algorithms

MDPI

Article

Ego-Motion Estimation for Autonomous Vehicles Based on Genetic Algorithms and CUDA Parallel Processing

Abiel Aguilar-González * and Alejandro Medina Santiago *

Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), San Andrés Cholula 72840, Mexico

* Correspondence: abiel@inaoep.mx (A.A.-G.); amedina@inaoep.mx (A.M.S.)

Abstract: Estimating ego-motion in autonomous vehicles is critical for tasks such as localization, navigation, obstacle avoidance, and so on. While traditional methods often rely on direct pose estimation or AI-based approaches, these can be computationally intensive, especially for small, incremental movements typically observed between consecutive frames. In this work, we propose a brute-force-based ego-motion estimation algorithm that takes advantage of the constraints of autonomous vehicles, which are assumed to have only three degrees of freedom (x, y, and yaw). Our approach is based on a genetic algorithm to efficiently explore potential vehicle movements. By generating an initial seed of random motion candidates and iteratively mutating and selecting the best-performing individuals, we minimize the cost function that measures image similarity between frames. Furthermore, we implement the algorithm using CUDA to exploit parallel processing, significantly improving computational speed. Experimental results demonstrate that our approach achieves accurate ego-motion estimation with high efficiency, making it suitable for real-time autonomous vehicle applications.

Keywords: ego-motion estimation; autonomous vehicles; genetic algorithms

Academic Editor: Binhai Zhu

Received: 29 October 2024
Revised: 23 December 2024
Accepted: 27 December 2024
Published: 3 January 2025

Citation: Aguilar-González, A.; Medina Santiago, A. Ego-Motion Estimation for Autonomous Vehicles Based on Genetic Algorithms and CUDA Parallel Processing. *Algorithms* **2025**, *18*, 19. https://doi.org/10.3390/a18010019

1. Introduction

Estimating ego-motion in autonomous vehicles is critical for tasks such as localization, navigation, and obstacle avoidance [1–4]. Traditional pose estimation methods often rely on direct calculation techniques, which can be computationally expensive due to the need to process large volumes of data and the complexity of calculating incremental movements between consecutive frames. For example, these methods require an exhaustive search for all possible states in high-dimensional spaces, leading to high computational costs, even in environments with limited degrees of freedom (x, y, and yaw) [5,6]. In such scenarios, achieving both accuracy and efficiency becomes a challenge.

Recent advances in artificial intelligence (AI) have led to the development of deep learning-based methods that improve the accuracy of pose estimation. However, these methods often rely on large-scale training datasets, which can be difficult to obtain, particularly in dynamic environments where vehicles must adapt to unknown and evolving obstacles [7,8]. But, AI-based methods may struggle to perform reliably without prior training under specific conditions, which poses a challenge in real-time autonomous vehicle systems where environments change rapidly. In contrast, our approach mitigates the reliance on large datasets by leveraging genetic algorithms, which can work effectively even with limited or no prior training data.

To address these limitations, in this work, we propose a brute-force-based approach that uses genetic algorithms to explore possible vehicle movements. By generating an

initial population of random motion candidates, we iteratively evaluated and selected the best-performing individuals based on image similarity between frames. This method not only improves accuracy but also enhances computational efficiency through parallel processing with CUDA [9,10]. Our application domain is for an autonomous vehicle scenario. Thus, our algorithm is specifically designed for this case, that is, it takes advantage of the constraints of autonomous vehicles, which are assumed to have only three degrees of freedom (x, y, and yaw). Our primary contribution lies in the efficient integration of genetic algorithms and CUDA to achieve real-time performance while maintaining high accuracy. Experimental results demonstrate that our approach effectively estimates ego-motion with significant improvements over traditional approaches.

1.1. Genetic Algorithms

Genetic algorithms (GAs) are optimization techniques inspired by the principles of natural selection and genetics. They are particularly effective in solving complex optimization problems, especially when traditional methods struggle with high-dimensional search spaces. GAs simulate the process of evolution by iteratively improving a population of candidate solutions through selection, crossover, and mutation [11,12].

In the context of ego-motion estimation, GAs explore multiple possible movements of the vehicle in an iterative manner. The process begins with an initial population of random motion candidates, each representing a potential ego-motion trajectory. These candidates are evaluated based on a fitness function, which, in this case, measures image similarity between consecutive frames. The best-performing candidates are selected and, through crossover and mutation, new offspring are generated, allowing the algorithm to explore the solution space while preserving high-quality solutions [13,14].

The parallel nature of GAs makes them particularly well suited for implementation on platforms like CUDA. By evaluating multiple candidate solutions simultaneously, CUDA significantly accelerates the optimization process. This parallelism not only enhances the efficiency of the algorithm but also enables real-time performance in autonomous navigation systems.

1.2. CUDA for Parallel Processing

CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) developed by NVIDIA, which enables the use of graphics processing units (GPUs) for general-purpose computation. Using the massive parallelism inherent in modern GPUs, CUDA allows for the acceleration of complex tasks, significantly reducing the computational time required for large-scale optimization problems [15,16].

In the context of ego-motion estimation, CUDA is employed to parallelize the evaluation of multiple candidate solutions simultaneously. This parallelism accelerates the optimization process of genetic algorithms, enabling real-time performance even for demanding applications in autonomous navigation systems. By distributing the computational workload across thousands of threads, CUDA makes it possible to handle high-dimensional problems efficiently, providing the necessary speed and scalability for real-time systems. The key contributions of integrating CUDA into our genetic algorithm-based solution can be summarized as follows.

- Parallelized Evaluation: CUDA facilitates the simultaneous evaluation of image similarity for multiple motion candidates, significantly reducing the computational time required for each iteration of the genetic algorithm [17,18].

- Optimization of Memory Hierarchies: The ability to manage memory hierarchies and streamline data transfers between the host (CPU) and device (GPU) minimizes latency, ensuring efficient utilization of computational resources.
- Real-Time Capabilities: By addressing computational bottlenecks traditionally associated with large candidate populations, our method ensures rapid iterations, making it suitable for real-time applications.
- Relevance to Autonomous Vehicles: The integration of CUDA enables immediate and precise decision making, which is critical to safe navigation in autonomous vehicle scenarios with strict real-time constraints.
- Novel Combination of Precision and Efficiency: The proposed approach effectively combines high accuracy in the estimation of ego-motion with unprecedented computational efficiency. This dual achievement addresses the challenges of speed and accuracy, making our method particularly suitable for deployment in resource-constrained environments common in autonomous systems.

2. Related Works

Multiple studies have explored both feature-based and CNN-based approaches for ego-motion estimation in autonomous systems. Researchers have investigated feature extraction techniques, such as SIFT, SURF, and ORB, along with their variants, to capture essential motion information from visual data. Additionally, recent works have emphasized the use of deep learning architectures, particularly Convolutional Neural Networks (CNNs), to enhance accuracy and robustness in ego-motion estimation. These efforts have advanced the state of the art by improving feature representation, improving network architecture design, and optimizing computational strategies for real-time applications.

2.1. State of the Art for Ego-Motion Estimation

The current state-of-the-art in ego-motion estimation has presented significant advancements, particularly through the use of deep learning techniques. Approaches based on Convolutional Neural Networks (CNNs) have shown promising results in accurately estimating the motion of vehicles in multiple environments. For example, [19] proposed a novel CNN architecture designed to capture spatial and temporal features from consecutive image frames, achieving notable improvements in the accuracy of motion estimation. This model demonstrates the effectiveness of using deep learning to address the complexities associated with real-world scenarios.

However, traditional methods such as visual odometry have been widely employed to estimate ego-motion. These methods typically rely on feature extraction techniques such as SIFT [20] and ORB [21], which provide robust performance in detecting and matching keypoints across frames. However, they can suffer from limitations under challenging conditions, such as low texture or significant motion blur. Despite their strengths, reliance on feature matching can result in inaccuracies, particularly when the environment lacks distinctive features.

Recent studies have also explored the integration of IMU (Inertial Measurement Unit) data with visual input to enhance ego-motion estimation. For example, ref. [22] demonstrated that the combination of IMU data with visual odometry results in improved accuracy and robustness. This hybrid approach capitalizes on the strengths of both sensors, compensating for the weaknesses of visual-only methods in fast-moving or visually sparse environments. Similarly, recent advances in automotive radar-based systems have demonstrated their potential for robust and accurate estimation of ego-motion [23–25]. Furthermore, event-based methods such as those discussed by [26] highlight innovative

approaches to optical flow and ego-motion estimation, pushing the boundaries of existing technologies.

