big data and cognitive computing

MDPI

Article

EIF-SlideWindow: Enhancing Simultaneous Localization and Mapping Efficiency and Accuracy with a Fixed-Size Dynamic Information Matrix

Javier Lamar Léon *, Pedro Salgueiro, Teresa Gonçalves and Luis Rato

Centro Algoritmi, LASI, University of Évora, 7005-854 Évora, Portugal; pds@uevora.pt (P.S.); tcg@uevora.pt (T.G.); lmr@uevora.pt (L.R.)

* Correspondence: jlamarleon@uevora.pt or jlamarleon@gmail.com

Abstract: This paper introduces EIF-SlideWindow, a novel enhancement of the Extended Information Filter (EIF) algorithm for Simultaneous Localization and Mapping (SLAM). Traditional EIF-SLAM, while effective in many scenarios, struggles with inaccuracies in highly non-linear systems or environments characterized by significant non-Gaussian noise. Moreover, the computational complexity of EIF/EKF-SLAM scales with the size of the environment, often resulting in performance bottlenecks. Our proposed EIF-SlideWindow approach addresses these limitations by maintaining a fixed-size information matrix and vector, ensuring constant-time processing per robot step, regardless of trajectory length. This is achieved through a sliding window mechanism centered on the robot's pose, where older landmarks are systematically replaced by newer ones. We assess the effectiveness of EIF-SlideWindow using simulated data and demonstrate that it outperforms standard EIF/EKF-SLAM in both accuracy and efficiency. Additionally, our implementation leverages PyTorch for matrix operations, enabling efficient execution on both CPU and GPU. Additionally, the code for this approach is made available for further exploration and development.

Keywords: SLAM; Kalman filter; extended Kalman filter (EKF); Gaussian noise

Citation: Léon, J.L.; Salgueiro, P.; Gonçalves, T.; Rato, L. EIF-SlideWindow: Enhancing Simultaneous Localization and Mapping Efficiency and Accuracy with a Fixed-Size Dynamic Information Matrix. *Big Data Cogn. Comput.* **2024**, *8*, 193. https://doi.org/10.3390/bdcc8120193

Academic Editor: Salvador García López

Received: 10 November 2024
Revised: 5 December 2024
Accepted: 11 December 2024
Published: 17 December 2024

1. Introduction

Simultaneous Localization and Mapping (SLAM) is a core problem in robotics and autonomous systems involving the estimation of a robot's pose (its position and orientation) and the creation of a map of an unknown environment using sensor data. SLAM is inherently complex and typically requires the integration of multiple sensor types, including cameras, LIDAR, and inertial measurement units (IMUs), to yield accurate and reliable estimates of the robot's pose and the surrounding environment map.

SLAM is commonly framed as a probabilistic inference problem, where the objective is to estimate an optimal robot pose and map by maximizing the posterior probability, based on sensor measurements. Various algorithms exist to solve SLAM, notably the Extended Kalman Filter (EKF) [1], Particle Filters (PFs) [2], and Graph-based SLAM [3].

The EKF (and its variants) and GraphSlam are among the most widely used SLAM algorithms (see examples of applications listed (https://www.cvlibs.net/datasets/kitti/eval_odometry.php, accessed on 1 November 2024) using [4,5]. However, GraphSlam heavily depends on detecting loop closures, which are crucial for correcting accumulated errors but can be challenging to identify. This requires robust and efficient methods to determine when a robot revisits a previously seen location. Additionally, GraphSlam's complexity can increase quadratically with the number of landmarks and poses in the environment.

Other proposals focus on optimizing the efficiency of the algorithm, as demonstrated in approaches that employ sub-maps [6–8]. However, dividing a map into sub-maps can lead to inaccuracies when integrating these sub-maps, as error accumulation across boundaries can disrupt global consistency. Additionally, sub-map management often

requires assumptions about independence or minimal correlation between regions, which may not hold in complex or densely featured environments, potentially degrading accuracy. Moreover, while sub-mapping reduces immediate computational load, it can introduce overhead in tracking and updating sub-map boundaries, which becomes particularly challenging in dynamic or large-scale environments. Lastly, the memory required to store and update multiple sub-maps in real-time scales linearly with the number of sub-maps, potentially constraining performance in resource-limited systems.

Recent advancements in the field include approaches that incorporate deep learning, as outlined in [9]. However, the quasi-analytic solutions derived from the EKF and its SLAM variants remain among the most effective methods currently available [4,10]. Deep learning solutions are increasingly applied in Visual SLAM (camera-based data), benefiting from advancements in image and video processing [11].

The EKF is arguably the most widely used estimation algorithm for nonlinear systems [12,13]. A notable variant of the EKF is the Extended Information Filter (EIF); mathematically, the EKF and EIF are equivalent, but EIF matrices link poses and landmarks, which is critical to our proposed method, EIF-SlideWindow.

EIF-SLAM is a recursive algorithm that estimates the robot's pose and environment map by iteratively updating the state estimate using sensor data. The EIF operates under the assumptions of Gaussian noise and linearity, which may lead to inaccuracies in highly nonlinear systems or in environments with significant non-Gaussian noise. Additionally, the computational complexity of the EIF/EKF can grow substantially with larger environments, resulting in performance issues.

In this paper, we present a novel approach to EIF-SLAM, the EIF-SlideWindow, designed to address these limitations by maintaining accuracy and improving efficiency. Our approach processes each step in constant time, regardless of the trajectory length, making it suitable for real-time scenarios.

Our EIF-SlideWindow approach modifies the EIF algorithm to keep the information matrix and vector at a fixed size. This sliding window is designed to act as a queue, where outdated landmarks are removed, and new ones are inserted as the robot moves, preserving a consistent matrix size along the trajectory. The sliding window is centered on the robot's current pose, allowing for efficient computation without sacrificing recent information.

It is important to note that our approach does not use the sub-map technique; instead, the environment is treated as a whole, maintaining correlation throughout the entire trajectory. However, we leverage the sliding window concept due to the limited influence of more distant or disconnected points relative to the current points [14], allowing us to use only a data window that achieves an optimal balance of efficacy and efficiency in the calculations.

We evaluate our proposed method using simulated data and compare its performance to the standard EIF/EKF-SLAM algorithm. Results indicate that our approach improves both accuracy and efficiency. We further demonstrate the real-time capabilities of our approach, highlighting its potential for practical applications.

For experimentation, we developed our code in Python, leveraging PyTorch for efficient matrix operations and compatibility with both CPU and GPU processing. This setup also facilitates future integration with the Python Rosbag package for real-data evaluation.

The rest of this paper is organized as follows: Section 2 provides a detailed overview of the EIF-SLAM algorithm and its limitations. Section 3 describes our proposed EIF-SlideWindow approach. Section 4 presents our experimental results, comparing the performance of our approach with that of the standard EKF and EIF-SLAM algorithms. Finally, in Section 5, we conclude the paper and discuss future research directions.

