A Quantitative Analysis of Kalman-Filter based Visual SLAM systems

Sougato Bagchi
University at Buffalo, USA
sougatob@buffalo.edu

***Abstract*—With the ever-increasing applications of autonomous robots, there remains a fundamental problem that needs to optimize, i.e. how do these machines create a map of their surroundings and localize themselves in it in real-time and efficiently? This is known as SLAM (Simultaneous Localization Mapping). Here I will explain the Kalman-Filter Based SLAM systems by taking [1]OPENVINS as an example.**

***Index Terms*—SLAM, Extended Kalman-Filter, MSCKF, computer vision**

I. Introduction

The use of different probabilistic filters in the field SLAM is common and it's due to the fact that When these are deployed in real-life, in most cases the environments are generally unpredictable and dynamic.

Most robots depend on 2 important components for navigation, and these are:-

- Sensors
- Actuators(involves motors)

Each of these components incurs errors over the course of time. For this reason, we implement the backbone for localization using the Bayes-Filter or some modified versions for better performance. Many SLAM systems use methods other than the Bayes Filter, but there is a significant advantage to these filter-based systems. Some are:-

- lower time complexity
- can be deployed in resource-constrained hardware
- hybrid filter-based SLAM systems have a very less ATE accuracy trade-off

From a theoretical aspect, we have $bel(x_{t-1})$, u_t, and the z_t of the robot for each and every time, these translate to the belief of the state variable at time $t-1$, control data at time t and sensor data at time t respectively. So our objective here is to calculate the $bel(x_t)$. Also when we apply our algorithms in real world scenarios then we need to calculate the $z_i^{(j)}$ and this is done using equation 1

$$z_i^{(j)} = \frac{1}{{}^{C_i}Z_j} \begin{bmatrix} {}^{C_i}X_j \\ {}^{C_i}Y_j \end{bmatrix} + n_i^{(j)} \tag{1}$$

$${}^{C_i}p_{f_j} = \begin{bmatrix} {}^{C_i}X_j \\ {}^{C_i}Y_j \ {}^{C_i}Z_j \end{bmatrix} \tag{2}$$

In equ 1 $i = 1...n$; no. of cameras and equ 2 denotes Feature position in the camera C_i frame.

OpenVins deploys a hybrid architecture to overcome the fundamental restrictions for the Bayes filter and improves the accuracy in estimating the $bel(x_t)$. It uses the [2]MSCKF and EKF(Extended Kalman Filter) for different tracked features in different situations.

A. Reason for the Hybrid Implementation

From the time complexity and the behavior of both the algorithms (i.e., MSCKF and the EFK) explained later, we know that when the no. of features is huge but they have been tracked in a few images/frames then MSCKF is a good choice for execution. But when we have few features and they have already been tracked in many frames its better to execute a less costly algorithm like the EKF (its be susceptible to linearization errors, but as these features has been tracked for too long, they have good estimates).

Each and every tracked feature has a different length (no. of frames it has been tracked). Here the algorithm tracks a feature over 11 frames, and it captures a maximum of 100 features per frame using optical flow. Ultimately it keeps 50 landmarks in the state, where we can say that a feature translates to a landmark only when it has been tracked in a minimum of 11 frames.

Here which module (EKF or the MSCKF) will be executed for the features depends on the threshold m

- If feature i's track is lost after fewer than m frames (i.e., $l < m$), then the feature is processed using the MSCKF equations.
- If a feature is still actively being tracked after m images, it is initialized into the state vector, and used for SLAM

This hybrid method helps in reducing the time complexity with optimized estimation. This hybrid method of OPENVINS has been inspired by [3]Li, Mingyang, and Anastasios I. Mourikis. So in simpler terms when a feature gets tracked for the 1st time, The MSCKF algorithm is executed on it till the no of times it has been tracked doesn't cross the threshold m, and when it crosses that then the EKF takes the responsibility. After the execution of the EKF, these features from the state vector are designated as SLAM features and the rest are designated as MSCKF fearures. These MSCKF features are basically one those are discarded from being used in the OPENVINS slam system, but if these MSCKF features crosses the threshold in the future then they can be designated as SLAM features.

B. Select the optimal value of m

- Example :- Let's say the sliding window length m = 20, no of features = 10 & they have been tracked in 20 images.
- Now there are 2 possibilities:-
  - increasing the size of the sliding window or
  - Include these 10 features in the state vector.
- This depends on the future behavior of the features. How?
  - if these features end up being tracked for a very large number of frames ($>>$ 20), then it would be preferable to include the features in the state vector.
  - If, on the other hand, the features end up being tracked for only 21 frames, it would be preferable to increase m by one.