2.2. Real-Time Ego-Motion Estimation

In the domain of estimation of ego-motion, GAs offer a compelling solution to explore potential vehicle movements within a limited set of degrees of freedom. Each candidate solution represents a possible motion trajectory, and the fitness of each candidate is assessed on the basis of its ability to minimize the difference between image frames. For example, ref. [27] illustrated how GAs can optimize path planning in robotics, demonstrating their adaptability to dynamic environments.

In [28], the authors proposed a new stereo visual odometry system for automotive applications that integrates advanced monocular techniques. The results indicate that adapting these techniques to a stereo context can significantly improve the accuracy and robustness of the system, validated through the results obtained on the KITTI dataset. The authors in [29] proposed an introspective vision for SLAM (IV-SLAM) that addresses the issue of assuming a uniform distribution of errors in feature extraction and matching. Experimental results demonstrate that IV-SLAM can accurately predict error sources in input images and significantly reduce tracking errors compared with previous algorithms.

In [30], a network architecture that processes 3D point clouds in an end-to-end manner was presented without the need for predefined corresponding pairs of points. Experiments on the KITTI Vision Benchmark Suite have shown that the performance is comparable to other works, even by employing significantly fewer parameters. Finally, in [10], an estimation of ego-motion was formulated based on a lookup table approach and a new feature matching algorithm. The results indicate that this methodology not only reduces the complexity of the algorithm but also provides higher estimation accuracy, achieving significant processing speed improvements compared with previous approaches.

3. The Proposed Algorithm

In this section, we present our approach for estimating ego-motion in autonomous vehicle scenarios using a brute-force method. The algorithm is designed to handle three degrees of freedom: x, y, and a single rotational angle. We assume that the vehicle does not move along the z-axis or rotates about two of its axes, allowing for efficient computation of ego-motion.

3.1. Parameter Selection Criteria

In our proposed ego-motion estimation approach, the selection of parameters plays a crucial role in ensuring both computational efficiency and high estimation accuracy. One of the key considerations when choosing parameters is the compatibility with the GPU architecture, which is optimized for processing data in powers of two. This approach maximizes the parallel processing capabilities of modern GPUs, ensuring efficient memory management and fast execution of kernel functions.

We specifically recommend selecting parameters such as the number of candidates or the resolution of image patches in powers of two, such as 32, 64, or 128. This choice stems from the fact that GPU architectures handle memory and data processing more efficiently when data structures are aligned to power-of-two sizes. When parameters are selected in powers of two, memory access becomes more optimized, improving memory coalescing and reducing fragmentation. This, in turn, improves the overall performance of the genetic algorithm running on the GPU.

For example, when defining the number of population candidates in the genetic algorithm, selecting values such as 32 or 64 ensures that memory operations are streamlined.

This alignment improves kernel execution by facilitating better load balance, reducing memory latency, and minimizing the overhead caused by non-aligned memory accesses. Additionally, selecting powers of two helps to optimize the memory hierarchy, particularly with shared memory on GPUs. This optimization ensures that the algorithm can make efficient use of GPU resources, thus reducing the time required for each iteration.

By adhering to these power-of-two values for parameter selection, we achieve both higher processing speeds and more efficient utilization of the available resources, enabling real-time performance for ego-motion estimation in autonomous vehicle systems. These choices align with industry best practices for high-performance parallel computing on GPUs.

3.2. Overview of the Algorithm

In Figure 1, an overview of our algorithm is shown, it consists of the following key steps:

1. Seed Initialization: Randomly generate n candidate movements.
2. Ego-motion Application: Apply ego-motion transformations to the second image and compare it with the first image.
3. Selection and Mutation: Retain the top $n/2$ best candidate movements based on a cost function, and generate $n/2$ new candidates by slightly mutating the retained movements.
4. Iterative Optimization: Repeat the process until a predetermined threshold is reached in the cost function.

For additional details, in Figure 2, a complete flowchart for the proposed algorithm is shown.

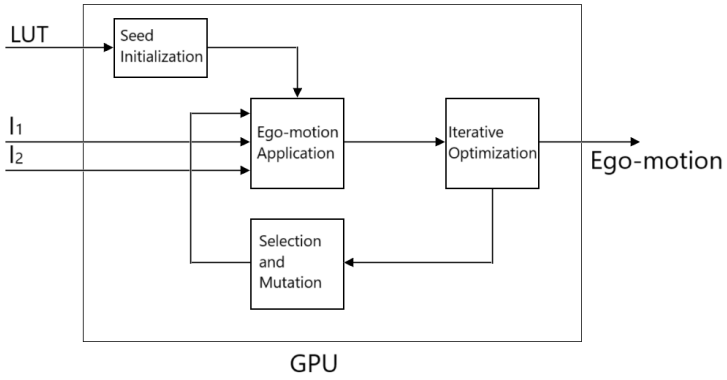


Figure 1. Block diagram of the proposed algorithm.

3.3. Input Lookup Table (LUT)

In this work, we use the KITTI dataset [31], a widely recognized benchmark in the field of autonomous vehicles. The dataset consists of 10 sequences that provide publicly available ground truth for evaluation. Initially, a set of candidate seeds M_k is randomly generated. Each seed represents a potential movement characterized by small displacements in the x and y directions, as well as a rotation angle defined as follows:

$$M_k = \{(dx_k, dy_k, \theta_k)\} \quad \text{for } k = 1, 2, \ldots, N \tag{1}$$

where dx_k and dy_k represent the displacements in the x and y axes, respectively, and θ_k denotes the rotation angle.

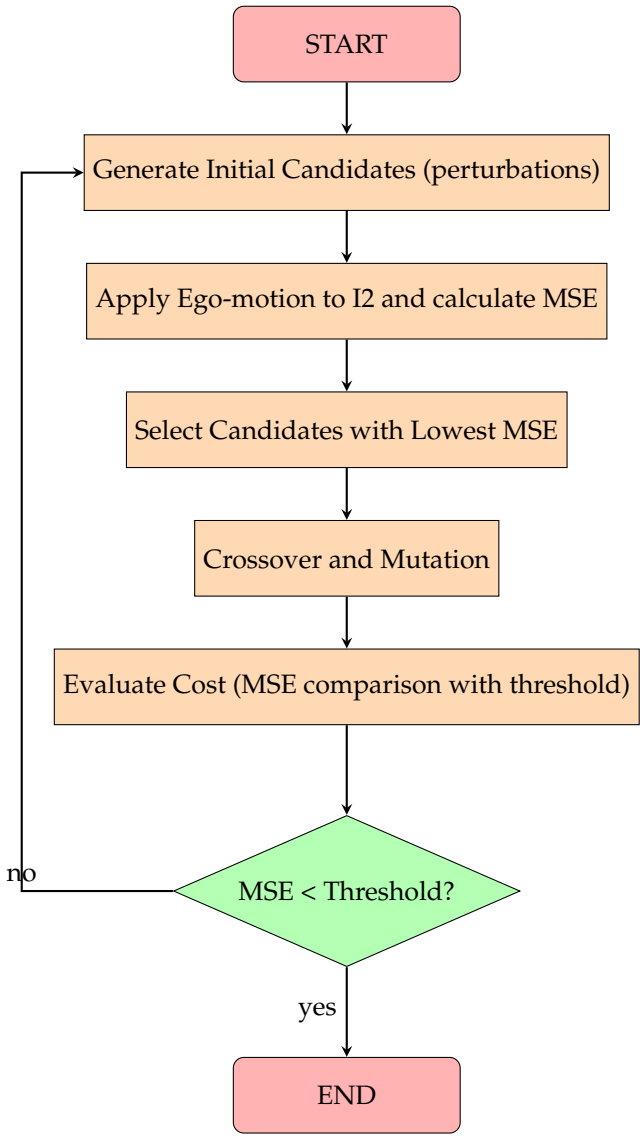


Figure 2. Flow chart of the proposed algorithm.

To ensure that the movements are representative of the actual dynamics observed in the KITTI sequences, we propose a strategy for selecting a subset of movements. This involves analyzing ground-truth data to extract key movements that capture the range of potential maneuvers a vehicle may experience in multiple scenarios. We can define a representative movement set R as

$$R = \{M_j \in M_k \mid \text{movement is statistically significant}\} \tag{2}$$

where M_j are the selected representative movements.

To select a representative set of ego-motion movements from the ground truth data, we propose an approach based on clustering as follows:

1. Clustering: Cluster the ego-motion data based on movement characteristics (we proposed direction and speed.
2. Representative Selection: From each cluster, select a representative movement that captures the key features of that cluster. For more details please see Algorithm 1.