2. Extended Information Filters (EIFs)

The Extended Information Filter (EIF) builds on the same mathematical foundation as the Extended Kalman Filter (EKF) to solve the Simultaneous Localization and Mapping (SLAM) problem, but it represents the information in a different form. The EKF extends

the standard Kalman filter to handle nonlinear systems, which makes it suitable for SLAM applications. Both the EKF and EIF estimate a posterior distribution over the robot state, denoted by E, which is modeled as a multivariate Gaussian distribution. The posterior distribution is conditioned on previous measurements (e.g., from LIDAR) and past robot motion commands.

The mathematical formulation of the EKF/EIF assumes that measurement errors follow a Gaussian distribution $N(\mu, \sigma^2)$, whose probability density function, evaluated at $x = E$, is given by:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{1}$$

When applying the EKF/EIF to the SLAM problem, the likelihood function seeks to estimate the most probable configuration of the state E given the robot motion commands and measurements (e.g., LIDAR-based landmark detections). Since we are interested in the maximum-likelihood estimate, rather than the absolute probability values, we can disregard constant factors in Equation (1). Thus, Equation (1) simplifies to:

$$f(x \mid \mu, \sigma^2) \propto e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{2}$$

Given the large number of measurements typically available in real-world scenarios (e.g., multiple landmark detections per robot step), a matrix representation is more computationally efficient.

Let p_t represent the robot's pose at time t, where we define $p_t = (x_t, y_t, \theta_t)$. Here, x_t and y_t are the Cartesian coordinates at time t, and θ_t is the robot's orientation (heading angle) from time $t-1$ to t. Let N denote the total number of landmarks in the environment, with each landmark l_n defined by its Cartesian coordinates (x_n, y_n), where $1 \leq n \leq N$.

The complete state E of the environment at time t can then be defined as:

$$E_t = (p_t, l_1, \ldots, l_N), \tag{3}$$

which can be expanded to:

$$E_t = (x_t, y_t, \theta_t, x_1, y_1, x_2, y_2, \ldots, x_N, y_N).$$

Equation (3) represents a vector that includes the robot's pose at time t, as well as the coordinates of all landmarks in the environment (see Figure 1).

Then, the posterior distribution of the robot state is defined as:

$$p(E_t \mid z^t, u^t) \propto e^{-\frac{1}{2}(E_t-\mu)^T S_t^{-1}(E_t-\mu)}, \tag{4}$$

where:

- $z^t = \{z_1, \ldots, z_N\}$ denotes the set of observations up to time t, with each observation $z_n = (\text{dist}_n, \theta_n)$ representing the distance and bearing from the robot's pose p_t to a landmark l_n, for $1 \leq n \leq N$.
- $u^t = \{u_1, \ldots, u_t\}$ represents the sequence of motion commands up to time t, with each motion command $u_t = (\text{dist}_t, \theta_t)$ specifying the distance and bearing over the interval $[t-1 : t]$, i.e., from pose p_{t-1} to p_t.
- μ is the mean of the distribution.
- S_t is the covariance matrix.
- T denotes the transpose operation.

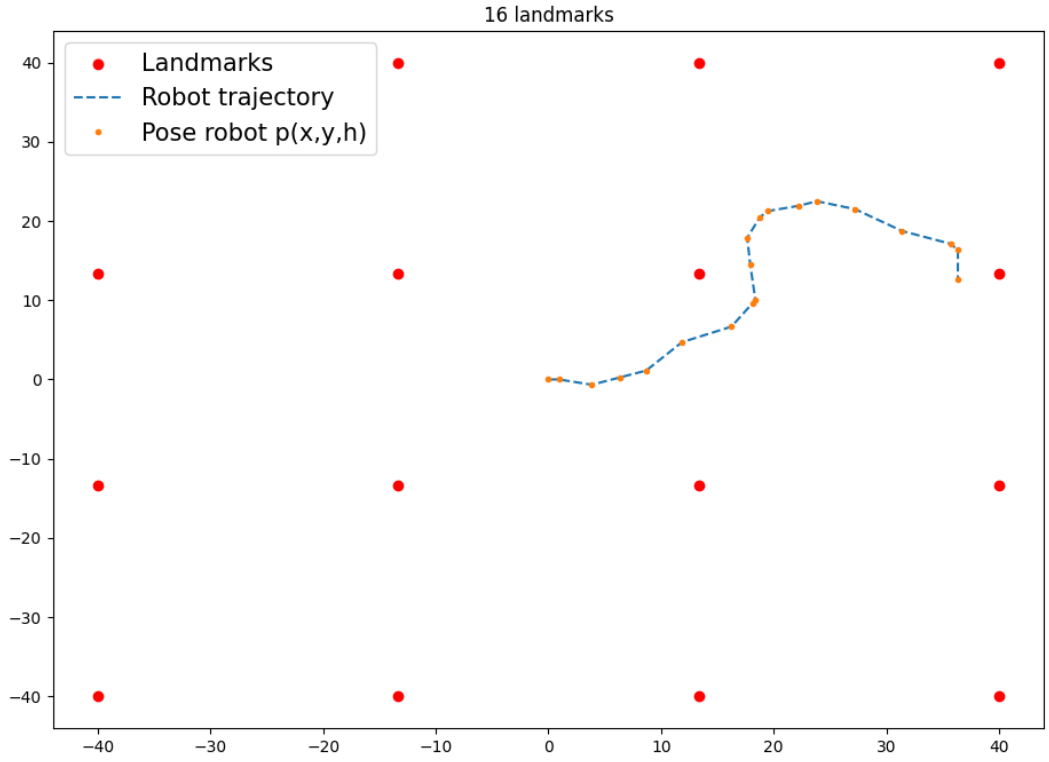

Figure 1. Example of trajectory and landmarks.

Starting from Equation (4), we expand the exponent term, yielding:

$$p(E_t \mid z^t, u^t) \propto e^{-\frac{1}{2}E_t^T S_t^{-1} E_t + \mu^T S_t^{-1} E_t - \frac{1}{2}\mu^T S_t^{-1}\mu}. \tag{5}$$

Following the simplification used in Equation (1), the constant term (last term in Equation (5)) can be removed without affecting the outcome of this development. Thus, we have:

$$p(E_t \mid z^t, u^t) \propto e^{-\frac{1}{2}E_t^T S_t^{-1} E_t + \mu^T S_t^{-1} E_t}. \tag{6}$$

The term S_t^{-1} in Equation (6) is known as the information matrix, denoted by H_t, and $\mu^T S_t^{-1}$ is called the *information vector*, denoted by b_t. Therefore, we define:

$$H_t = S_t^{-1}, \quad b_t = \mu^T S_t^{-1} = \mu^T H_t.$$

Using this alternative representation, we obtain the Extended Information Filter (EIF) formulation:

$$p(E_t \mid z^t, u^t) \propto e^{-\frac{1}{2}E_t^T H_t E_t + b_t^T E_t}. \tag{7}$$

Finally, the estimate of the state E_t can be computed as:

$$E_t = H_t^{-1} b_t. \tag{8}$$

From a mathematical perspective, the EKF and EIF formulations are equivalent. However, the information matrix H_t is symmetric, and its entries directly correspond to the elements (i.e., poses and landmarks) in the environment. Figure 2 illustrates the matrix H_t and its relationships with the robot's poses and landmarks in the environment.

