UNIVERSITY OF CALIFORNIA
RIVERSIDE

A New Resource-Efficient 3D SLAM Framework using Adaptive Interval Rates

A Thesis submitted in partial satisfaction
of the requirements for the degree of

Master of Science

in

Computer Engineering

by

Seungjoon Lee

June 2023

Thesis Committee:

Dr. Hyoseung Kim, Chairperson
Dr. Nael Abu-Ghazaleh
Dr. Jiasi Chen

The Thesis of Seungjoon Lee is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I am immensely grateful to my advisor, Hyoseung Kim, for his guidance, expertise, and continuous support throughout the entire research process. His invaluable insights and constructive feedback have been instrumental in shaping this work. I would like to extend my heartfelt appreciation to my wife Sori, my son Junseo, and my parents for their endless love, understanding, support, encouragement and sacrifices they have made throughout my academic pursuits. I would also like to acknowledge the support and encouragement I have received from my brother, SeungHwan. His belief in my abilities and bright insights have been a driving force behind my accomplishments. Furthermore, I want to express my deepest gratitude to my parents and father-in-law for their love. Their belief in my potential and relentless encouragement have been the foundation of my success. Lastly, I would like to express my appreciation to the Korea Navy for providing me with the opportunity to pursue my master's degree. Their support and belief in my abilities have been instrumental in my personal and professional development. I truly thankful for people who love and protect my family in belief. Without your help, I would not have been able to accomplish what I have today. Thank you all from the bottom of my hear.

ABSTRACT OF THE THESIS

A New Resource-Efficient 3D SLAM Framework using Adaptive Interval Rates

by

Seungjoon Lee

Master of Science, Graduate Program in Computer Engineering
University of California, Riverside, June 2023
Dr. Hyoseung Kim, Chairperson

Simultaneous Localization and Mapping (SLAM) is a fundamental task in robotics and computer vision, allowing a robot to build a map of its environment while estimating its own pose. In real-time SLAM scenarios, the processing time plays a critical role in achieving accurate and timely results. However, varying processing times due to computational load or system constraints can directly impact the performance of SLAM algorithms.

In this thesis, a new resource-efficient 3D SLAM framework is proposed using adaptive interval rates under consistent uncertainty. The proposed approach incorporates an adaptive mechanism that dynamically adjusts SLAM parameters based on the available processing time, allowing the system to adapt to changing computational loads and achieve real-time performance while maintaining accuracy. Specifically, upgrading the popular method, named Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping (LIO-SAM)[17] is utilized, which combines LiDAR and inertial sensor data for 3D SLAM.

The proposed approach utilizes processing time measurement and control of a key SLAM parameter such as key-frame frequency. By dynamically adapting this parameter,

the system can optimize its performance based on the available processing time, maximizing the efficiency of system resources. We conduct extensive experiments and evaluations on real-world datasets to validate the effectiveness of the proposed approach and compare it with existing SLAM approaches.

The results demonstrate that the adaptive-SLAM approach enhances the adaptability and efficiency of LIO-SAM in real-time visual odometry scenarios. It achieves accurate and timely pose estimation while effectively utilizing the available processing time, making it suitable for applications that require real-time and adaptive SLAM capabilities, such as autonomous robotics, augmented reality, and virtual reality. The findings of this thesis contribute to the field of SLAM by providing a novel approach for maximizing the utilization of processing time in SLAM algorithms, opening up new possibilities for robust and efficient localization and mapping in dynamic environments.

List of Figures

List of Tables

Chapter 1

Introduction

Simultaneous Localization and Mapping (SLAM) is a fundamental task in robotics and computer vision that enables a robot to construct a map of its environment while estimating its own position. In real-time scenarios, robots equipped with multiple sensors, such as three-dimensional (3D) light sensing and ranging (LiDAR) sensors or vision sensors, often face resource limitations while performing tasks, including SLAM. However, existing 3D SLAM frameworks, such as Lidar Odometry and Mapping (LOAM)[22] and Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping (LIO-SAM)[17], struggle to efficiently utilize limited resources due to their fixed processing cycles. This limitation becomes particularly challenging for robots based on embedded systems, where regular processing of large data sets can quickly deplete the battery, hindering the robot's long-term mission.

Figure 1.1 illustrates the need for an adaptive processing interval. 3D LiDAR sensors typically generate 600,000 point clouds per second. Continuous processing of these

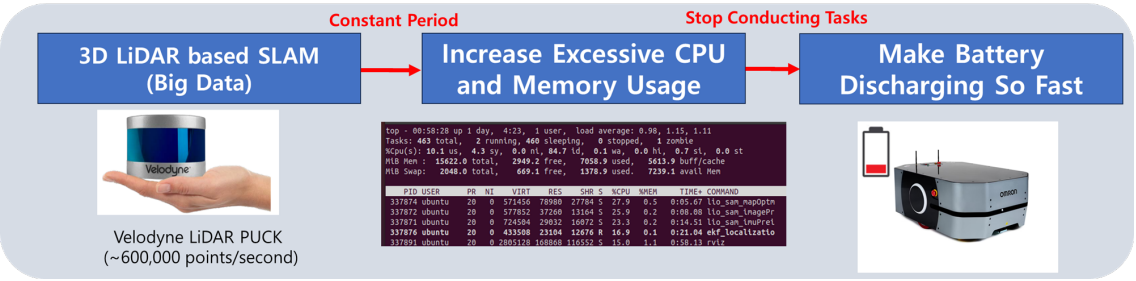

Figure 1.1: Need for Adaptive Processing Intervals in 3D LiDAR-based SLAM

3D point clouds using CPU and memory consumes significant amounts of power, leading to premature battery drain and requiring the robot to halt operations for recharging.

In this thesis, I propose a novel resource-efficient 3D SLAM framework with adaptive intervals, aiming to utilize limited resources effectively. The processing interval is dynamically calculated based on the current pose covariance, a measure of uncertainty in the estimated robot pose. Most SLAM algorithms operate by minimizing the covariance of SLAM state variables. Therefore, the processing interval is determined as a function of pose covariance and is controlled to prevent its increase.

The remaining chapters of this thesis are organized as follows. Chapter 2 provides an overview of related research works and briefly introduces the fundamental framework, LIO-SAM. In Chapter 3, I present a detailed description of the proposed framework. The experiments conducted and their analysis are presented in Chapter 4. Lastly, Chapter 5 summarizes the entire contents and concludes the thesis.

Chapter 2

Related Works

2.1 SLAM

Several research works related to SLAM have been classified and summarized in [3]. Additionally, J. A. Placed et al. conducted a comprehensive investigation of SLAM in their study [14]. They compared various studies in terms of SLAM methods, sensor equipment, map representation, formulation, exploration methods, utility function, validation environments, stopping criteria, and availability of public resources. Furthermore, they presented challenging objectives in the field.

There is a wide range of 2-dimensional (2D) SLAM approaches. The initial solution for the SLAM problem involved the use of extended Kalman filter (EKF) [4] that are the extended version of a Kalman filter. The Kalman filter [18]utilizes observers, which are mathematical models of linearized systems that aid in estimating the behavior of the real system. These filters also incorporate an optimal state estimator that deals with white noise in system measurements [12]. In the context of SLAM, the EKF predicts the state

of a robot pose and maps in real-time using a state equation of the robot's movement and environment. In several research works, the commonly employed sensor is LiDAR (Light Detection and Ranging), although solutions using sonars or monocular cameras [2] have also been developed.

For more accurate SLAM, another strategy involves the use of particle filters with a number of particles [1]. Although there are a lot of differences between particle filters and Kalman filters, the main distinction lies in the type of data distribution they can handle. Kalman filters are designed for linear Gaussian distributions, whereas particle filters can handle arbitrary non-Gaussian distributions and non-linear process models. In SLAM, particle filters include a set of particles, each representing a specific estimation of the robot state (pose and map) [13]. As the robot moves through the given environment and gathers sensor data, the filter consistently eliminates erroneous particles (those with the low probability of occurrence) and introduces new particles that are close to those with the highest probability of occurrence. Those are called particle rejection and replication. Over time, the erroneous particles are removed while the correct ones converge to similar estimates of pose and map. It is called a particle poverty problem [10], which sometimes can cause another problems. Typically, LiDAR sensors are used in particle filter-based SLAM. The representative approaches include Gmapping [21], and Hector SLAM [9].