Algorithm 1: Representative Selection of Movements from Ground Truth

Input: Ground truth ego-motion data GT, number of clusters K
Output: Representative movement set R
Extract direction and velocity from GT
$\theta \leftarrow \text{ComputeDirection}(GT)$
$v \leftarrow \text{ComputeVelocity}(GT)$
Perform clustering on (θ, v) into K clusters
$Clusters \leftarrow \text{ClusterEgoMotion}((\theta, v), K)$
Initialize $R \leftarrow \emptyset$
for each cluster $C_k \in Clusters$ **do**
    Select a representative movement M_j from cluster C_k
    Add M_j to R
end for
return R

Note: For practical purposes, it is recommended to use values of K in the form of 2^α to ensure efficient utilization of GPU resources. For more details see Section 3.1.

In order to calculate the direction and velocity of ego-motion from the ground truth data, we analyze the position and orientation information associated with ego-motion at different time frames. Given the positions in two consecutive time frames, $P(t) = (x(t), y(t))$ and $P(t + \Delta t) = (x(t + \Delta t), y(t + \Delta t))$, we can compute the direction and velocity as follows: The direction θ of the ego-motion can be determined using the change in position over time, calculated as

$$\theta = \tan^{-1}\left(\frac{y(t + \Delta t) - y(t)}{x(t + \Delta t) - x(t)}\right) \tag{3}$$

This angle θ represents the direction of motion relative to the x-axis.

The velocity v of the ego-motion is computed based on the displacement over a specified time interval. The displacement D is given by

$$D = \sqrt{(x(t + \Delta t) - x(t))^2 + (y(t + \Delta t) - y(t))^2} \tag{4}$$

Then, the velocity can be calculated as follows

$$v = \frac{D}{\Delta t} \tag{5}$$

where $\Delta t = t_2 - t_1$, t_n are the timestamps in the KITTI dataset.

Once representative movements are established, we construct a lookup table (LUT) that encodes the relationships between these movements and their corresponding outcomes. The LUT is defined as follows:

$$\text{LUT}(M_j) = \{(x', y', \theta')\} \quad \text{for each } M_j \in R \tag{6}$$

where (x', y', θ') are the new states that result from applying the movements M_j.

3.4. Seed Initialization

Given the lookup table (LUT), we generate 2^α random variants of the data from the LUT. Each variant is created by applying small random perturbations to the states (x', y', θ') defined in the LUT.

The perturbation can be expressed as

$$(x_{\text{variant}}, y_{\text{variant}}, \theta_{\text{variant}}) = (x' + \Delta x, y' + \Delta y, \theta' + \Delta\theta) \tag{7}$$

where $(\Delta x, \Delta y, \Delta\theta)$ are small random perturbations.

We can represent the generation of these variants in pseudocode as shown in Algorithm 2.

Algorithm 2: Generate Random Variants from LUT

Input: Lookup table LUT(M_j), number of variants $N = 2^{\alpha}$
Output: Set of generated variants *Variants*
Initialize *Variants* $\leftarrow \emptyset$
for each movement $M_j \in R$ **do**
  for each variant index $i = 1$ to N **do**
    $\Delta x \leftarrow$ RandomPerturbation()
    $\Delta y \leftarrow$ RandomPerturbation()
    $\Delta\theta \leftarrow$ RandomPerturbation()
    $(x_{\text{variant}}, y_{\text{variant}}, \theta_{\text{variant}}) \leftarrow (x' + \Delta x, y' + \Delta y, \theta' + \Delta\theta)$
    Add $(x_{\text{variant}}, y_{\text{variant}}, \theta_{\text{variant}})$ to *Variants*
  end for
end for
return *Variants*(M_k)

Random perturbations can be generated using a CUDA kernel to enable parallel computation (for more details, refer to the cuRAND library documentation provided by NVIDIA [32]), where each thread can handle the perturbation for different movements simultaneously.

3.5. Ego-Motion Application

For each candidate movement M_k, we apply the ego-motion transformation to the second image I_2 to generate a transformed image $I_{2,k}$:

$$I_{2,k} = \text{Ego-motion}(I_2, M_k) \tag{8}$$

The transformation of ego-motion can be expressed as follows: given a movement M_k defined by displacements $(\Delta x, \Delta y)$ and a rotation angle $\Delta\theta$, the transformation for a pixel coordinate (x, y) in the image I_2 can be represented as follows:

$$\begin{aligned} x' &= x \cdot \cos(\Delta\theta) - y \cdot \sin(\Delta\theta) + \Delta x \\ y' &= x \cdot \sin(\Delta\theta) + y \cdot \cos(\Delta\theta) + \Delta y \end{aligned} \tag{9}$$

This defines how each pixel of the image I_2 is transformed based on the movement parameters. The transformed image is then compared with the first image I_1 using a cost function defined in terms of the differences in pixel intensity to evaluate the quality of the candidate movement.

The main goal of this step is to assess how well a transformed image $I_{2,k}$ (obtained through a candidate movement M_k) aligns with the reference image I_1. This alignment is quantified through a cost function that measures the dissimilarity between the two images. For that purpose, we adopt the Mean Squared Error (MSE), which is a widely used measure that computes the average squared differences between pixel intensities of the two images. It is defined as follows:

$$\text{MSE}(I_1, I_{2,k}) = \frac{1}{N} \sum_{i=1}^{W} \sum_{j=1}^{H} (I_1(i,j) - I_{2,k}(i,j))^2 \tag{10}$$

where W and H are the width and height of the images, respectively, and $N = W \cdot H$. For more details please see Algorithm 3.

Algorithm 3: Ego-Motion Transformation and Similarity Evaluation

Input: Images I_1, I_2, movement $M_k = (\Delta x, \Delta y, \Delta\theta)$
Output: Transformed image $I_{2,k}$, MSE value
Initialize $I_{2,k} \leftarrow$ empty image of the same size as I_2
Initialize $MSE \leftarrow 0$
Initialize $N \leftarrow W \cdot H$
for each pixel (x, y) in I_2 **do**
    Compute the transformed coordinates:

$$x' = x \cdot \cos(\Delta\theta) - y \cdot \sin(\Delta\theta) + \Delta x$$
$$y' = x \cdot \sin(\Delta\theta) + y \cdot \cos(\Delta\theta) + \Delta y$$

    if x' and y' are within image bounds **then**
        Assign the pixel value: $I_{2,k}(x', y') = I_2(x, y)$
    end if
end for
for each pixel (i, j) in I_1 **do**
    Compute the squared difference:

$$\text{squared\_difference} = (I_1(i, j) - I_{2,k}(i, j))^2$$

    Update MSE:
    $MSE \leftarrow MSE + \text{squared\_difference}$
end for
$MSE \leftarrow \frac{MSE}{N}$
return $I_{2,k}, MSE$

Parallel Ego-motion Application: Given the computational demands of evaluating multiple candidate movements, we leverage CUDA to implement the ego-motion application in parallel. This allows for the simultaneous processing of numerous image pairs, significantly accelerating the process.

3.6. Selection and Mutation

3.6.1. Selection Mechanism

To select the most promising candidate movements for mutation and crossover, we evaluate their performance based on previously computed mean square error (MSE) values. The selection process involves choosing movements that exhibit lower MSE values, indicating that they are more effective in transforming the images to match the reference.

Let M_k be the set of candidate movements, and let MSE_k represent the MSE associated with each movement M_k. We can define a selection criterion as follows:

$$\text{Select}(M_k) \text{ if } MSE_k < T \tag{11}$$

where T is a predefined threshold for acceptable performance. Movements that meet this criterion will be retained for mutation and crossover operations.

3.6.2. Mutation Mechanism

To maintain genetic diversity and explore new potential solutions, we apply a mutation process to the retained seeds. This involves making small, random adjustments to the parameters of the selected movements. The mutation can be defined as follows:

$$M_k^{\text{mutated}} = M_k + \Delta M \tag{12}$$

where ΔM represents a random perturbation of the candidate seed parameters, ensuring that the new movement remains within a reasonable range around the original movement.

3.6.3. Crossover Operation

In addition to mutation, we implement a crossover operation to combine the characteristics of two parent movements to create offspring. This can be performed by averaging the parameters of the selected seeds:

$$M_k^{\text{offspring}} = \frac{1}{2}(M_i + M_j) \tag{13}$$

where M_i and M_j are two selected parent movements. The offspring inherit traits from both parents, which can lead to better candidate movements. For more details about this whole step please see Algorithm 4.

