From Timing Variations to Performance Degradation: Understanding and Mitigating the Impact of Software Execution Timing in SLAM

Ao Li, Han Liu, Jinwen Wang, and Ning Zhang

***Abstract*— Timing is an important property for robotic systems that continuously interact with our physical world. Variation in program execution time caused by limited computational resources or system resource contention can lead to significant impact on algorithmic result accuracy. Even though recent work has found Simultaneous Localization And Mapping (SLAM) to be timing-sensitive, little exists in understanding the interactions between the timing variations in SLAM systems and the corresponding degradation. In this paper we conduct a systematic analysis of nine state-of-the-art SLAM systems and dissect the root causes of their degradation. We discovered that timing-induced errors are generated either from delayed execution in certain critical tasks, or from desynchronization in sensor fusion. Based on the insights from our analysis, we propose a solution that combines selective fusion on data in the front end and temporal budget optimization on bundle adjustment in the backend to mitigate the impacts of unexpected timing variation adaptively. Experimental results show that our proposed method makes it possible to migrate expensive algorithms to low-cost platforms without laborious tuning, while making the SLAM system robust against the effects of abnormal timing.**

I. Introduction

As a widely used algorithm in robotic systems, SLAM covers a broad domain of applications, e.g., cartography, autonomous delivery, home services, and augmented reality. Consequently, the platforms SLAM runs on are equally diverse in terms of computational resources, ranging from high-end platforms, such as self-driving cars, to low-cost ones, such as robotic bees [1]. Maintaining system performance on such a diverse set of platforms remains a key research challenge for SLAM.

Existing approaches to make SLAM adaptive follow the idea of input reduction, where a subset of the original information is selected for processing [2], [3], [4], [5], [6]. This selection reduces both front-end feature points and back-end frames using a strategy obtained through predefined metrics [7], [8], [9], [10], [11]. However, these methods rely on the manual tuning of parameters for each target platform, making it time-consuming to deploy.

Recent studies of localization drifts introduced by timing delay have connected timing characteristics to performance [12]. In this paper, we leverage system timing feedback from software/hardware platforms to enable adaptive input reduction to achieve better performance. Intuitively, input reduction is only needed to the degree that it preserves stable execution timing. To guarantee that reduction remains only as conservative as required by the real-time constraints of the system, we need to address two challenges: First, we characterize the manifestation of timing issues in performance. Second, we leverage these insights to strike better performance via adaptive input reduction. *Understanding Timing Issues:* Existing research efforts on timing analysis generally focus on two aspects: the temporal offsets between sensors [13], [14], [15], [16] and the processing latency of algorithms [17], [18], [19]. Little attention is paid to the internal causal relationships between timing and performance, partly because timing problems are not always observable in multi-threaded systems. The non-determinism and architecture-based performance-boosting also exacerbate the challenge in timing analysis. In this paper, we conduct a comprehensive study of these timing problems over several state-of-the-art SLAM algorithms, including both visual-based and LiDAR-based SLAMs [20], [21], [22], [23], [24], [25], [26], [27], [28], [29]. We found the root causes to be desynchronization and delay. Desynchronization often leads

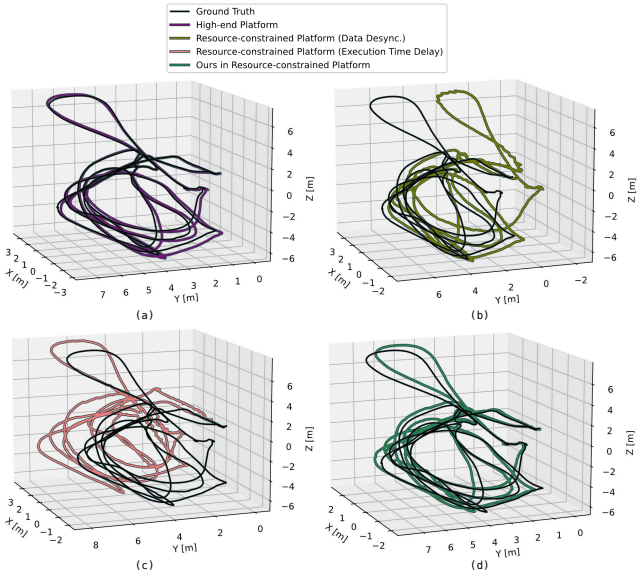


Fig. 1. (a) Localization of ORB-SLAM on a high-end PC (AMD Ryzen 9 3900 12-Core processor with 128G RAM). (b) and (c) illustrate localization results on a resource-constrained platform (Intel i7-7700HQ with 8GB RAM). In particular, in (b) the result is majorly impacted by desynchronization of threads while using data, and in (c) it is majorly impacted by the execution time delay of bundle adjustment. (d) illustrates the result of our proposed method, which is designed to alleviate degradation from timing problems.

Ao Li, Han Liu, Jinwen Wang, and Ning Zhang {ao, h.liu1, jinwen.wang, zhang.ning}@wustl.edu are with Department of Computer Science and Engineering, Washington University in St. Louis, MO, USA. This work is supported in part by US National Science Foundation under grants CNS-1837519, CNS-1916926 and CNS-2038995, CNS-2154930, and by Army Research Office under contract W911NF-18-1-0305.

the SLAM tasks to select temporally misaligned data in sensor fusion, while delay often causes large task jitters, degrading the quality of map states.

Timing-aware Input Reduction: Leveraging these insights, we propose a mitigation design that explicitly considers the time dimension. This method contains an inference model and an efficient online searching algorithm which enable a SLAM system to find the best strategy according to its real-time status. Concretely, it selectively fuses sensor data at the front-end to avoid desynchronization problems and predicts the timing budget for the back-end to stabilize the execution of bundle association in real-time. Our method is complementary to existing works that focus on computation reduction metrics. The experimental results show that our method is able to mitigate harmful timing variation while maintaining sufficient accuracy. Moreover, it can serve as an adapter that enables migration of state-of-the-art SLAMs to resource-limited platforms without laborious tuning (as illustrated in Figure 1).