A more recent approach to SLAM involves the utilization of graph-based methodologies [2]. This approach deals with a graph structure consisting of nodes and edges. Nodes correspond to the robot poses at different points in time, and edges denote physical constraints between the poses. The graph is constructed through environment observations

or the robot's movement actions. Once the graph is optimized, the map can be computed by finding the spatial configuration of the nodes that best aligns with the measurements represented by the edges. This process is typically performed using standard optimization methods such as Gauss-Newton or Levenberg–Marquardt [6], or nonlinear sparse optimization techniques [19].

In recent years, there has been significant development in 3D LiDAR-based SLAM approaches. The LOAM algorithm suggested by Zhang [22] played a pivotal role in popularizing and garnering attention for LiDAR-based 3D SLAM technology. Zhang's work focused on extracting crucial information from complex point clouds, particularly effective edge and plane feature points. The algorithm leveraged point-to-line and point-to-plane distances to construct an error function and solve the nonlinear optimization problem of the robot poses. However, LOAM lacks loop closure detection and global pose optimization in the back-end. Subsequent to LOAM, many LiDAR-IMU loosely coupled systems have been improved and refined. The works presented in this section not only focuses on sensor data fusion, but also improving the point cloud registration of the front end and the overall optimization of the back end. Shan proposed the LeGO-LOAM algorithm [16] based on LOAM, introducing point cloud clustering and ground segmentation into the data preprocessing stage to expedite point cloud registration. Furthermore, a simple acceleration formula is employed to process IMU data for point cloud distortion correction and provide a priori pose. The IMU plays a similar role in line/plane feature-based LOAM and two-stage LOAM. Feature extraction has received increased attention, with both methods utilizing the normal vector of points to extend the types of considered features. CSS-based LOAM

[5]and ALeGO-LOAM[11] further enhance feature quality compared to previous methods. However, this loosely integrated approach does not effectively mitigate the influence of IMU measurement bias, with the IMU serving merely as a supplementary means.

2.2 LIO-SAM

LIO-SAM[17], as recently introduced, represents a notable method for 3D SLAM. This approach excels in achieving highly accurate robot trajectory estimation and real-time map-building capabilities. By combining LiDAR and inertial measurements, LIO-SAM enables real-time motion estimation for a robot. The estimation problem is formulated as a factor graph, facilitating the integration of diverse measurement types, including relative and absolute measurements, loop closures, and others.

Initially, the system employs inertial measurements from an IMU to generate an initial motion estimate for the robot. Subsequently, this estimate is refined by using LiDAR measurements. The LiDAR odometry solution also aids in estimating the IMU bias. To achieve high real-time performance, LIO-SAM utilizes a selective approach when integrating LiDAR scans. Instead of matching every new scan to a global map, the system marginalizes older scans and registers new scans with a fixed-size set of prior "sub-keyframes." This strategy greatly improves real-time performance, especially when combined with the introduction of keyframes based on their importance. As a result, the system excels in performing local-scale scan-matching, enabling efficient and accurate real-time motion estimation. In figure 2.1, the scheme of LIO-SAM within the Robot Operating System (ROS) [15], a collection of software libraries and tools, is depicted. The LIO-SAM workflow begins with the

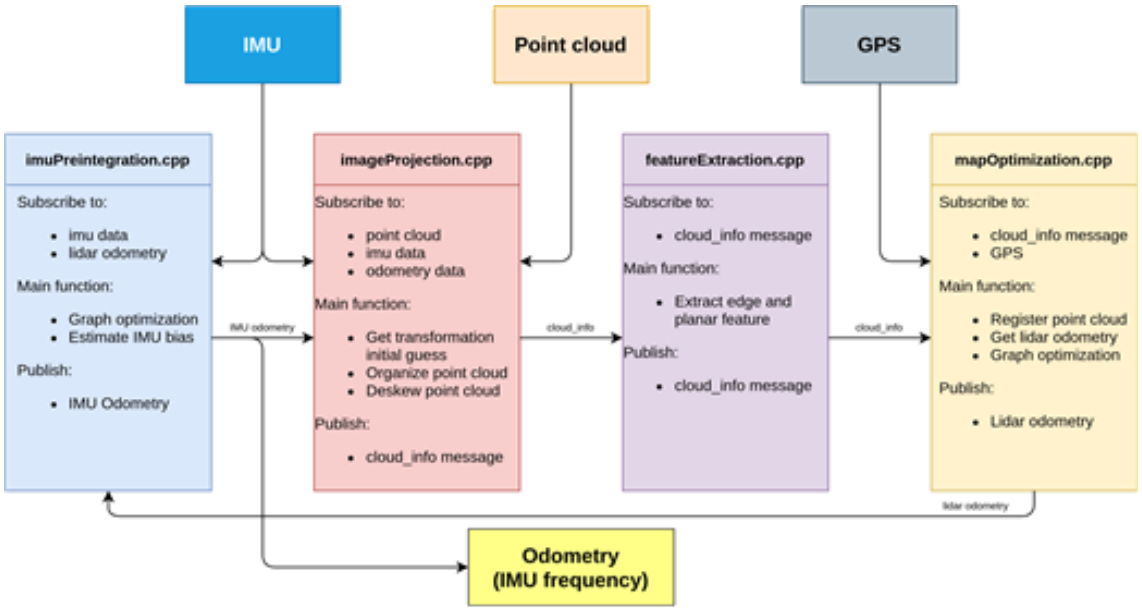

Figure 2.1: LIO-SAM Scheme in ROS [17]

generation of a cloud information message using the imageProjection module. Subsequently, this message is passed to the featureExtraction module, which extracts the features from the point cloud. The cloud information message, along with the extracted features, is then delivered to the mapOptimization module.

Within the mapOptimization module, a series of crucial operations are conducted to enhance the accuracy and reliability of the mapping process. Firstly, the point cloud registration algorithm is employed to align the successive point clouds, ensuring seamless integration and minimizing inconsistencies. Additionally, LiDAR odometry acquisition is executed to estimate the robot's motion based on the LiDAR measurements. By analyzing the changes in the point cloud data over time, the system can accurately track and predict the robot's position and orientation, crucial for effective mapping. Finally, graph optimization is carried out to refine the estimated motion based on the available constraints.

These techniques utilize the available constraints and measurements to iteratively optimize the estimated motion trajectory, ensuring alignment with the observed data and minimizing errors. The LiDAR odometry information is further passed to the imuPreintegration module.

In the imuPreintegration module, various tasks are performed. Firstly, IMU bias estimation takes place utilizing the LiDAR odometry information. Subsequently, graph optimization is performed to further refine the motion estimation using the IMU and LiDAR odometry measurements. Finally, IMU odometry, representing the estimated robot motion based on the IMU measurements, is generated. The IMU odometry is then fed back to the imageProjection module. Within this module, initial value estimation is performed based on the IMU odometry information. Additionally, point cloud distortion removal is carried out to mitigate any distortions present in the point cloud data.

However, since SLAM in robots is based on embedded systems, it cannot be free from energy efficiency issues. In [7], they produced strong statistical evidence, based on the pose error, map accuracy, CPU usage, and memory usage. Especially, in their work, KARTO-SLAM outperformed all the other algorithms because it balances the use of resources and holds a good SLAM performance. Therefore, in addition to the SLAM accuracy problem, an energy-efficient strategy on the embedded systems is required.

Chapter 3

Proposed Framework

In this Chapter, I provide a comprehensive description of the proposed framework. Initially, I present the overall structure of the framework, outlining its key components and their interrelationships. This serves as a foundation for understanding the subsequent detailed discussion.

