UNIVERSITY OF BLIDA 1

Faculty of Technology

Department of Electronics

DOCTORAL THESIS

In Electronics

VISUAL ODOMETRY OF DENSE RGB-D IMAGES USING DIFFERENT OPTIMIZATION METHODS

Realized by:

DJEMA SLIMANE

In front of the jury made up of :

Mr. YKHLEF Farid	Professor, Blida1 University	President
Mr. DJENDI Mohamed	Professor, Blida1 University	Examiner
Mr. HOCINE Abdelfettah	MCA, University of Djilali Bounaama Khemis Miliana	Examiner
Mr. Zoubir Abdeslem Benselama	Professor, Blida1 University	Thesis director
Mr. Ramdane Hedjar	Professor, King Saud University	Thesis co-director

Blida, April 06,2025

Abstract

The goal of this thesis is to accurately estimate the motion of a camera embedded in a robot or a moving object in a static scene using RGB-D images. These images can be provided by a stereo camera or by using a color digital camera as well as one that provides the depth of the scene. Therefore, RGB-D images are the only information acquired by the system from the environment. Thus, the movement of the system is estimated using different consecutive images. The unknown camera motion can be determined by minimizing the intensity error between every two consecutive images. Hence, the challenge of motion estimation is transformed into a non-linear least squares optimization problem, with robot motion being the unknown solution. The solution of such problems typically involves iterative approaches. Exact methods use the linearization of the least square equation to resolve this problem. Alternatively, we can use metaheuristic optimization methods to solve this non-linear equation. Note that the optimal solution will be used to estimate the position of a mobile robot. To evaluate the visual odometry methods, both exact and metaheuristic methods, we apply the root mean square error to an extensive set of images.

Key words: RGB-D images, static scene, stereo camera, metaheuristic method, visual odometry.

ملخص

الهدف من هذه الأطروحة هو تقدير حركة الكاميرا المدمجة في روبوت أو جسم المتحرك بدقة في مشهد ثابت باستعمال صور RGB-D. هذه الصور يمكن توفيرها بواسطة كاميرا استريو أو باستخدام كاميرا رقمية ملونة بالإضافة إلى كاميرا توفر عمق المشهد. حيث صور RGB-D هي المعلومات الوحيدة التي يحصل عليها النظام من المحيط. وبالتالي، يتم تقدير حركة الكاميرا باستخدام صور متتابعة مختلفة. يمكن تحديد حركة الكاميرا، غير المعروفة، عن طريق تقليل خطأ الكثافة بين كل صورتين متتابعتين. ومن ثم، يتم تحويل تحدي تقدير الحركة إلى مشكلة تحسين تقليل المربع غير الخطي، حيث تكون حركة الروبوت هي الحل المجهول. عادةً ما يتضمن حل مثل هذه المشكلات أساليب تكرارية. تستخدم الطرق الدقيقة الخطية لمعادلة تقليل المربع غير الخطي لحل هذه المشكلة. وبدلاً من ذلك، يمكننا استخدام طرق التحسين الميتايورستية لحل هذه المعادلة غير الخطية. يُلاحظ أن الحل الأمثل سيُستخدم لتقدير موضع روبوت متحرك. لتقييم طرق قياس المسافة البصرية، كل من الطرق الدقيقة والميتايورستية، نطبق متوسط جذر مربع الخطأ على مجموعة واسعة من الصور.

الكلمات المفتاحية: صور RGB-D، مشهد ثابت، كاميرا ستيريو، طريقة الميتايورستية، القياس البصري للمسافة.

بسم الله الرحمن الرحيم

قَالُوا سُبْحَانَكَ لَا عِلْمَ لَنَا إِلَّا مَا عَلَّمْتَنَا إِنَّكَ أَنتَ الْعَلِيمُ الْحَكِيمُ

سورة البقرة , الآية 31

Acknowledgments

First and foremost, I would like to thank Allah 'Âzza wa Jal' for giving me the courage, patience, health, and will to finish this work.

This thesis is a result of years of hard work, and it is with the help of people who did not give up. During these years, I learned a lot in my field and that is what led me to make contributions and continue research n order to improve my knowledge.

I am very grateful for the support of my thesis directors, who have helped me through the PhD. I would like to express my deep gratitude to Mr. Zoubir Abdeslem Benselama, professor at the University of BLIDA 1 and director of this thesis, and without your continuous support and encouragement, I would probably not have finished this modest work. I would also like to thank Mr. Ramdane Hedjar, Professor at King Saud University and co-director of this thesis. I have benefited from our many interesting discussions regarding this work, and I have really appreciated your help in wrapping my head around some of the more technical aspects and for the support you were kind enough to give me throughout my work and, above all, your confidence in me as well as your incessant encouragement.

My sincere thanks also go to the members of the jury: Pr. YKHLEF Farid, president of the jury, Pr. DJENDI Mohamed, and Pr. HOCINE Abdelfettah, examiners of this thesis, for agreeing to evaluate this work.

I would like my friends, my siblings, and members of LATSI Laboratory to find here the expression of my most sincere and deep thanks in recognition of their support, sacrifices and encouragement.

Dedication

I dedicate this modest work as a testimony of my great respect to my dearest. I would like to thank all of them, particularly and above all, MY PARENTS, for all their sacrifices, their support, and their prayers all along my studies, and I am happy to have been able to read joy and pride in their eyes. May Allah preserve your health and grant you a long life.

To my brothers, sisters, wife, sons, all my grand family DJEMA, friends, and classmates who shared their words of advice and encouragement to finish this doctorate study.

Table of contents

III. Optimization Methods

IV. Performance evaluation metrics

V. Experimental setup

List of abbreviations

2D	Two Dimension
3D	Three Dimension
3-DOF	Tree Degree of Freedom
ABC	Artificial Bee Colony
AR	Augmented Reality
Cumsum	The cumulative sum.
det(R)	determinant of matrix *R*
DMS	Down Mean Sampling
DSR	Down-Sampled Resolution
DTAM	Dense Tracking and Mapping
EB	Energy-Based method
FA	Firefly Algorithm
FAST	Features from Accelerated Segment Test
fr1_xyz	dataset freiburg 1 in xyz direction
fr2_desk	dataset freiburg 2 desk
GA	Genetic Algorithm
Gbest	Global best solution
GPU	Graphics Processing Unit
GUI	graphical user interfaces
ICP	Iterative Closest Points
IFA	Iterative Firefly Algorithm
IMU	Inertial Measurement Unit
IR	Infrared
LIDAR	LIght Detection And Ranging sensor
LIMO	LIdar-Monocular visual Odometry
MER	Mars Exploration Rovers
NASA	National Aeronautics and Space Administration
Pbest	Personal best solution
PNG	Portable Network Graphics
POSIX	Portable Operating System Interface for uniX
$\boldsymbol{probc_i}$	The cumulative probability sum of the particle index *i*

$prob_i$	The select probability of the particle index i
PSO	Particle Swarm Optimization
PTAM	the Parallel Tracking and Mapping
quat2dcm	quaternion to direction cosine matrix
RGB-D	Red Green Blue - Depth
RMSE	Root Mean Square Error
ROS	Robot Operating System
RPE	Relative pose error
RPEI	Relative pose error between current and precedent image
SE(3)	Special Euclidean group in three dimensions
se(3)	Lie algebra of Special Euclidean group in three dimensions
SIFT	Scale-Invariant Feature Transform
SLAM	Simultaneous Localization And Mapping
SO(3)	special orthogonal group
so(3)	Lie algebra of Special Orthogonal group
SURF	Speeded-Up Robust Features
TGZ	Tape Archive compressed using Gzip
UNIX	Uniplexed Networked Interface eXecution
UTC	Coordinated Universal Time
VectError	Vector of error
VGA	Visual graphics array
VR	Virtual Reality

List of figures

List of tables

General introduction

General introduction

In literature, odometry involves exploiting data provided by different sensors, such as speed and orientation sensors, to estimate the motion of a robot, drone, or car. However, this approach has its limitations, as the wheels may experience slippage on slick surfaces or remain stationary while spinning in sandy terrain, and flying drones in bad weather conditions, such as rain and strong winds makes it difficult for them to gauge their speed and direction of movement, leading to inaccuracies in movement calculations. Over time, these errors can accumulate, resulting in a growing disparity between the ground truth and the estimated motion as the path length increases.

The process of calculating motion using consecutive frames from a moving camera is known as visual odometry, and various algorithms perform this process. It improves navigation accuracy for mobile entities employing various forms of locomotion, including robots with legs or wheels operating on surfaces with different difficulties, such as sticky or sandy terrain. This process enhances the overall performance of the robots and drones in executing their designated tasks.

Over the past decade, substantial advancements in data processing technology and computer science, driven by the growing demand in the fields of autonomous driving, robotics, drones, and the Internet of Things, have significantly enhanced visual odometry. The Internet of Things has become one of the most important technologies, and locating objects is one of its six main objectives.

Various techniques, such as sparse and dense methods, exist for visual odometry. The sparse technique [1] extracts features from an image and uses them to estimate motion. Conversely, dense techniques make use of every pixel in the image for navigation.

Furthermore, various optimization strategies can be identified, including both exact and metaheuristic techniques. Exact methods employ traditional mathematical rules to calculate the optimal motion. On the other hand, metaheuristic methods are employed to compute an approximate motion, which has proven acceptable for practical applications.

Metaheuristics can be used in a wide range of artificial intelligence domains, including visual odometry, due to their effectiveness in addressing both routine and complex challenges. It is an efficient technique for enhancing the solution to make it optimal by modifying the initial parameters, and it is considered one of the optimization methods.

In these experiments, we made use of the dense visual odometry method, which utilizes the complete RGB-D image that a Microsoft Kinect sensor [2] provides. RGB-D cameras are digital devices that furnish color information about the scene for each pixel in two images: red, green,

and blue, as an RGB image, as well as a depth (*D*) frame. The website [3] provides RGB-D images along with associated information, such as the ground truth path and the calibration of the camera used.

The first chapter focuses on the related work of this thesis because this field is rich in research and ideas, as much as it is important in the fields of navigation [4], transportation, and robotics, which have become the backbone of the economy and a tool for achieving prosperity. A brief reference to previous works is important to gain a comprehensive overview of this field. We categorized the various methods of visual odometry into sparse and dense methods, and some works in this field use the metaheuristics method; moreover, we recalled some common work that is widespread for evaluating these optimization methods.

To solve a real problem in our daily lives by exact or metaheuristic methods, we need to develop an objective function in the form of a mathematical equation; this is called the modeling of the system. This modeling is mentioned in Chapter II, where we explain the components of the objective function as described in [5]. Here, we will describe how to use the images captured by a moving camera to deduce the motion using a mathematical model that we can solve by various techniques.

An overview of the different types of optimization methods is mentioned at the beginning of Chapter III. Then, we describe four optimization methods that are most commonly used; one is a classic method called energy-based as mentioned in [6], and the others are metaheuristic methods that we have worked with and developed some of these methods in the field of visual odometry.

The evaluation of optimization methods in visual ego-motion lies mainly in experimenting with them over the same sequence of images and under the same initial parameters, and then comparing the results in terms of their closeness to the true trajectory. For this purpose, we used the root mean square error [7], which represents the drift of the estimated trajectory from the real path in all parts of the experiment by a numerical value, and we provided our experiments with a platform to monitor the results in real-time, as described in Chapter IV. Additionally, we included *3D* drawings on the same graph of the true and estimated trajectories.

The dataset used in our experiments was downloaded from a website, and we will explain the type of dataset and the camera used to capture the images in Chapter V. Then we will explain the experiments conducted and discuss the results obtained, and we will end our thesis with a conclusion and perspectives for future work.

Chapter I

Related work

I. Related work

I.1. Introduction

Visual odometry, or visual ego-motion, computes the motion of the camera using images captured during movement. There are different techniques published for visual ego-motion. We will mention in this chapter some work related to our ideas; such as sparse and dense visual odometry, which uses the exact or metaheuristic methods, in addition to the previous methods published to evaluate these techniques.

I.2. Sparse Visual Odometry

Sparse visual odometry is a method employed for calculating the motion of a robot or any moving body that carries a camera. It relies solely on unique features identified from images by focusing on particular points of interest. Sparse visual odometry allows for more efficient computation, particularly in scenarios with limited computational resources, as mentioned in [1], [8], [9], and [10].

By comparing characteristics between two photos, the sparse visual odometry approach has been widely employed to manage a range of moving equipment, including autonomous ground vehicles in [11], [12], and recently quadcopters [13], [14], and [15].

Engel [14] and Weiss [15] utilize the Parallel Tracking And Mapping (PTAM) system [16]. Huang [13] employs a comparable system but evaluates various alternatives for each component of the processing pipeline and selects the optimal one based on the balance between accuracy and runtime. All have in common that the visual odometry estimates are fused with measurements from the IMU mounted on the quadcopter. In a sparse visual odometry system, the typical trajectory is estimated like this: First, using detectors such as FAST [17] or Harris [18], feature points are retrieved from the newly acquired image. Subsequently, associations are built between the newly added features and those from the preceding frame. Comparing small regions surrounding the feature points will help achieve this.