Algorithm 4: Selection, Mutation, and Crossover of Candidate Movements

Input: Candidate movements M_k, MSE values MSE_k, threshold T
Output: New population of movements M_{new}
Initialize $M_{new} \leftarrow \emptyset$
for each movement M_k in M_k **do**
  if $MSE_k < T$ **then**
    Add M_k to the retained movements
  end if
end for
for each pair of retained movements M_i, M_j **do**
  Generate offspring movement:

$$M_k^{\text{offspring}} = \frac{1}{2}(M_i + M_j)$$

  Add $M_k^{\text{offspring}}$ to M_{new}
end for
for each movement M_k in the retained movements **do**
  Generate random perturbation ΔM
  Mutate movement:

$$M_k^{\text{mutated}} = M_k + \Delta M$$

  Add M_k^{mutated} to M_{new}
end for
return M_{new}

Parallel Selection and Mutation To optimize the selection, mutation, and crossover steps of the algorithm, we can use the parallel processing capabilities of CUDA. By implementing the selection mechanism in parallel, each thread can independently evaluate the MSE of candidate movements against the threshold T. This approach allows for simultaneous processing of multiple movements, significantly reducing the time required for selection. Similarly, the mutation process can be parallelized, where each thread generates a random perturbation ΔM and applies it to a different retained movement. This ensures that multiple mutations occur simultaneously, further accelerating the overall computa-

tion. Furthermore, the crossover operation can also benefit from parallelization; pairs of movements can be selected and averaged in parallel across different threads, producing offspring movements rapidly. Using CUDA, we enhance the efficiency of these critical steps in the algorithm, allowing the exploration of a larger search space and facilitating quicker iterations within the optimization process.

3.7. Iterative Optimization

The iterative optimization process is crucial for refining the candidate movements and achieving an accurate estimation of the vehicle's ego-motion. The above steps are repeated iteratively until the cost function converges to a predefined threshold th or a maximum number of iterations MAX_{iter} is reached.

$$\text{Terminate if } \text{cost}(I_1, I_{2,k}) < \text{th} \quad \text{or} \quad \text{if iterations} \geq MAX_{\text{iter}} \tag{14}$$

This ensures that the optimization process does not continue indefinitely and allows for a controlled exploration of the search space. The iterative nature of the algorithm allows for continual refinement of the candidate movements, as adjustments are made based on the results of the cost function evaluations.

Although individual steps of seed initialization, ego-motion application, and mutation can be executed in parallel, the overall iteration process remains sequential due to dependencies on the results of the previous iterations. Specifically, the frequency of the GPU will limit the rate at which these steps can be executed. As such, careful consideration is given to the number of iterations to balance computational efficiency with the quality of the ego-motion estimation.

The general algorithm can be summarized as shown in Algorithm 5.

Algorithm 5: Iterative Optimization for Ego-Motion Estimation

Input: Initial candidate movements M, predefined threshold th, maximum iterations MAX_{iter}
Initialize iteration count $iter \leftarrow 0$
while true **do**
  Seed Initialization: Randomly generate n candidate movements.
  Ego-motion Application: Apply ego-motion transformations to the second image
  and compare it with the first image.
  Selection and Mutation: Retain the top $n/2$ best candidate movements based on a
  cost function, and generate $n/2$ new candidates by slightly mutating the retained movements.
  Calculate cost $\text{cost}(I_1, I_{2,k})$
  if $\text{cost}(I_1, I_{2,k}) < \text{th}$ or $iter \geq MAX_{\text{iter}}$ **then**
    Terminate the optimization process
  end if
  Increment $iter \leftarrow iter + 1$
end while
return Best candidate movement

3.8. Computational Complexity Analysis

The computational complexity of the proposed ego-motion estimation algorithm is primarily influenced by the genetic algorithm (GA) optimization process, which iteratively

evaluates multiple candidate solutions. In order to provide a clear understanding of the algorithm's performance, we analyze its complexity in terms of the number of motion candidates (N) and the number of iterations (T) required for convergence.

- Genetic Algorithm Iterations: The genetic algorithm begins with an initial population of N candidate solutions, where each candidate represents a potential ego-motion trajectory. In each iteration, the algorithm evaluates the fitness of each candidate based on image similarity between consecutive frames. The evaluation of each candidate involves calculating the similarity function, which is computationally intensive and requires the processing of image patches. Given that each candidate is evaluated in parallel using CUDA, the time complexity to evaluate each candidate is $O(1)$ with respect to the number of candidates, due to parallelism.
  The total complexity for one iteration of the genetic algorithm is $O(N)$, where N is the number of candidates. The algorithm iterates T times to arrive at an optimal solution, where T represents the number of iterations required for convergence. Therefore, the overall time complexity of the genetic algorithm is $O(N \cdot T)$.
- CUDA Optimization: Parallel execution on the GPU significantly reduces computational time by allowing the simultaneous evaluation of all candidates. CUDA's parallel processing capabilities enable the algorithm to handle large populations efficiently. The GPU handles N evaluations simultaneously, with minimal overhead. As a result, the computational bottleneck is mainly determined by the memory bandwidth and the efficiency of the kernel functions.
  The complexity of memory management and data transfer between the CPU and GPU also contributes to overall performance. By optimizing the memory hierarchy and ensuring that data is transferred efficiently between the CPU and GPU, the algorithm minimizes latency and maximizes throughput.
- Memory Usage: The memory complexity is dominated by the storage of candidate solutions and image patches. Each candidate requires storage for its motion parameters and the associated image patch, leading to a memory requirement of $O(N)$. In addition, intermediate results and image data must be stored during the evaluation process. However, by utilizing memory optimization techniques, such as memory pooling and shared memory on the GPU, we can minimize memory overhead and ensure that the algorithm operates efficiently even with large populations.

In summary, the proposed algorithm has an overall time complexity of $O(N \cdot T)$ and memory complexity of $O(N)$, where N is the number of candidates and T is the number of iterations. Through efficient parallel processing with CUDA, the algorithm is able to achieve real-time performance in autonomous vehicle systems while maintaining high accuracy in ego-motion estimation.

4. Results

For all experiments, we tested our algorithm on an MSI Raider GE76 12U laptop equipped with an Intel Core i7-12700H CPU and a GTX 3080 laptop GPU with 7424 CUDA cores and a maximum of 1024 threads per block (see Table 1). All experiments were performed in MATLAB 2022a, utilizing CUDA 8.1 as the GPU processing library. The dataset used was KITTI [31], a well-known benchmark for visual odometry challenge. This dataset was created using a stereo camera mounted on a vehicle navigating through urban environments. The images have a resolution of 1241 $\times$ 376 pixels with a frame rate of 10 fps. For evaluation, KITTI provides 11 training sequences (00–10) with public ground truth data, while an additional 11 sequences (11–21) lack public ground truth and are reserved for evaluation purposes.

First, we evaluated cross-validation across the training sequences by constructing the lookup table (LUT Section 3.3) using all sequences from the training dataset, excluding the sequence under evaluation (see Table 2). This approach enables the assessment of the generalization of the model in different driving scenarios. Given the provided LUTs and their associated ground truth values, we have access to 21,732 elements in M_k (Equation (1)). We chose a LUT length of 100, capturing the 100 most representative movements, which yielded approximately 97% accuracy while achieving an ego-motion estimation of 81 frames per second.

Figure 3 presents qualitative results corresponding to the data outlined in Table 2, illustrating the algorithm's capacity to adapt to multiple conditions in different driving environments. These cross-validation findings suggest that our approach effectively balances speed and accuracy, a balance that is particularly advantageous for real-time applications in dynamic environments, such as autonomous driving. Moreover, by focusing on the most significant movements within the LUT, our approach ensures robust ego-motion estimation without requiring extensive computational resources, making it suitable for embedded implementations in autonomous vehicle scenarios.

Table 1. Thread configuration for the GPU implementation.

Variable	Value
Input data size (independent)	$[x, y]$
Output data size (independent)	$[x, y]$
Threads per block (independent) *	T
Blocks size (dependent)	$[\text{floor}(\sqrt{T}), \text{floor}(\sqrt{T}), 1]$
Grid size (dependent)	$[\text{floor}(x/\sqrt{T}), \text{floor}(y/\sqrt{T}), 1]$

* This value depends on the GPU device; the most typical value is $T = 1024$.

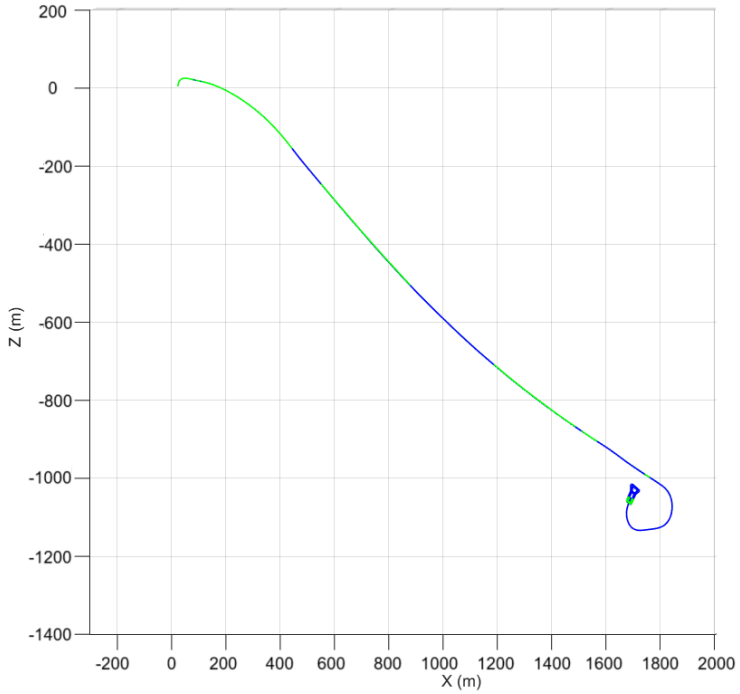
Table 2. Quantitative results for the KITTI dataset. Training sequences 00–10.