Subsequently, I delve into each part of the framework, offering an in-depth analysis of its functionalities and contributions to the overall framework. I highlight the unique features and innovations brought forth by each component, emphasizing their significance in enhancing the effectiveness and performance of the proposed framework.

3.1 Framework Design

Technically, the proposed framework builds upon the foundation of LIO-SAM, which is recognized as one of the prominent 3D LiDAR SLAM frameworks that have gained popularity in recent times. In the previous chapter, we provided a brief overview of LIO-

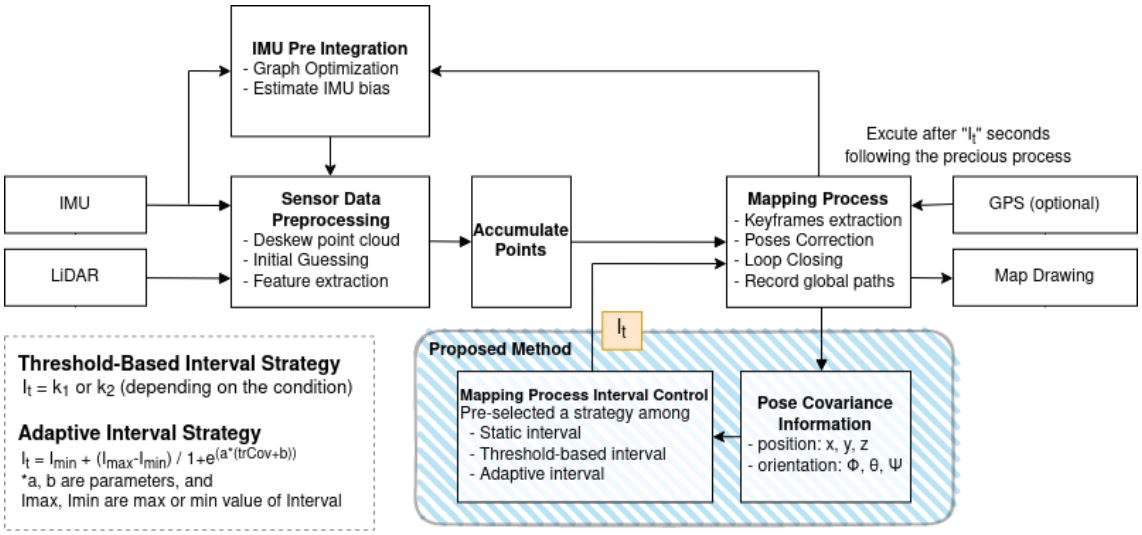

Figure 3.1: Proposed Framework

SAM, highlighting its capabilities and contributions in the field of SLAM. While LIO-SAM has demonstrated powerful SLAM performance, especially with the integration of IMU data, it is important to acknowledge that its regular execution interval may pose certain challenges in embedded robot systems characterized by limited resources. Specifically, these challenges manifest in the form of excessive CPU and memory usage, as well as accelerated battery discharge.

In the context of the proposed framework, I aim to address these concerns and optimize the performance of LIO-SAM for deployment in resource-constrained embedded robot systems. By carefully considering the system's limitations and the specific requirements of such systems, we devise strategies and optimizations to mitigate the negative effects associated with regular execution intervals.

Through my proposed framework, I strive to strike a balance between maintaining SLAM performance and ensuring efficient resource utilization. This entails devising tech-

niques that minimize CPU and memory usage, thereby alleviating the burden on limited resources, and implementing power management strategies to optimize battery consumption. By doing so, I enable the application of LIO-SAM in embedded robot systems without compromising their operational efficiency.

The structure of the proposed framework is illustrated in the figure 3.1. A notable addition to the conventional LIO-SAM structure is the inclusion of an adaptive interval update component. This component plays a crucial role in determining the interval at which the SLAM process is executed. The interval is determined based on the three strategies, which include a constant interval, a threshold based interval, and an adaptive interval. All of these approaches, with the exception of the first one, specifically address the handling of uncertainty related to the 6D position estimation results, focusing on the pose covariance information.

The 6D position estimation results, denoted as X, consist of the coordinates x, y, and z, as well as the orientation angles ϕ, θ, and ψ. A 6x6 covariance matrix that provides information about the uncertainty or confidence in the estimated pose at time t is represented as follows:

$$P_t = \begin{bmatrix} P_{xx} & \dots & P_{x\psi} \\ \vdots & \ddots & \vdots \\ P_{\psi x} & \dots & P_{\psi\psi} \end{bmatrix} \tag{3.1}$$

where the total variance related to a current estimated 3D geometric position is represented as $\mathrm{tr}(P_{1:3,1:3})$. It is abbreviated as $trPCov$. $\mathrm{tr}(\cdot)$ is the sum of the diagonal elements in a matrix. In addition, the total variance of 3D orientations is computed using $\mathrm{tr}(P_{4:6,4:6})$, which is described as $trOCov$. Even though the purpose of SLAM is accurate localization

and mapping (minimization of P), energy-efficient processing is also significant in embedded robot systems. Therefore, it is important to ensure that SLAM does not diverge and that energy efficiency can be maximized. However, those are a trade off relationship. Therefore, the problem to be solved in this thesis can be defined as a loss function L_t at time t as follows:

$$L_t = \min P_t + \Sigma_{i=0}^{N} \lambda E_{power} \tag{3.2}$$

where N is the number of SLAM operations up to time t. $\mathrm{E}_{t,power}$ is the amount of the power consumption from total resources. λ is a transfer coefficient between the uncertainty and the power consumption.

In this thesis, $\mathrm{E}_{i,power}$ is assumed to be a constant value, which means $\mathrm{E}_{i,power}$ equals to E_{power}. Eq(3.2) can be reformulated as follows:

$$L_t = \min P_t + N \lambda E_{i,power} \tag{3.3}$$

where N should be minimized while maintaining the minimum of P_t. P_t can be separated by terms of the 3D position and orientation. If $\min P_t$ is replaced with the minimization of total variances, it can be represented as follows:

$$L_t = \min(trPCov_t + trOCov_t + N \lambda E_{i,power}) \tag{3.4}$$

where N, $trPCov_t$ and $trOCov_t$ are the functions of the SLAM interval I. $\Delta trPCov$ and $\Delta trOCov$ represent the rate of change of $trPCov$ and $trOCov$, respectively, per unit of time. As I increases, N tends to decrease. If the interval is fixed, $N = T_{total}/I$. Conversely, as

I increases, $trPCov_t$ and $trOCov_t$ decrease. Based on these correlations, several strategies are suggested to minimize L_t by controlling I in the following sections.

3.2 Static Interval-based Strategy

Intuitively, it is possible to find a proper interval I that minimizes the loss function $L_t(\cdot)$ by considering various candidates of I within a specific environment. The proper I_F satisfies the following equation:

$$I_F = arg\min_I L(I) \tag{3.5}$$

where $L(I)$ is denoted as $L_t(I)$, representing the loss function calculated at the final time. The candidates for I can range from $I_{c,min}$ to $I_{c,max}$. $I_{c,min}$ represents the minimum interval at which the processor can handle the 3D SLAM process, while $I_{c,max}$ represents the maximum interval within which the SLAM process does not diverge. This approach relies on selecting a fixed static interval for SLAM processing. However, it does not allow for modifications in the middle of the SLAM process. Since the covariance matrix P_t is continuously updated over time, it becomes challenging to adequately adapt a fixed constant interval to accommodate the changes in P_t at each moment.