If there is little error between two patches, it is assumed that they match. Feature descriptors such as SIFT [19] or SURF [20] can be utilized in place of patches. These descriptors are vector representations that are computed from a feature point's surrounding pixels. While they provide enhanced robustness against mismatches relative to image patches, they are significantly more computationally expensive. Then, by reducing the reprojection error between each pair of matched feature points, the transformation between the two images is calculated. It is possible to guarantee precise feature association and raise the motion estimate's correctness by utilizing a variety of advanced strategies. Scaramuzza and Fraundorfer examine these points in [16], [17], and also

discuss matching strategies and feature extractors and descriptors for sparse visual odometry in addition to giving a thorough summary of visual odometry research over the previous few decades. In order to estimate the camera motion, knowledge of the depth of points is necessary. Otherwise, only a homography aligning the images can be estimated. When using a monocular camera, such as in PTAM, stereo initialization is required initially to provide depth for the first feature points. Subsequently, the depth of new points can be calculated through triangulation once the camera motion is estimated from points with known depth. However, the depth of the points can only be determined up to a scale factor, not in metric values. Engel et al. [14] and Weiss et al. concurrently estimated the unknown scale using the IMU readings and an extended Kalman filter. The difficulty is simplified when RGB-D cameras are used, as in [13], as the absolute depth is known.

As a next step, the identified features and camera motion can be combined into a comprehensive map. By employing optimization methods, both the map and the camera's path can be refined to achieve more accurate position estimates and counteract drift over time. These methodologies are referred to as simultaneous localization and mapping (SLAM) [21].

Alternatively, instead of minimizing the error between images, we can minimize the geometric error between 3D surfaces using iterative closest point (ICP) algorithms. These algorithms have various variants [22]. However, they come with drawbacks such as the requirement for structured 3D surfaces and the involvement of computationally expensive nearest neighbor searches to establish point correspondences. For small displacements, the projective lookup algorithm [23] can overcome this by finding correspondences in 2D depth maps. By representing 3D surfaces as 2D depth maps, the correspondence for a point in one depth map can be found by applying rigid body motion and projecting it to 2D coordinates, simplifying the correspondence lookup by computing the memory address.

To accelerate the process, Henry et al. [24] extracted features from the color images and then applied ICP to match these features with their corresponding 3D points computed from the depth map.

Every RGB-D image was transformed into a surfel octree by Stuckler et al. [25], where each node represented a Gaussian distribution that modeled the color and point distribution. Next, feature descriptors for each octree node were calculated. Establishing point correspondences between features in two octrees and then applying ICP is how alignment is accomplished.

I.3. Dense Visual Odometry

Dense visual odometry methods, unlike sparse ones, utilize all the image data. They estimate camera motion by aligning consecutive images, and minimizing an error equation.

Comport et al. proposed one of the earliest dense visual odometry methods using stereo image pairs [8]. Steinbrucker et al. [26] and Tykkala et al. [27] have recently introduced similar dense methods employing data from RGB-D images. These approaches all aim to minimize the photometric difference between two consecutive frames, which can be viewed as an extension of the image alignment algorithm described by Lucas-Kanade [28]. Baker and Matthews [29] discuss some optimizations to the Lucas-Kanade algorithm in detail in their article.

Most odometry techniques that accumulate motion estimates between successive images are susceptible to long-term drift [30]. Consequently, [31] and Dense Tracking and Mapping (DTAM) [32] construct a global environment model in parallel with camera tracking. The camera's motion is tracked by aligning the current image with synthesized views generated from the model, and each new frame is integrated into the model. DTAM employs a monocular camera and requires the model to determine the scene's three-dimensional structure [32]. Kinect Fusion [31] uses a variant of the ICP algorithm to align surfaces from an RGB-D camera to the model, while DTAM [32] employs a photometric error similar to that of Steinbrucker et al. These model-based methods reach real-time speed thanks to GPU general-purpose computing. A dense approach utilizes all pixels in the image for trajectory estimation, as demonstrated in [33]. The initial dense method, as mentioned in [34], [28], and [35], involves frame alignment and the reduction of geometric error.

Following the discovery of the RGB-D image, this format has recently become widely used in visual odometry, as shown in [5], [36], [26], or [37], and [38]. In this thesis, we use dense visual odometry with RGB-D frames.

Visual odometry is used in Virtual Reality (VR) and Augmented Reality (AR) applications to enhance spatial awareness [28]. It enables the accurate overlay of virtual objects onto the real-world environment, creating a seamless and immersive experience.

In addition, visual odometry aids in stabilizing and controlling the flight of drones by providing real-time information about their position and orientation [39]. This is crucial for tasks such as aerial photography, surveillance, and package delivery.

In manufacturing [4] and logistics [40], visual odometry is employed to track the movement of objects on conveyor belts or within warehouses. This facilitates automation processes and ensures efficient material handling.

In modern warfare, unmanned systems rely on visual odometry to navigate complex terrains without GPS, which is essential in hostile environments or GPS-denied areas. Visual odometry can assist in real-time tracking and positioning, improving the accuracy of surveillance and reconnaissance missions and minimizing human risks. The popularity of small unmanned aircraft

systems (SUAS), developed using visual odometry as mentioned in [41] and [42], has exploded in recent years and has seen increasing use in both commercial and military sectors.

Space rovers use visual odometry to navigate and explore extraterrestrial terrains, where GPS is unavailable, making it an indispensable technology for future space missions. The two Mars Exploration Rovers (MER) operated by NASA have successfully shown off their robotic visual odometry capabilities on a different planet, as stated in [43] and [44]. This gives every rover precise positional information, enabling it to recognize and adjust for unanticipated slippage during a drive independently. Since it has decreased the number of days needed to drive to intriguing locations, it has enhanced mission science return and allowed the rovers to drive more safely and effectively in highly sloped and sandy terrains.

Visual odometry is a fundamental component of SLAM systems, enabling devices to create maps [13] of unknown environments while simultaneously tracking their own position within those environments. This is valuable in applications ranging from robotics [45] to augmented reality (AR).

It is used in medical robotics [46] to provide real-time feedback during surgical procedures. It helps in precisely tracking the movement of surgical instruments and maintaining accurate alignment with preoperative imaging.

I.4. Metaheuristic method

Numerous optimization techniques utilizing metaheuristic methods have been developed in the literature for motion estimation by vision issue employing sparse methods. A block matching approach for movement estimation based on the artificial bee colony (ABC) algorithm was proposed by Cuevas *et al*. [47]; this algorithm significantly minimizes the computation of search locations. Marco [48] introduced an optical flow estimation method that utilizes a genetic algorithm (GA). The technique segments the image into generic shape regions based solely on luminance and color information. Subsequently, for each region, a motion model is estimated using a GA. Additionally, Shahbazi *et al.* [49] used genetic algorithms and RANSAC to improve robust sparse matching method in movement estimation.

The odometry estimation technique LIdar-Monocular visual Odometry (LIMO) combines a camera and Light Detection and Ranging sensor (LIDAR) dataset for visual localization. It tracks features from both the camera and LIDAR measurements to estimate motion using bundle adjustment based on robust key frames. Adarsh *et al.* argued in [50] that the utilization of the genetic algorithm to optimize parameters for LIMO aims to enhance its localization and motion

estimation performance. In addition, the genetic algorithm can find the rotation and translation of a device accurately when the 3D structure of the device is given. Yu and Wong described in their paper [51] a method to estimate the pose using the genetic algorithm of a real object. They showed that the proposed approach applies to visual odometry and augmented reality applications.

The diagnosis of diseases often makes use of the ultrasound image sequence of the soft tissue. A novel algorithm for evaluating soft tissue motion has been created. The proposed iterative firefly algorithm (IFA) outlined in [52] selects a few candidate points to determine the optimal motion vector.

However, dense methods use all pixels in the frame to estimate the motion. Although this method has a high computational cost, it produces very valuable results. If the amount of input information increases, the movement will be estimated with high accuracy. In [53] , Baik et al. presented a new particle filtering-based system for visual odometry that shows remarkable resilience to sudden camera movements. Additionally, by reorganizing the traditional vector space PSO method while taking the geometry of the special Euclidean group *SE(3)* into account, they were able to perform the suggested visual ego-motion estimation approach in real-time.

Kostusiak and Piotr described a particle swarm optimization (PSO) method and an evolutionary algorithm variant in various robots as a means of finding the best parameters of a simple RGB-D visual odometry system as mentioned in [54].

PSO has already been used for several vision tasks, including visual SLAM [55], [56] and visual tracking [57], [58]. References [55] and [56] also exclusively address 3-DOF ego-motion estimation issues, meaning that the state space is limited to a 2-D plane. We address generic 6-DOF ego-motion estimation issues, unlike references [55] and [56]. Our goal in this study is to solve visual odometry in the 6-DOF search space by using various optimization methods.

I.5. RGB-D Benchmark

The RGB-D benchmark created by Sturm et al. [7] offers a framework for evaluating algorithms that use RGB-D images, such as SLAM or visual odometry. It includes multiple consecutive RGB-D images along with a dataset of ground truth camera motion information and camera calibration. This information is used to evaluate the different optimization methods.

Furthermore, the RGB-D benchmark includes some tools that may be used to determine quality measures for an estimated motion in relation to the ground truth. The absolute trajectory error and the relative pose error are two examples of these measurements. The absolute trajectory error

evaluates the difference between the estimated and real endpoints and provides insight into the performance of the visual odometry method. On the other hand, the relative pose error measures the translational or rotational drift of the estimation relative to the ground truth over a specific temporal distance, such as drift per frame or per second, making it suitable for evaluating visual odometry approaches. For each metric, we can compute the Root Mean Square Error (RMSE), mean, median, standard deviation, minimum, and maximum values.

However, a drawback of these diverse options is the absence of a standardized metric, leading to varying measurements among authors; some may measure the median drift per frame while others measure the RMSE drift per second.

Utilizing the RGB-D benchmark for evaluation offers the benefit of enabling an objective comparison among various methods. The extensive array of datasets featuring diverse scene content guarantees robustness in the generalization of the evaluated approach. Additionally, it frees researchers from the laborious process of capturing real-world datasets with ground truth data.

I.6. Conclusion

In this section, we mentioned some related work associated with our project. Our work represents a continuation of previous research. Subsequently, we focus on optimization methods for dense visual odometry.

Our research builds upon the findings of earlier studies in this field. Specifically, we delve into optimization techniques tailored for dense visual odometry. The significance of our work lies in its contribution to advancing the understanding and implementation of methods that enhance the accuracy and efficiency of visual odometry processes. This detailed examination serves as the foundation for our subsequent discussions and findings.

Chapter II

Visual odometry method

II. Visual odometry method

II.1. Introduction

In this part, we will briefly explain the development of the visual odometry system in a static scene, from reality to an abstract model expressed as a mathematical equation, which we call system modeling, as described in [28] and [59]. Only a sequence of images is received from the scene by a camera mounted on a moving device whose path of movement needs to be accurately tracked. The warping of an image is an important component in this equation, and this is related to the parameters of the camera that captures the images. If the camera moves quickly, it becomes difficult to determine its exact location, and this problem can be solved by using a multi-resolution pyramid, which we will discuss next.

II.2. System modeling

Visual odometry estimates the motion between two consecutive frames (I_t, I_{t+1}) that are taken by a camera fixed to the top of a moving robot. We will demonstrate how a mathematical equation has been created to represent this problem.

An image *r*, named residual, is produced by subtracting the intensities of each pair of pixels at the same position from two consecutive gray images; the pixels' intensity values of these pixels in these RGB images is calculated as:

$$I_t=(I_R+I_G+I_B)/3 \qquad \text{II.1}$$

where I_R, I_G, and I_B are the pixels' color components, respectively, red, green, and blue.

The pixels' intensity in *r* is lower, resulting in a small error value. The intensity error *E* between two successive RGB frames of *N* pixels, as mentioned in [5], is represented by the following function:

$$E(\xi)=\frac{1}{N}\sum_{i=1}^{N}|I_{t+1}(\omega(\xi,p_i))-I_t|^2=\frac{1}{N}\sum_{i=1}^{N}|r_i(\xi)|^2 \qquad \text{II.2}$$

Where:

- N is the pixel number of the full image, when image resolution is (640,480); N=640×480.
- p_i is the pixel at index i.
- $\xi \in \mathbb{R}^6$ is the estimated motion represented in six degrees of freedom.
- $\omega(\xi, p_i)$ is the warping function of the next image captured at $t+1$.
- I_t is the image captured at time t.
- $I_{t+1}(\omega(\xi,p_i))$ is the warping of the image I_{t+1} and it consists of the migration of the pixels towards new positions in the image by the inverse of the assumed motion value. Thus, an image is created that is identical to the image taken at time t.

- $r_i(\xi)$ is the residual image, which is produced from the difference between the current image I_t and the warped image $I_{t+1}(\omega(\xi,p_i))$.

The estimation of the camera motion is based on minimizing the average value of image residual intensity *r*, or the error value produced by equation II.2. In the ideal case, when this error is equal to zero, it implies that the assumed motion is equal to the motion in reality. However, in practice, this error is never null due to capture noise, variations in the visibility angles of objects during the motion, and other factors.