This property will be crucial in our proposed approach, as the information matrix H and vector b will serve as the basis for the sliding-window EIF.

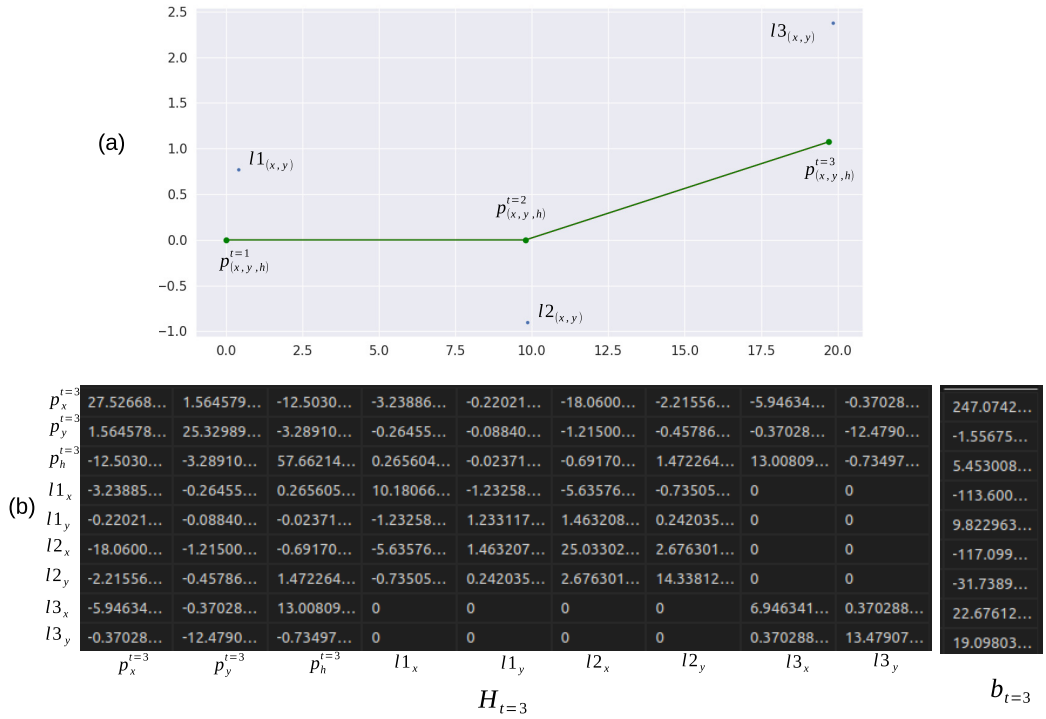

Figure 2. Information matrix $H_{t=3}$ at time step $t = 3$, showing the connections between the robot's pose $P^{t=3}$ and the environmental landmarks. Panel (**a**) depicts the robot's pose and the environmental landmarks, while panel (**b**) shows the information matrix $H_{t=3}$ (**left**) and the information vector $b_{t=3}$ (**right**).

According to the SLAM process, with each LIDAR measurement, the information matrix H and vector b must be updated as follows:

1. Prediction of the robot pose and covariance S from $(t-1)$ to t;
2. Update (or correction) of the robot pose and covariance: Adjust the state estimate with the new LIDAR measurements and refine the covariance.

It is worth noting that the entries in H_t in Figure 2b do not directly represent coordinates in the environment as shown in Figure 2a.

In a real-world scenario, the robot operates in a nonlinear environment (such as 2D space with x and y coordinates), which complicates the analytical computation of the Gaussian distribution. To address this, a linearization process is commonly used, making it possible to apply either the Extended Kalman Filter (EKF) or the Extended Information Filter (EIF).

To incorporate both the motion command $u_t = (\text{dist}, \theta)$ (between consecutive poses) and LIDAR measurements (between pose and landmark), both quantities must be linearized before updating the information matrix H and vector b.

Following traditional Kalman filter methodology, we approximate nonlinear functions with a first-order Taylor series expansion, replacing the first derivative with the Jacobian matrix of partial derivatives.

We avoid further mathematical details here, which are available in [15]. However, a critical component of the Kalman filter algorithm is this linearization process, applied in each of the two steps outlined above.

In the first step (motion prediction), the next robot pose is modeled by a deterministic nonlinear function g plus independent Gaussian noise:

$$E_t = E_{t-1} + \Delta_t \quad \text{where} \quad \Delta_t = g(E_{t-1}, u_t) + P\delta_t, \tag{9}$$

where Δ_t is the change from state E_{t-1} to state E_t, δ_t is a Gaussian random variable with zero mean and covariance determined by the sensor properties (LIDAR or IMU) or manually

set for simulated scenarios, P is a projection matrix that scales δ_t, and g is the nonlinear function of the robot's motion.

The function g can be approximated using a first-degree Taylor series expansion as follows:

$$g(E_{t-1}, u_t) \approx g(\mu_{t-1}, u_t) + \frac{\partial g(\mu_{t-1}, u_t)}{\partial E}(E_{t-1} - \mu_{t-1}), \tag{10}$$

where the Jacobian matrix ∂g (also written as $\frac{\partial g(\mu_{t-1}, u_t)}{\partial E}$) captures the partial derivatives of g with respect to the state E.

In the second step (update or correction), a second deterministic nonlinear function h plus independent Gaussian noise ϵ_t is used to model the robot's measurements, e.g., LIDAR observations:

$$l_t = h(E_t) + \epsilon_t, \tag{11}$$

where h is similarly approximated by a Taylor series expansion:

$$h(E_t) \approx h(\mu_t) + \frac{\partial h(\mu_t)}{\partial E}(E_t - \mu_t). \tag{12}$$

Here, ∂g and ∂h are the Jacobians of g and h, respectively.

Choosing the values of μ_{t-1} and μ_t is crucial for accurate linearization of these nonlinear functions. A typical choice is to use the following approximation:

$$\mu_t = b_{t-1} H_{t-1}^{-1} + \hat{\Delta}_{xy\theta},$$
$$\mu_{t-1} = b_{t-1} H_{t-1}^{-1},$$

where $\hat{\Delta}_{xy\theta}$ represents the displacement in x, y, and θ from time $(t-1)$ to t, calculated using $u_t = (\text{dist}, \theta)$. This estimate may lead to drift over time, as errors accumulate with each movement. One solution is to repeat the linearization several times, each time using progressively better estimates to linearize g and h, achieving convergence after sufficient iterations [16].

However, this iterative approach can be computationally expensive, especially during the update step, as each landmark measurement requires a separate linearization and update. The prediction step, on the other hand, involves a single linearization of g.