3.3 Threshold-based Interval Control Strategy

A threshold-based approach in the context of SLAM provides a mechanism to dynamically adjust the processing interval based on a specific covariance threshold. This threshold serves as a reference point to detect when the covariance value reaches a critical

level that may lead to the divergence of the SLAM process. In this approach, the interval I becomes a time-varying variable denoted as I_t. I_t is defined as follows:

$$I_t = \begin{cases} k_1, & \text{if } \Delta trPCov_t \leq p_1 \text{ or } \Delta trOCov_t \leq p_2 \\ k_2, & \text{otherwise} \end{cases} \tag{3.6}$$

$$\Delta trPCov_t = \frac{trPcov_t - trPCov_{t-1}}{T_t - T_{t-1}},\ \Delta trOCov_t = \frac{trOcov_t - trOCov_{t-1}}{T_t - T_{t-1}} \tag{3.7}$$

where p_1 and p_2 represent upper bounds on the total variance of 3D positions and orientations, respectively. If they are more than that, SLAM can diverge. The interval k_1 is typically set to be larger than k_2. As previously mentioned, N, $trPCov_t$, and $trOCov_t$ are related to the interval I. Consequently, N is reformulated as follows:

$$N = N_1 + N_2 \tag{3.8}$$

$$N_1 = \frac{T_{total,k_1}}{k_1}, \tag{3.9}$$

$$N_2 = \frac{T_{total,k_2}}{k_2}, \tag{3.10}$$

where T_{total,k_1} represents the total time affected by the interval k_1, while T_{total,k_2} represents the total time affected by the interval k_2. The problem can be reformulated as follows:

$$L_t = \min(trPCov_t + trOCov_t + \lambda(N_1 + N_2)E_{power}) \tag{3.11}$$

Since k_1 is larger than k_2, it is possible for $trPCov_t$ and $trOCov_t$ to be relatively large when using k_2 as the interval.

3.4 Adaptive Interval-based Strategy

The final approach in our study is the adaptive interval-based strategy, which offers a different approach compared to the threshold-based interval control strategy. While the threshold-based strategy discretely modifies the processing interval I_t based on predefined constraints such as $trPCov_t \leq p_1$ and $trOCov_t \leq p_2$, the adaptive interval-based strategy focuses on consistently adjusting I_t to maximize the efficient utilization of available resources. In this approach, L_t is defined as a function of I_t, which means N_t, $trPCov_t$ and $trOCov_t$ are the function of I_t as well as follows:

$$L_t(I_t) = \min(trPCov_t(I) + trOCov_t(I) + \lambda N_t(I) E_{power}) \tag{3.12}$$

In the adaptive interval-based strategy, the interval should be changed consistently during a short time. $trPCov_t(I)$ and $trOCov_t(I)$ are represented by the first-order Taylor series approximation as follows:

$$trPCov_t(I) \simeq trPCov_t(0) + trPCov'_t(I)I \tag{3.13}$$

$$trOCov_t(I) \simeq trOCov_t(0) + trOCov'_t(I)I \tag{3.14}$$

where $trPCov'_t(I)I$ and $trOCov'_t(I)$ are the derivatives of each trace of the covariance. If $N_t(I)$ is proportional to the inverse of the interval, I, the loss function, $L_t(I_t)$ is reformulated as follows:

$$L_t(I_t) = \min(trPCov_t(0)+trPCov'_t(I)I+trOCov_t(0)+trOCov'_t(I)I+\lambda(\frac{1}{I_t})E_{power}) \quad (3.15)$$

The minimization of the loss function $L_t(I_t)$ is equal to the $\frac{\partial L_t(I_t)}{\partial I_t} = 0$. It is computed as follows:

$$\frac{\partial L_t(I_t)}{\partial I_t} = trPCov'_t(I) + trOCov'_t(I) - \lambda(\frac{1}{I_t^2})E_{power} = 0 \quad (3.16)$$

$$I_t = \frac{\lambda E_{power}}{trPCov'_t(I) + trOCov'_t(I)} \quad (3.17)$$

$$I_t = \frac{\lambda E_{power}}{\Delta trPCov_t + \Delta trOCov_t} \quad (3.18)$$

where $\Delta trPCov$ and $\Delta trOCov$ denote the difference of the trace of covariance between t and $t-1$, respectively. Although I_t should be larger than zero, $\Delta trPCov_t$ and $\Delta trOCov_t$ can be negative or zero. To solve this, I_t is considered as a sigmoid function with $\Delta trPCov_t$ and $\Delta trOCov_t$. It can be represented as follows:

$$x = \lambda_P \cdot w_P \cdot \Delta trPCov_t + \lambda_O \cdot w_O \cdot \Delta trOCov_t \quad (3.19)$$

$$I_t \ = \ I_{\min} + \ (I_{\max} - I_{\min}) \cdot \frac{1}{1 + e^{\alpha \cdot (x+\beta)}} \quad (3.20)$$

$$w_P + w_O = 1 \quad (3.21)$$

Here, λ_P and λ_O are scaling factors that determine the relationship between the interval and the inverse of the variance. w_P and w_O represent the weights assigned to the total pose variance and orientation variance, respectively, in order to determine appropriate intervals. α and β are smoothing parameters that affects the steepness of the adaptive interval change. It is important to note that the sum of w_P and w_O should be equal to one. Overall, these parameters and their relationships are essential in the adaptive interval control strategy, allowing the system to dynamically adjust processing intervals based on the level of uncertainty and the specific requirements of pose and orientation estimation tasks.

3.5 Discussion of Three Strategies

The discussion for the three interval-based strategies (static interval, threshold-based interval control, and adaptive interval) can be summarized as follows:

static Interval-based Strategy: The static interval-based strategy suggests selecting a fixed interval, denoted as I, for the SLAM process. The goal is to find a proper I that minimizes the loss function $L_t(I)$. This approach involves considering various candidates for I within a specific environment. The range of candidates typically spans from the minimum interval that the processor can handle ($I_{c,min}$) to the maximum interval within which the SLAM process remains stable without divergence ($I_{c,max}$). However, this approach does not allow for dynamic modifications of I during the SLAM process, which can make it challenging to adapt to changes in the pose covariance information over time.

Threshold-based Interval Control Strategy: The threshold-based interval control strategy introduces a dynamic adjustment of the interval value based on predefined covariance thresholds. The interval, denoted as I_t, becomes a time-varying variable. If the total variances of 3D positions and orientations, represented as $trPCov_t$ and $trOCov_t$ respectively, exceed certain threshold values (p_1 and p_2), the interval is set to a larger value k_1. Otherwise, if the variances are below the thresholds, the interval is set to a smaller value k_2. The aim is to prevent divergence of the SLAM process by adapting the interval based on the pose covariance information. This strategy allows for discrete modifications of I_t at specific instances.

Adaptive Interval-based Strategy: The adaptive interval-based strategy aims to consistently adjust the interval I_t to efficiently utilize resources. The interval is set as a function of the pose covariance information, specifically $trPCov_t$ and $trOCov_t$. In this thesis, the proposed function is defined as a linear relationship between the inverse of the variances and the interval. This adaptive strategy allows for continuous adjustments of the interval based on the varying pose covariance information.

Overall, these three strategies aim to optimize the performance of the LIO-SAM framework in resource-constrained embedded robot systems by considering different approaches for interval control. The static interval strategy selects a fixed interval, the threshold-based strategy dynamically adjusts the interval based on predefined thresholds, and the adaptive interval strategy continuously adapts the interval based on the pose covariance information. Each strategy has its own advantages and trade-offs, and the choice depends on the specific requirements and constraints of the system.

Chapter 4

Performance Evaluation

In this thesis, I have investigated three strategies for determining the processing interval in my proposed framework: a constant interval, a threshold-based interval, and an adaptive interval. This chapter focuses on analyzing the characteristics and performance of these strategies. To facilitate a comprehensive comparison and evaluation, I utilized public datasets in my experimental setup.

Several performance factors were considered in my analysis, including processing time over cycles, the number of cycles, total consumed time, maximum CPU usage, memory usage, and SLAM errors. These metrics provide valuable insights into the effectiveness and efficiency of each strategy. Note that, the goal is to find the better way having both high resource efficiency and moderate accuracy with these factors.