This is why our goal is limited to minimizing the error in estimating the optimal motion vector by optimizing the following function:

$$\xi=\min_{\xi}E(\xi)=\min_{\xi}\ \frac{1}{N}\sum_{i=1}^{N}\ |r_i\,(\xi)|^2. \qquad \text{II.3}$$

This function is solved by various optimization methods in this thesis. The warping function is considered the principal component for calculating the error $E_i\,(\xi)$.

II.3. Construct the Warp function

Most of the pixels in the frame I_{t+1} change their position using the warping equation $\omega(\xi, p)$ in order to create the warped frame I_{t+1} (ω(ξ, p)) noted in equation II.2 . This frame would have been captured by a camera if the robot returned to the position where it originally captured the frame I_t, after moving in a direction inverse to the real motion ξ. Subsequently, we subtract this newly warped image from the original I_t, and the effectiveness of the proposed motion ξ is assessed by calculating the error using equation II.2. If the error is smaller and tends towards 0, then the corresponding movement ξ is closer to the real movement because the two images, I_{t+1} (ω(ξ, p)) and I_t ,are almost identical. This operation of warping serves as a critical step in determining the optimal motion.

The warp function is made up of the collection of transformations depicted in Figure II.1 as described in [5], [6] and [60]. A pixel *p* of the frame I_{t+1}, which has the coordinates (*u;v;d*), is projected onto a *3D* point *M (X; Y; Z)* by the transformation P^{-1}, as mentioned in equations II.7 . After that, *M* is transformed from the landmark associated with I_{t+1} to a *3D* point *M′* in coordinates (*X′; Y′; Z′*) in the landmark attached to $I_{t+1}(\omega(\xi, p_i))$ by the transformation $g\ (\xi)$ as shown in the following equation:

$$P_{M'} = g\ (\xi)\times P_M. \qquad \text{II.4}$$

Finally, *M′* is projected onto a pixel *p′* in the frame $I_{t+1}(\omega(\xi,p_i))$ by the transformation *P* as mentioned in equations II.6. Thus, in the following, the function of the warp is formed as:

$$\omega(\xi,p)=P(g\ (\xi)P^{-1}(p)). \qquad \text{II.5}$$

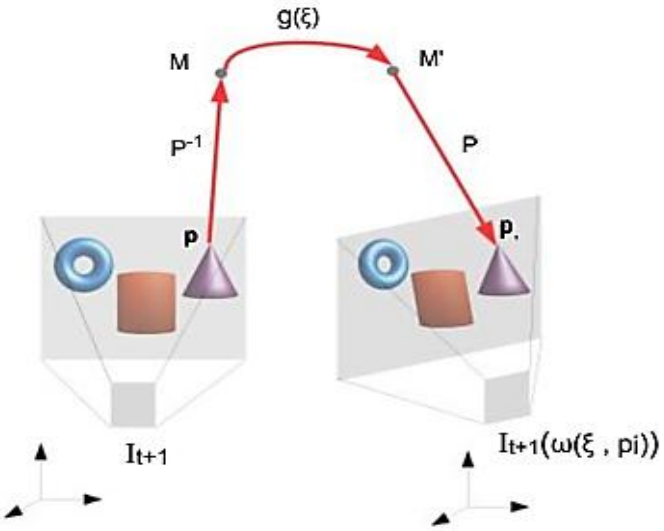

Figure II.1: The warp process consists of transforming each pixel in the frame I_{t+1} into another pixel in the warped frame $I_{t+1}(\omega(\xi, p_i))$.

The transformations P and P^{-1}, along with the rigid body motion g, are important components for constructing the warp function. In the following, we will demonstrate the formula for this transformation based on the camera model. After that, we will explain how to express the motion in the form of rigid body motion g.

The camera model provides the transformation of points in the 3D scene to the 2D plane as pixels in the image; this process is accomplished with a camera [61]. This transformation P from $3D$ to $2D$ plane is called projection and is represented by

$P: \mathbb{R}^3 \rightarrow \mathbb{R}^2$

A simple pinhole camera model, represented in Figure II.2, summarizes the working principle of the camera as a 3D point $L(X; Y; Z)$ being projected to a pixel $S(u; v; d)$ in the image plane of an ideal pinhole camera. This figure aids in deducing the mathematical relationship between the 3D scene and the 2D plane as a transformation P and P^{-1}.

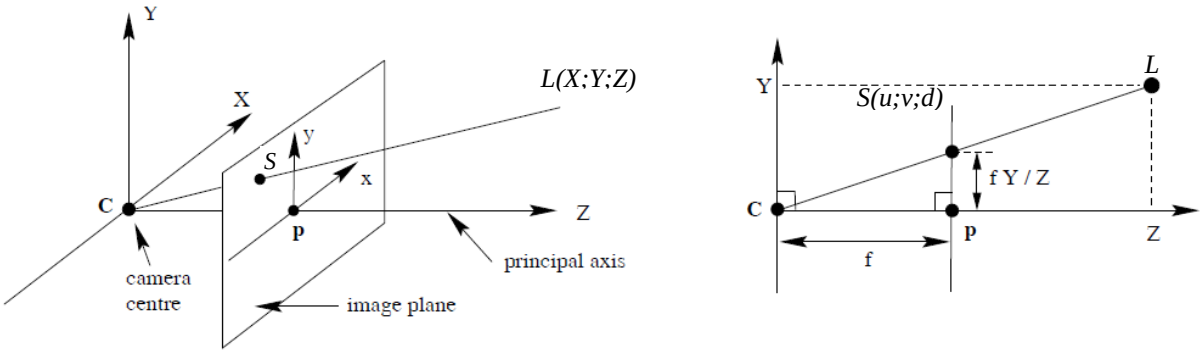

Figure II.2: Pinhole camera model.

The projection of a spatial point *L* to plane point *S* as presented in Figure II.2 is called a perspective projection. Only light rays that pass through the pinhole and are projected onto the image plane are translated into a two-dimensional image. The position of the pinhole is the camera optical center *C*. The distance between the image plane and the optical center is the focal length *f*. In practice, the image plane is situated behind the optical center rather than in front of it. However, for the sake of the model's generality, this rearward displacement can be disregarded without sacrificing accuracy. Every 3D point, at where the line connecting it to the optical center intersects the image plane, is effectively represented in the image according to this simplified model.

Camera calibration is a critical process that involves the determination of both intrinsic and extrinsic parameters of a camera. These parameters play a pivotal role in numerous computer vision applications, including image analysis and augmented reality.

Intrinsic parameters, which encompass key internal characteristics such as the principal point and focal length, are acquired through a standardized camera calibration procedure as outlined in references [62], [63], and [64]. This calibration process typically entails capturing images of a known calibration object from various viewpoints, enabling the accurate determination of intrinsic camera characteristics, where:

- Focal length (f) indicates the distance between the image plane and the optical center of the camera. While the physical focal length (f), measured in millimeters, cannot be directly determined, the focal lengths in the *x* and y directions, represented by f_x and f_y, can be obtained through camera calibration. f_x and f_y are fundamentally the focal lengths represented in pixels. These focal lengths in pixels account for the rectangular nature of each pixel on a typical imager, where the lengths in the *x* and *y* directions are distinct. Since each pixel on a common imager is rectangular, we use two extra parameters with varying pixel lengths in *x* and *y*.

- Principal Point (C) indicates the optical center's coordinates in the image and is conventionally denoted as (c_x, c_y), where c_x represents the horizontal coordinate and c_y represents the vertical coordinate. These two parameters, c_x and c_y, deal with any potential misalignment between the image's center and the principal point. It is important to remember that c_x and c_y are given in pixel units, providing a precise indication of the offset from the image center and facilitating accurate camera calibration procedures.

The following formula links each *3D* point of coordinates *(X; Y; Z)* in space to its matching 2D pixel *(u;v;d)* as mentioned in [59] and [61]:

$P: \mathbb{R}^3 \to \mathbb{R}^2; \quad (X;\ Y;\ Z\ ) \to (u, v)$

$$\begin{cases} u = \frac{X \times f_x}{Z} + c_x. \\ v = \frac{Y \times f_y}{Z} + c_y. \\ \mathrm{d} = Z\ . \end{cases} \qquad \text{II. 6}$$

Where f (f_x, f_y) is the focal length and c (c_x, c_y) is the principal point of the camera. d is the depth of the pixel returned by the camera. The intrinsic parameters (f, c) can be obtained by a standard camera calibration procedure [62].

The RGB-D image allows the reconstruction of the *3D* point of the scene from the pixel.

It is possible to transition from a *3D* scene to a *2D* image using the projection *P*. The transformation P^{-1} links every 2D pixel (u, v, d) to its corresponding 3D point with coordinates (X, Y, Z) in space, as seen in the following equation:

$P^{-1}: \mathbb{R}^2 \to \mathbb{R}^3; \quad (u, v, d) \to (X; Y; Z)$

$$\begin{cases} X = \frac{u - c_x}{f_x} \times d \\ Y = \frac{v - c_y}{f_y} \times d \\ \mathrm{Z} = \mathrm{d} \end{cases} \qquad \text{II. 7}$$

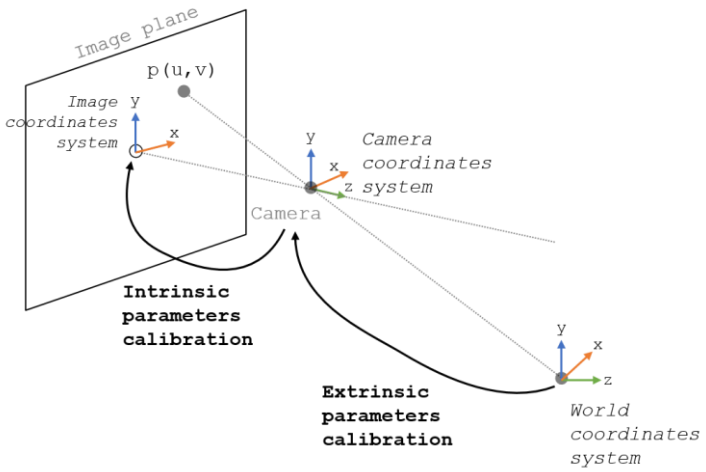

Figure II.3: Parameters camera calibration

On the other hand, the camera's motion in the outside world is described by extrinsic parameters, as represented in Figure II.3, where:

- The translation vector (t) typically refers to the displacement of the robot's position in the Cartesian coordinate system. This can be along the X, Y, and Z directions in three-dimensional space.

- The rotation matrix (R) is an action of turning a robot around a particular axis. Rotation can occur around the X, Y, or Z axis in three-dimensional space.

In the context of camera extrinsic parameters or robotics, the combination of translation and rotation captures the overall transformation or pose of an object or camera in three-dimensional space. The combination of these movements is often represented by matrices [R,t], such as the extrinsic matrix in the case of a camera's pose.

Extrinsic parameters define the camera's location and orientation in the external environment. As the camera moves or is relocated, it may also change.

A rigid object's location in relation to a fixed reference point at all times precisely describes its motion in three-dimensional space. R and t can be combined into one rigid transformation matrix, g, as shown below:

$$g = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}. \qquad \text{II.8}$$

Where $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ is the rotation of movement, and $t = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$ is the translation of movement.

The concise and effective representation of the vector ξ with six degrees of freedom simplifies the complexity associated with describing the motion of a rigid body. This streamlined approach is particularly beneficial when employing optimization methods for motion determination. This vector is represented as follows:

$$\xi = [\, v_1 \; v_3 \; v_2 \; w_1 \; w_2 \; w_3]. \qquad \text{II.9}$$

With $v = (v_1; v_2; v_3)^T$ as the linear velocity and $w = (w_1;w_2;w_3)^T$ as the angular velocity.

Multiple rigid body motions can be chained by multiplying consecutive matrices g, which represent small transformations, to build all the trajectories. This shows that there is significant importance in writing the movement in the form of g. The identity transformation, meaning no translation and no rotation, is given by $R = I$ and $t = 0$.

The translation vector t stands out as canonical due to its equivalence to the first three degrees of freedom of the motion vector ξ. The simplicity of having three components in the vector mirrors the translational degrees of freedom, facilitating a clear and concise representation that aids both understanding and computation.

In contrast, the representation of rotation as a matrix R is non-canonical. Despite comprising nine parameters, it inherently possesses only three degrees of freedom associated with the rotational motion of the motion vector ξ. Despite its non-canonical nature, the matrix representation remains

a powerful tool for capturing the nuances of rotational motion within the broader context of rigid body transformations.

A minimal representation ξ for a robot motion g can be deduced by using the parameters of its associated Lie algebra *se(3)* given in [65]. Each transformation matrix in the Lie group *SE(3)* describing a rigid body movement has a representation with a 6 × 1 parameter vector $\xi = [v, w]$ in its corresponding Lie algebra.