Finally, we can express H and b for the two steps as follows:

1. Prediction of the robot pose:

$$\hat{H}_t = \left[(I + \partial g) H_{t-1}^{-1} (I + \partial g)^T + P \delta_t P^T \right]^{-1}, \tag{13}$$

$$\hat{b}_t = \mu_t^T H_t. \tag{14}$$

2. Update or correction of $\hat{H}_t$ and $\hat{b}_t$:

$$H_t = \hat{H}_t + \partial h \epsilon^{-1} \partial h^T, \tag{15}$$

$$b_t = \hat{b}_t + \left(Z + \partial h^T \mu \right) \epsilon^{-1} \partial h^T, \tag{16}$$

where:

$$Z = h(\xi_t) - h(\mu_t),$$

and ξ_t is an initial estimate of the robot state, computed as $\xi_t = g(\xi_{t-1}, u_t)$, with $\xi_{t=0} = E_{t=0}$.

Here, I is the identity matrix with dimensions $[\dim(E), \dim(E)]$.

3. Our Proposal: EIF-SlideWindow

As is well known, nonlinear Kalman filter (KF) variants share a common limitation: computational cost and memory requirements grow with the trajectory length, even in environments with limited spatial extent.

Specifically, the update step in the Extended Kalman Filter (EKF), where measurements for N active landmarks are processed at each robot timestep, represents a computational bottleneck.

For the EKF, the dimensions of the observation matrix H and the measurement vector b are determined primarily by the total number of timesteps T and the number of landmarks.

In practical scenarios, two distinct cases may arise:

1. Online Estimation : Data from sensors (e.g., LIDAR) are processed in real-time at each timestep. Here, the final values of T and N are unknown, so H and b grow incrementally as new observations are incorporated.
2. Offline Estimation: All data are collected before processing, so T and N are fixed in advance.

In both cases, significant data volumes must be handled. For instance, a typical LIDAR sensor may capture thousands of points per second, posing challenges in both processing time and memory usage.

This results in a key trade-off for the EKF algorithm and its variants between accuracy, computational load, and memory usage. Reducing the number of landmarks or robot poses over a fixed trajectory length reduces both memory and processing time but at the cost of reduced estimation accuracy and vice versa.

In this paper, we propose a modification to the Extended Information Filter (EIF) algorithm to maintain fixed dimensions for H and b. At each timestep, newly observed landmarks are incorporated and the robot's current state is updated. To keep the dimensions of H and b constant, we employ a sliding-window approach where older landmarks are removed from consideration as new ones are added. This window centers on the current robot pose, effectively maintaining a fixed-size state representation over the entire trajectory.

Figure 3 illustrates the sliding window (highlighted in blue) associated with a robot pose (indicated by a circle) at time t. As shown in Figure 3, the state E_t consists of the current robot pose and the landmarks within the window (see Equation (3)).

Figure 3. Slide window (square) associated with a robot pose (highlighted with a circle); $RectangleSize = size(H)$.

This strategy aims to achieve constant processing time as the robot continues its measurements along its path.

To mitigate the potential loss of accuracy due to reduced constraints (i.e., fewer landmarks), the time saved can be reallocated to enhancing the linearization process,

particularly during the prediction step, which has the potential to significantly improve estimation accuracy. However, it is important to note that this aspect has not been explicitly explored within the scope of the current work.

3.1. Size of Sliding Window

At each robot pose, a large set of landmarks in the environment is detected. However, only a fixed subset is associated with each pose for use in the EIF algorithm. The number of landmarks associated with each pose depends on factors such as computational power, available memory, and the desired accuracy or error in the trajectory estimation.

By adjusting the sliding window size, these variables (computing load, accuracy, and memory usage) can be balanced. In real-time applications, the sliding window size is often constrained by available memory and computational resources.

We define the sliding window size SW as:

$$SW = \left(\dim(p) + k \cdot N_{lp} \cdot \dim(l)\right) \qquad k, N_{lp} \geq 1 \tag{17}$$

where N_{lp} is the number of landmarks selected per pose step, k is the number of previous robot steps, $\dim(p)$ is the dimension of the robot state vector, and $\dim(l)$ is the dimension of the measurement vector. For example, in a planar setting, $\dim(p) = 3$ and $\dim(l) = 2$, where the robot features vector p is $(\text{dist}_x, \text{dist}_y, \theta)$ and the measurement features vector l is (dist, θ).

Using SW, we can now define the dimensions of H, b, and the state vector as follows:

$$\text{size}(H) = [SW, SW]$$

$$\text{size}(b), \text{size}(\text{state}) = [SW, 1]$$

Thus, SW determines the dimensions of H, b, and the state vector for the EIF-SlideWindow algorithm.

3.2. Updating H, b, and E According to the Sliding Window Size

In our approach, *updating* refers to the process of *remove*, *move*, or *insert* operations on a matrix or vector. Specifically, we define these operations as follows:

$$E_{\text{updating}} = \text{Upd}_E(E, L_t) \Rightarrow \text{Chain Process}(\text{remove} \rightarrow \text{move} \rightarrow \text{insert}) \tag{18}$$

$$[H_{\text{updating}}, b_{\text{updating}}] = \text{Upd}_{Hb}(H, b) \Rightarrow \text{Chain Process}(\text{remove} \rightarrow \text{move}) \tag{19}$$

where L_t is the set of landmarks associated with the current robot pose. This *updating* task incurs minimal computational cost.

Note that until the number of robot steps reaches k (as defined in Equation (17)), no updating is performed. Once updating begins on H, b, and the state vector, the oldest block of landmarks of size N_{lp} is removed, and the remaining blocks are shifted to make room for the new landmark block. Figures 4–6 illustrate the updating process for the state vector, vector b, and matrix H, respectively.

Algorithm 1 presents a detailed pseudocode of our proposed algorithm, EIF-SlideWindow, closely following the structure of the corresponding Python implementation. We incorporate both mathematical notation and Python-style pseudocode to enhance clarity and facilitate understanding of the algorithm's logic and flow. Line 19 performs the *updating* process, while lines 24 to 31 carry out the prediction step, during which the linearization process can also be refined.

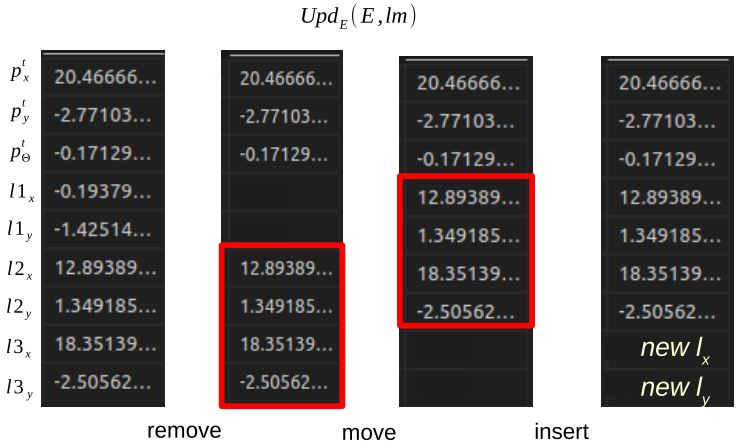

Figure 4. A simple example of updating the state vector with $N_{lm} = 1$ and $k = 3$. The red box highlights the items that are moved during the update.

Figure 5. Simple example of updating matrix b where $N_{lm} = 1$ and $k = 3$. The red box highlights the items that are moved during the update.