KITTI Sequence	Accuracy
00	96.77%
01	95.67%
02	98.23%
03	97.56%
04	96.75%
05	98.01%
06	96.24%
07	97.34%
08	97.21%
09	96.76%
10	98.11%

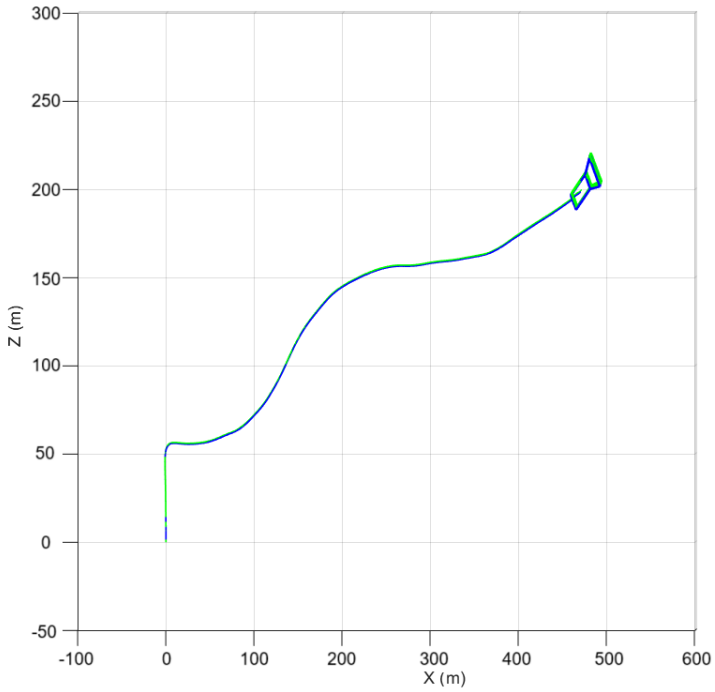
In addition, we conducted a comparative performance analysis with previous work in the current literature. For these comparisons, performance metrics from previously published manuscripts were referenced. For our algorithm, we submit the results of the test sequences (11–21) to the KITTI benchmark suite and then compare the results obtained with previous work.

Quantitative comparisons are presented in Table 3. Our algorithm demonstrates higher performance compared with traditional methods like [33,34], outperforming both accuracy and processing speed. This advantage lies in the fact that many conventional approaches rely on binary feature description and matching techniques, which tend to be sensitive to image quality degradation. In contrast, the genetic processes proposed in this work enhance robustness against such degradations, leading to improved overall performance.

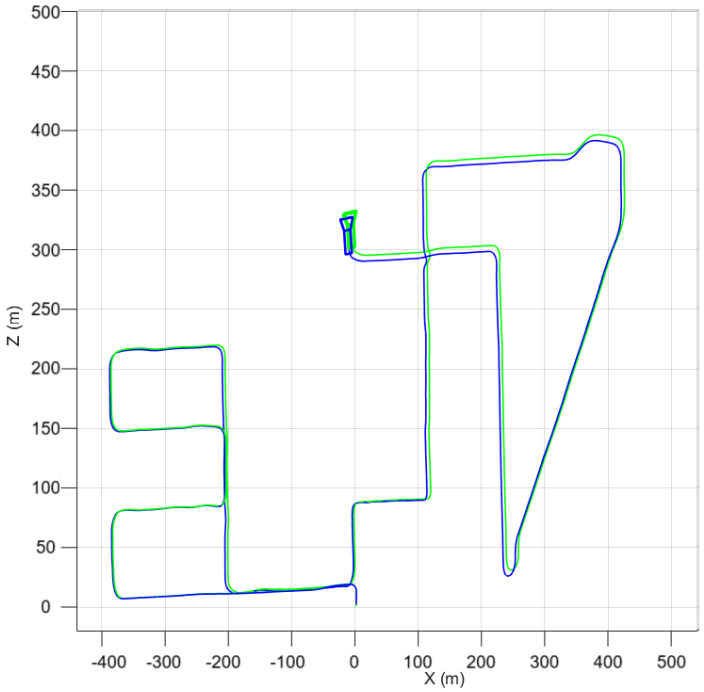
Compared with [35–38], our algorithm achieves a good trade-off between accuracy and processing speed. Specifically, we outperform [36] by achieving an increase in accuracy of 5% and an improvement of 8% over [37]. In terms of processing speed, we achieve a remarkable speed of 91 fps, which is three times faster than [36]. However, ref. [37] exceeds our method with a processing speed advantage of 67 fps.



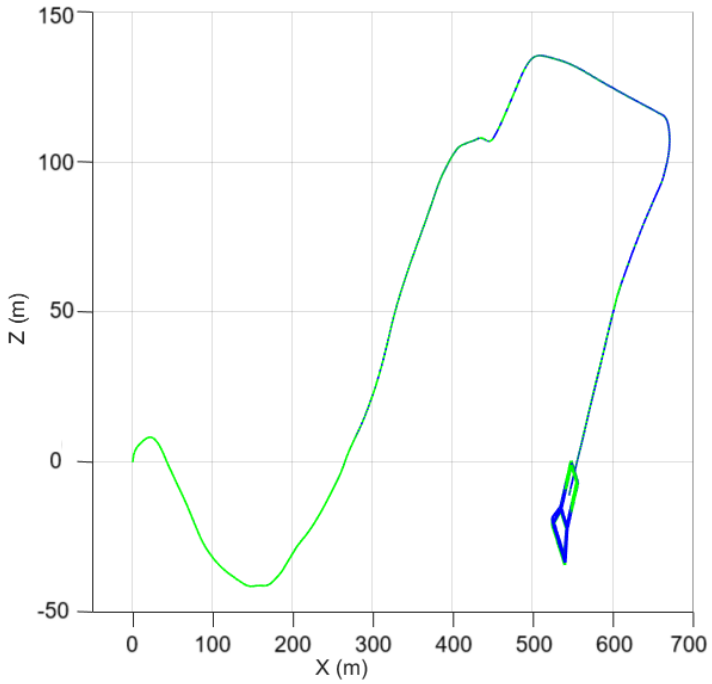
(**a**) 01



(**b**) 03



(**c**) 08



(**d**) 10

Figure 3. The algorithm's performance was evaluated on the KITTI dataset's training sequences, with results demonstrating consistently high accuracy across multiple test cases, closely matching the ground truth. These outcomes confirm that the model effectively captures vehicle motion and spatial consistency within diverse urban and suburban environments featured in KITTI. The blue line represents the ground truth, while the green line shows the estimated ego-motion using the proposed algorithm.

Although [38] reports a 2% higher accuracy than our approach, it does so at the cost of processing speed. Their reported figure of 333.3 fps does not account for the intensive operations involved in image reading, feature extraction, and feature tracking, which are critical components in the visual odometry framework. Similarly, recent AI-based

methods, such as [29,39,40], achieve high accuracy levels but face significant limitations in processing speed. For example, ref. [40] reports an impressive accuracy of 98.3% but only 3 fps using a high-performance Nvidia Tesla V100 GPU, which makes it unsuitable for real-time applications. Furthermore, refs. [29,39] achieve accuracies of 99.89% and 98.83%, respectively, but both operate at only 10 fps. In particular, ref. [39] relies solely on a CPU, which limits its computational efficiency in dynamic scenarios. In contrast, our proposed algorithm delivers a robust balance between accuracy (97.71%) and processing speed (91.78 fps) on a GPU platform, demonstrating its suitability for real-time performance in autonomous navigation tasks.

In Figure 4, we present qualitative results for sequences 11–14 from the KITTI dataset. The results obtained by applying our proposed algorithm were submitted to the KITTI evaluation platform. In all cases, qualitative results demonstrate that high precision is possible, underscoring the effectiveness of the proposed algorithm in multiple urban environments.