The robot motion g can be computed from its Lie algebra vector ξ using the exponential map, as described in [5] and [66]:

$$exp: se(3) \rightarrow SE(3); \ \xi \rightarrow g$$

$$g(\xi) = e^{\hat{\xi}} \qquad \text{II.10}$$

With $\hat{\xi}$ is the skew symmetric matrix and it is written as:

$$\hat{\xi} = \begin{bmatrix} [w]_\times & v \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -w_3 & w_2 & v_1 \\ w_3 & 0 & -w_1 & v_2 \\ -w_2 & w_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \qquad \text{II.11}$$

The operator $[k]_x$ creates a 3×3 skew symmetric matrix from a 3×1 vector $k = (x; y; z)$,

$$[k]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix} \qquad \text{II.12}$$

The exponential matrix $e^{\hat{\xi}}$ is written in the following form as mentioned in [67] and [68]

$$e^{\hat{\xi}} = \begin{bmatrix} e^{[w]_\times} & V v^T \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \qquad \text{II.13}$$

$$e^{[w]_\times} = I + \frac{\sin(||w||)}{||w||}[w]_\times + \frac{1-\cos(||w||)}{||w||^2}[w]_\times{}^2 \qquad \text{II.14}$$

and V is :

$$V = I + \frac{1-\cos(||w||)}{||w||^2}[w]_\times + \frac{||w|| - \sin(||w||)}{||w||^3}[w]_\times{}^2 \qquad \text{II.15}$$

The inverse of the exponential map is called the logarithm map.

$$log: SE(3) \rightarrow se(3); \ g \rightarrow \xi$$

$$\xi = log(g) \qquad \text{II.16}$$

The code MATLAB of the exponential and logarithm map is mentioned in annex C.

Where the identity transformation is obtained for $\xi = 0$.

The corresponding w is given by

$$w = \frac{||w||}{2\sin(||w||)} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix} \quad \text{II.17}$$

Where:

$$||w|| = \cos^{-1}\left(\frac{\text{trace(R)} - 1}{2}\right) \quad \text{II.18}$$

And

$$v = V \setminus t \quad \text{II.19}$$

The vectors v and w will be used to construct the motion ξ *as* mentioned in equation II.9. We have demonstrated how to move from expressing motion in the form ξ to the form g and vice versa. In addition, we mentioned the importance of the need for the two forms to implement the algorithms in optimization methods.

Thus, we explained all the parts of the warp function II.5 as ξ, $g(\xi)$, P, and P^{-1}, which construct the warping of each pixel in the frame I_{t+1} to $I_{t+1}(\omega(\xi, p_i))$, and this frame is used to deduce the residual image r and the error using the Function II.2. Regarding ξ, the metaheuristic approach suggests a range of values as motions for the particles; we then compute the associated error value for every particle, as we shall detail in our suggested method. However, the exact algorithm is an iterative method. At each iteration, an increment is calculated and added to the solution obtained at the previous iteration. This process continues for several times until the stopping condition is reached.

II.4. Pyramid Multi-resolution

In the context of visual odometry, a pyramid multi-resolution approach is a technique used to enhance the efficiency of motion estimation algorithms. Visual odometry is the process of estimating the motion of a camera by analyzing sequential images. The pyramid multi-resolution technique involves creating a multi-level image pyramid, with each level representing a different scale or resolution of the original image. This pyramid is then utilized to perform motion estimation at different scales, as mentioned in [6], [69], and [70]. Here's how the pyramid multi-resolution is used in our visual odometry experiments. The original image is down-sampled to create a series of images at different resolutions, forming a pyramid structure, where the down-sampled resolution (*DSR*) of the captured frame is performed by a factor of 2, as represented in Figure II.4.

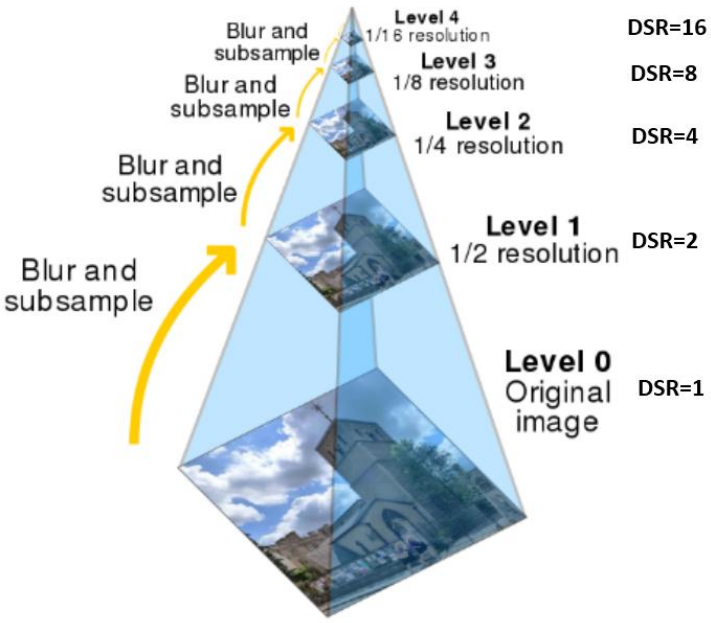

Figure II.4: Representation of an image pyramid with 5 levels.

Each level of the pyramid represents a different scale of the image, with the bottom level having the original resolution and subsequent levels having reduced resolutions.

The visual odometry algorithm starts motion estimation at the highest resolution level and calculates an initial motion estimate. This initial estimate is then used as an initialization for the next lower resolution level, as represented in Figure II.5.

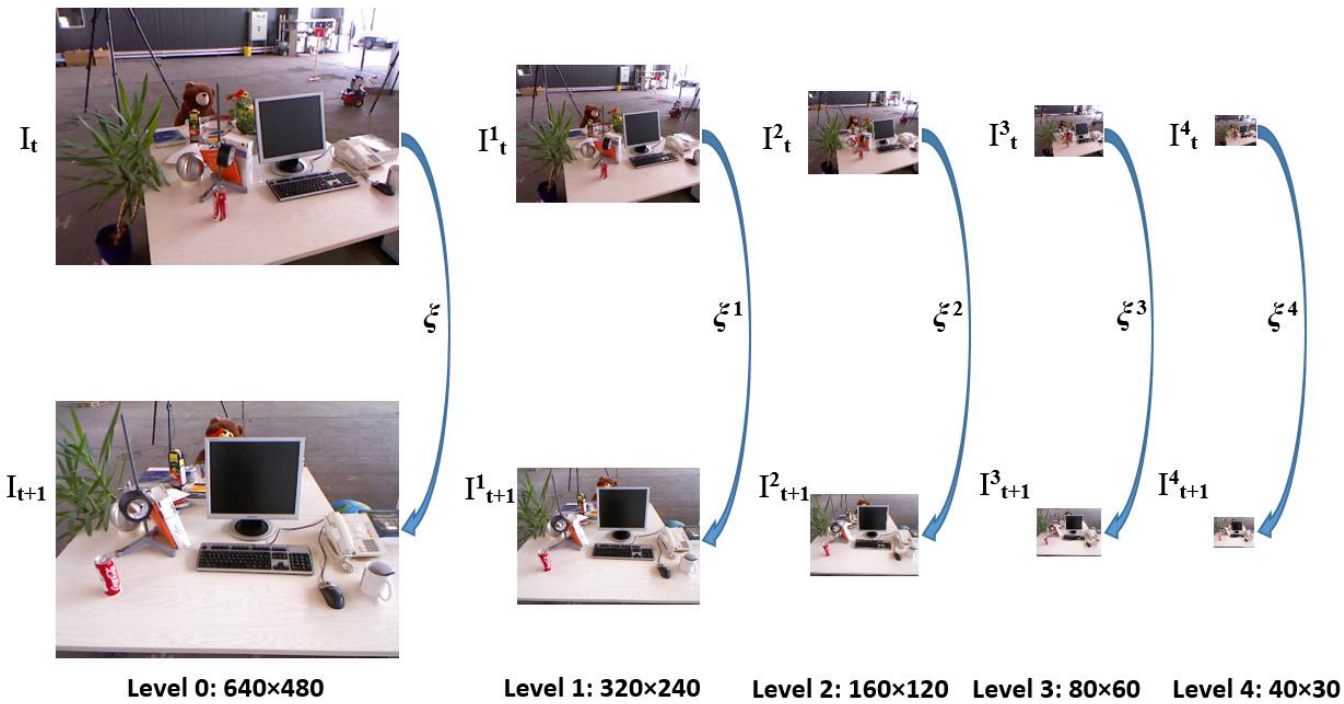

Figure II.5: Iterative image alignment process using a multi-resolution image pyramid.

In the first phase, we compute the motion ξ^4 using the image corresponding to high-level *DSR* (level 4). This motion will be utilized as the initialization for the next lower level in the pyramid down to the original resolution of the image, where we deduce the optimal motion ξ. Starting with a rough motion estimate at high levels helps in quickly converging to a solution. This initial estimate provides a good starting point for the lower levels, where the algorithm can focus on refining the motion with greater precision.

The motion estimation process is iteratively refined at each level of the pyramid. Starting from a coarser level allows for capturing large-scale motion, and as the algorithm progresses to finer levels, it refines the Function II.3 and it is solved by minimizing the photometric intensities of the residual image *r*. The solution will be closer to the ground truth motion in the case of tiny motion ξ or a small image resolution. To improve the accuracy of motion estimation in the case of a large displacement between two consecutive frames and the variations in scale due to changes in scene depth, visual odometry should be able to handle variations in scale using pyramid multi-resolution. The multi-resolution approach helps handle variations in scale and perspective across different frames. By initially processing lower-resolution images, the algorithm reduces computational demands, making it more efficient. This pyramid multi-resolution strategy is widely used in visual odometry algorithms to improve their performance in various environments and conditions and ensure convergence to the optimal solution.

II.4. Conclusion

Modeling a real-life problem into a mathematical equation while simplifying all its parts is the first step toward solving this problem. We have devoted an important part of this chapter to explaining the modeling of the visual odometry problem through a mathematical function as an error equation that uses two consecutive images as input. The error equation of visual odometry was explained in detail, in addition to the benefits of using the multi-resolution pyramid to solve the problem of large motion and the breadth of the scene in which the camera moves. This equation will be solved using several optimization methods in the next chapter, which is the core of our work in this thesis.

Chapter III

Optimization Methods

III. Optimization Methods

III.1. Introduction

At the outset of this section, an overview of optimization methods is presented, with the focus being on what was used in the experiments of exact or approximate methods. The subsequent narrative delves into a meticulous exploration of the Gauss-Newton method. Additionally, attention is directed towards an evolutionary method, specifically the genetic algorithm, and its implementation.

Then there are two noteworthy swarm-intelligence methods, called particle swarm optimization and the firefly algorithm. The intricate workings of these two algorithms are explained in this section. This comprehensive exploration of the diverse optimization methods landscape not only enriches the reader's understanding, but also sets the stage for a nuanced comparison and evaluation within the context of the experiments, and opens new horizons for discovering other methods.

III.2. Overview optimization methods

There exists a large number of real-life problems that are complex and difficult to solve, taking a significant amount of time to solve or being unsolvable at all mathematically. In these cases, exact algorithms are not appropriate or require a large amount of resources (e.g., computational cost) for using them. Therefore, approximate algorithms are needed. Among approximate algorithms, we can find two types: heuristics and metaheuristics. We focus in this work on metaheuristics. Figure III.1 shows a simple classification of optimization methods used throughout the history of computer science as described in [71].

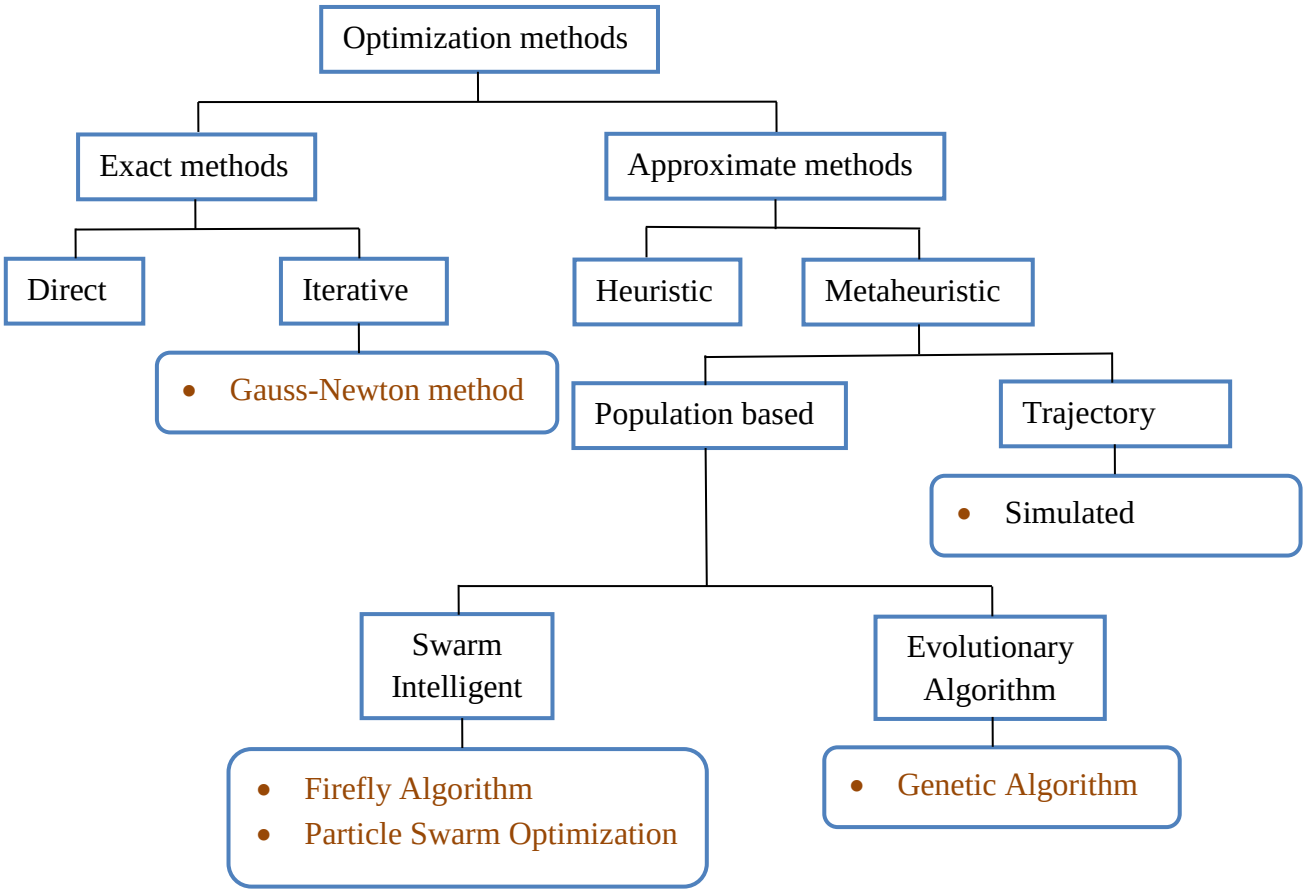

Figure III.1: General classification of the optimization techniques.