By examining the processing time over cycles, I assessed the computational efficiency of the strategies throughout the SLAM process. Additionally, the number of cycles provided an indication of the frequency at which processing occurred, further contribut-

ing to my understanding of the strategies' behavior. To evaluate the impact on system resources, I measured the total consumed time, maximum CPU usage, and memory usage associated with each strategy. These metrics offer valuable insights into the resource utilization patterns and efficiency of the strategies.

Furthermore, I considered SLAM errors as an essential performance metric. By comparing and analyzing the errors produced by each strategy, I gained insights into their accuracy and reliability in mapping and localization tasks.

The results of these evaluations provide a comprehensive understanding of the characteristics and performance of the proposed strategies. By considering multiple performance factors, I ensure a thorough assessment of their suitability for resource-constrained robotic systems.

In a nutshell, this chapter focuses on the public dataset and setup used in the study, as well as the presentation of three strategies. Additionally, an analysis of the results is conducted to compare each strategy based on the aforementioned factors.

4.1 Public Dataset and Setup

The dataset used in my experiments was obtained from the open-source data provided by LIO-SAM. The dataset employed a Velodyne VLP-16 LiDAR, a MicroStrain 3DM-GX5-25 IMU, and a Reach M GPS device. Park dataset is collected in Pleasant Valley Park, New Jersey. This is open source data which was once used by LIO-SAM. The trajectory direction is from left to right in figure 4.1. The park where data collection took place is characterized by a significant presence of trees, resembling a trail. For specific

Table 4.1: Dataset details

Dataset	Scans	Elevation change(m)	Trajectory length(m)	Max rotation speed(°/s)	Play time(s)
Park	24691	19.0	2898	217.4	560

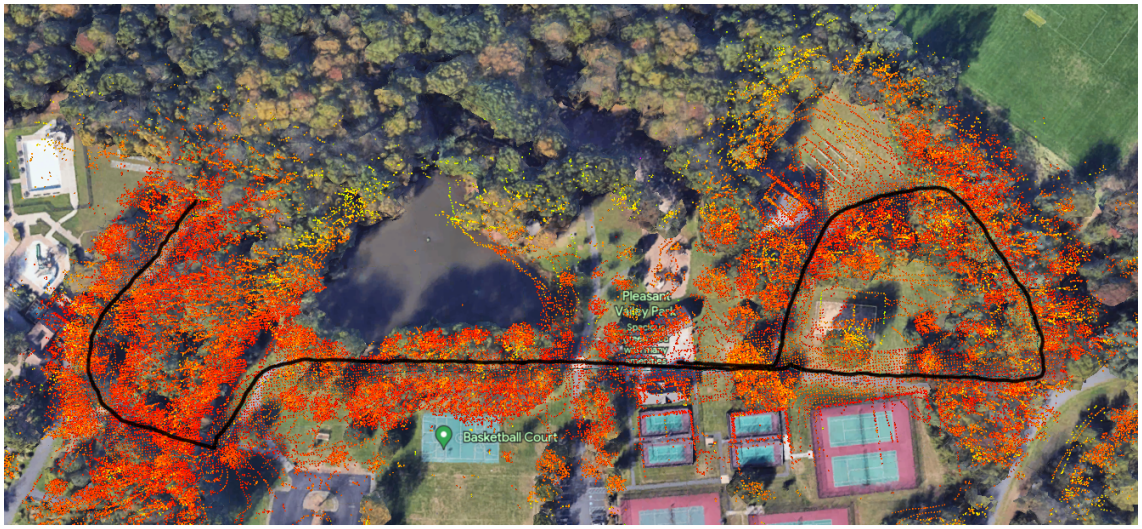

Figure 4.1: Map of Park dataset aligned with Google Earth

details regarding the dataset, please refer to the table 4.1. In this experiments, a laptop computer with Intel i7-12650, 16GB RAM and Nvidia GeForce GTX 3070M was used.

There are several evaluation factors, such as the number of processing time, the number of cycles N_t, the total processing time T_{total}, power consumption E_{power}, the maximum usage of CPU and memory related to max(E_{power}), and the SLAM error E_P. E_P is computed as follows:

$$E_P = \sqrt{X_{L,true} - \hat{X}_L} \tag{4.1}$$

4.2 Static Interval

Typically, the static interval strategy is the simplest approach used in SLAM (Simultaneous Localization and Mapping), where the interval value i is fixed as a constant. This chapter addresses the various processing metrics according to each interval value. In the evaluation, I examined the impact of different fixed processing interval values on the mapping process. Table 4.2 shows key metrics such as mapping process duration, CPU utilization, memory usage, and GPS position error corresponding to each fixed interval value.

The findings indicate that as the interval lengthens, the processing time for a single mapping cycle exhibits a slight increase. However, the total number of cycles decreases, leading to reduced CPU utilization and overall processing time. Moreover, elongating the interval results in a decreased number of poses that need to be stored in memory, consequently leading to lower memory usage. Note that, if the distance between two adjacent points are too far, pose estimation may fail. This implies that accurate pose estimation cannot be achieved if the interval exceeds the optimal threshold. Table 4.2 effectively demonstrates the failure of mapping processing when it exceeds 2.2 seconds, highlighting this fact.

These observations provide valuable insights into the performance and trade-offs associated with different fixed processing interval values in the proposed memory-efficient 3D SLAM framework.

The table 4.2 reveals the processing time per a cycle, the number of processing cycles during mapping, the maximum CPU utilization (%), the resident memory size in megabytes, and the distance error compared to GPS, with respect to the processing interval

Table 4.2: Processing metrics by interval time using Park dataset

Interval	0.1 sec	0.3 sec	0.5 sec	1sec	1.5 sec	2 sec	2.2 sec
P.T./Cycle(ms)	59.27	65.12	69.02	72.37	73.95	77.47	Fail
Num of Cycles	2990	1812	1091	546	364	273	Fail
Consumed time(s)	177.2	117.9	75.3	39.4	26.9	21.1	Fail
MaxCPU(%)	100	90.7	54.0	48.7	32.9	20.3	Fail
Mem(MB)	325	309	280	269	216	201	Fail
Error(m)	0.53	0.72	1.03	1.02	1.89	23.29	Fail

time, using the park dataset. As shown in table 4.2, the experimental results using LIO-SAM indicate that as the processing time interval becomes longer, the processing time per cycle increases slightly, but the overall efficiency in terms of SLAM processing time, CPU resource consumption, and memory utilization is improved. In other words, as the time interval increases, resource efficiency improves, but the uncertainty of the robot's position increases, and localization and mapping may fail after a certain point.

The value of trPCov represents the uncertainty of the robot's x, y, z positions, while trOCov represents the uncertainty of the roll, pitch, and yaw information. figure 4.2 and 4.3 depict the variations of trPCov and trOCov, respectively. The significant changes occurring within the first 30 seconds are attributed to the influence of GPS, which records differences in GPS positions to track accuracy. While trPCov shows minimal changes from its initial value, trOCov exhibits significant variations over time due to the robot's motion. It implies that the uncertainty in the position variables are stable, while the uncertainty in the orientation variables varies over time.