C. How do we obtain future info?

- During the filter's operation we learn the probability mass function (pmf) of the
  - feature track lengths $p(l_i)$
  - probability of failure of the Mahalanobis gating test
  - pmf of the number of features tracked in the images
- Using the learned pmfs, we compute the average number of operations needed for each EKF update. f(m)
- The learning of the pmfs as well as the selection of the optimal threshold in consecutive time windows spanning a few seconds (15 sec in the [3]paper's implementation)

II. Extended Kalman Filter

1. **Algorithm Extended Kalman Filter** (μ_t, $\sum_t$, u_t, z_t):
2. $\overline{\mu_t} = g(u_t, \mu_{t-1})$
3. $\overline{\sum_t} = G_t \sum_{t-1} G_t^T + R_t$
4. $K_t = \overline{\sum_t} H_t^T (H_t \overline{\sum_t} H_t^T + Q_t)^{-}1$
5. $\mu_t = \overline{\mu_t} + K_t(z_t - h(\overline{\mu_t}))$
6. $\sum_t = (I - K_t H_t)\overline{\sum_t}$
7. **return** $\mu_t, \sum_t$

In this algorithm μ_t, $\sum_t$ makes up for $bel(x_{t-1}$. Line 2 and 3 calculates the $\overline{bel(x_t)}$ before incorporating z_t This is a modified version of the Kalman filter which itself is based on the Bayes filter. Here the nonlinearity of the observations and the next state is considered. So here the state transition variable x_t is a function of x_{t-1} and similarly the measurement variable z_t is also a function of z_{t-1}

$$x_t = g(u_t, x_{t-1}) + \epsilon_t \tag{3}$$

$$z_t = h(x_t) + \delta_t \tag{4}$$

The EKF handles all the shortcomings of the Naive KF, by implementing linearization. So the accuracy in the estimation depends on how well we linearize the required variables and also on the prior and other input data.

A. *Time Complexity*

The time complexity of a typical Kalman Filter is $O(K^2.4+ n^2)$, where K denotes the dimension of the z vector and n denotes the state vector x_t.

III. Multi-State Constraint Kalman Filter

The MSCKF is the backbone of the OPENVINS as this algorithm poses some unique advantages.

- z_t (measurement model) is able to express the geometric constraints(like, features found in both left & right camera images for OpenVins & in the stereo system) that arise when a static feature is observed from multiple camera poses. This also helps in attaining higher estimation accuracy.
- Reduced time complexity.

A. MSCKF as published by the authors of [2]

Algorithm MSCKF

Propagation: For each IMU measurement received, propagate the filter state and covariance.

Image registration: Every time a new image is recorded,

- augment the state and covariance matrix with a copy of the current camera pose estimate
- image processing module begins operation

Update: When the feature measurements of a given image become available, perform an EKF update.

The Propagation adjusts the $\overline{X_{IMU}}$ w.r.t different control inputs. The Image registration module deals with Detecting features from the incoming camera images and also due to the unavailability of z_t, it is derived from the camera poses per tracked feature. During the execution of the Update module, we have our u_t, z_t, & X_{IMU}^{t-1} and so we are ready to calculate the EKF update.

B. Modified MSCKF algorithm as published by the authors of [3]

Hybrid MSCKF/SLAM algorithm

Propagation: For each IMU measurement received, propagate the filter state and covariance. **Update:** Once camera measurements become available:

- Augment the state vector with the latest camera pose.
- For features to be processed in the MSCKF (feature tracks of length smaller than m), do the following
  - Calculate the residual and the Jacobian matrix for each an every feature to be processed.
  - Perform the Mahalanobis gating test
  - Using all features that passed the gating test, form the residual vector and the Jacobian matrix
- For features that are included in the state vector, compute the residuals and measurement Jacobian matrices, and form the residual $\tilde{z_k}$ and matrix H_k
- Update the state vector and covariance matrix, using $\tilde{z_k}$ and matrix H_k
- Initialize features tracked in all m images of the sliding window.

State Management:

- Remove SLAM features that are no longer tracked, and change the anchor pose for SLAM features anchored at the oldest pose.

- Remove the oldest camera pose from the state vector. If no feature is currently tracked for more than m_o poses (with $m_o < m - 1$), remove the oldest $m - m_o$ poses.

C. Analysis of the ***Time Complexity*** for the modified MSCKF as implemented in [3]

- We have features, with feature track lengths l_i, $i = 1....n$
- The "computation of the residual z & the Jacobian Matrix(H) takes $O(\sum_{i=1}^{n} l_i^3)$ operations
- Mahalanobis test, (calculated for each frame), for selecting the residuals take $O(\sum_{i=1}^{n} l_i^3)$ operations
- Similarly the QR Factorization (of H), to solve the linear least squares problem & then calculate $\tilde{z}^r$ for updates take $O(\sum_{i=1}^{n} l_i^3)$ operations
- So all of the above listed steps depends linearly w.r.t the no. of features and also with the track length cubed.
- The last step consists of calculating the Kalman Gain & updating the covariance matrix. The time complexity is $O(\frac{r^3}{6} + r(15+6m)^2)$. Here r denotes the no. of rows in H^r(no. of independent constraints for the camera poses) and it can be derived by using $r = 2(l_1 + l_2 + l_3) - 7$, where $(l_1 + l_2 + l_3)$ denotes 3 longest feature tracks. $(15 + 6m)$ denotes the size of the covariance matrix.