III.2.1. Exact method

Exact optimization methods aim to find the global optimum of a given objective function with a high degree of precision. They guarantee convergence to the global optimum, provided certain conditions are met. These methods often rely on mathematical proofs and theorems to ensure the correctness of their results. Exact methods are essentially composed of two types: direct and iterative.

A. Iterative method

In the realm of exact methods, the iterative approach plays a pivotal role. Iterative optimization techniques dynamically enhance solutions through incremental adjustments, iterating until convergence is achieved. One exemplary instance of an iterative method is the Gauss-Newton method. This method iteratively refines solutions, ultimately converging toward an optimal

solution. These methods improve the solution iteratively, making small adjustments at each step until convergence. Gradient descent [72] is another example of an iterative method.

B. Direct method

Direct methods involve solving problems in a finite number of steps (in theory), providing an exact solution if no rounding or computational errors occur. They are often contrasted with iterative methods, which converge to the solution through repeated iterations. Direct methods are commonly used in linear algebra, especially for solving systems of equations. Direct methods often involve mathematical transformations or explicit formulas to identify optimal points in the solution space.

Direct methods solve specific mathematical problems (like linear equations) exactly and are more focused on solving broader classes of optimization problems (like linear programming (LP) [73], integer programming, and Gaussian elimination [74], etc.).

III.2.2. Approximate methods

The exact methods are limited when it comes to solving complex and highly nonlinear problems. These methods may struggle with complex models, fail to find feasible solutions, or fail to converge to local optima, particularly in cases of non-linearity.

To address these challenges, the passage towards approximate methods is considered a successful trend. These approaches are required due to their potential to combine efficiency with the ability to find global optima. Unlike traditional techniques, approximate methods are positioned as a solution to overcome the lack of global optimum attainment in nonlinear optimization as well as the computational intensity associated with complex systems. Two methods can be distinguished in the approximate approach: heuristic and metaheuristic methods.

A. Heuristic methods

Heuristics and metaheuristics are used to solve optimization problems in different fields, such as artificial intelligence and computer vision, but their applications differ.

Heuristic methods, as outlined in references [75] and [76], are problem-solving strategies that rely on rules of thumb or practical approaches. While not guaranteeing an optimal solution, heuristics are instrumental in swiftly finding feasible solutions, particularly in computationally expensive scenarios involving complex problems.

These methods are used to solve a specific problem and are tailored to the characteristics of a given problem. Notably, heuristics do not guarantee an optimal solution but prioritize efficiency, providing solutions deemed "good enough" based on available information and constraints. Greedy algorithms are a common example of heuristics, as noted in [77]. They make locally optimal choices at each stage with the hope of finding a global optimum.

On the other hand, metaheuristics represent a higher-level approach, functioning as techniques or heuristics. The primary goal of a metaheuristic is to yield a heuristic that is capable of delivering a sufficiently good solution to an optimization problem. Unlike heuristics, metaheuristics operate on a broader scale, providing a framework for guiding the search for an optimal solution rather than being addressed to a specific problem.

B. Metaheuristic methods

Metaheuristic methods are an optimization approach designed to solve difficult and non-linear problems. Two Greek words are combined to form the word “metaheuristic”:

- The word "heuristic" is derived from the verb "heuriskein" (ευρισκειν), meaning better approximations that do not ensure the attainment of an exact solution.

- The prefix "meta," which means "beyond" or "on an elevated level," and the inclusion of the term "meta" indicate that the area of using metaheuristics must be able to extend to many varied problems that they can adapt to with more or less difficulty rather than being restricted to a specific problem.

A metaheuristic is officially characterized as an iterative generation method that implements learning strategies to organize information systematically, enhancing the efficiency of identifying near-optimal solutions. It cleverly directs a subordinate heuristic through the clever integration of diverse concepts, facilitating the exploration and exploitation of the search area.

Metaheuristics represent approximate algorithms that have proven to be effective stochastic optimization tools, capable of delivering satisfactory results for complex optimization problems [78], [79], and [80]. Metaheuristics, being generic tools, generally do not assume anything about the problem to be solved; instead, they merely use suitable representations of the solution, quality or fitness functions, and specific operators to direct the solution towards a good result. Nature is considered the inspiration for many metaheuristics, including swarm intelligence, evolutionary algorithms (EAs), and simulated annealing (SA).

Simulated annealing (SA) is one of the earliest bio-inspired optimization methods and it relies on the metal and crystal annealing process described in [81]. It is regarded as the oldest technique that explicitly outlines a plan for getting out of local optima.

These algorithms, which draw inspiration from nature, can be categorized as either population-based or trajectory-based. The first group works with a group of components called a population or swarm, whereas the second group deals with one element in the search space at a time, such as SA. In this chapter, we will explain these algorithms and their applications in visual odometry, the basic principles of swarm intelligence (SI) and evolutionary algorithms (EA), and then some of the major types of these methods.

a. Swarm Intelligent

The term swarm intelligence was first used to describe cellular robotic multi-agent systems, where a group of basic agents interacted with one another according to local rules in a given area [82]. These days, this term refers to the process of creating algorithms or tools for addressing problems inspired by the group behavior of social creatures. Several swarm intelligence methods are employed to tackle challenging optimization problems, such as particle swarm optimization (PSO) [83].

b. Evolutionary Algorithm

The Darwinian theory of evolution, as presented in [84], serves as the foundation for evolutionary algorithms. According to Darwin's theory, as stated in [85], a population of individuals with the ability to reproduce through genetic diversity after selection would produce new populations of individuals that are progressively better adapted to their surroundings. When these straightforward natural methods are used in computation, a variety of algorithms are produced, including genetic algorithms (GA) in [86].

III.3. Energy-Based using Gauss-Newton method

Visual odometry (VO) refers to the process of estimating the position and orientation of a robot or vehicle using visual data from a camera. There are several methods for visual odometry, including feature-based, exact, and approximate methods. Among these, the energy-based method, which falls under exact methods, is particularly interesting.

In the context of visual odometry, as detailed in [26] and [6], an energy-based method typically involves the formulation of an energy function that estimates the motion based on a visual dataset, as in our experiments, using RGB-D images captured from a Microsoft Kinect camera. Visual odometry is a computer vision technique used in robotics and autonomous systems to estimate the motion of a camera or vehicle by analyzing sequential images.

This method does not rely on extracting and matching features (such as points or edges) between images. Instead, it uses the entire image dataset, which can be more effective for estimation. The main step of the energy-based method is the photometric error minimization between successive images. An optimization algorithm, such as the Gauss-Newton method, is used to minimize the photometric error.

The basic steps of an energy-based visual odometry method can be outlined as follows: we suggest an energy Function II.2 that seeks to determine the optimal rigid body motion that transforms one RGB-D image into another, assuming that these images were captured by a moving camera in a static scene. Next, we propose linearizing the energy function, which leads to a normal equation for the twist coordinates that represent the movement of the rigid body ξ.

The resolution of the least squares equation II.2, as outlined in [87], aims to determine the unknown variables ξ.

In the quest to find this unknown, the method minimizes the quadratic sum of the difference between two successive RGB-D frames. Once the derivative with regard to ξ equals zero, the minimum of this function is attained, and vice-versa:

$$\frac{\partial E(\xi)}{\partial \xi} = \frac{2}{N}\sum_{i=1}^{N} \frac{\partial r_i(\xi)}{\partial \xi} r_i(\xi) = 0 \qquad \text{III.1}$$

III.3.1. linearization of the least squares method

In cases where the equation $E(\xi)$ is nonlinear, achieving a direct solution becomes challenging. Multiple approaches exist for addressing this nonlinear case, and here we use the Gauss-Newton method. This method provides an iterative solution by making the function $r_i(\xi)$ linear to establish a linear dependence with k. Therefore, using the first order Taylor series, as described in [88], $r_i(\xi_k)$ is linearized at each iteration k in the vicinity of the solution of the preceding iteration $\xi = \xi_{k-1}$ so that:

$$r_i(\xi_k)|_{\xi=\xi_{k-1}} \approx r_i(\xi_{k-1}) + \frac{\partial r_i(\xi)}{\partial \xi}\Big|_{\xi=\xi_{k-1}} . (\xi_k - \xi_{k-1}) \qquad \text{III.2}$$

The Jacobian $J_i(\xi_{k-1})$ is equal to the derivative of residual $r_i(\xi_k)$ with respect to the camera pose parameters ξ.

$$\frac{\partial r_i(\xi)}{\partial \xi} \Big|_{\xi = \xi_{k-1}} = J_i(\xi_{k-1}) \qquad \text{III.3}$$

This is the Jacobian matrix and $\xi_k - \xi_{k-1}$ is considered the increment, represented as $\Delta\xi$. Thus, the Function III.2 can be constructed as follows:

$$r_i(\xi_k)|_{\xi = \xi_{k-1}} = r_i(\xi_{k-1}) + J_i(\xi_{k-1}).\ \Delta\xi \qquad \text{III.4}$$

By replacing the equivalence of the equations $r_i(\xi_k)$ and $\frac{\partial r_i(\xi)}{\partial \xi}$ into equation III.2, it produces:

$$\sum_{i=1}^{N} (J_i(\xi_{k-1})^T .(r_i(\xi_{k-1}) + J_i(\xi_{k-1}).\ \Delta\xi)) = 0 \qquad \text{III.5}$$

After developing and rearranging the terms, we obtain:

$$\sum_{i=1}^{N} J_i(\xi_{k-1})^T .J_i(\xi_{k-1}).\ \Delta\xi = -\sum_{i=1}^{N} J_i(\xi_{k-1})^T .\ r_i(\xi_{k-1}) \qquad \text{III.6}$$

The Equation III.6 is written in matrix form as follows:

$$J(\xi_{k-1})^T .J(\xi_{k-1}).\ \Delta\xi = -\ J(\xi_{k-1})^T .\ r(\xi_{k-1}) \qquad \text{III.7}$$

The solution to this equation is obtained by multiplying by the term $(J(\xi_{k-1})^T .J(\xi_{k-1}))^{-1}$ on both sides:

$$\Delta\xi = -(J(\xi_{k-1})^T .J(\xi_{k-1}))^{-1}.\ J(\xi_{k-1})^T .\ r(\xi_{k-1}) \qquad \text{III.8}$$

With $J(\xi_k)$ being the Jacobian matrix of size $n \times m$. Therefore, at each iteration k, the increment $\Delta\xi$ is first calculated using Equation III.8, and the solution at iteration k is obtained by adding to the value of obtained at iteration $k-1$:

$$\xi_k = \xi_{k-1} + \Delta\xi \qquad \text{III.9}$$

The iteration value corresponds to the number of pyramid multiresolution levels. At each level, the increment is calculated and added to the motion of the previous iteration until the first level is reached, where the image is complete, and thus the optimal solution is found using this method.