If we look at figure 4.4, which represents the graph of the difference in $trOCov$, it provides an easier way to understand the variations. The values of $trOCov$ are close to

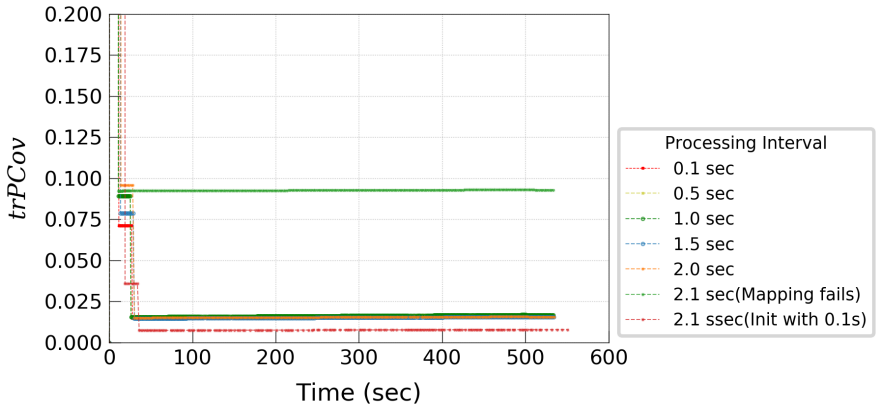

Figure 4.2: Trace of position covariance matrix by varying processing interval

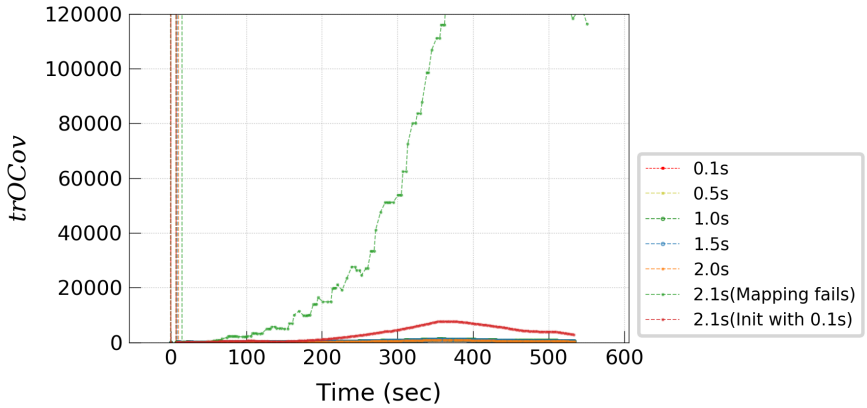

Figure 4.3: Trace of orientation covariance matrix by varying processing interval

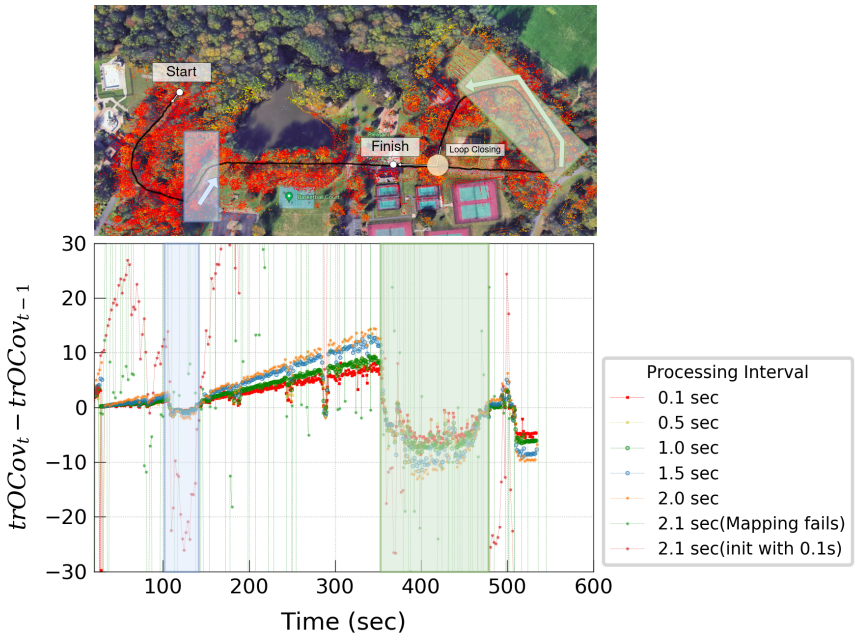

Figure 4.4: The graph showing the difference in consecutive $trOCov$ values along with the Google Earth map.

zero or negative when there is information from the already traversed path, and they show a linear increase when moving into unexplored areas. This phenomenon is related to the given information at each state. In the blue and green regions, the $trOCov$ values tend to be below 0, indicating that the pose estimation is converging. As a result, even if the robot has a longer interval, I believe the system will be safe from divergence. This is because there is existing information that aligns with the expected movement, leading to reduced uncertainty. Similar to how we pay more attention to unfamiliar paths compared to known areas, adjusting the interval speed in resource-limited regions and slowing it down in stable positions can help optimize resource usage.

4.3 Threshold-based Interval Control Strategy

As for the equation 3.6 mentioned earlier, the threshold-based interval control strategy returns one of two constants based on the system's state. Additionally, figure 4.4 provides insights on setting the values of $p1$ and $p2$. In this experiment, the values of $k1$ and $k2$ are assumed to be 0.1 and 2.0, respectively, while both $p1$ and $p2$ are set to 0. Each $k1$ and $k2$ represents the minimum and maximum intervals. The values of $p1$ and $p2$ are set to 0 when the pose estimation converges, specifically when the $trPCov$ value reaches 0. Therefore, the processing interval I_t follows equation 4.2.

$$I_t = \begin{cases} 0.1, & \text{if } \Delta trPCov_t \leq 0 \text{ or } \Delta trOCov_t \leq 0 \\ 2.0, & \text{otherwise} \end{cases} \tag{4.2}$$

The result is shown in figure 4.5, where the processing interval in the blue and green boxes corresponding to figure 4.4 increases to 2.0 seconds, while in other regions, it is set to a shorter duration of 0.1 seconds.

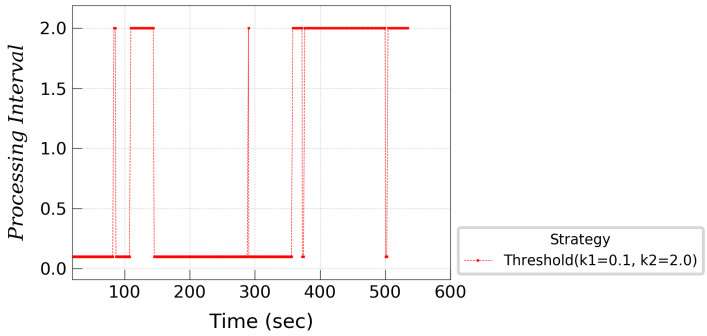

Figure 4.5: The graph illustrating the variation of the processing interval over time when applying the threshold-based interval strategy

4.4 Adaptive Interval Control

The adaptive interval control strategy in this study employs a sigmoid function, as defined in Equation 3.20. This function allows for a smooth and gradual adjustment of the processing interval based on the input factor values. In the experiment conducted, the parameters α and β are set to 0.4 and 2, respectively.

The sigmoid function used in the adaptive interval control strategy is represented by the equation $I_t = I_{\min} + (I_{\max} - I_{\min}) \cdot \frac{1}{1+e^{\alpha \cdot (x+\beta)}}$, where I_t denotes the processing interval at time t, $I_{\min}$ and $I_{\max}$ represent the minimum and maximum intervals, respectively. The parameters α and β control the shape and steepness of the sigmoid curve. In this experiment, α is set to 0.4 and β is set to 2.

By using the sigmoid function, the adaptive interval control strategy ensures a smooth and continuous transition of the processing interval in response to changes in the input factor values. This allows the system to dynamically adjust the interval based on the level of uncertainty or complexity in the environment, providing a more efficient and accurate motion estimation process.

Similar to the threshold-based interval control, the adaptive interval control also ranges between minimum and maximum intervals of 0.1 second and 2.0 seconds, respectively. The graph in figure 4.6 illustrates the shape of the sigmoid function used for adaptive interval control. As shown in the graph, the processing interval smoothly adapts to changes in the input factor values.

Figure 4.7 demonstrates the variation of the processing interval when applying the adaptive interval strategy on the park dataset. While the overall pattern of interval changes

is similar to the threshold-based interval strategy, the adaptive interval strategy exhibits a smoother transition of intervals. This smooth transition of the processing interval in the adaptive interval strategy can potentially improve accuracy in situations where uncertainty abruptly increases. By adjusting the interval smoothly, the system can allocate more processing resources and time to handle challenging or uncertain conditions, thereby enhancing accuracy.