Table 3. Quantitative results for the proposed algorithm compared with previous works. In most cases, our algorithm outperforms previous works in terms of accuracy and processing speed.

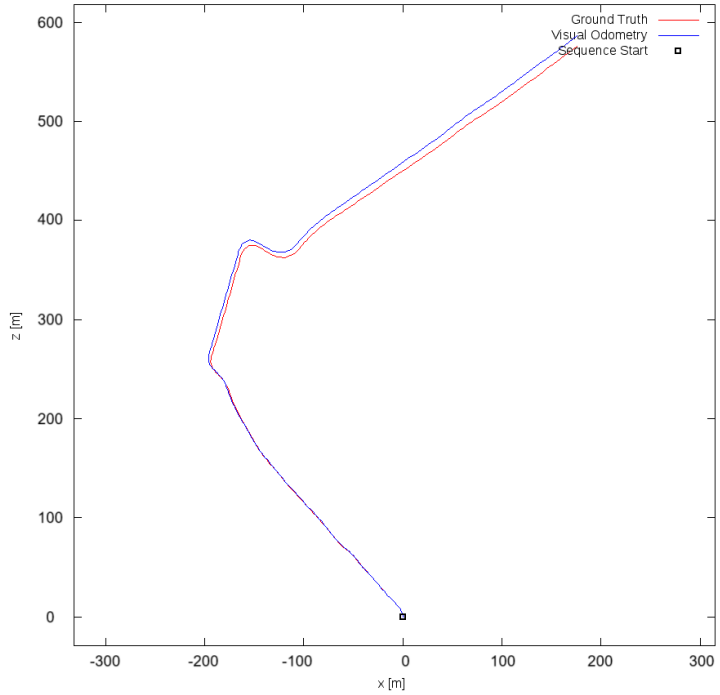
Algorithm	Accuracy/Speed/Hardware
Geiger et al. (2011) [33]	83.71%/16.39 fps/CPU (i7-4720HQ)
Ciarfuglia et al. (2014) [34]	85.56%/9.09 fps/CPU (i7-4720HQ)
Costante et al. (2016) [36]	91.04%/3.27 fps/CPU (i7-4720HQ)
Costante et al. (2016) [36]	91.04%/20.83 fps/GPU (Tesla K40)
Mohanty et al. (2016) [35]	94.50%/111.11 fps/Intel Xeon @4 + GPU (GTX 970)
Weber et al. (2017) [37]	88.53%/158.73 fps/GPU (GTX 970)
Pillai and Leonard (2017) [38]	99.72%/333.3 fps /CPU (i7-3920XM)
Aguilar et al. (2019) [10]	96.07%/86.34 fps/GPU (GTX 970M)
Chen et al. (2021) [39]	99.89%/10.00 fps/GPU (N/A)
Yoon et al. (2021) [40]	98.3%/3.00 fps/GPU (Nvidia Tesla V100 GP)
Rabiee et al. (2020) [29]	98.83%/10.00 fps/GPU (Unknown)
This work	97.71%/91.78 fps/GPU (GTX 3080 Laptop)

To further evaluate the robustness of the proposed algorithm, we applied it to a benchmark dataset consisting of indoor video sequences. This dataset, contains 130 different video sequences, each recording various movements recorded at six degrees of freedom (DOF). This indoor dataset offers a diverse set of movement scenarios, providing valuable insight into the performance of the algorithm under different conditions compared with the KITTI dataset.

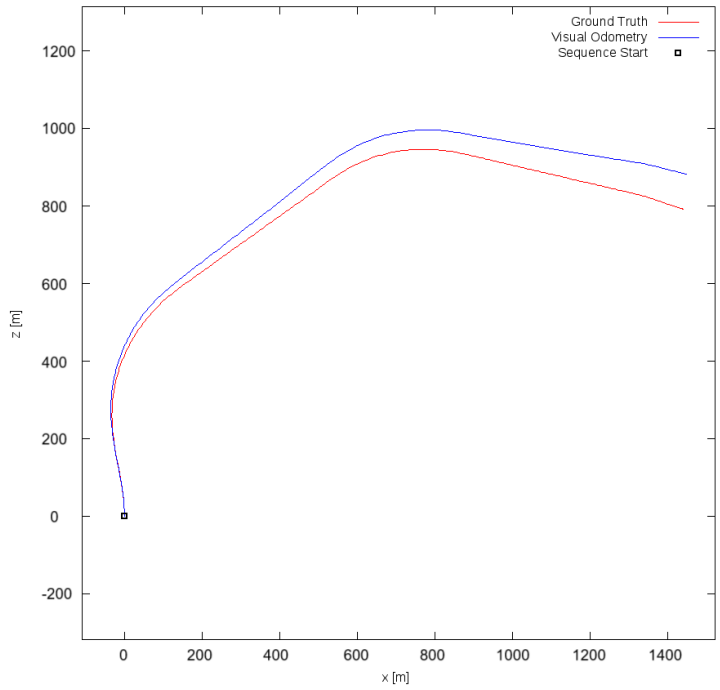
The results obtained from the indoor dataset, shown in Figure 5, demonstrate that the proposed method maintains high precision in both qualitative and quantitative terms. Despite the challenging indoor environment, where visual characteristics differ significantly from those in urban outdoor settings (as seen in the KITTI dataset), the algorithm consistently achieved strong performance across all test cases. For more details on the indoor benchmark dataset, including its creation and the full set of sequences used in this work, we refer the reader to the dataset webpage [41].

Concerning real-world disturbances, including motion blur, occlusions, and noisy sensor data, such factors are often present in dynamic environments where autonomous vehicles operate. In this scenario, our algorithm employs genetic algorithms, which are inherently adaptive and capable of selecting the best candidate solutions based on fitness criteria, such as image similarity between consecutive frames. This process allows the algorithm to effectively handle minor disturbances in the input data, as it can evolve potential movements even when certain frames are noisy or partially occluded. Moreover,

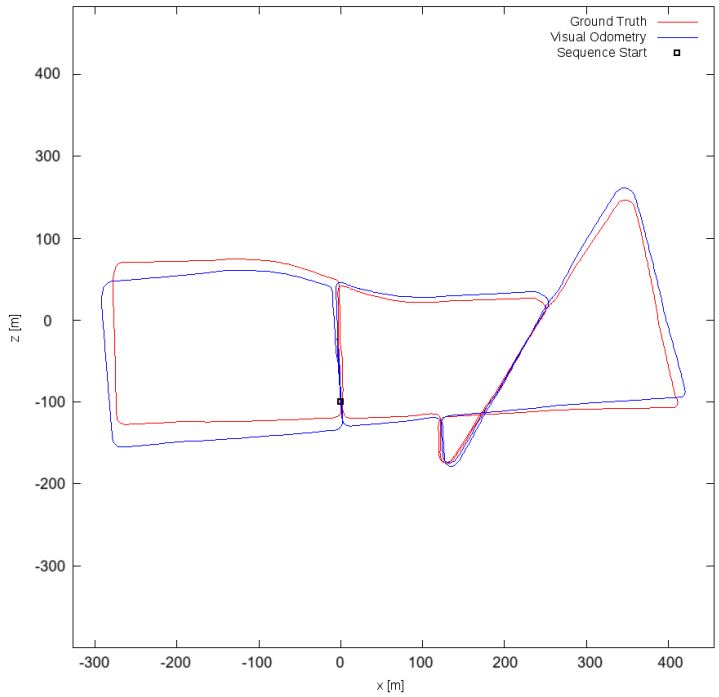
the ability to refine and combine candidate solutions iteratively contributes to resilience in the face of such challenges.



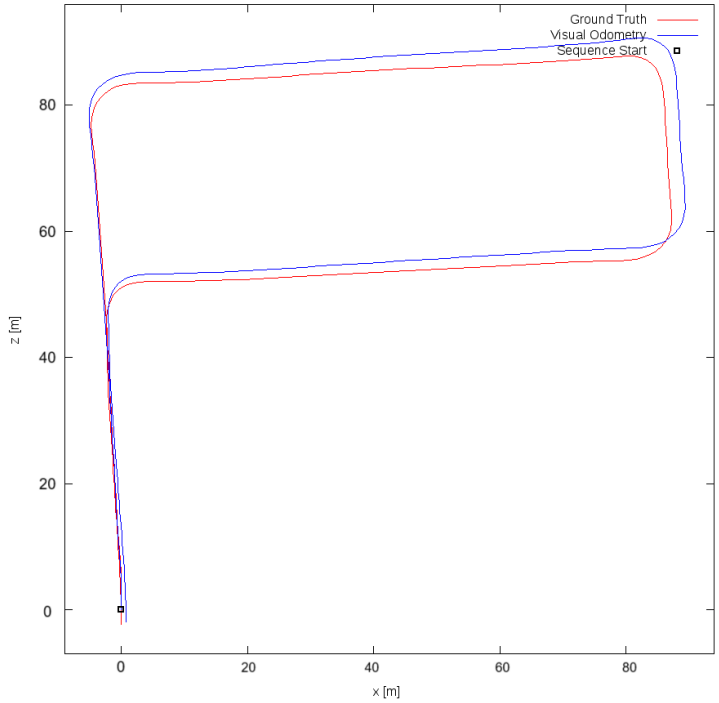
(**a**) 11



(**b**) 12



(**c**) 13



(**d**) 14

Figure 4. The performance of the algorithm, sequences 11 to 14 from the KITTI dataset, without ground truth data. The results were obtained from the KITTI evaluation platform, where our algorithm was submitted for evaluation. These evaluations demonstrated that the proposed algorithm maintains a high level of precision across all tested scenarios.