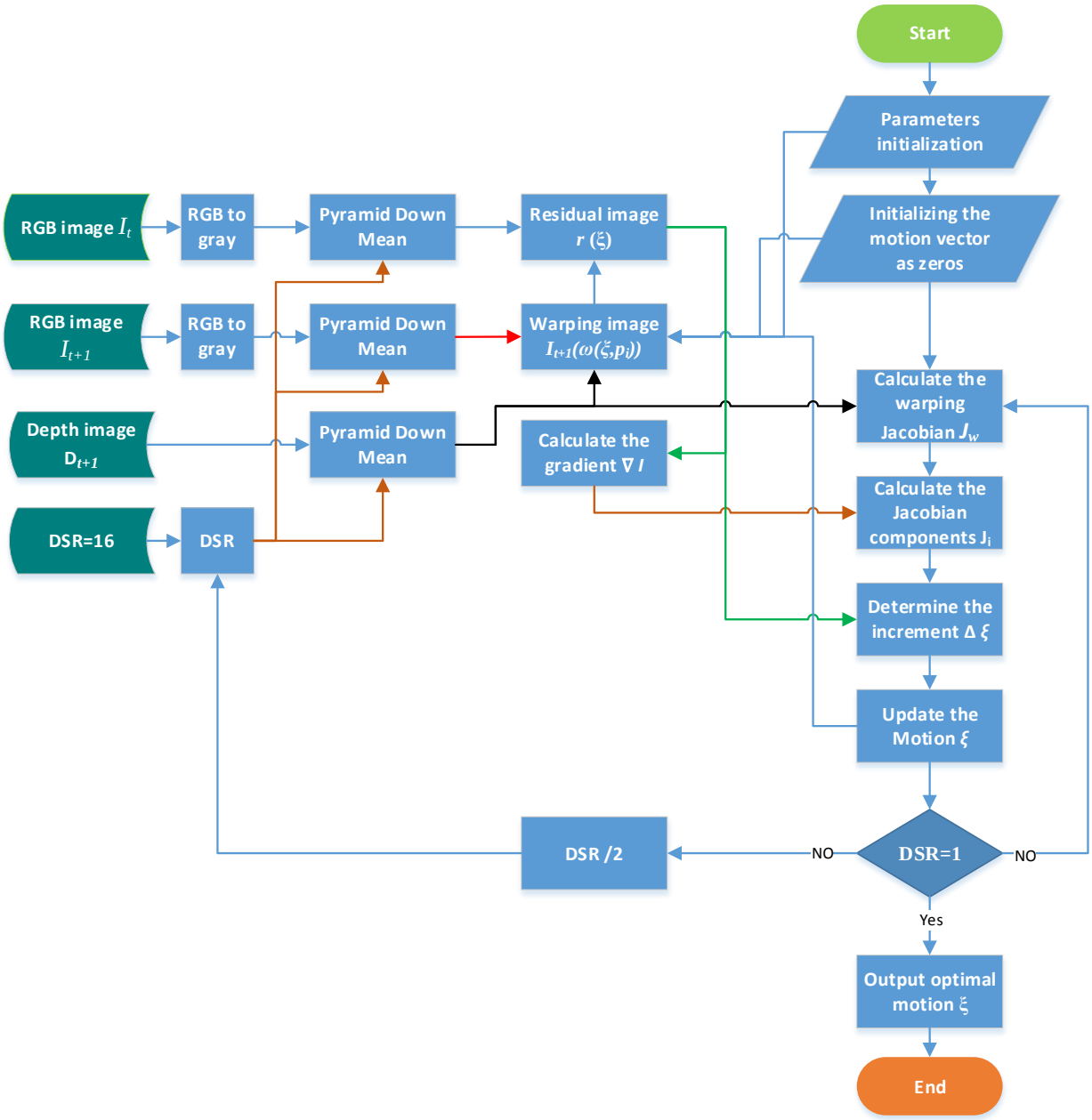

Figure III.2: Flowchart of the motion estimation using the energy-based method

Exact or direct methods for visual odometry estimate the motion directly from image intensities by minimizing the photometric error. The main steps in the energy-based method involve minimizing the photometric error between two consecutive images, thereby calculating the pixel displacement in the second image with the first image as a reference. The photometric error $r_i(\xi)$ is defined as the difference in intensity values of corresponding pixels in two consecutive images; the first image I_t is considered the reference and the second is the warped image $I_{t+1}(\omega(\xi, p_i))$.

As previously explained, the energy-based method converges to the nearest global minimum and may exhibit divergence in certain situations, such as large motions. To handle this and improve

convergence, a multi-scale pyramid approach is often used. Images are processed at different resolutions, starting from a coarse level and refining the estimates at finer levels. The initial motion ξ_1, as mentioned in Figure II.5, is crucial in this iterative process and should be close to the global minimum for successful convergence. Utilizing a multi-resolution pyramid helps guide the algorithm towards a favorable selection of ξ_1 during its execution. At level 4 of the multi-resolution pyramid, where *DSR* = 16, the process begins.

Figure III.2 presents a flowchart describing the execution process of the energy-based method. The initial parameters include the focal lengths in both the *x* and *y* directions (f_x, f_y), and the pixel coordinates of the principal point (c_x, c_y), which are input into the algorithm. The motion vector is initialized as zero. These initial parameters, along with the motion and the RGB-D image I_{t+1}, are used to calculate the warping image using Equation II.5, subsequently deducing the composition of the residual image $r_i(\xi)$ by Function II.2.

This prior motion is used to calculate the warping Jacobian J_w, as shown in Figure III.2 and described in Equation A.12. The next step is to deduce the Jacobian components using the warping Jacobian *Jw* and the gradient ∇I of the residual image $r_i(\xi)$. This process paves the way for calculating the increment using Equation III.8 and updating the motion by relation III.9.

Thus, as iteration *k* increases, the image resolution moves to the lower level of the pyramid until the *DSR* equals one, corresponding to the use of the original image at full resolution. An optimal solution is produced at the end of the energy-based method's execution.

III.3.2. Calculation of the Jacobian matrix

The Jacobian matrix $J(\xi_k)$ takes the form of an n×6 matrix with the following structure:

$$J(\xi_k) = \begin{bmatrix} \frac{\partial r1}{\partial v1} & \frac{\partial r1}{\partial v2} & \frac{\partial r1}{\partial v3} & \frac{\partial r1}{\partial w1} & \frac{\partial r1}{\partial w2} & \frac{\partial r1}{\partial w3} \\ \frac{\partial r2}{\partial v1} & \frac{\partial r2}{\partial v2} & \frac{\partial r2}{\partial v3} & \frac{\partial r2}{\partial w1} & \frac{\partial r2}{\partial w2} & \frac{\partial r2}{\partial w3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial ri}{\partial v1} & \frac{\partial ri}{\partial v2} & \frac{\partial ri}{\partial v3} & \frac{\partial ri}{\partial w1} & \frac{\partial ri}{\partial w2} & \frac{\partial ri}{\partial w3} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial rN}{\partial v1} & \frac{\partial rN}{\partial v2} & \frac{\partial rN}{\partial v3} & \frac{\partial rN}{\partial w1} & \frac{\partial rN}{\partial w2} & \frac{\partial rN}{\partial w3} \end{bmatrix} = \begin{bmatrix} J1(\xi\, k) \\ J2(\xi\, k) \\ \vdots \\ Ji(\xi\, k) \\ \vdots \\ JN(\xi\, k) \end{bmatrix} \qquad \text{III.10}$$

Where *N* denotes the number of pixels in the image, and 6 represents the number of degrees of freedom, encompassing linear speed (v_1; v_2; v_3), and angular motion (w_1; w_2; w_3). Each row $J_i(\xi_k)$ within this matrix signifies the derivative of the residue r_i associated with pixel p_i with respect to

the iteration *k*. It's essential to note that r_i corresponds to the intensity error of pixels in the two frames.

$$r_i(\xi) = I_{t+1}(\omega(\xi, p_i)) - I_t(p_i) \qquad \text{III.11}$$

Moreover, we have the following equation:

$$J_i(\xi_k) = \frac{\partial r_i}{\partial \xi} \mid_{\xi = \xi_k} \qquad \text{III.12}$$

Applying the theorem of compound function differentiation, as described in Annex A, we can express $J_i(\xi_k)$ in the following manner:

$$J_i(\xi_k) = \nabla I . J_w(\xi_k) \qquad \text{III.13}$$

The gradient of each pixel in the image in the direction of *x* and *y* is represented by a 1×2 matrix called ∇I as follows:

$$\nabla I = [\, \nabla I_x \,, \nabla I_y \,] \qquad \text{III.14}$$

The detailed computation of the warping Jacobian J_w is available in Annex A. Ultimately; the final Jacobian matrix as described in [89], takes the following form:

$$J_i(\xi_k) = [\, \nabla I_x \,, \nabla I_y \,] . \begin{bmatrix} \frac{fx}{Z'} & 0 & -\frac{fx.X'}{Z'^2} & -\frac{fx.X'.Y'}{Z'^2} & fx + \frac{fx.X'^2}{Z'^2} & -\frac{fx.Y'}{Z'} \\ 0 & \frac{fy}{Z'} & -\frac{fy.Y'}{Z'^2} & -fy - \frac{fy.Y'^2}{Z'^2} & \frac{fy.X'.Y'}{Z'^2} & \frac{fy.X'}{Z'} \end{bmatrix} \qquad \text{III.15}$$

The calculation of $J_i(k)$ depends on the coordinates of the transformed pixel *M'(X'; Y'; Z')* and so must be calculated at each *DSR*. Backer and Matthews introduced a technique in [29] known as the compositional inverse, enabling the pre-computing of the Jacobian on the reference image I_t. This pre-calculated Jacobian remains constant throughout the minimization process, significantly enhancing the algorithm's execution speed by eliminating the need to recalculate the extensive Jacobian matrix $\mathbb{R}^{n\times 6}$ at each iteration. Consequently, employing the inverse compositional method transforms the new Jacobian, denoted as J_i in the subsequent discussion, as follows:

$$J_i = [\, \nabla I_x \,, \nabla I_y \,] . \begin{bmatrix} \frac{fx}{Z} & 0 & -\frac{fx.X}{Z^2} & -\frac{fx.X.Y}{Z^2} & fx + \frac{fx.X^2}{Z^2} & -\frac{fx.Y}{Z} \\ 0 & \frac{fy}{Z} & -\frac{fy.Y}{Z^2} & -fy - \frac{fy.Y^2}{Z^2} & \frac{fy.X.Y}{Z^2} & \frac{fy.X}{Z} \end{bmatrix} \qquad \text{III.16}$$

Nevertheless, in our experiments, we recalculated the $J_i(\xi_k)$ at each *DSR* as a choice.

III.4. Genetic Algorithm for motion estimation

In the 1970s, researchers created genetic algorithms [90], an important type of optimization algorithm, to understand how genes behave and how living things reproduce. Over time, these algorithms have found applications in the fields of artificial intelligence and computer science. The primary function of a genetic algorithm is to generate multiple motions by employing predetermined equations based on the existing motions of parents. Subsequently, the algorithm selects motions that yield minimal error, and this error is determined by Function II.2. This process is iteratively repeated until the specified stopping conditions are met. Ultimately, the optimal motion, denoted as ξ, is determined using Equation II.3.

The following explains the genetic algorithm (GA) design as it is mentioned in [91]:

III.4.1. Representation

The position ξ is recognized as a chromosome, where the decision variables within ξ represent genes, each comprising six alleles. Furthermore, the position of an element (gene) within a chromosome is denoted as a locus. The initial three alleles are specifically designated for linear velocity, while the last three are allocated for angular velocity.

III.4.2. Population initialization

Every particle within the population is required to possess an initial position ξ, established through an array of continuous, uniform random numbers. Each variable is constrained by defined lower and upper bounds. These boundaries are related to the speed of the robot in the considered direction. If we have a predefined speed, we choose the smallest possible values; otherwise, we select the boundaries to be as wide as possible.

III.4.3. Objective function

The objective function defines the desired goal, associating a numerical value with each solution within the search space to quantify its quality and thus determines the efficacy of the method. This assigned result is given as an absolute value, enabling a comprehensive ranking of all solutions within the scene. The objective function plays a crucial role in the development of a metaheuristic, steering the search towards optimal motion within the solution space. This function is expressed as a mathematical equation, with its minimum value indicating the most favorable motion within the search domain. The objective function in our case is the error Function II.2. However, the term "fitness" is denoted by the Function III.17. It is written in the form of a fraction, and each individual has a percentage, which represents the probability of choosing parents for mating, and the Function III.17 was used only in the genetic method.

III.4.4. Selection strategy

At this point in the process, the selection of particles for reproduction is carried out, and various approaches can be employed for this purpose.

Tournament Selection:

Tournament selection involves the random selection of k particles, where k represents the tournament group size. Subsequently, a tournament is conducted among the particles to determine the best one. To choose μ individuals, the tournament process is repeated μ times.

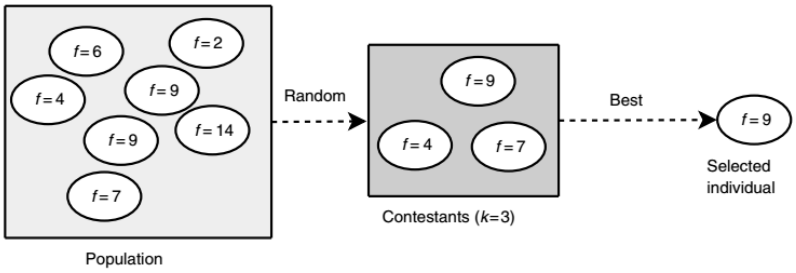

Figure III.3: strategy tournament selection consists of selecting a group of the best particles or individuals randomly from the population, and then the best solution from the selected individuals is chosen.