Through this work, we hope that our timing analysis results and methodology serve as a new tool for the development of robust SLAM algorithms, and that our mitigation method facilitates the smoother migration of computationally-intensive algorithms to platforms with limited resources, while improving its robustness against timing variations. We have also made our source code available for the community [30].

II. Related work

A. Temporal Offset Estimation

The temporal offset, including communication delays or time synchronization errors, between the onboard hardware has long been considered as a non-trivial factor of performance degradation of SLAM. Prior works focus on estimating fixed offsets [31], [15], [32] based on the characteristics of data from different sensors. Varying offsets can be treated as unknown variables and modeled as Gaussian noise [16]. The timing problems discussed in this paper are different from this hardware-induced delay. While the uncertainty due to hardware is typically modeled stochastically, such models often cannot predict timing variations induced by the operating system. For example, the processing latency of a single task can surge dramatically due to a dynamic environment or interference from other tasks. Existing efforts to analyze the impact of system-induced timing consider the latency of visual matching as a factor in localization drifts [12]. However, due to the complex implementations of modern SLAM systems, timing problems are not merely limited to IMU forward propagation, which was the focus of [12]. Instead, it happens at almost every critical component in SLAM. To ensure a robust SLAM system, it is necessary to perform comprehensive analysis of component timing behaviors.

B. Computation Reduction

Since many SLAM algorithms make use of complex mathematical computations, their real-time performance cannot be met if they are deployed on resource-constrained platforms. To address this problem, existing works mainly focus on reducing computation by executing over a subset of the data. The design space includes feature selection [2], [3], [4] aiming to reduce the computation of matching and subgraph selection [5], [6] aiming to reduce the computation of bundle adjustment. By designing effective metrics, such as entropy [7], covisibility [8], [9], covariance [10], [11], minimum eigenvalue [4], and log-determinant [5], these methods can achieve faster computation in a limited timing budget without severely sacrificing accuracy. However, these methods require manual tuning for each target platform, which can be a key limiting factor for wider adoption. Moreover, because available timing budgets dynamically change with system overhead, online budget prediction is necessary to handle unexpected timing. Motivated by those problems, we propose an adaptive method which adjusts SLAM parameters online. The aforementioned work on subset selection metrics are orthogonal to our method and can be utilized to further improve its performance.

III. Dissecting Timing Impacts on SLAMs

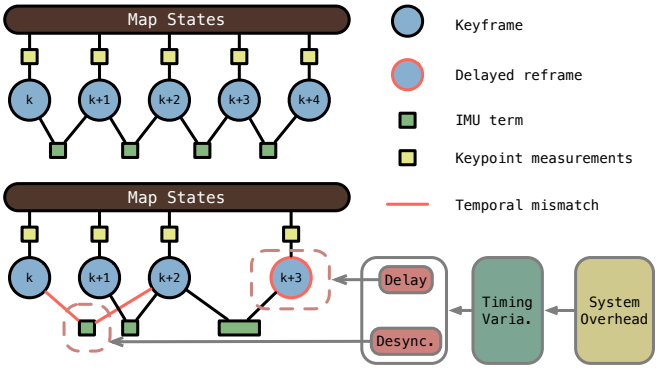


Fig. 2. The impacts of timing variations. Overheads from stressed computational resources induce timing variations during execution, which we categorize as delay and desynchronization. Here, we illustrate keyframe delay and the desynchronization of frames and inertial data.

A. Timing Variations

Performance anomalies are frequently observed problems in operating systems, usually accompanied by unexpected execution timings. The variations of workloads [33], the interference between co-running threads [34], and semantic bugs [35] are the main contributors to timing variations. We attribute timing variations of SLAM systems to two root causes. First, expensive algorithms deployed in resource-constrained platforms may cause concurrently executing tasks to contend for shared resources. Second, a dynamic environment and adaptive software behavior can induce significant execution timing variations.

To better understand the interactions between timing variation and performance degradation, we experimentally analyzed several well known SLAM systems, including visual-based [20], [21], [22], [23], [24], [25], [26] and LiDAR-based [27], [28], [29] (more results are shown in Section V). By dissecting the root causes of timing-induced performance

TABLE I

MANIFESTATIONS OF DESYNCHRONIZATION AND DELAY UNDER DIFFERENT SYSTEM OVERHEADS

Overhead	Desync.		Delay		Total RPE
	Prob.	RPE	Exe. Time	RPE	
None	0.20%	0.005m	33ms	0.008m	0.021m
Low	5.60%	0.029m	44ms	0.035m	0.094m
Medium	16%	0.13m	62ms	0.12m	0.28m
High	19%	0.22m	81ms	0.25m	–

Overhead is characterized by CPU utilization. "None" indicates average CPU usage under 10%. Low, Medium, and High indicate approximately 30%, 60%, and 90% utilization respectively.

degradation, we systematize the two contributing factors as desynchronization and delay, as illustrated in Figure 2. Since SLAM covers a variety of architectural and algorithmic implementations, the manifestations of timing problems are diverse. To better illustrate the timing impacts, we chose the keyframe-based visual-inertial SLAM as our example of elaboration.

B. Desynchronization in Sensor Fusion

As modern SLAM systems generally take multiple sensor inputs to achieve a robust perception, temporal alignment is crucial while fusing data from different sensors. However, when there are unexpected timing variations among different threads, data streams of two different sensors are prone to temporal misalignment. For example, in ROS-based systems, the sensor data are read via callback threads and then stored in buffers. If system overhead causes a delay in scheduling the IMU callback thread, the inertial data in buffer will be desynchronized with images. Even if the camera and IMU are perfectly synchronized in hardware, software and operating system issues, such as priority inversion and concurrency problems, can still cause their data to temporally misalign. Table I presents the probability of desynchronization under different system overheads and their corresponding relative pose errors. High system overhead often occurs on resource-constrained platforms, and occasionally occurs as corner cases on powerful platforms.