The adaptive approach in interval control enables the system to dynamically respond to changes in the environment and allocate resources effectively. This leads to improved performance in uncertain or challenging scenarios. By adapting the processing interval based on the input factors, the system can optimize its performance and provide more accurate and reliable results.

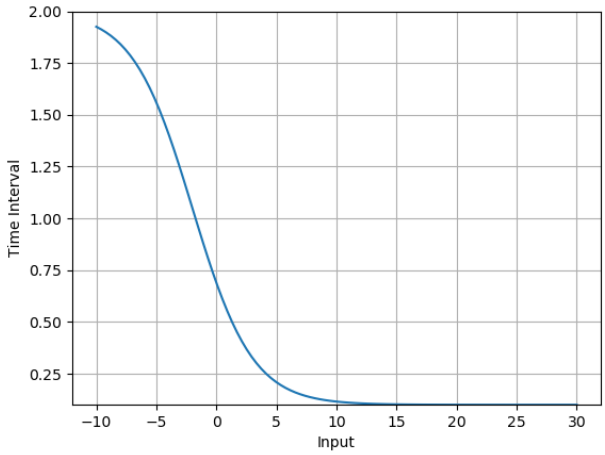

Figure 4.6: Graph of the Adaptive Strategy Function for Mapping Processing Time Interval (0.1s - 2.0s)

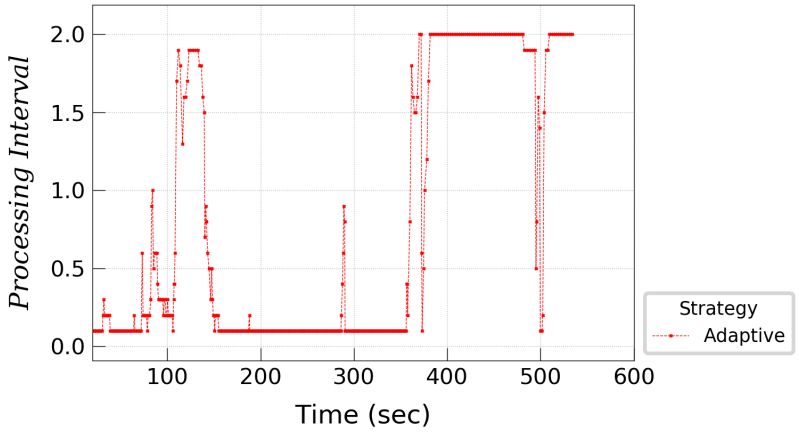

Figure 4.7: The Graph Depicting the Temporal Variation of the Processing Interval with the Adaptive Interval Strategy.

4.5 Result Analysis

In the evaluation of the performance of the discussed methods, several metrics were analyzed, including CPU utilization, memory utilization, and error distance. Figure 4.8 illustrates the changes in CPU utilization over time. When the interval is fixed at 0.1 seconds, the CPU maintains full utilization (100%). Conversely, with a static interval of 2.0 seconds, the CPU utilization remains around 20%. However, both the threshold-based and adaptive approaches, as shown in Figures 4.5 and 4.7, exhibit CPU utilization that increases or decreases in proportion to the interval size. The memory utilization, measured in terms of resident memory size (RES) in MB, is depicted in Figure 4.9. The amount of memory used varies based on the number of keyframes and point cloud data stored in memory. Comparing the threshold-based and adaptive approaches to the static 0.1-second and static 2.0-second strategies, it can be observed that the amount of information lost is

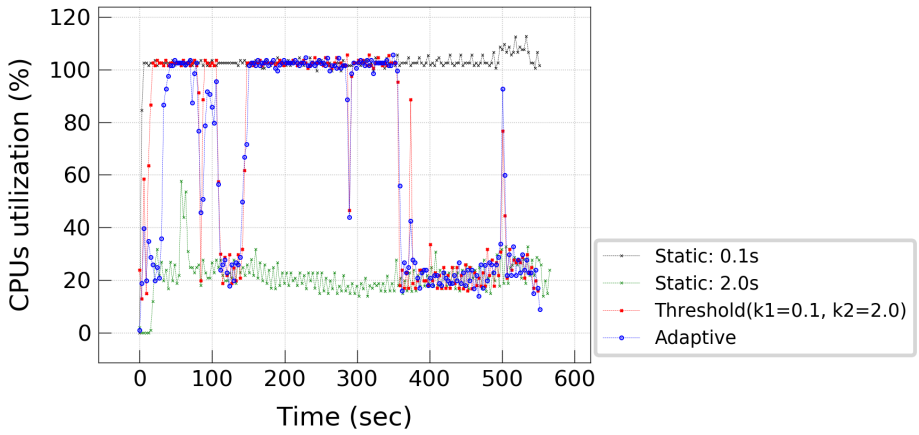

Figure 4.8: Comparative Analysis of CPU Utilization Across Different Methods.

relatively low. This is further supported by Figure 4.10, where the error distance is nearly negligible due to the minimal information loss.

These evaluations demonstrate the efficiency and effectiveness of the threshold-based and adaptive approaches in terms of resource utilization and estimation accuracy. The dynamic adjustment of intervals allows for optimal allocation of CPU resources, resulting in improved performance. Additionally, the minimal information loss and low error distance indicate the robustness of the methods in preserving the accuracy of motion estimation while efficiently managing memory usage.

Starting from approximately 150 seconds, the error distance gradually increases for both the threshold-based and adaptive approaches, indicating that a smooth transition of intervals during directional changes can reduce errors compared to abrupt changes. In the time range of 480 to 490 seconds, where loop closing takes place, the error distance decreases significantly once again. Figure 4.11 illustrates the trajectories obtained by each

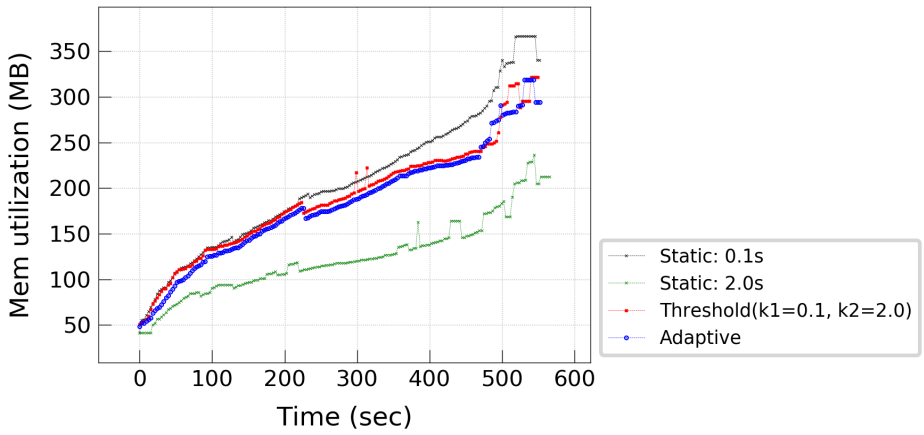

Figure 4.9: Temporal Variation of Memory Utilization Across Different Methods.