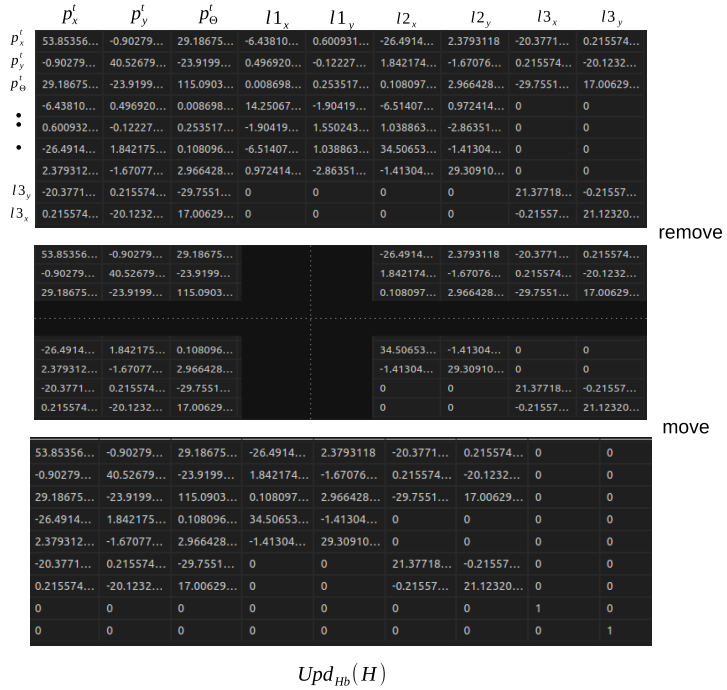

Figure 6. Updating the matrix H with $N_{lm} = 1$ and $k = 3$. The middle image is the result of "remove" items from the top image, and the bottom image is the result of "move" items from the middle image.

Algorithm 1 EIF-SlideWindow Algorithm

Require:

Commands $\rightarrow u_t(dist, \theta_{rumbo})$.
$N_l p$, landmarks number by pose.
k, number of robot previous steps
$L_t\{l_1, l_2, \ldots l_{n=N_{lm}}\}$ Landmarks from a robot step

Initialize:

$ListStates = [], t = 1, maxItr > 0$
$SW = (dim(p) + k \times N_{lm} \times dim(l)) \quad dim(p) = 3, dim(l) = 2$
State $E_{t=0} = zeros(SW, 1)$, insert $p_{t=0}$ and $L_{t=0}$
Simple estimate State $\xi_{t=0} = E_{t=0}$
Information matrix $H_{t=0} = diagOnes\Big(SW, SW\Big)$
Information vector $b_{t=0} = E_{t=0}^T H_{t=0}$
Covariance Motion $\delta \rightarrow size\Big(dim(p), dim(p)\Big)$
$noise_{mot} = RanMulN\ \Big(mean{=}0, \delta\Big) \rightarrow size\Big(dim(p), 1\Big)$
Covariance measurements $\varepsilon \rightarrow size\Big(dim(l), dim(l)\Big)$
$noise_{mes} = RanMulN(mean{=}0, \varepsilon) \rightarrow size\Big(dim(l), 1\Big)$
while $L_t \neq$ NULL **do**
  $\xi_t = \xi_{t-1} + \Delta_{xy\theta} + noise_{mot}$
  $\Delta_{xy\theta} = [dist_{u_t} \times cos(\theta_{\xi_{t-1}}), dist_{ut} \times sin(\theta_{\xi_{t-1}}), \theta]$
  $E_t = E_{t-1} + \Delta_{xy\theta}$
  if $t > k$ **then**
    $\xi = Upd_E(\xi, L_t) \quad E = Upd_E(E, L_t)$
    $H = Upd_H b(H) \quad b = Upd_{Hb}(b)$
  else
    $\xi \leftarrow \begin{bmatrix} \xi \\ L_t \end{bmatrix}$ and $E \leftarrow \begin{bmatrix} E \\ L_t \end{bmatrix}$
  end if
  while $itr \leq maxItr$ **do** ▷ Prediction and linearization
    $\mu_{t-1} = bH^{-1}$
    $\mu = \mu_{t-1} + \Delta_{xy\theta} \qquad \Delta_{xy\theta} = [dist_{u_t} \times cos(\theta_{\mu_{t-1}}), dist_{u_t} \times sin(\theta_{\mu_{t-1}}), \theta]$
    $G_{mot} = diagOnes(ZW, ZW)$
    G_{mot}[2,0:2]=$[dist_{u_t} \times (-sin(\theta_\mu)), dist_{u_t} \times (cos(\theta_\mu))]$
    $H = G_{mot} H^{-1} G_{mot}^T + P\delta P^T$
    $b = \mu^T H$
    $itr+ = 1$
  end while
  for each $l \in E_t$ **do** l is x,y ▷ measure Update
    if l is considered from this robot pose **then**
      $\mu_t = bH^{-1}$
      $z = [dist, difAng] + noise_{mes} \qquad dist{=}L2_{dis}(\xi_t, l); \qquad difAng = atan^2(\xi, l) - \theta_{\xi}$
      $\hat{z} = [dist, difAng] \qquad dist{=}L2_{dis}(\mu_t, l); \qquad difAng = atan^2(\mu, l) - \theta_{\mu}$
      $C{=}zeros(ZW, dim(l))$
      $C[0{:}dim(p){-}1,0]$ = diff/d diff=$\mu_t - l$; d=$L2_{dis}(\mu_t, l)$
      $C[ind_l, 0]$ = $-$(diff/d) ind_l is the index of l in C
      $C[0 : dim(p){-}1,1]$ = [$-$diff[1]/d^2, diff[0]/d^2]
      $C[ind_l, 1]$ = [diff[1]/d^2, $-$diff[0]/d^2]
      $C[dim(p){-}1,1] = -1$
      $H = H + C\varepsilon^{-1}C^T$
      $b = b + \Big((z - \hat{z}) + C^T \mu_t\Big)\varepsilon^{-1}C^T$
    end if
  end for
  $ListStates.Add(bH^{-1})$
  $t+ = 1$
end while

4. Experimental Design

To validate our proposal, we present results that demonstrate its accuracy and efficiency through the use of simulated data, offering a controlled environment for rigorous evaluation. This methodology enables benchmarking across a range of predefined scenarios, ensuring that the algorithm adheres to established performance standards. By leveraging simulations, we can conduct a comprehensive assessment, allowing for refinement and optimization prior to real-world implementation.

The experiments were conducted on a computer with the following specifications:

- RAM: 15 GiB
- Processor: $Intel\ Core\ i5-8300H\ CPU$ @ 2.30 GHz $\times$8 cores
- GPU: $GeForce\ GTX\ 1050\ Ti$ with 4 GiB of VRAM

Using the pseudocode in Algorithm 1, we developed an implementation of the EIF-SlideWindow algorithm in Python. To optimize performance, particularly for matrix operations, we leveraged the PyTorch library, allowing us to utilize both CPU and GPU processing for comparative analysis.

