of the data and not have all the landmarks jumbled up together near the start of the path.

Another strategy that was implemented to optimize speed was to dynamically allocate the size of the Jacobian and the Kalman gain and perform operations based on the max index of the initialized landmarks. Since the landmarks will eventually be all initialized, this increases the speed significantly near the beginning but will be back to normal speeds towards the end.

To make the program modularized, a helper functions file and 2 classes were made: one for EKF Mapping and one for EKF Visual-Inertial SLAM. Both employ EKF localization so are very similar but have some differences as discussed above. Running the mapping or SLAM is also very simple through one main loop.

IV. Results

A. Results: Analysis and Images

The overall results are positive and everything worked as expected. Several tests were run varying the amount of features, the prior motion noise covariance and the prior observation noise covariance. Running times were also exponentially longer with more features.

It was found that even with high noise, the EKF mapping and SLAM were both very resillient to a high landmark covariance. The trajectories for both were also very reliable and similar. This is could be due to the IMU data being very accurate, resulting in a strong prediction step.

As seen in the resulting plotted 2D maps, the trajectories and landmarks weren't paticularly affected despite adding more noise to the motion by a magnitude of 10. Even in subsequent tests with EKF SLAM when noise was absurdly large, the paths and landmarks did not deviate much. This could be due to the algorithm being too resilient to changes by using the Kalman gain gained from the innovation step, which adjusts the weight to counteract absurd predictions, meaning the update step is doing a very fine job.

From the graphs, it is also shown that the tests were ran only for skipping 10 and 20 features per feature. This is because skipping less features would take very long to run, especially for the 10.npz dataset which has $13,000$ features.

Following are some of the EKF mapping and SLAM results from the 2 data sets.