Desynchronization can occur anywhere that sensor data are fused, including tracking, initialization, and pose extrapolation components, etc. For example, visual-inertial SLAM performs tracking in the front-end to estimate pose at frame rate. It is often formulated as a MAP (maximum a posteriori estimation) problem as follows:

$$\theta^* = \underset{\theta}{\operatorname{argmin}} \left(\sum_k \mathbf{r}_{\mathrm{c}}(k,j) + \mathbf{r}_{\mathrm{I}}(i,j) \right) , \tag{1}$$

where $\theta = \{\mathbf{R}, \mathbf{t}\}$, $\mathbf{r}_c(k,j)$ is the residual of the visual feature of frame j with the matched points k in the map, and $\mathbf{r}_I$ is the residual of inertial data. Since IMU data typically arrives at a higher rate, a stream of inertial data $\{t_i, ..., t_j\}$ is pre-integrated based on the kinematics equation:

$$\begin{aligned} \mathbf{p}_{m+1} &= \mathbf{p}_m + \mathbf{v}_m \Delta t - \frac{1}{2}\mathbf{g}\Delta t^2 + \mathbf{R}_m \hat{\boldsymbol{\alpha}}^m_{m+1} , \\ \mathbf{v}_{m+1} &= \mathbf{v}_m - \mathbf{g}\Delta t + \mathbf{R}_m \hat{\boldsymbol{\beta}}^m_{m+1} , \end{aligned} \tag{2}$$

where $\mathbf{p}, \mathbf{v}$ are position, velocity respectively and $\hat{\boldsymbol{\alpha}}^m_{m+1}, \hat{\boldsymbol{\beta}}^m_{m+1}, \hat{\mathbf{q}}^m_{m+1}$ are the pre-integration terms between inertial frames m and $m+1$.

For joint optimization, the camera and IMU have to be temporally aligned. However, as discussed previously, high system overhead can cause misalignment: Assume that two consecutive images have respective timestamps t_i and t_j. If desynchronization occurs, $\mathbf{r}_I$ is instead determined by integration over $\{\tilde{t}_i, ..., \tilde{t}_j\}$, where $\tilde{t} = t + \Delta t^a$, with Δt^a as the (unknown) time variation. This leads to data inconsistency in Equa. 1. [16] proposes to model Δt^a with a covariance matrix $\mathbf{\Sigma}$, but this does not eliminate the error. Furthermore, in many SLAM systems, the initial value of θ in Equa. 1 is often given by the IMU pre-integration Equa. 2 such that a false initial value causes θ^* in Equa. 1 to be prone to fall into a local minimum.

C. Execution Time Delay

Most SLAM systems have multithreading implementations; with strong inter-thread data dependencies, execution delay in any single thread can introduce disruption to the data timeliness in other threads, leading to degradation of localization accuracy. Though errors induced by delay can happen in many places, we primarily focus on the delay in tracking and bundle adjustment (BA) because these errors can accumulate over time, which have longer-lasting impacts. Other errors, such as the drifts in IMU forward propagation [12], can be mitigated once new visual matching results are produced.

Once tracking cannot meet the camera frame rate, additional images will be dropped, causing two consecutive frames to have fewer common features to associate. Similarly, untimely BA leads to keyframes being dropped such that the landmarks in the map are sparser. Both of these cases impact the quality of the map, which in turn has negative effects on all future results. Since image dropping ultimately causes BA to drop keyframes, we consider this no differently than other forms of keyframe dropping, and instead focus on degradation caused by delay in BA in general. The problem is commonly formulated as follows:

$$\underset{\mathbf{x_c}, \mathbf{x_p}}{\arg\min} \sum_{i,j} \left\| \mathbf{r}\left(\mathbf{x_c}(i), \mathbf{x_p}(j)\right) \right\|^2_{\mathbf{\Sigma_{ij}}} , \tag{3}$$

where the $\mathbf{x_c}$ is the vector of camera states, $\mathbf{x_p}$ is the vector of map states, and $\mathbf{\Sigma_{ij}}$ is the covariance matrix.

Generally, when more information is supplied to Equation 3, it produces more accurate results. However, due to the complexity of the BA algorithm, the timing budget might not be sufficient to produce a full solution; in this case, working on a smaller-scale subproblem of the original BA may be more cost-effective. Prior work suggests the use of metrics such as entropy [7], covisibility [8], [9], and covariance [10],

[11] to select a subset of available keyframes to provide to BA. The selection is performed according to:

$$\max_{\mathbf{S}\subseteq\{0,1,\ldots,n\},|\mathbf{S}|=k} \mathbf{M}([\mathbf{\Lambda}(\mathbf{S})])\,, \tag{4}$$

where $\mathbf{M}$ is the metric function and $\mathbf{S}$ is the selected subset of keyframe indices. Under heavy computational constraints, too many keyframes are discarded such that consecutive keyframes in the selected subset are spaced too far in time. This inevitably leads to a loss of information (e.g., fewer constraints in the factor graph) in the produced map. Where additional keyframe dropping occurs due to unexpected execution timing, BA executes over a subset of keyframe indices $\mathbf{S}^- \subset \mathbf{S}$. Invariably, $\mathbf{M}([\mathbf{\Lambda}(\mathbf{S}^-)]) < \mathbf{M}([\mathbf{\Lambda}(\mathbf{S})])$, implying further degradation of accuracy. Lack of information (i.e., a priori knowledge of dropped keyframes) supplied to the optimization problem in Equa. 4 therefore results in an outcome significantly worse than if selection of the BA subproblem had been able to predict this behavior and remove dropped keyframes from the searched indices.