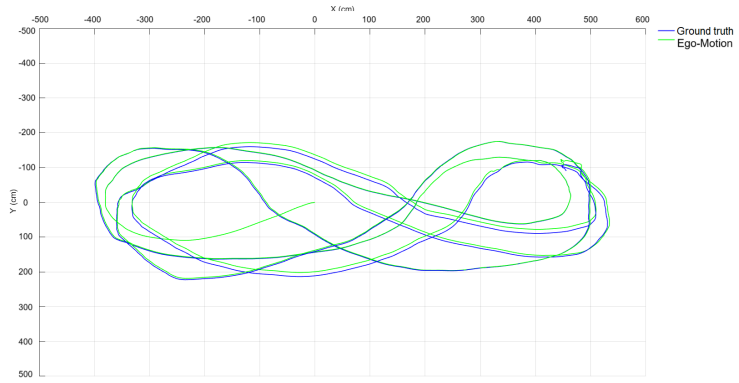


Figure 5. Performance for the pose estimation step under the proposed dataset. Sequence 48, which consists of an x, y, and yaw camera movement, is used to validate the performance under loop trajectories. By using the proposed algorithm, an accuracy of around 97.4% can be reached.

Additionally, we tested the algorithm on the KITTI dataset, which, while not specifically designed to simulate extreme motion blur or occlusions, presents a diverse range of urban and suburban driving scenarios. These conditions include varying lighting, partial occlusions, and sensor noise. The algorithm performed consistently with high accuracy (up to 97.4%), demonstrating its effectiveness in real-world conditions where such disturbances are common. The robustness observed in these tests suggests that the approach can handle moderate motion blur and occlusions, as well as noisy sensor data, which are typical challenges for ego-motion estimation systems in autonomous vehicles. Although further testing with more severe conditions is always beneficial, the performance on the KITTI dataset suggests that our method is already well equipped to manage common sensor imperfections encountered in autonomous driving scenarios.

Finally, in terms of energy consumption, we acknowledge that energy efficiency is a significant consideration in the design of autonomous vehicle systems, particularly when high-performance algorithms are deployed in embedded platforms. CUDA implementation, leveraging the parallel processing power of GPUs, is inherently more energy efficient than traditional CPU-based computations for large-scale optimization tasks. By reducing the processing time required to evaluate multiple candidate solutions in real time, our algorithm minimizes the overall energy consumption. However, it is important to note that exact energy consumption depends on factors such as the GPU model used, the complexity of the task, and the size of the input data. For instance, our experiments using the GTX 3080 laptop GPU yield high processing speeds with relatively lower energy overhead compared with traditional methods that rely on sequential CPU processing.

For safety implications, real-time processing failures in autonomous vehicles can have significant consequences, including potential accidents or system malfunctions. To mitigate these risks, our algorithm is designed with multiple layers of redundancy and error handling. The iterative approach of the genetic algorithm ensures that candidate solutions are continuously evaluated and refined, making it more resilient to occasional computational errors or sensor anomalies. Furthermore, the use of CUDA for parallel processing enables the rapid completion of tasks, which is crucial to minimizing latency and avoiding processing bottlenecks that could lead to delays in decision-making.

Although the proposed algorithm has been tested with the KITTI dataset under typical driving conditions, real-world safety assurance would require additional testing in more critical scenarios, including extreme environmental conditions or hardware malfunctions. In future work, we plan to investigate fault-tolerant mechanisms and safety features such as sensor fusion and redundancy to improve the robustness of the system in real-time autonomous navigation.

5. Extension to Six Degrees of Freedom

Although this work focuses on the estimation of ego-motion using three degrees of freedom (x, y, and yaw), extending the proposed method to six degrees of freedom (x, y, z, roll, pitch, and yaw) will be explored in future work. The extension to six DoF involves additional challenges, primarily due to the increased complexity in capturing and accurately predicting the vertical movements (z) and rotational motions (roll and pitch) of the vehicle. To address these challenges, the integration of more sophisticated optimization techniques, such as multi-objective genetic algorithms, will be considered to handle the increased dimensionality effectively.

Furthermore, the use of advanced sensor fusion approaches, incorporating LiDAR or IMU data alongside visual information, could be explored to improve the robustness and accuracy of ego-motion estimation in three-dimensional spaces. Using the strengths of these additional sensors, the proposed method could achieve more precise and reliable results for the full six degrees of freedom in diverse environments, including urban and off-road scenarios.

In general, extending the method to six degrees of freedom will be a significant step towards improving the generalization and applicability of the algorithm for real-world autonomous vehicle navigation tasks, particularly in more complex and dynamic environments.

6. Conclusions

In this work, we presented a novel brute-force-based ego-motion estimation algorithm suitable for autonomous vehicles. Taking advantage of the unique constraints of such systems, which operate with only three degrees of freedom (x, y, and yaw), our approach effectively leverages these limitations to streamline the estimation process. Using a genetic algorithm, we systematically explored potential vehicle movements, enabling efficient generation and refinement of motion candidates. This iterative process not only minimizes the cost function measuring image similarity between consecutive frames but also enhances the robustness of our method against environmental variations.

The implementation of our algorithm using CUDA for parallel processing demonstrated significant improvements in computational speed, making it suitable for real-time applications. Experimental results demonstrate that our approach achieves high accuracy in ego-motion estimation while maintaining high efficiency for the dynamic demands of autonomous navigation.

In addition, our experimental results highlight the importance of integrating advanced computational techniques with domain-specific constraints to enhance the performance of ego-motion estimation in practical scenarios. The successful application of our algorithm to the KITTI dataset guarantees its potential for real-world deployment, allowing for future research in optimizing visual odometry and related tasks in autonomous systems.

Author Contributions: Conceptualization, Investigation: A.A.-G. Validation and Writing—Original Draft: A.M.S. All authors have read and agreed to the submitted version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The datasets used in research are publicly available at: https://www.cvlibs.net/datasets/kitti/eval_odometry.php accessed 1 January 2025. The MATLAB source code that support the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: Acknowledgements to INAOE for supporting the development of this postdoctoral research under the supervision of Alejandro Medina Santiago (Researcher for Mexico); this work will strengthen Project 882 of Conahcyt.