Roulette wheel selection method:

In the experiments carried out in this thesis, we implemented the genetic algorithm using the roulette wheel selection method. This method assigns a probability value, denoted as $prob_i$, to each particle p_i within the population, and this value is proportional to the particle's fitness as outlined in Equation III.18. It is important to highlight that Ei represents the error of individual p_i, and E_{min} signifies the minimum error of the particles. In the subsequent discussion, the fitness f_i of particle p_i is expressed as:

$$f_i = exp(-8 \times E_i / E_{min}). \quad \text{III.17}$$

Its probability of being chosen is

$$prob_i = f_i / (\sum_{i=1}^{n} f_i). \quad \text{III.18}$$

Then we apply, to each selection probability, the cumulative sum of elements

$$probc_i = cumsum(prob_i). \quad \text{III.19}$$

The selection of μ particles for mating involves a random process, akin to the independent spin of a roulette wheel, influenced by an indiscriminate variable. Individuals demonstrating superior qualities, characterized by minimal error and thus maximum fitness, are more likely to be chosen for the subsequent stage, as determined by Equation III.19.

To simplify the roulette wheel selection process, assume a pie chart where each particle in the population has a section on the pie that corresponds to their level of fitness:

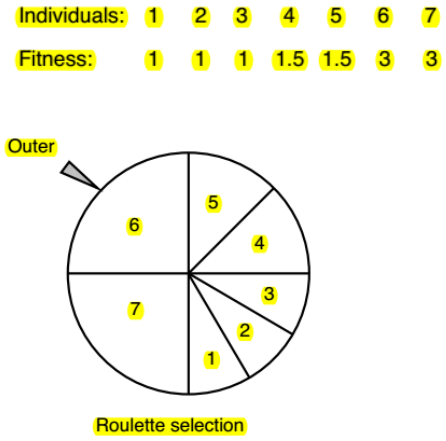

Figure III.4: The strategies of roulette wheel selection include choosing a single individual for each spin.

An outer roulette wheel surrounds the pie. The process of selecting μ particles involves μ independent spins of the roulette wheel, with each spin targeting a single particle. Superior individuals are allocated more space, increasing their likelihood of being chosen.

In our experiment, we used the *cumsum* given in Equation III.19 to compute the cumulative sum along the first non-singleton dimension of $prob_i$, which performs the same function as the pie. The cumulative sum of the probability values given in Equation III.18 is calculated to select the particles. The goal is to find the first index i where μ is less than or equal to an element of $probc_i$. Since μ is a random number between 0 and 1, it will be compared against the values in $probc_i$.

For example, let's assume $\mu = 0.25$ and $probc = [0.1, 0.3, 0.6, 1.0]$. The condition $\mu \leq probc$ gives: [false, true, true, true]. The first true occurs at index 2, so $i = 2$.

This process effectively randomly selects an index based on the probability distribution defined in *probc*. The elements of *probc* act as weights (probabilities summing to 1).

This is commonly used in roulette wheel selection for probabilistic sampling in genetic algorithms.

However, in roulette wheel selection, outstanding particles can introduce a bias in the initial stages of the search, potentially leading to premature convergence and a reduction in divergence because the error E_i will be close to the E_{min}.

III.4.5. Reproduction strategy

The process of reproduction consists of two stages required to generate a new individual.

• *Mutation:* every particle undergoes this process independently. A random selection of thirty percent of the population is chosen to experiment with the mutation. The likelihood that a change in particle genes will be chosen for mutation is known as the mutation rate (p_m = 0.1). The genes in the chromosome in this work represent the elements composed of the motion vector ξ.

The mutation formula is:

$$\xi' = \xi + M. \quad \text{III.20}$$

In addition, M is a mutation random vector calculated as follows:

$$M = \delta i \times randn(size\_\xi) \quad \text{III.21}$$

Where the equation δ_i is:

$$\delta_i = p_m \times (\xi_i^U - \xi_i^L) \quad \text{III.22}$$

And ξ_i^U(respectively, ξ_i^L) represents the upper bound (respectively, lower bound) for ξi.

• *Crossover:* either the parents selected through the roulette wheel selection method or the tournament selection strategy will undergo recombination, specifically the crossover process. The objective of recombination is to generate offspring that inherit genetic material from both parents. One of the commonly employed crossover techniques is the intermediate crossover [92], which attempts to average the positions corresponding to the two parents. Through the equations of crossover, two individuals, O_1 and O_2, are produced using a weighted average:

$$\begin{cases} O_1^i = \alpha \xi_1^i + (1-\alpha)\xi_2^i \\ O_2^i = \alpha \xi_2^i + (1-\alpha)\xi_1^i \end{cases} \quad \text{III.23}$$

The additional crossover factor, denoted as α, signifies the proportion to which parents are chosen as random arrays from the continuous uniform distribution.

Subsequently, the combination of the old and new individuals generated after mutation and crossover will be merged and will give rise to the future population for the next stage.

III.4.6. Replacement strategy

The future population is determined through a competition between the newly generated offspring and the old particles, including the parents. This is achieved by forming a combined population that includes both the previous elements and the offspring produced through crossover and mutation strategies. Then, all the particles are sorted based on the errors resulting from equation II.2 for each individual, and particles with the minimum error value, based on the required number of individuals, are selected. The process concludes with an update to the minimum error ever recorded and the best individual corresponding to it.

III.4.7. Stopping criteria

Numerous stopping criteria are applied during the execution of the metaheuristic code. We employed two stopping procedures, as outlined below:

• Static procedure: The end of execution is based on a maximum iteration value that is predetermined and specified for each *DSR* level of image resolution.

• Diversity procedure: The code execution stops when the error of the best individual remains stagnant after a predefined number of iterations, rendering further algorithm execution with more iterations futile.

III.4.8. Overall algorithm

We present the key steps that GA takes to determine the optimal motion in the following:

1. Initialization

a. Set the number of particles as N.

b. Set the number of GA iterations as M.

c. Set the variables bound

d. Set crossover and mutation percentage

e. For i=1,…,N, set ξ_0^i*=rand(1).*

f. Set initial parameters camera intrinsic

2. Main Loop

for j=1:M iterations.

-Select parents using the roulette wheel selection

for i=1:ns particles selected (also called Parents)

-Update the particles via Crossover and Mutation

end

-Evaluate $f(P_j^i)$ *and update* P^i *and get* ξ of *Pbest*

end

Return ξ

The Genetic Algorithm (GA) is a metaheuristic method used to solve various optimization problems. This method is widely used in the artificial intelligence field, particularly for visual odometry. It is based on how genes behave and how a population reproduces to find an optimal solution by the end of the algorithm's execution. The chromosome of a particle in the population represents its motion, ξ, with six genes or six degrees of freedom. The motion of these particles occurs within a limited search space.

Figure III.5 presents a flowchart describing the execution process of the genetic algorithm. The initial parameters include the camera intrinsic, the focal lengths in both the x and y directions (f_x, f_y), and the pixel coordinates of the principal point (c_x, c_y), which are defined as the input to the algorithm.

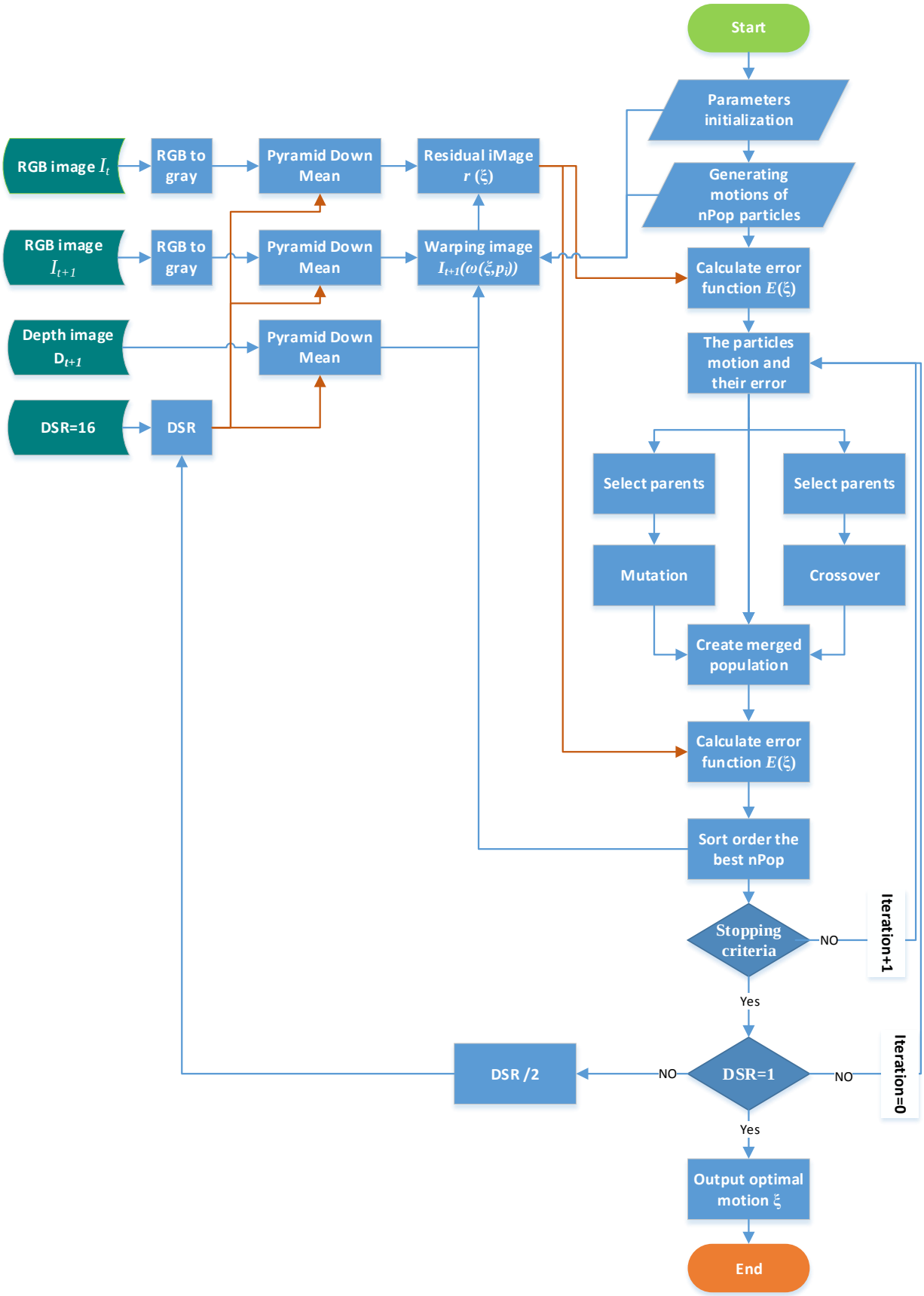

Figure III.5: Flowchart of the motion estimation using the genetic algorithm

Initially, the algorithm assigns a random initial motion ξ to each item in the particle group. The initial parameters, along with the motion of each particle and the RGB-D image I_{t+1}, are used to calculate the warping image using Equation II.5. This subsequently deduces the composition of the residual image $r_i(\xi)$ by Function II.2. The residual image of each particle is then used to calculate the error $E(\xi)$ associated with each particle's motion. This step is crucial for evaluating the motions of the particles and ranking them in order of increasing error $E(\xi)$.

A predetermined number of particles are then chosen to create new particles with different motions using crossover and mutation. The new and old particles will form a merged population. After that, the best of these new particles replaces the worst older ones in the previous population, which have a greater error $E(\xi)$. Thus, one iteration using the same *DSR* value is completed. The algorithm then selects a new set of particles for the subsequent iteration, repeating the main loop. The process continues until the stopping criteria are reached.

Afterward, the original image is down sampled to create images at higher resolutions by down sampling the two RGB-D images' resolution by dividing the previous *DSR* by 2. The GA then uses the population resulting from the previous down sampled resolution (*DSR*) and repeats the main loop. The process continues until the stopping criteria are reached and $DSR = 1$, where the image used is the original with a resolution of 640×480 pixels. At this point, the motion ξ of the best individual is calculated. The algorithm's performance will be evaluated through various experiments.

III.5. Geometric particle swarm optimization for visual ego-motion Estimation

Visual ego-motion estimation, also known as visual odometry, involves the continuous determination of *3D* camera movement using sequences of *2D* images taken by a camera. This process is crucial in numerous computer vision and robotics applications, including visual simultaneous localization and mapping (SLAM) and augmented reality.

This estimation method is derived from reworking the traditional vector space geometric particle swarm optimization (PSO) algorithm to account for the geometry of the special Euclidean group *SE(3)*. *SE(3)* is a Lie group that characterizes the space of *3D* camera poses, as outlined in [53].

In this section, we introduce the geometric PSO metaheuristic method, designed with consideration for the geometry of *SE(3)*. First, we provide a concise overview of traditional PSO within a vector space. For a more detailed examination of PSO refer to the reference [93].