IV. Mitigation of Timing Impacts

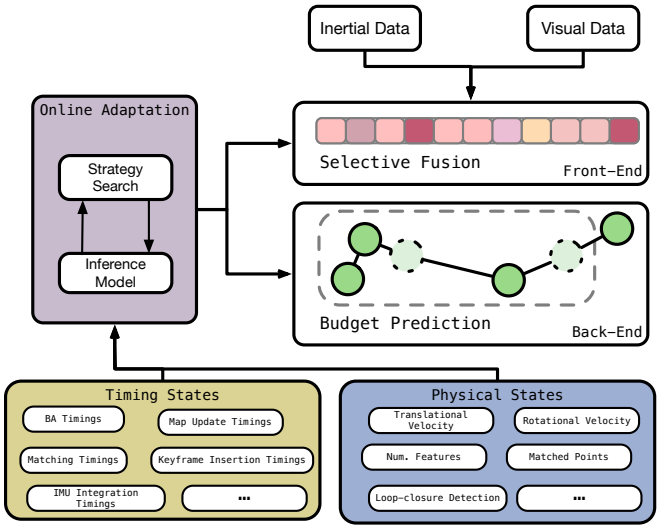


Fig. 3. Overview of the proposed method.

Based on the analysis above, we propose a mitigation method to alleviate the timing impacts. The mitigation method has two objectives: selecting proper sensor data at the front-end to avoid desynchronization problems and predicting the timing budget for the back-end to stabilize the execution timings of BA, which are illustrated in Figure 3.

Selective Fusion: The candidates of selection include the raw data from the camera and IMU. This mechanism is based on two considerations. First, when the system overhead is high, the execution time will increase such that tracking cannot keep up with the frequency of the camera. Consequently, dropping some frames is required to meet real-time performance. Second, the high system overhead introduces misalignment, out-of-order data, or loss of data (including both images and IMUs), which can be uniformly treated as desynchronization. In this case, better-matched data will be prioritized for processing.

Timing Budget Prediction: Generally, the timing variations of the back-end are greater than that of the front-end. This is because the back-end is composed of more optimization-related computations, which are more sensitive to resource contention (e.g., CPU cache contention). Large fluctuations of back-end start time will cause keyframe insertion time jitter. To avoid this problem, our method adaptively manages the computation time of the back-end. Concretely, it predicts the timing budget of the back-end, and uses this to select the number of keyframes to supply the BA, which is the major back-end task.

A. Overview

Our mitigation is an online adaptation method (illustrated in Figure 3) that adjusts the strategy according to the current system state. At run-time, our method reads timing states and physical states from the SLAM system and feeds them into the strategy search. The strategy search generates strategies and inputs them into the inference model. By using the current system status and the input strategy, the inference model predicts future performance. Based on feedback from the model, the searching algorithm changes the candidate strategy. Through iterative search over different strategies, our method finds the most effective one and applies it to the next execution loop of the SLAM system.

B. Inference Model

Input and Output: The input of the model includes the strategy and the system states. The strategy is modeled as a tuple $< N, C, I >$, where N is the allowed number of frames to perform BA, and $C = \{c_1, .., c_i\}$ and $I = \{i_1, .., i_i\}$ are arrays composed of 1 or 0, which indicates if the corresponding image frames or IMU frames should be dropped or not. We consider two types of system states: *timing states* and *physical states*. *Timing states* contain the execution timings of critical components (denoted as E) – such as feature tracking, keyframe insertion, bundle adjustment, etc. – which are primary targets we aim to stabilize. *Physical states* are composed of the physical variants (denoted as P) in software, such as velocity, number of queued frames in BA, number of features, etc. These states can negatively amplify the impacts of timings, as explained in Section V. The model prediction output is the gain on localization accuracy when the input strategy is enacted, and is formulated as A_{no}/A_s. (For simplicity, we measure A_{no} and A_s using relative translational error without and with strategy.)

Model Selection: As navigation accuracy may have a highly nonlinear dependence on system state, a reinforcement learning (RL) model, which has proven effective in mitigating unexpected timings in web services [33], is one potential solution. However, RL is too computationally expensive to be adopted for low-cost platforms. Instead, we consider lightweight models, evaluating random forest regression, decision tree regression, Gaussian process regression, and

support vector machine (SVM) regression on ten-fold cross-validation datasets. We select the random forest regression for our model since it produced the best R^2 scores over our validation datasets. This is not surprising: as the input feature size is large (above 20), a single integrated model (decision tree, SVM, etc.) may be over-fitted on such a complex feature space. However, random forest regression may split the features on different trees and then combine them to produce better-fitted results.

C. Online Strategy Search

The goal of searching is to find the best combination of N, C, and I which maximize the navigation accuracy under the given states. We denote our objective function as:

$$\{N, C, I\}^* = \arg\min_{N,C,I} T(N, C, I, E_1, P_1 ..., E_n, P_m). \quad (5)$$

An efficient search strategy is important as excessive queries to the model will induce extra overhead, exacerbating time delay. Thus, instead of using grid search or random search, we develop an efficient search strategy based on the analysis above. The strategy is a trade-off problem between processing more data and achieving better efficiency. More data will produce a more accurate result but excessive frames will exacerbate timing problems. Hence, the navigation accuracy will initially increase and then decrease with the increase of N, N_C, and N_I, where N_C and N_I are the number of ones in C and I respectively. We design a greedy search strategy to find the best N, N_C, N_I (pseudo-code is given in Alg.1). Based on the assumption that execution timing often depends on the number of frames and inertial data points, the strategy actually predicts timing budgets for the SLAM system. Under the constraint of budgets (N_C and N_I), we select the best temporally-aligned images and inertial data by marking them with ones in the array C and I. Our experiment shows that the searching algorithm can find the best N and N_C in dozens of queries, costing less than $10\mu s$.