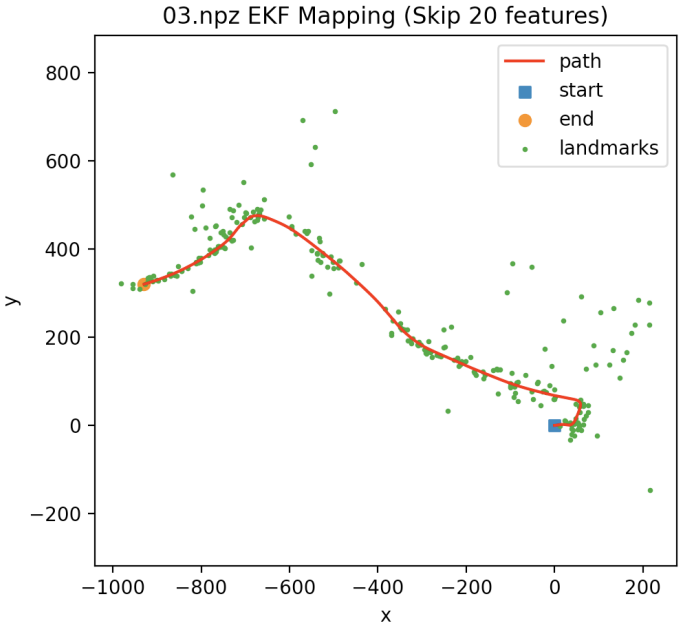

Fig. 1. $1e-3*np.eye(6)$ motion noise

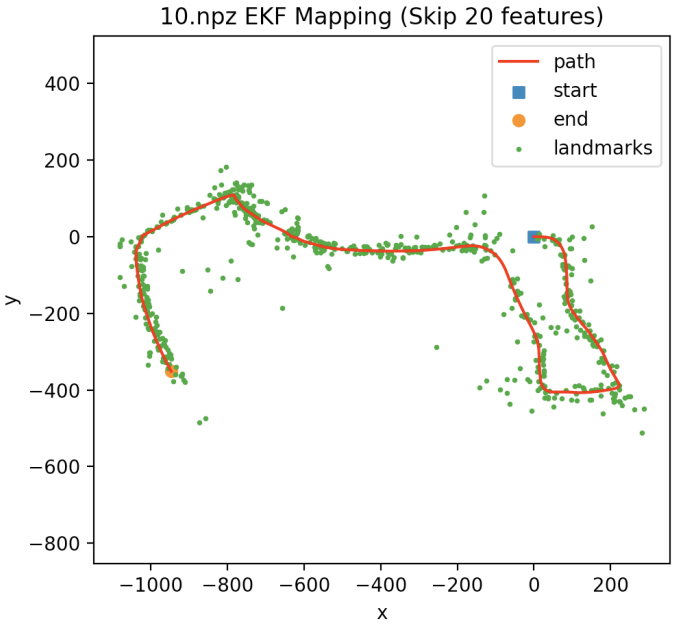

Fig. 2. $1e-3*np.eye(6)$ motion noise

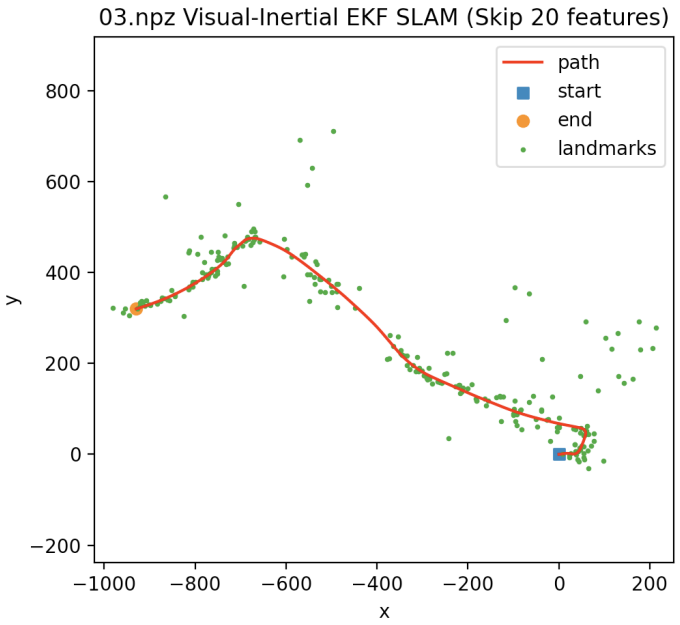

Fig. 3. $1e-3*np.eye(6)$ motion noise

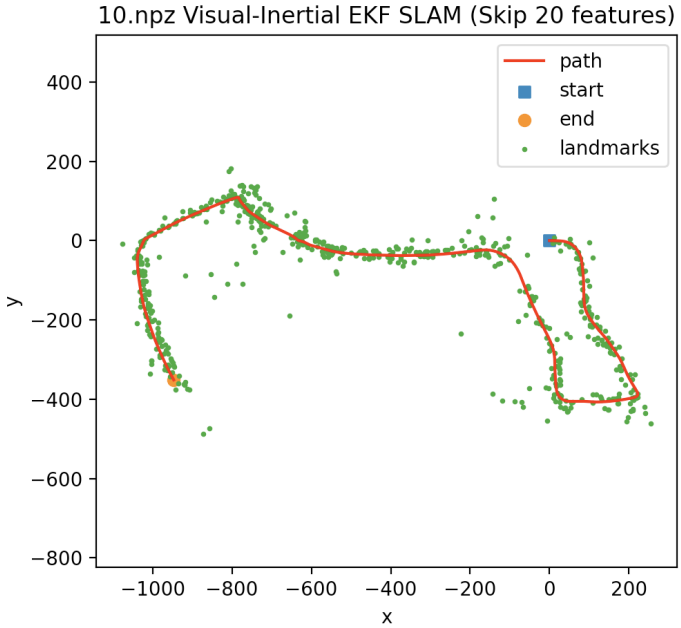

Fig. 4. $1e-3*np.eye(6)$ motion noise

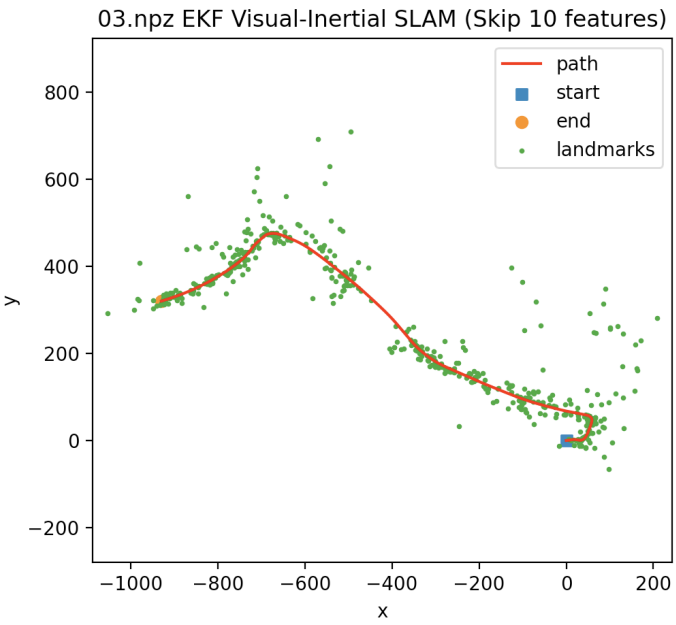

Fig. 5. $diag([0.3, 0.3, 0.3, 0.05, 0.05, 0.05])$ motion noise

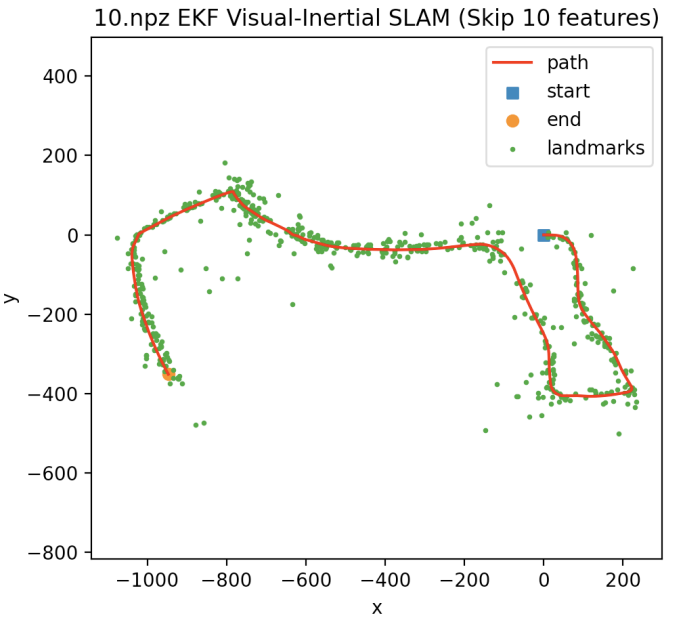

Fig. 6. $diag([0.3, 0.3, 0.3, 0.05, 0.05, 0.05])$ motion noise

B. Results: Conclusion

Overall, the implementation of EKF Visual-Inertial SLAM was a success. For future improvements, spending more time on fine tuning parameters could lead to an even more accurate SLAM. For speed, eliminating the use of for-loops when building the Jacobian could be a possible route too. Also, the main bottleneck of the program is calculating the inverses of poses and the very large Kalman gain term. If there were quicker implementations to solve that other than np.linalg.solve, this could also reduce the time complexity.