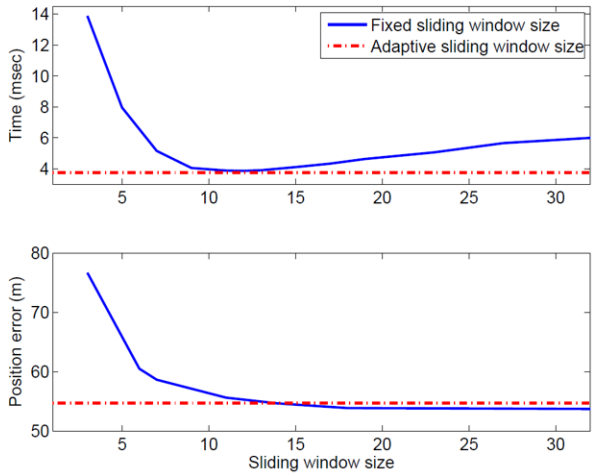

Fig. 1. Monte-Carlo simulation results: Timing performance and RMS position accuracy of the hybrid filter, for changing values of m. Timing measured on a laptop computer. [3]Img src.

IV. Evaluation

A. Evaluation based on Euroc V1_01_easy ($58m$ long)

Modules	Min	Avg	Max
Tracking	2.54	4	11.19
Propagation	0.1	0.25	1.41
MSCKF Update	0	1.4	22.45
SLAM Update	0	6.19	14.6
SLAM delayed	0	1.2	22.3
re-tri & marg	0.2	0.84	4.49
Total	4.32	14	45.59

Time taken by different modules at incoming frequency of images 20Hz

B. Evaluation based on Euroc V1_01_easy (lab VM) ($58m$ long)

Modules	Min	Avg	Max
Tracking	2.29	7	32.98
Propagation	0	0.21	2.86
MSCKF Update	0	1.156	16.4
SLAM Update	0	3.55	11.78
SLAM delayed	0	1.7	27.92
re-tri & marg	0.2	0.6	4.44
Total	3.9	14.37	50.61

Time taken by different modules at incoming frequency of images 100Hz
ATE : 0.055

Modules	Min	Avg	Max
Tracking	2.28	4	20
Propagation	0	0.18	1.52
MSCKF Update	0	1.2	16.93
SLAM Update	0	5.3	9.5
SLAM delayed	0	1.01	21
re-tri & marg	0.2	0.56	1.84
Total	4.96	12.4	37.07

Time taken by different modules at incoming frequency of images 20Hz
ATE : 0.046

C. Evaluation based on TUM_VI room1 (lab VM) ($147m$ long)

Modules	Min	Avg	Max
Tracking	2.48	5.8	35.83
Propagation	0	0.164	2.12
MSCKF Update	0	1.1	12.91
SLAM Update	0	2.054	10.65
SLAM delayed	0	1.47	14.91
re-tri & marg	0.27	0.6	6.85
Total	3.66	11.26	44.03

Time taken by different modules at incoming frequency of images 100Hz
ATE : 0.066

Modules	Min	Avg	Max
Tracking	2.28	4.39	20.85
Propagation	0	0.1557	3.25
MSCKF Update	0	1.07	14.67
SLAM Update	0	2.65	8.74
SLAM delayed	0	1.42	26.69
re-tri & marg	0.2	0.524	1.71
Total	4.96	10.23	32.65

Time taken by different modules at incoming frequency of images 20Hz
ATE : 0.06

V. Conclusion and Future Work

From our evaluation results we can conclude that the OpenVINS is performing fairly when it's fed with incoming image streams with original frequency i.e., $20hz$. So we have tried to increase the incoming image frequency to the point where it breaks like $100hz$, still its working well enough with a slight drop in ATE. In my present experimentation with these 2 datasets (EUROC & TUM_VI) its seems like OPENVINS is a fairly competitive model w.r.t. other full-SLAM models like the ORBSLAM. But I am yet to see the performance of OPENVINS in longer datasets like the KITTI, which has trajectory lengths in kms.

From my experimentation I have checked that the OPENVINS doesn't face many of the problems that the other full-slam system does, like concurrency. This is due to the fact that this system doesn't implement many of the steps like the creation of the map, loop closure, global bundle adjustments which

are costly to execute. Its a fairly simple architecture with the Hybrid-MSCKF being the backbone. And one of the most costly steps in the whole system is the Tracking, other modules are more or less either the feature update in the state variable and propagating those features in the present timeframe.

REFERENCES

[1] Geneva, Patrick, et al. "Openvins: A research platform for visual-inertial estimation." 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020.

[2] Mourikis, Anastasios I., and Stergios I. Roumeliotis. "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation." ICRA. Vol. 2. 2007.

[3] Li, Mingyang, and Anastasios I. Mourikis. "Optimization-based estimator design for vision-aided inertial navigation." Robotics: Science and Systems. Germany: Berlin, 2013.