Algorithm 1 Online Strategy Search

Input: Strategy bounds x_{max}, x_{min}, States S, Step Δ
Output: Strategy x

procedure ONLINESEARCH(x_{max},x_{min}, S, Δ)
  $H_x \leftarrow x_{max}$, $L_x \leftarrow x_{min}$
  Init variable $x, i, step$
  while $step > threshold$ **do**
    $x_i \leftarrow x_{i-1} + step$
    $T_i \leftarrow Model(x_i, S)$
    if $T_i \geq T_{i-1}$ **then**
      $x_i \leftarrow L_x + step$
    else
      $L_x \leftarrow x_{i-2}, H_x \leftarrow x_i$
      $step \leftarrow step - \Delta$
    end if
  end while
  $x* = H_x - step$
  return x^*
end procedure

V. Experimental Results

Our experiments are divided into two parts: the first presents the analysis on the interactions between timing variations and performance degradation, and the second part demonstrates the effectiveness of the proposed method in mitigating timing impacts.

TABLE II
Experiment Platforms

Sys. Overh.	None	Low	Medium	High
Platform	AMD Ry. 9 3900 (128GB)	AMD Ry. 5600G (16GB)	Intel i7-7700HQ (8GB)	Nvidia Jeston Nano (4GB)

A. Experimental Setup

Implementation: We conducted timing sensitivity on nine well-known SLAM systems, including visual-based, such as *ORB-SLAM* [20], [21], *VINS-Fusion* [22], *DSO* [23], *SVO* [24], *OKVIS* [25], and *MSCKF* [26]; as well as LiDAR-based, such as *Cartographer* [28], *LOAM* [27], and *AMCL* [29]. The aforementioned timing problems exist on all of these SLAM systems; for brevity, only the key results are highlighted. To demonstrate the effectiveness of the proposed method, we implemented our mitigation method on top of ORB-SLAM. We chose ORB-SLAM because it is a well-engineered system with good real-time performance. For testing data, we used the EuRoC [36] for the visual-based SLAM and utilized simulation in Gazebo for LiDAR-based SLAM. To train our inference model, we manually injected workloads into the system to achieve different overheads. To create the stochastic system overhead, we used the Linux tool commands *tc*, *stress-ng*, and *pmbw* to inject network, CPU, and memory workloads respectively. Some results appear in our prior work [37] and in github [30], and are therefore omitted here.

System Setup: We run the entire software stack on three different platforms: AMD Ryzen 5 5600G with 16GB RAM, Intel Core i7-7700HQ with 8GB RAM, and Nvidia Jeston Nano with 4GB RAM. The distinct computational resources on these three platforms induce different extents of performance degradation when running SLAMs. We refer to them as platforms with **`low`** overhead, **`medium`** overhead, and **`high`** overhead as listed in Table II. We also refer to the localization accuracy not impacted by any timing problems as the baseline performance. The baseline is obtained on a powerful PC with an AMD Ryzen 9 3900 12-Core processor and 128G RAM. We denote the baseline as **`none`** overhead.

B. Accuracy Degradation Caused by Timing Variations

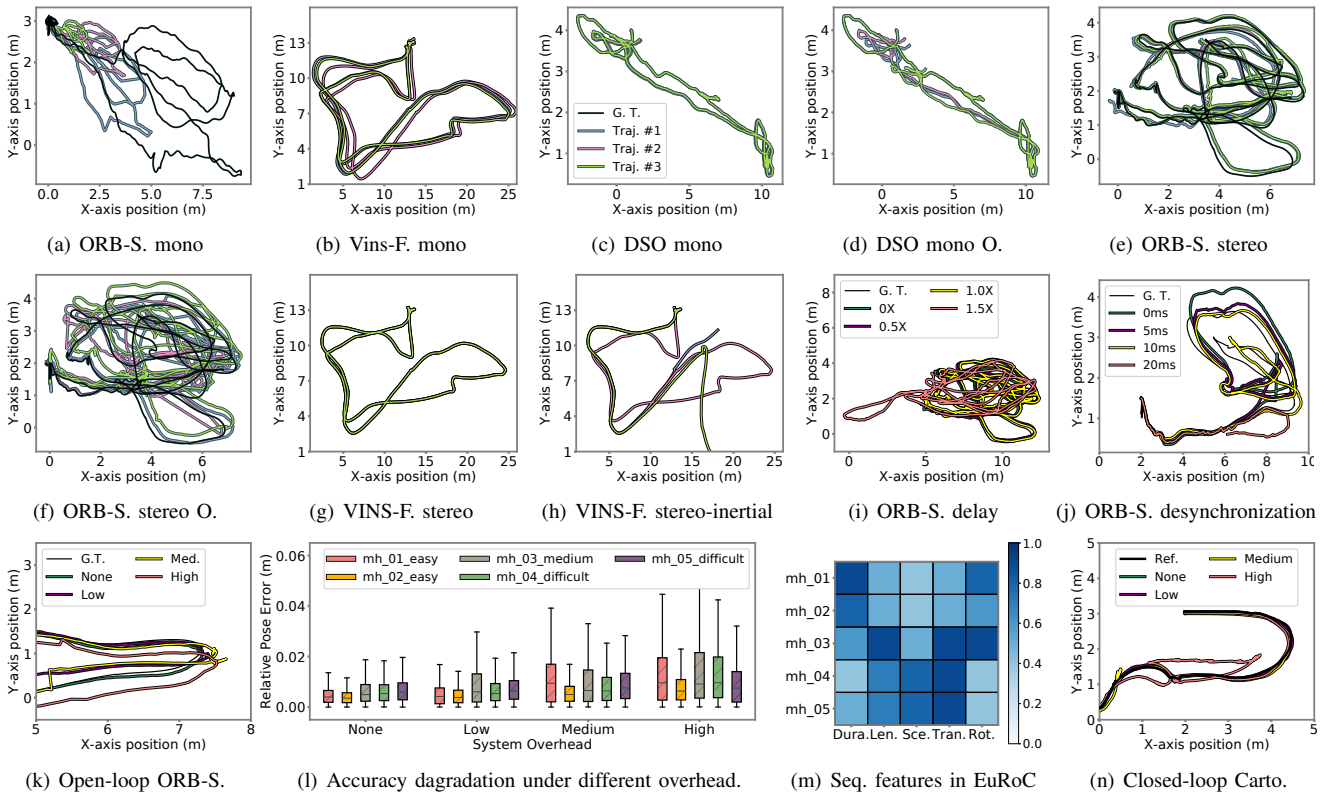
Timing can cause software misbehavior and degradation in many components. To find the factors, we stress the system and quantitatively compare the results. To avoid randomness, each set was run 10 times. Selected results are shown in Figure 4. (4(a)-4(h) share the same legend.)