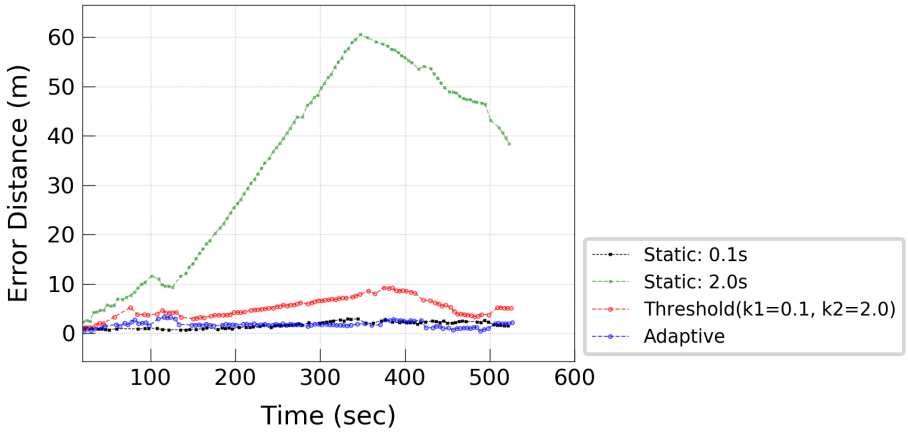

Figure 4.10: Temporal Variation of Error Distance Across Different Methods.

method on a map, with the loop closing observed around the x-axis value of 200. From the observations, it can be seen that the error increases gradually as time passes after a heading transformation of the robot. Particularly, the adaptive approach (blue) closely overlaps with the Static 0.1-second interval (black) method, while the threshold-based method (red) gradually diverges from the position estimation results of the Static 0.1-second interval

method as it moves from left to right. This implies that the adaptive approach provides more accurate and consistent position estimation results, especially in scenarios involving heading changes.

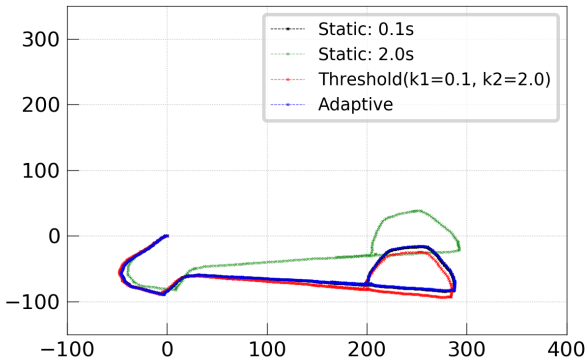

Figure 4.11: Path map for each method.

The experimental results at the end state are summarized in table 4.3. In comparison to the Static Interval 0.1-second strategy, the Threshold-based approach showed a total processing time of 67.2 seconds, while the Adaptive approach required 77.2 seconds. These values correspond to 38.1% and 43.5% of the minimum static interval, respectively, indicating a substantial improvement in resource efficiency of more than twofold. This highlights the effectiveness of adjusting the processing interval based on covariance information, enabling efficient mission execution in resource-limited scenarios where multiple tasks must be performed simultaneously.

In terms of accuracy, the Adaptive approach outperformed the Threshold-based approach by reducing the error by approximately 3 meters. This demonstrates the efficacy of employing the adaptive processing interval to achieve superior accuracy while efficiently

Table 4.3: Comparison of Memory and Error for Fixed, Threshold-based, and Adaptive Interval Mapping Processing

Interval	0.1s(static)	2.0s(static)	Threshold-based	Adaptive
N (Num of Cycles)	2990 (100%)	273 (10.9%)	1171 (39.1%)	1243 (41.5%)
Total Computation time(s)	177.2 (100%)	21.1 (11.9%)	67.6 (38.1%)	77.2 (43.5%)
Resident Memory Size(MB)	340.3 (100%)	212.6 (62.4%)	321.5 (94.4%)	294.3 (86.4%)
Error from GPS(m)	1.63 (0m)	23.29 (21.6m)	5.21 (3.58m)	2.16 (0.53m)

utilizing resources.

The results presented in Table 4.3 clearly demonstrate the advantages of the adaptive interval control strategy in terms of both resource efficiency and accuracy. By dynamically adjusting the processing interval based on covariance information, the system can effectively allocate resources and optimize performance in real-time SLAM applications. These findings contribute to the advancement of SLAM techniques in resource-constrained embedded environments, allowing for improved accuracy and efficient utilization of available resources.

Chapter 5

Conclusions

In this thesis, I have presented a novel 3D SLAM framework that addresses the challenge of limited memory resources by incorporating adaptive interval rates. The proposed framework builds upon the well-established LIO-SAM 3D SLAM framework, utilizing it as the fundamental basis for our approach. By dynamically adjusting the processing interval rates based on the current covariance of states, I aim to optimize the utilization of available resources.

To evaluate the effectiveness of our framework, I conducted extensive experiments using various publicly available datasets and modified parameters. The performance of our SLAM approach was assessed in terms of position error, as well as the efficiency of resource utilization, including memory and CPU usage. Through comparative analysis with conventional SLAM frameworks, I demonstrated that our proposed framework maintains robust SLAM performance while achieving efficient resource utilization. These findings highlight the significance of the resource-efficient 3D SLAM framework, which not only

maintain the overall SLAM performance but also ensures the optimal utilization of limited resources. By adaptively adjusting the processing interval rates, our framework provides a practical solution for real-time robotics applications that face resource constraints.

The outcomes of this research contribute to the advancement of SLAM techniques and provide valuable insights for the development of resource-efficient robotics systems. Further research can explore additional enhancements and optimizations to improve the adaptability and performance of our framework in diverse real-world scenarios.

Bibliography

[1] Particle filter. https://en.wikipedia.org/wiki/Particle_filter. Accessed: 2023-05-22.

[2] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017.

[3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[4] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, 2006.

[5] Clayder Gonzalez and Martin Adams. An improved feature extractor for the lidar odometry and mapping (loam) algorithm. In *2019 International Conference on Control, Automation and Information Sciences (ICCAIS)*, pages 1–7, 2019.

[6] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[7] M. Bolanos J. Fallas K. Trejos, L. Rincon and L. Marin. 2d slam algorithms characterization, calibration, and comparison considering pose error, map accuracy as well as cpu and memory usage. *Sensors*, 22(18):1–37, 2022.

[8] Stefan Kohlbrecher, J Meyer, K Petresen, and T Graber. Hector slam for robust mapping in usar environments. *ROS RoboCup Rescue Summer School Graz*, 2012.

[9] Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160, 2011.

[10] Nosan Kwak, In-Kyu Kim, Heon-Cheol Lee, and Beom-Hee Lee. Analysis of resampling process for the particle depletion problem in fastslam. In *RO-MAN 2007 - The*

16th IEEE International Symposium on Robot and Human Interactive Communication, pages 200–205, 2007.

[11] Sheng-Wei Lee, Chih-Ming Hsu, Ming-Che Lee, Yuan-Ting Fu, Fetullah Atas, and Augustine Tsai. Fast point cloud feature extraction for real-time slam. In *2019 International Automatic Control Conference (CACS)*, pages 1–6, 2019.

[12] Leonardo Marín, Marina Vallés, Ángel Soriano, Ángel Valera, and Pedro Albertos. Event-based localization in ackermann steering limited resource mobile robots. *IEEE/ASME Transactions on Mechatronics*, 19(4):1171–1182, 2014.

[13] Michael Montemerlo, Sebastian Thrun, Daphne Roller, and Ben Wegbreit. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI'03, page 1151–1156, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc.

[14] Julio A. Placed, Jared Strader, Henry Carrillo, Nikolay Atanasov, Vadim Indelman, Luca Carlone, and José A. Castellanos. A survey on active simultaneous localization and mapping: State of the art and new frontiers. *IEEE Transactions on Robotics*, pages 1–20, 2023.

[15] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[16] Tixiao Shan and Brendan Englot. Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4758–4765. IEEE, 2018.

[17] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Rus Daniela. Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142. IEEE, 2020.

[18] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. John Wiley & Sons, Hoboken, NJ, USA, 2006.

[19] Cyrill Stachniss, John J. Leonard, and Sebastian Thrun. Simultaneous localization and mapping. In *Springer Handbook of Robotics*, pages 1153–1176. Springer International Publishing, Berlin/Heidelberg, Germany, 2016.

[20] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. Fast-lio2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, 38(4):2053–2073, 2022.

[21] Rauf Yagfarov, Mikhail Ivanou, and Ilya Afanasyev. Map comparison of lidar-based 2d slam algorithms using precise ground truth. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1979–1983, 2018.

[22] Ji Zhang and Sanjiv Singh. Loam : Lidar odometry and mapping in real-time. *Robotics: Science and Systems Conference (RSS)*, pages 109–111, 01 2014.