To evaluate trajectory estimation accuracy and computational efficiency, we employed the Extended Kalman Filter (EKF) and the standard Extended Information Filter (EIF) as reference benchmarks. The code for the EKF and EIF, along with parameter initialization, was obtained from [15] (https://github.com/theevann/SLAM, accessed on 1 November 2024). This setup enables direct comparison with validated implementations, and all algorithms were executed under identical hardware conditions to ensure a fair comparison.

In accordance with [15], the following parameters were defined:

- The noise associated with robot motion and sensor measurements is modeled by zero-mean Gaussian noise:

$$\delta = \begin{pmatrix} 0.0001 & 0 & 0 \\ 0 & 0.0001 & 0 \\ 0 & 0 & 0.001 \end{pmatrix}$$

$$\varepsilon = \begin{pmatrix} 0.002 & 0 \\ 0 & 0.003 \end{pmatrix}$$

- Dimension of robot state: $dim(p) = 3$; dimension of measurement features: $dim(l) = 2$.
- Number of active landmarks per robot step: $N_{lm}^{active} = 10$, as specified in line 33 of Algorithm 1.
- Length of the trajectory (number of robot steps): $T = 100$.
- Command for robot motion: $u(dist, \theta)$, where $0 \leq dist \leq 5$ and $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$.
- Number of iterations for linearization: $maxItr = 1$, as we do not consider iterations to improve linearization in these experiments.

It is important to distinguish between N_{lm}^{active} and N_{lm}. In this context, N_{lm}^{active} refers to the number of landmarks accepted (true) in line 33, while N_{lm} denotes the number of landmarks selected by each robot step, typically derived from LIDAR measurements.

During each robot pose update, landmarks are measured from the surrounding environment. In this implementation, N_{lm} landmarks are distributed around each pose within an area of approximately 100 meters, consistent with standard LIDAR configurations.

To assess the accuracy of the estimated trajectory, we utilize the Root Mean Square Error (RMSE), specifically the RMSE between the estimated trajectory and the actual motion. We employ the Python package `evo` [17] for this purpose (https://github.com/MichaelGrupp/evo, accessed on 1 November 2024).

Initially, it is crucial to understand the accuracy behavior for varying slide window (SW) sizes. Figure 7 illustrates the results of accuracy (RMSE) as the size of SW is incrementally increased.

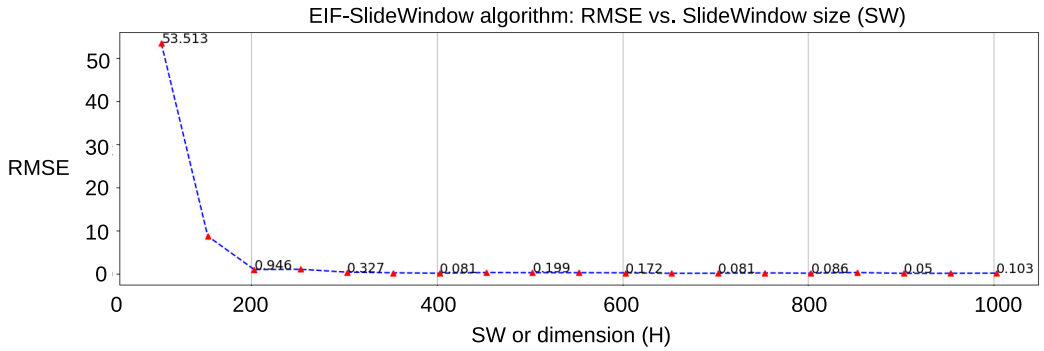

Figure 7. EIF-SlideWindow accuracy (RMSE) behavior in the trajectory estimation for different SW sizes.

Note that with each iteration, where the slide window (SW) size is fixed, a trajectory consisting of 100 steps is generated. Each trajectory is distinct because the robot's motion is determined by a random command u (see Figure 8). Furthermore, the minor variations in RMSE observed for $SW > 400$ can be attributed to the random Gaussian noise added to both motion and measurement processes. Notably, the RMSE for the EIF-SlideWindow algorithm stabilizes when $SW > 400$.

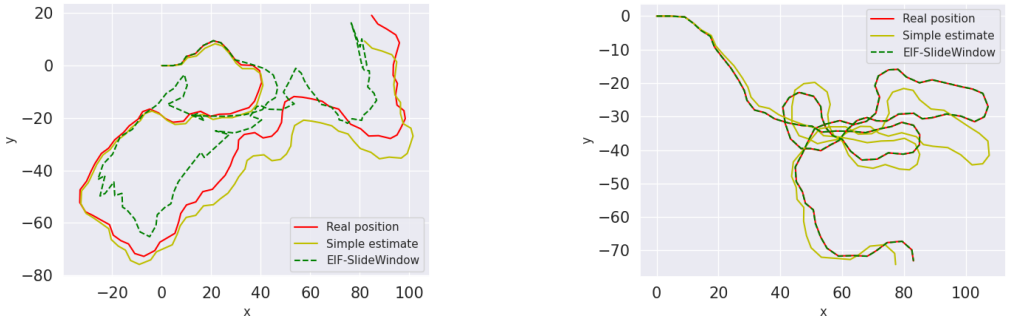

Figure 8. According to Figure 7, estimated trajectory with SW = 103 and RMSE = 53.5 on right and estimated trajectory with SW = 1003 and RMSE = 0.1 on left.

On the other hand, we need to compare our approach with the results obtained from the EKF and EIF algorithms. Given our approach, it is reasonable to anticipate a decrease in accuracy due to the utilization of fewer landmarks. As in the previous experiment, we adhere to the parameters established in [15], as defined above. Table 1 presents the results obtained.

Table 1. Results using a total of 400 landmarks, with 10 active landmarks (N_{lm}^{active}) per pose over 100 robot steps. The EKF website was accessed on 5 June 2024.

Methods	N_{lm}^{active}	SW	RMSE	Time (s) per Step
EKF (https://github.com/theevann/SLAM)	10	803	0.57	0.31
EIF (https://github.com/theevann/SLAM)	10	803	0.55	1.36
EIF-SlideWindow	10	803	0.1	0.13
EIF-SlideWindow	10	403	0.08	0.06

From Table 1, we observe that our proposal yields superior results in trajectory estimation. This improvement may stem from the fact that the number of active landmarks per robot pose (as indicated in line 32 of Algorithm 1) is fixed to manage computational costs across all algorithms in the state of the art. This constraint on accuracy arises from not utilizing all available landmarks. However, it is important to note that an abundance of non-active landmarks may adversely affect accuracy.

Our experimental results indicate a relationship among the number of active landmarks, the slide window size SW (which corresponds to the size of the information matrix H), and the RMSE (see Figure 9). It is evident that achieving a stable minimum RMSE while increasing the number of active landmarks necessitates a corresponding increase in SW.

Additionally, from Figure 7 and Table 1, we can conclude that our approach demonstrates improved accuracy compared to the EKF for values of $SW > 400$.