TABLE III

[AVERAGE/MAXIMUM] OF RPE AND ATE WITH DIFFERENT EXECUTION TIME DELAYS IN EACH MAIN THREAD IN ORB-SLAM

Delay	Tracking				Mapping				Loop Closing			
	RPE (m)		ATE (m)		RPE (m)		ATE (m)		RPE (m)		ATE (m)	
0%	0.008	0.34	0.09	0.54	0.008	0.31	0.08	0.61	0.009	0.31	0.07	0.64
50%	0.008	0.31	0.13	0.48	0.009	0.55	0.16	0.74	0.009	0.31	0.12	0.75
100%	0.011	0.34	0.17	0.64	0.012	0.79	0.29	0.79	0.007	0.68	0.19	1.2
150%	0.021	0.97	0.28	1.14	–		–		0.013	0.60	0.22	1.5

The experimental results are generated on AMD Ryzen 5600G with 16GB RAM. The original tracking time is 23ms on average.



(a) ORB-S. mono (b) Vins-F. mono (c) DSO mono (d) DSO mono O. (e) ORB-S. stereo

(f) ORB-S. stereo O. (g) VINS-F. stereo (h) VINS-F. stereo-inertial (i) ORB-S. delay (j) ORB-S. desynchronization

(k) Open-loop ORB-S. (l) Accuracy dagradation under different overhead. (m) Seq. features in EuRoC (n) Closed-loop Carto.

Fig. 4. Timing impacts on different SLAMs. 4(a), 4(b), and 4(c) present respectively the trajectories of ORB-SLAM2, VINS-Fuison, and DSO in monocular SLAM mode; 4(d) illustrate the trajectories of Monocular DSO with medium level overhead; 4(e) and 4(f) are trajectories of ORB-SLAM2 with and without overhead in stereo SLAM mode; 4(g) and 4(h) show respectively the trajectories of stereo and stereo-inertial SLAM VINS-Fusion under high overhead. 4(i) and 4(j) present the performance of visual-inertial ORB-SLAM under different average delays and under different temporal offsets between visual and inertial data; 4(k) is the zoomed fragments of trajectories of ORB-SLAM under different overhead; 4(l) contains the degradation in different sequences of the EuRoC dataset; 4(m) shows the characteristics of sequences in EuRoC; and 4(n) illustrates the performance degradation of Cartographer under different overheads in our simulated environment.

Timing Impacts on Different Modes: From the results, we found that monocular SLAMs are the most vulnerable to timing variations. Figures 4(a) and 4(b) contain three runs of monocular ORB and VINS under **`low`** overhead. Even with minor timing variations, they are prone to have falsely estimated depth scales. This is because the initialization is sensitive to timing.

Execution Time Delay in Multiple Threading: Figures 4(c), 4(d) and Figures 4(e), 4(f) show trajectories produced by DSO and ORB SLAM under no system overhead and low system overhead respectively. The initialization results are ensured to be identical in each run. Since there is no sensor fusion and no image frame is dropped in the processing, interleaved multithreading is the only cause of performance degradation, as it interferes with software execution logic. For example, in ORB-SLAM, untimely tracking can interrupt the execution of BA, causing large jitters. To further investigate how execution delay of each thread may negatively impact the final result, we manipulate the execution time of each main thread in ORB-SLAM: *tracking*, *mapping*, and *loop_closing*. The results are shown in Table III, where we observe that the delay of *mapping* is most critical. This is because the timings of BA directly impact the frequency of keyframe insertion. Moreover, by inspecting the internal states, we found that the delay in each thread ultimately impacts the frequency of keyframe insertion (or map update).

Desynchronization in Sensor Fusion: Figures 4(g) and 4(h) show the results of stereo-only mode and stereo-inertial mode of VINS-Fusion [22] under high system overhead. Stereo

VINS-Fusion processes the image data frame by frame, which means there was no frame dropping. Initialization is also ensured to be identical in each run. However, final trajectories are quite different in the two sets. By tracing the logs, we found that high system overhead causes the inertial data to be temporally mismatched with frames, which consequently causes false matching results because it is highly sensitive to the initial value given by inertial data. Inertial data is introduced into the SLAM system for more accurate and robust estimation, but it could be harmful if the system overhead is high. To quantify the impacts of different extents of desynchronization, we manipulate the temporal offset between images and IMU data from $0ms$ to $20ms$. This manipulated desynchronization is realistic because the temporal offset caused by high system overhead is usually greater than $50ms$. As shown in Figure 4(j), ORB-SLAM is likely to fail once the temporal offset is more than $5ms$.

Timing Impacts on Different Scenarios: The impacts of scenarios is shown in Figures 4(l) and 4(k). It can be observed that timing is more sensitive in the data sequences for which the trajectory is more curved, such as *MH_03*. Additionally, other characteristics (Figure 4(m)) equally exacerbate the degradation. Moreover, simulation-based testing on Cartographer [28] in Figure 4(n) shows that a vehicle is more likely to be affected by timing variation if we put the controller in the loop. These are the reasons that we consider the physical states while designing our inference model.