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

References

1. Park, H.S.; Hwang, J.J.; Niu, Y.; Shi, J. Egocentric future localization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 4697–4705.
2. Olson, C.F.; Matthies, L.H.; Schoppers, M.; Maimone, M.W. Rover navigation using stereo ego-motion. *Robot. Auton. Syst.* **2003**, *43*, 215–229. [CrossRef]
3. Wang, N.; Zhang, B.; Chi, H.; Wang, H.; McLoone, S.; Liu, H. DUEL: Depth visUal Ego-motion Learning for autonomous robot obstacle avoidance. *Int. J. Robot. Res.* **2024**, *43*, 305–329. [CrossRef]
4. Khan, N.H.; Adnan, A. Ego-motion estimation concepts, algorithms and challenges: an overview. *Multimed. Tools Appl.* **2017**, *76*, 16581–16603. [CrossRef]
5. Gao, Y.; Tian, F.; Li, J.; Fang, Z.; Al-Rubaye, S.; Song, W.; Yan, Y. Joint optimization of depth and ego-motion for intelligent autonomous vehicles. *IEEE Trans. Intell. Transp. Syst.* **2022**, *24*, 7390–7403. [CrossRef]
6. Arif, U.; Razzaq, W.; Akram, A.; Muhammad, W. Self-Driving Car Control Using Visual Ego-Motion Estimation. In Proceedings of the 2019 15th International Conference on Emerging Technologies (ICET), Peshawar, Pakistan, 2–3 December 2019; pp. 1–4.
7. Bian, J.; Li, Z.; Wang, N.; Zhan, H.; Shen, C.; Cheng, M.M.; Reid, I. Unsupervised scale-consistent depth and ego-motion learning from monocular video. *Adv. Neural Inf. Process. Syst.* **2019**, *32*. [CrossRef]
8. Gao, R.; Xiao, X.; Xing, W.; Li, C.; Liu, L. Unsupervised learning of monocular depth and ego-motion in outdoor/indoor environments. *IEEE Internet Things J.* **2022**, *9*, 16247–16258. [CrossRef]
9. Ouerghi, S.; Boutteau, R.; Savatier, X.; Tlili, F. CUDA accelerated visual egomotion estimation for robotic navigation. In Proceedings of the 12th International Conference on Computer Vision Theory and Applications, Porto, Portugal, 27 February–1 March 2017; SCITEPRESS-Science and Technology Publications: 2017; pp. 107–114.
10. Aguilar-González, A.; Arias-Estrada, M.; Berry, F.; de Jesús Osuna-Coutiño, J. The fastest visual ego-motion algorithm in the west. *Microprocess. Microsyst.* **2019**, *67*, 103–116. [CrossRef]
11. Forrest, S. Genetic algorithms. *ACM Comput. Surv. (CSUR)* **1996**, *28*, 77–80. [CrossRef]
12. Lambora, A.; Gupta, K.; Chopra, K. Genetic algorithm-A literature review. In Proceedings of the 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon), Faridabad, India, 14–16 February 2019; pp. 380–384.
13. Martínez, J.L.; González, J.; Morales, J.; Mandow, A.; García-Cerezo, A.J. Mobile robot motion estimation by 2D scan matching with genetic and iterative closest point algorithms. *J. Field Robot.* **2006**, *23*, 21–34. [CrossRef]
14. Villaverde, I.; Echegoyen, Z.; Graña, M. Neuro-evolutive system for ego-motion estimation with a 3D camera. In Proceedings of the Advances in Neuro-Information Processing: 15th International Conference, ICONIP 2008, Auckland, New Zealand, 25–28 November 2008; Revised Selected Papers, Part I 15; Springer: Berlin/Heidelberg, Germany, 2009; pp. 1021–1028.
15. Farber, R. *CUDA Application Design and Development*; Elsevier: Amsterdam, The Netherlands, 2011.
16. Garland, M.; Le Grand, S.; Nickolls, J.; Anderson, J.; Hardwick, J.; Morton, S.; Phillips, E.; Zhang, Y.; Volkov, V. Parallel computing experiences with CUDA. *IEEE Micro* **2008**, *28*, 13–27. [CrossRef]
17. Ouerghi, S.; Tlili, F. CUDA accelerated visual relative motion estimation. In Proceedings of the 2016 International Symposium on Signal, Image, Video and Communications (ISIVC), Tunis, Tunisia, 21–23 November 2016; pp. 302–307.
18. Gómez-Luna, J.; Endt, H.; Stechele, W.; González-Linares, J.M.; Benavides, J.I.; Guil, N. Egomotion compensation and moving objects detection algorithm on GPU. In *Applications, Tools and Techniques on the Road to Exascale Computing*; IOS Press: Amsterdam, The Netherlands, 2012; pp. 183–190.
19. Muñoz, B.; Troni, G. Learning the Ego-Motion of an Underwater Imaging Sonar: A Comparative Experimental Evaluation of Novel CNN and RCNN Approaches. *IEEE Robot. Autom. Lett.* **2024**, *9*, 2072–2079. [CrossRef]
20. Lowe, G. Sift-the scale invariant feature transform. *Int. J* **2004**, *2*, 2.
21. Rublee, E.; Rabaud, V.; Konolige, K.; Bradski, G. ORB: An efficient alternative to SIFT or SURF. In Proceedings of the 2011 International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; pp. 2564–2571.
22. Wu, H.; Gao, Y.; Li, S. Odometry Estimation Utilizing 6-DOF Force Sensors and IMU for Legged Robot. In Proceedings of the 2020 Chinese Automation Congress (CAC), Shanghai, China, 6–8 November 2020; pp. 6901–6905.
23. Yuan, S.; Fioranelli, F.; Yarovoy, A. 3drudat: 3d robust unambiguous doppler beam sharpening using adaptive threshold for forward-looking region. *IEEE Trans. Radar Syst.* **2024**, *2*, 138–153. [CrossRef]
24. Yuan, S.; Wang, D.; Fioranelli, F.; Yarovoy, A. Improved accuracy for 3D ego-motion estimation using automotive FMCW MIMO radar. In Proceedings of the 2024 IEEE Radar Conference (RadarConf24), Denver, CO, USA, 6–10 May 2024; pp. 1–6.
25. De Araujo, P.R.M.; Noureldin, A.; Givigi, S. Towards Land Vehicle Ego-Velocity Estimation using Deep Learning and Automotive Radars. *IEEE Trans. Radar Syst.* **2024**, *2*, 460–470. [CrossRef]
26. Shiba, S.; Klose, Y.; Aoki, Y.; Gallego, G. Secrets of event-based optical flow, depth and ego-motion estimation by contrast maximization. *IEEE Trans. Pattern Anal. Mach. Intell.* **2024**, *46*, 7742–7759. [CrossRef]

27. Tu, J.; Yang, S.X. Genetic algorithm based path planning for a mobile robot. In Proceedings of the 2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422), Taipei, Taiwan, 14–19 September 2003; Volume 1, pp. 1221–1226.
28. Persson, M.; Piccini, T.; Felsberg, M.; Mester, R. Robust stereo visual odometry from monocular techniques. In Proceedings of the 2015 IEEE Intelligent Vehicles Symposium (IV), Seoul, Korea, 28 June–1 July 2015; pp. 686–691.
29. Rabiee, S.; Biswas, J. IV-SLAM: Introspective vision for simultaneous localization and mapping. In Proceedings of the Conference on Robot Learning, PMLR, 2021; Held in virtual on 16-18 November 2020 pp. 1100–1109.
30. Adis, P.; Horst, N.; Wien, M. D3DLO: Deep 3D LiDAR Odometry. In Proceedings of the 2021 IEEE International Conference on Image Processing (ICIP), Anchorage, AK, USA, 19–22 September 2021; pp. 3128–3132.
31. Geiger, A.; Lenz, P.; Stiller, C.; Urtasun, R. Vision meets robotics: The kitti dataset. *Int. J. Robot. Res.* **2013**, *32*, 1231–1237. [CrossRef]
32. Corporation, N. cuRAND—CUDA Random Number Generation Library. 2024. Available online: https://docs.nvidia.com/cuda/curand/index.html (accessed on 20 October 2024).
33. Geiger, A.; Ziegler, J.; Stiller, C. Stereoscan: Dense 3d reconstruction in real-time. In Proceedings of the Intelligent Vehicles Symposium (IV), Baden-Baden, Germany, 5–9 June 2011; pp. 963–968.
34. Ciarfuglia, T.A.; Costante, G.; Valigi, P.; Ricci, E. Evaluation of non-geometric methods for visual odometry. *Robot. Auton. Syst.* **2014**, *62*, 1717–1730. [CrossRef]
35. Mohanty, V.; Agrawal, S.; Datta, S.; Ghosh, A.; Sharma, V.D.; Chakravarty, D. DeepVO: A deep learning approach for monocular visual odometry. *arXiv* **2016**, arXiv:1611.06069.
36. Costante, G.; Mancini, M.; Valigi, P.; Ciarfuglia, T.A. Exploring representation learning with CNNs for frame-to-frame ego-motion estimation. *IEEE Robot. Autom. Lett.* **2016**, *1*, 18–25. [CrossRef]
37. Weber, M.; Rist, C.; Zöllner, J.M. Learning temporal features with CNNs for monocular visual ego motion estimation. In Proceedings of the 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Yokohama, Japan, 16–19 October 2017; pp. 1–6.
38. Pillai, S.; Leonard, J.J. Towards visual ego-motion learning in robots. *arXiv* **2017**, arXiv:1705.10279.
39. Chen, X.; Li, S.; Mersch, B.; Wiesmann, L.; Gall, J.; Behley, J.; Stachniss, C. Moving object segmentation in 3D LiDAR data: A learning-based approach exploiting sequential data. *IEEE Robot. Autom. Lett.* **2021**, *6*, 6529–6536. [CrossRef]
40. Yoon, D.J.; Zhang, H.; Gridseth, M.; Thomas, H.; Barfoot, T.D. Unsupervised learning of lidar features for use ina probabilistic trajectory estimator. *IEEE Robot. Autom. Lett.* **2021**, *6*, 2130–2138. [CrossRef]
41. INAOE/DREAM Benchmark Dataset. Available online: https://dream.ispr-ip.fr/ispr-benchmark-dataset/#page-content (accessed on 17 December 2024).