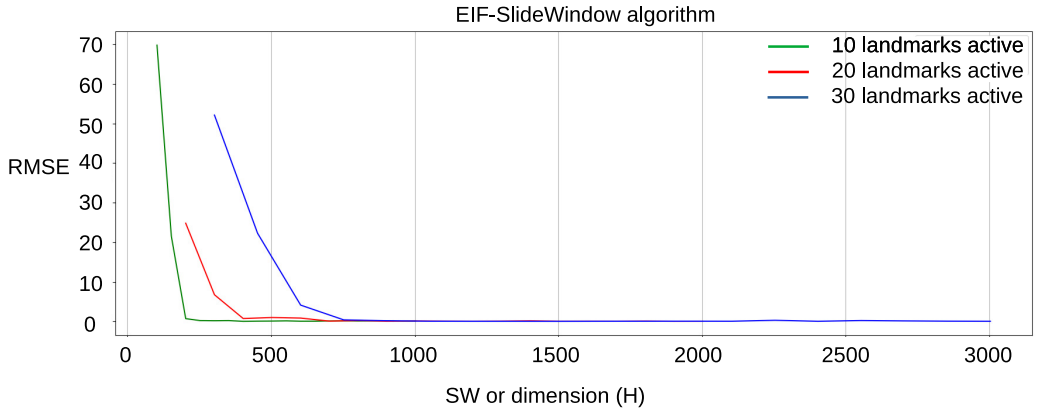

Figure 9. The result shows that we need to increase SW when more active landmarks are used.

The next aim is to evaluate the efficiency of the EIF-SlideWindow algorithm. In this case, we observe that the algorithm maintains a constant processing time regardless of trajectory length, given that both the slide window (SW) and the number of landmarks (N_{lm}) are fixed. Figure 10 illustrates the results for robot trajectories ranging from 100 to 1000 steps, with $SW = 1203$ and $N_{lm}^{active} = 10$.

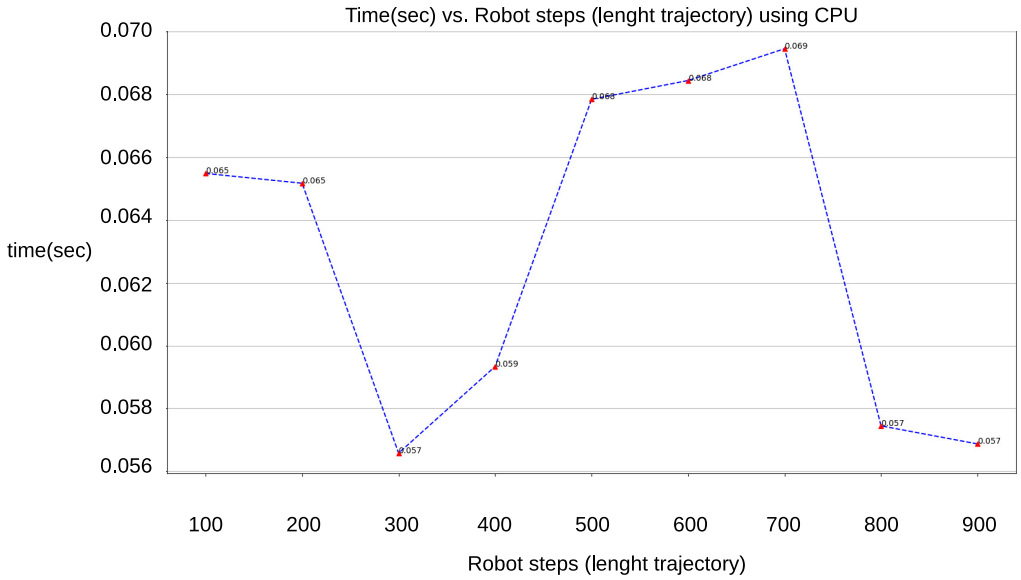

Figure 10. Average time per robot step for varying trajectory lengths with a fixed slide window size of $SW = 1203$.

Finally, we highlight the advantages of utilizing a GPU. When using the CPU, the processing time increases exponentially after $SW > 1800$ (see Figure 11, top). In contrast, the GPU exhibits a linear increase in processing time (see Figure 11, bottom). Specifically, we compute the average processing time (in seconds) for each robot pose in a trajectory, with SW fixed. According to Algorithm 1, this average time accounts for all iterations between lines 15 and 48, which correspond to a single trajectory.

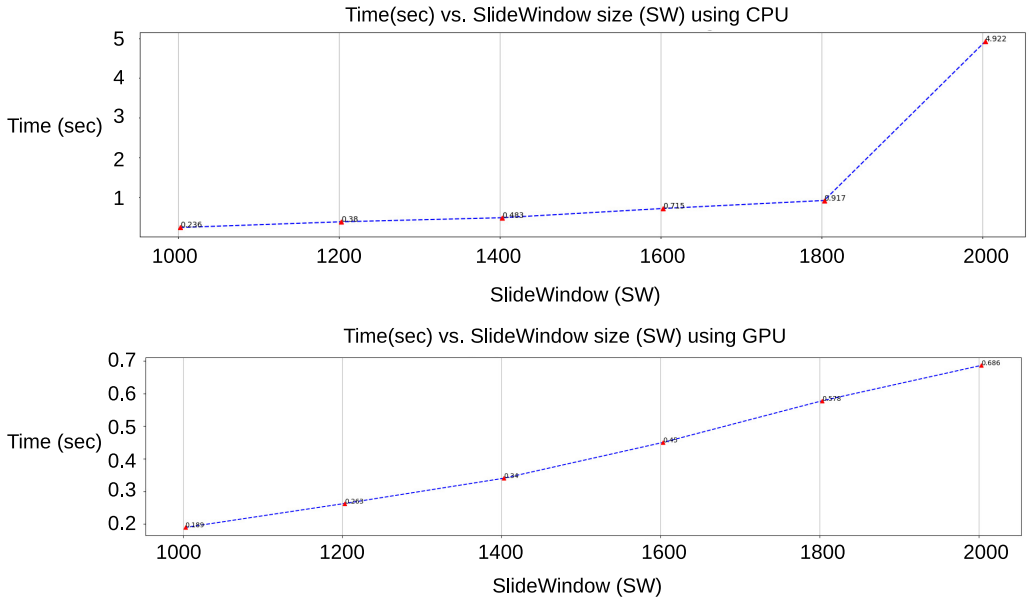

Figure 11. Processing time when the GPU is used at the **top** and CPU processing time at the **bottom**.

4.1. Performance Considerations in Complex Environments

The sliding window approach offers distinct advantages and potential limitations in moderately complex environments. By maintaining a fixed-size state representation focused on recent poses and landmarks, it ensures computational efficiency and real-time performance by reducing memory and processing demands. In environments with moderate landmark density or overlapping features, the constrained representation may reduce accuracy by excluding global correlations. However, its adaptability to local data makes it effective in scenarios where complexity challenges global methods but remains manageable for local optimization. This balance highlights its practical benefits and areas for adaptation. Additionally, retaining the entire trajectory in RAM or physical memory (see Algorithm 1 line 49) enables retrieval and reinsertion to updates process (lines 20, 21).