C. Mitigation of the Proposed Method

The evaluation of the proposed method is from two perspectives: its capability to adapt on resource-constrained platforms and to mitigate harmful timings.

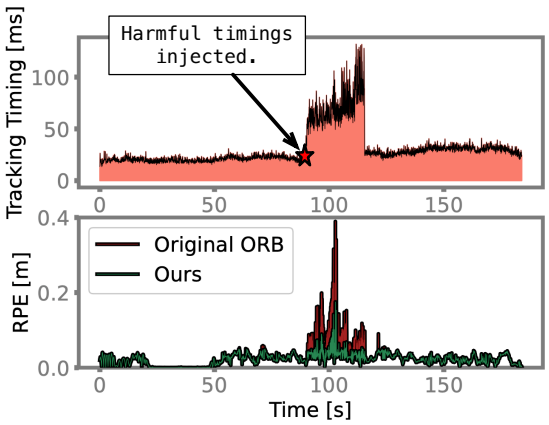


Fig. 5. Mitigation of harmful execution timings.

Resource-constrained Platforms: We conducted experiments with ORB-SLAM on the four platforms listed in Table II. For each platform except the baseline (**none** overhead), we conducted three sets of experiments: the original ORB-SLAM version, the modified version with manually tuned parameters, and the version integrated with our method. The results are shown in Table IV. We observe that on the resource-constrained platforms – Intel Core i7-7700HQ and Nvidia Jeston Nano – the original ORB-SLAM is very fragile. In particular, the Jetson Nano is unable to maintain consistent tracking for most sequences. We then manually tuned the parameters, such as frame dropping rate, iteration limits, and the number of keyframes in BA. The tuned version performed better since it was able to complete some trajectories without becoming lost. However, this manual-tuning method relies on expert knowledge and is labor-intensive. Conversely, the proposed method (which requires no manual tuning) outperforms the tuned version on low-end platforms. On these platforms, resource usage tends to reach the limit, which causes large timing variations; in such cases, our method provides the necessary dynamic adjustment.

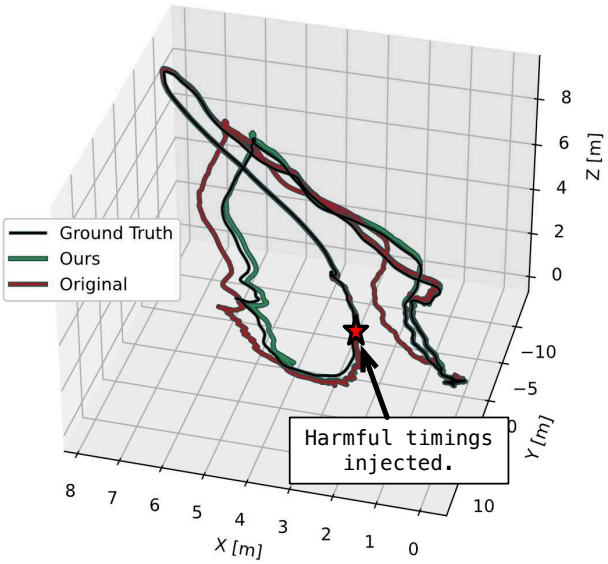


Fig. 6. Trajectories in normal and harmful timing conditions.

Harmful Timings: As discussed previously, due to the dynamic environment, the SLAM systems may suffer extremely abnormal software timing behaviors during a short period. Handling this harmful timing is critical, as otherwise the SLAM system is prone to fail. We investigate the case of occasional harmful timing on the resource-rich platform. Via manually injected system overhead in run-time for a period, we can observe that the execution time of tracking increased significantly in Figure 5. In the bottom figure, the original ORB-SLAM had relative pose errors of more than 0.3 m during this period, which results in an obvious deviation from the ground truth as in Figure 6. Instead, with our mitigation method, the SLAM can dynamically adjust its strategy. Even with similar execution timings of `tracking`, our methods can keep the RPE under 0.1 m, allowing the SLAM system to keep consistent tracking.

VI. Conclusion

This work systematically analyzed the interaction between software execution timings and performance degradation on SLAM systems. Through investigation on nine widely-used state-of-the-art SLAM systems, we found the impacts of timing often result in execution delay in multi-threading and desynchronization in multi-modal sensor fusion. The root cause behind these timing issues in the SLAM systems

TABLE IV

RMSE [M] OF ORIGINAL ORB-S. & MANUALLY-TUNED ORB-S & THE ORB-S. PATCHED WITH OUR METHOD ON DIFFERENT PLATFORMS.

Platform	Overhead		MH_01	MH_02	MH_03	MH_04	MH_05	V1_01	V1_02	V1_03	V2_01	V2_02	V2_03
AMD Ryzen 3900 (128GB RAM)	None		0.0163	0.0233	0.0271	0.0464	0.083	0.0265	0.0299	0.0294	0.0169	0.0228	0.0335
AMD Ryzen 5600G (16GB RAM)	Low	Orignal	0.0171	0.0303	0.0348	0.0493	0.0841	0.023	0.0311	0.0291	0.0285	0.0231	0.0321
		Tuned	0.0314	0.0218	0.0352	0.0481	0.0833	0.0352	0.0266	0.134	0.0247	0.0219	0.0373
		Ours	0.0383	0.0237	**0.309**	**0.0433**	**0.0814**	**0.0203**	0.0284	0.084	**0.0238**	0.0244	0.0361
intel i7-7700HQ (8GB RAM)	Medium	Orignal	0.0376	0.0281	0.0412	0.119	0.133	0.0452	0.122	0.0480	0.0553	0.035	0.0876
		Tuned	0.0291	0.0244	0.0399	0.087	0.094	0.0511	0.1343	0.0485	0.0609	0.03824	0.0576
		Ours	**0.0254**	0.0274	**0.0364**	**0.0661**	0.105	0.0535	**0.0974**	**0.0333**	**0.0415**	**0.301**	0.0633
Nvidia Jeston Nano (4GB RAM)	High	Orignal	–	0.155	–	0.283	0.337	–	–	–	0.587	0.171	–
		Tuned	–	0.143	1.17	0.247	0.221	0.326	0.0848	–	0.0584	0.0622	–
		Ours	**0.167**	**0.0884**	**0.154**	**0.142**	**0.121**	**0.136**	0.103	**0.0818**	**0.0377**	**0.0483**	**0.151**