4.2. EIF-SlideWindow: Parameterization and Design Guidelines

The window size in the EIF-SlideWindow algorithm directly affects its performance, balancing computational load, estimation accuracy, and adaptability to environmental characteristics. A larger window size retains more landmarks and historical data, which should improve global correlation and trajectory consistency, particularly in environments with dense or overlapping features. However, this comes at the cost of increased computational and memory demands, potentially hindering real-time performance. Conversely, a smaller window size reduces computational overhead and enhances efficiency but may sacrifice accuracy in complex environments by excluding critical correlations.

4.2.1. Relationship with Environmental Characteristics

Environmental complexity plays a pivotal role in determining the optimal window size. For sparse environments with well-separated landmarks, a smaller window suffices, as the reduced density minimizes the risk of losing important correlations. In contrast, medium-to-high-complexity environments, where landmarks are denser or feature overlaps occur, benefit from larger windows to maintain trajectory consistency and accurate mapping.

4.2.2. Computational Load Considerations

The computational load increases quadratically with the window size due to the dimensions of the associated matrices, such as H and b. Larger windows necessitate greater computational resources and may require high-performance hardware for real-time applications. The sliding window approach ensures that computational demands

remain predictable, enabling system designers to select a window size appropriate for the available resources.

4.2.3. Design Guidelines

To balance accuracy and efficiency, we propose the following design guidelines:

1. Assess Environmental Complexity: Choose larger windows for dense or overlapping features and smaller windows for sparse settings with minimal landmark interaction.
2. Consider Computational Resources: Ensure the window size aligns with the available hardware capabilities, particularly for real-time implementations.
3. Optimize Landmark Selection: Implement heuristics or prioritization strategies to ensure that the landmarks selected within the sliding window maximize informational gain and contribute effectively to the accuracy of the algorithm.

5. Conclusions and Future Work

This paper has delved into the workings of the EIF-SlideWindow algorithm, an approach within the field of Simultaneous Localization and Mapping (SLAM). EIF-SlideWindow represents a significant advance in addressing challenges typically encountered in SLAM algorithms. By leveraging the Extended Information Filter, the EIF-SlideWindow algorithm achieves improved efficiency and accuracy, demonstrating a capacity for constant-time performance across varying trajectory lengths.

The EIF-SlideWindow algorithm provides robust solutions to critical issues such as computational complexity and real-time operation constraints, enhancing the viability of SLAM for autonomous systems. The experimental results illustrate the method's effectiveness in maintaining accuracy while controlling computational overhead, especially when compared to traditional EKF and EIF approaches.

Future work will focus on validating this approach with real-world datasets and exploring the effects of increased iterations on the linearization process to further refine accuracy. For dynamic or varying environments, future development will aim to incorporate mechanisms for adapting the window size based on observed changes in landmark density or environmental conditions. These efforts will aim to expand the robustness and application of EIF-SlideWindow in diverse, real-time autonomous systems environments.

Author Contributions: Conceptualization, methodology, and code, J.L.L.; review and editing, P.S., T.G. and L.R. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

SLAM	Simultaneous Localization and Mapping
LIDAR	Light Detection and Ranging
IMU	Inertial Measurement Unit
EKF	Extended Kalman Filter
EIF	Extended Information Filter
PF	Particle Filters

References

1. Ribeiro, M.I. Kalman and extended kalman filters: Concept, derivation and properties. *Inst. Syst. Robot.* **2004**, *43*, 3736–3741.
2. Wills, A.G.; Schön, T.B. Sequential Monte Carlo: A Unified Review. *Annu. Rev. Control Robot. Auton. Syst.* **2023**, *6*, 159–182. [CrossRef]
3. Thrun, S.; Montemerlo, M. The graph SLAM algorithm with applications to large-scale mapping of urban structures. *Int. J. Robot. Res.* **2006**, *25*, 403–429. [CrossRef]

4. Geiger, A.; Lenz, P.; Urtasun, R. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR), Providence, RI, USA, 16–21 June 2012.
5. Tranzatto, M.; Dharmadhikari, M.; Bernreiter, L.; Camurri, M.; Khattak, S.; Mascarich, F.; Pfreundschuh, P.; Wisth, D.; Zimmermann, S.; Kulkarni, M.; et al. Team CERBERUS Wins the DARPA Subterranean Challenge: Technical Overview and Lessons Learned. *arXiv* **2022**, arXiv:cs.RO/2207.04914. Available online: http://arxiv.org/abs/2207.04914 (accessed on 11 December 2024). [CrossRef]
6. Leonard, J.J.; Feder, H.J.S. A computationally efficient method for large-scale concurrent mapping and localization. In *Proceedings of the Robotics Research: The Ninth International Symposium*; Springer: London, UK, 2000; pp. 169–176.
7. Leonard, J. Large-Scale Concurrent Mapping and Localization. *Proc. SPIE* **2000**, *4196*, 370–376.
8. Guivant, J.E.; Nebot, E.M. Optimization of the simultaneous localization and map-building algorithm for real-time implementation. *IEEE Trans. Robot. Autom.* **2001**, *17*, 242–257. [CrossRef]
9. Yang, N.; Stumberg, L.v.; Wang, R.; Cremers, D. D3vo: Deep depth, deep pose and deep uncertainty for monocular visual odometry. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 13–19 June 2020; pp. 1281–1292.
10. Mokssit, S.; Licea, D.B.; Guermah, B.; Ghogho, M. Deep Learning Techniques for Visual SLAM: A Survey. *IEEE Access* **2023**, *11*, 20026–20050. [CrossRef]
11. Han, J.; Dong, R.; Kan, J. BASL-AD SLAM: A Robust Deep-Learning Feature-Based Visual SLAM System With Adaptive Motion Model. *IEEE Trans. Intell. Transp. Syst.* **2024**, *25*, 11794–11804. [CrossRef]
12. Rauniyar, S.; Bhalla, S.; Choi, D.; Kim, D. Ekf-slam for quadcopter using differential flatness-based lqr control. *Electronics* **2023**, *12*, 1113. [CrossRef]
13. Ebadi, K.; Bernreiter, L.; Biggie, H.; Catt, G.; Chang, Y.; Chatterjee, A.; Denniston, C.E.; Deschênes, S.P.; Harlow, K.; Khattak, S.; et al. Present and future of slam in extreme environments: The darpa subt challenge. *IEEE Trans. Robot.* **2023**, *40*, 936–959. [CrossRef]
14. Smith, R.C.; Cheeseman, P. On the representation and estimation of spatial uncertainty. *Int. J. Robot. Res.* **1986**, *5*, 56–68. [CrossRef]
15. Thrun, S.; Koller, D.; Ghahramani, Z.; Durrant-Whyte, H.; Ng, A.Y. *Simultaneous Mapping and Localization with Sparse Extended Information Filters: Theory and Initial Results*; Springer: Berlin/Heidelberg, Germany, 2004.
16. Julier, S.J.; Uhlmann, J.K. Unscented filtering and nonlinear estimation. *Proc. IEEE* **2004**, *92*, 401–422. [CrossRef]
17. Grupp, M. Evo: Python Package for the Evaluation of Odometry and SLAM. 2017. Available online: https://github.com/MichaelGrupp/evo (accessed on 1 August 2024).