"–" means that the SLAM lost its localization during the experiment; "Tuned" means the parameters (e.g., time interval for map update) are assigned with suitable constants based on our empirical analysis.

is that some timing properties are not considered while implementing software. Based on insights from the analysis, an online adaptation method is proposed to mitigate the timing impacts. The proposed method dynamically adjusts the computation strategy according to the system state, achieving robust performance even when the system is congested. We demonstrate in our evaluation that, by selecting data that is properly temporally aligned and stabilizing the map updates, timing problems can be mitigated significantly. Moreover, experimental results show that our method can enable us to migrate expensive algorithms to low-cost platforms without manual tuning.

REFERENCES

[1] "Robot bee," https://wyss.harvard.edu/technology/robobees-autonomous-flying-microrobots/, accessed: 2021-10-29.

[2] B. Mu *et al.*, "Two-stage focused inference for resource-constrained minimal collision navigation," *IEEE T-RO*, 2017.

[3] S. Hochdorfer and C. Schlegel, "Landmark rating and selection according to localization coverage: Addressing the challenge of lifelong operation of slam in service robots," in *IEEE/RSJ IROS*, 2009.

[4] Y. Zhao and P. A. Vela, "Good feature matching: toward accurate, robust vo/vslam with low latency," *IEEE T-RO*, 2020.

[5] L. Carlone and S. Karaman, "Attention and anticipation in fast visual-inertial navigation," *T-RO*, 2018.

[6] Y. Zhao *et al.*, "Good graph to optimize: Cost-effective, budget-aware bundle adjustment in visual slam," *arXiv*, 2020.

[7] S. Zhang *et al.*, "Entropy based feature selection scheme for real time simultaneous localization and map building," in *IEEE/RSJ IROS*, 2005.

[8] C. Mei *et al.*, "Closing loops without places," in *IROS*, 2010.

[9] H. Strasdat *et al.*, "Double window optimisation for constant time visual slam," in *ICCV*. IEEE, 2011.

[10] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," *Robotics and autonomous systems*, 2009.

[11] F. A. Cheein *et al.*, "Feature selection criteria for real time ekf-slam algorithm," *IJARS*, 2009.

[12] Y. Zhao *et al.*, "Closed-loop benchmarking of stereo visual-inertial slam systems: Understanding the impact of drift and latency on tracking accuracy," in *IEEE ICRA*, 2020.

[13] G. Jacovitti and G. Scarano, "Discrete time techniques for time delay estimation," *IEEE Transactions on signal processing*, 1993.

[14] J. Kelly and G. S. Sukhatme, "A general framework for temporal calibration of multiple proprioceptive and exteroceptive sensors," in *Experimental Robotics*. Springer, 2014.

[15] T. Qin and S. Shen, "Online temporal calibration for monocular visual-inertial systems," in *IEEE/RSJ IROS*, 2018.

[16] Y. Ling *et al.*, "Modeling varying camera-imu time offset in optimization-based visual-inertial odometry," in *ECCV*, 2018.

[17] D. Falanga *et al.*, "How fast is too fast? the role of perception latency in high-speed sense and avoid," *RA-L*, 2019.

[18] A. Aalerud and G. Hovland, "Evaluation of perception latencies in a human-robot collaborative environment," in *ICRA*. IEEE, 2020.

[19] W. Jang *et al.*, "R-tod: Real-time object detector with minimized end-to-end delay for autonomous driving," in *IEEE RTSS*, 2020.

[20] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE T-RO*, 2017.

[21] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE T-RO*, 2021.

[22] T. Qin *et al.*, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE T-RO*, 2018.

[23] J. Engel *et al.*, "Direct sparse odometry," *T-PAMI*, 2017.

[24] C. Forster *et al.*, "Svo: Fast semi-direct monocular visual odometry," in *ICRA*. IEEE, 2014.

[25] S. Leutenegger *et al.*, "Keyframe-based visual–inertial odometry using nonlinear optimization," *IJRR*, 2015.

[26] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *ICRA*. IEEE, 2007.

[27] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *RSS*, 2014.

[28] W. Hess *et al.*, "Real-time loop closure in 2d lidar slam," in *ICRA*. IEEE, 2016.

[29] D. Fox *et al.*, "Monte carlo localization: Efficient position estimation for mobile robots," *AAAI*, 1999.

[30] Wustl, "Timing mitigation slam," https://github.com/WUSTL-CSPL/Timing-Adaptive-SLAM.

[31] M. Li and A. I. Mourikis, "3-d motion estimation and online temporal calibration for camera-imu systems," in *ICRA*. IEEE, 2013.

[32] D. Mori and Y. Hattori, "Simultaneous estimation of vehicle position and data delays using gaussian process based moving horizon estimation," in *IROS*. IEEE, 2020.

[33] H. Qiu *et al.*, "{FIRM}: An intelligent fine-grained resource management framework for slo-oriented microservices," in *OSDI*, 2020.

[34] J. Fried *et al.*, "Caladan: Mitigating interference at microsecond timescales," in *OSDI*, 2020.

[35] L. Weng *et al.*, "Argus: Debugging performance issues in modern desktop applications with annotated causal tracing," in *USENIX ATC*, 2021.

[36] M. Burri *et al.*, "The euroc micro aerial vehicle datasets," *IJRR*, 2016.

[37] A. Li, J. Wang, and N. Zhang, "Chronos: Timing interference as a new attack vector on autonomous cyber-physical systems," in *CCS*, 2021.