III.5.1. PSO on a vector space

The core concept of Particle Swarm Optimization (PSO) is to leverage the interactions and information-sharing among particles to efficiently identify the global optimum. Consider a collection of particles, denoted as $\xi \triangleq \{\xi_1, \cdots, \xi_M\}$ scattered randomly across the search space. Each particle $\xi_i \in \mathbb{R}^6$ navigates this space to locate the global optimum, guided by two relative position vectors: $p^{gb} - \xi_i$ and $p^{ib}{}_i - \xi_i$, where p^{gb} represents the globally best particle, while $p^{ib}{}_i$ is the best position recorded by each individual i.

The velocity $v_i \in \mathbb{R}^6$, which indicates where to go, of each individual is calculated by summing three vectors as

$$v^{t+1}{}_i = w\, v^t{}_i + c_1 r_1 \,(p^{ib}{}_i - \xi^t{}_i) + c_2 r_2 \,\, (p^{gb} - \xi^t{}_i) \qquad \text{III.24}$$

Where w represents inertia, c_1 and c_2 are the weighting coefficients for the two relative position vectors. r_1 and $r_2 \in \mathbb{R}^6$ are random vectors drawn from a uniform distribution ranging from 0 and 1, introducing stochasticity into the optimization process. The value of the constants used to execute the algorithms for the equation III.24 are mentioned in Annex B. Subsequently, each particle $\xi^{t+1}{}_i$ updates its position using $v^{t+1}{}_i$ as follows:

$$\xi^{t+1}{}_i = \xi^t{}_i + v^{t+1}{}_i \qquad \text{III.25}$$

Then, p^{gb} and $p^{ib}{}_i$ are updated as

$$p^{ib}{}_i = \xi^{t+1}{}_i \quad \text{if} \quad E(\xi^{t+1}{}_i) < E(p^{ib}{}_i) \qquad \text{III.26}$$

$$p^{gb} = \xi^{t+1}{}_i \quad \text{if} \quad E(\xi^{t+1}{}_i) < E(p^{gb}) \qquad \text{III.27}$$

Where E is the error Function II.2, i.e., an objective function that aims to minimize.

By repeating Equations III.24, III.25, III.26, and III.27 several times, particles can explore the solution space efficiently, and convergence can be guaranteed.

The mathematical operations of addition and subtraction contained in Equations III.24 and III.25 cannot be performed directly in the traditional formula but must formulate the geometric PSO algorithm on the special Euclidean group $SE(3)$ and perform these operations using a general Riemannian manifold.

III.5.2. The special Euclidean group *SE(3)*

A camera's position can be described using a rigid body transformation matrix structured as $\begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}$ in homogeneous coordinates, where R denotes a 3×3 rotation matrix in $\mathbb{R}^{3\times3}$, and t is a 3×1 vector in $\mathbb{R}^3$ space. These rigid body transformation matrices, along with the rotation matrices, correspond to matrix Lie groups: specifically, the special Euclidean group *SE(3)* and the special orthogonal group *SO(3)*. Formally, *SO(3)* and *SE(3)* are defined as follows:

$$SO(3)=\{ R \in \mathbb{R}^{3\times3} \mid R^TR=RR^T=I,\ det(R)=+1\} \qquad \text{III.28}$$

And $$SE(3)= \{ g = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times4} \} \qquad \text{III.29}$$

Where $R \in SO(3)$ and $t \in \mathbb{R}^3$.

A Lie group is a differentiable manifold that possesses a group structure with smooth product and inverse operations. The Lie algebra associated with a Lie group is defined as the tangent vector space at the identity of the Lie group. The exponential map is a fundamental tool that relates Lie algebras to Lie groups. It allows you to "exponentiate" elements of the Lie algebra to obtain elements of the Lie group, and vice versa. This map provides a bridge between the abstract group structure of the Lie group and the linear structure of the Lie algebra. The Lie algebras associated with the Lie group *SE(3)* and *SO(3)* are denoted by *se(3)* and *so(3)*, respectively. A Lie group and its Lie algebra can be related via the exponential map, i.e., $exp: so(3) \rightarrow SO(3)$ and $exp: se(3) \rightarrow SE(3)$, as presented in Figure III.6. The logarithmic (*log)* map is defined as the inverse of the exponential (*exp)* map.

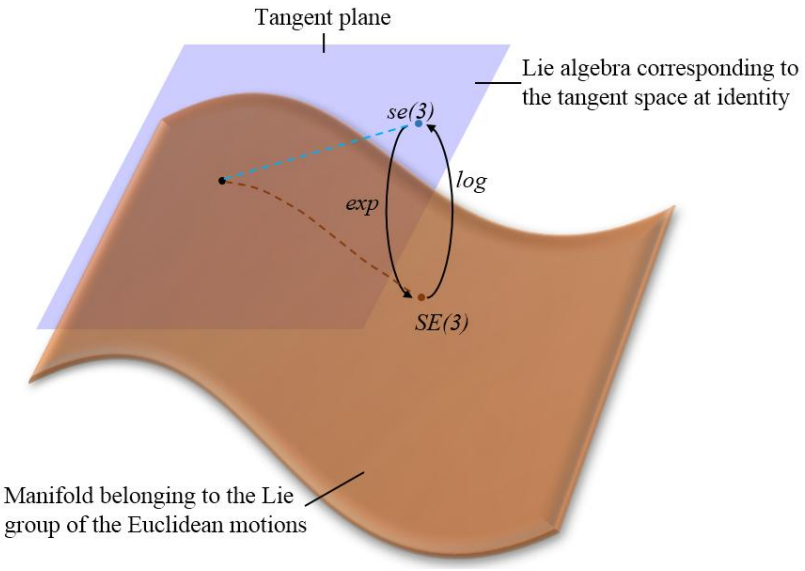

Figure III.6: *SE* and the corresponding Lie algebra as tangent space at the identity

For matrix Lie groups, the matrix exponential and *log* give the exponential and *log* maps. *so(3)* is a set of $\mathbb{R}^{3\times3}$ skew symmetric matrices of the form :

$$w=\begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix} \qquad \text{III.30}$$

And *se(3)* is given by $\begin{bmatrix} w & v \\ 0 & 0 \end{bmatrix}$, with $w \in so(3)$ and $v \in \mathbb{R}^3$.

In this part, we represent the camera pose as an *SE(3)* group, instead of employing the vector parameterization of rotation matrices such as Euler angles [94]. Using the *SE(3)* representation allows us to avoid the singularity issues associated with any $\mathbb{R}^3$ vector parameterization of rotation matrices. However, since *SE(3)* constitutes a curved space rather than a flat vector space, it is necessary to reformulate particle swarm optimization (PSO) to account for the geometric properties of *SE(3)* for improved optimization performance.

III.5.3. PSO on *SE(3)*

To formulate a geometric PSO algorithm on *SE(3)*, we first consider a general Riemannian manifold case. What we have to consider when formulating PSO on a Riemannian manifold is that the difference between particles' motion should be calculated as the minimal geodesic distance on a manifold. Since the Riemannian logarithmic and exponential maps are derived from the minimal geodesics on the Riemannian manifold, we can represent the difference between the elements on the Riemannian manifold as the one on its tangent vector space obtained via the Riemannian *log* map. In this manner, a particle can be considered to be a point ξ_i on a manifold, and its velocity v_i can be defined as the one on the tangent vector space of ξ_i.

Points on a manifold are projected onto the tangent vector space via the Riemannian *log* map at ξ_i^{old}, then the velocity calculation is performed on the tangent vector space according to the original PSO in a vector space. A new particle position ξ_i^{new} is obtained by the Riemannian exponential map at ξ_i^{old} of the resulting velocity vector in the tangent vector space.

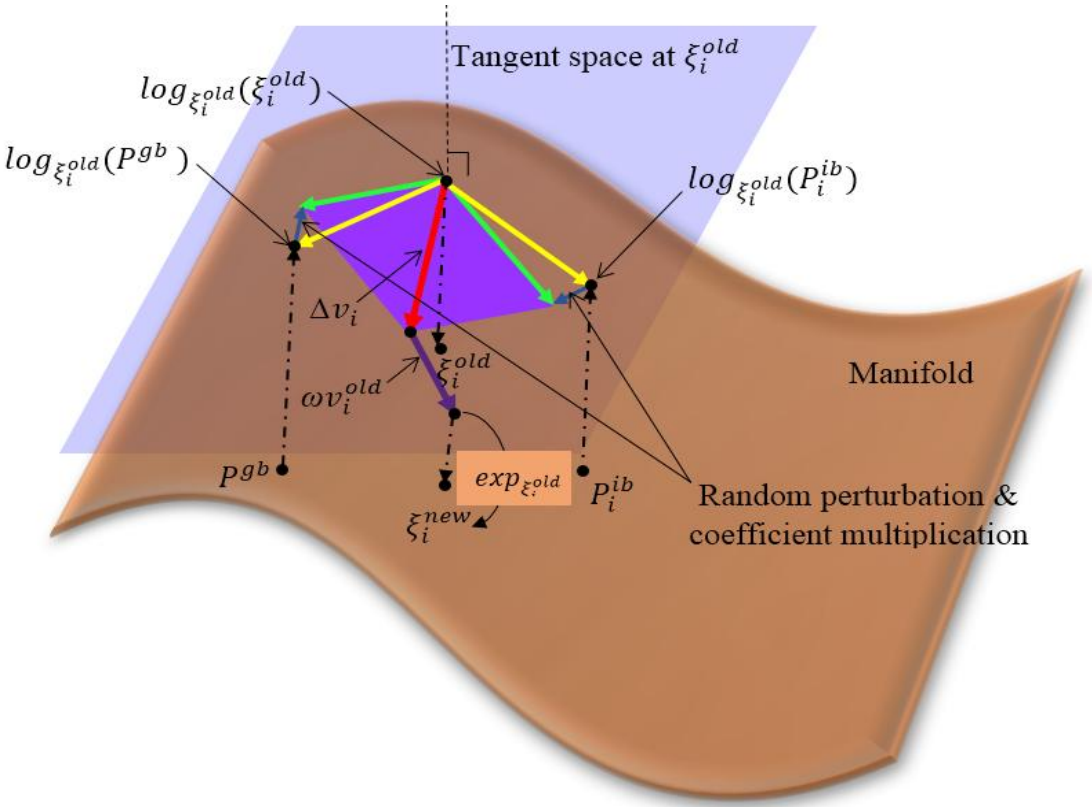

Figure III.7: Graphical representation of geometric PSO on a general Riemannian manifold.

The difference between ξ_i and the individual best $P^{ib}{}_i$ can be identified as a vector on the tangent space at ξ_i obtained via log_{ξ_i}, the Riemannian *log* map at ξ_i, and can be represented as $log_{\xi_i}(P_i^{ib})$. The difference between ξ_i and the global best P^{gb} also can be represented as $log_{\xi_i}(P^{gb})$. Then the velocity v_i is obtained by $log_{\xi_i}(P_i^{ib})$ and $log_{\xi_i}(P^{gb})$ similarly to Equation III.31 and the particle update with v_i is realized via exp_{ξ_i}, the Riemannian exponential map at ξ_i, as $exp_{\xi_i}(v_i)$. Figure III.7 depicts this geometric PSO procedure on a general Riemannian manifold.

The procedure of PSO on manifolds has some similarity to the nonlinear mean shift on manifolds, as described in [95], since the required operations are done on the tangent vector space of a manifold. However, it is not straightforward to directly apply this geometric PSO on a general Riemannian manifold to *SE*(*3*). The first requirement of geometric PSO is to obtain the Riemannian exponential and *log* maps on a specific manifold. Since the minimal geodesics on *SE(3)* is given by the union of the respective geodesics on *SO(3)* and $\mathbb{R}^3$ [96], it is hard to obtain a single expression of the Riemannian exponential and *log* maps for *SE(3)*. Fortunately, the Riemannian exponential and *log* maps for *SO (3)* are simply given by the left and right translations

of *exp* and *log*, which are the matrix exponential and logarithmic. Thus, we can perform geometric PSO on *SE(3)* appropriately by splitting $g_i \in SE(3)$ into $R_i \in SO(3)$ and $t_i \in \mathbb{R}^3$.

The calculations of particle velocity $v^R{}_i \in so(3)$ and particle update for R_i are given by:

$$\begin{cases} v^R{}_i\,(t+1) = w\, v^R{}_i\,(t) + c_1 r^R{}_1 \log\left(R^T{}_i\, R^{ib}{}_i\right) + c_2 r^R{}_2 \log\left(R^T{}_i\, R^{gb}\right)) & \text{III.31} \\ R^{t+1}{}_i = R_i \ \exp(v^R{}_i) & \text{III.32} \end{cases}$$

where $R^{ib}{}_i$ and R^{gb} are the rotation parts of the individual best particles and global best particles, respectively. Note that the differences are calculated by *log* after multiplying $R_i{}^T$ to $R^{ib}{}_i$ and R^{gb}. Then the exponential of the resulting vector on the tangent vector space is multiplied by R_i. This is to apply the exact Riemannian exponential and logarithmic maps of *SO(3)*. In Equation III.31, $r^R{}_1$ and $r^R{}_2$ represent $\mathbb{R}^3$ uniform random vectors, with *SO(3)* elements represented in $\mathbb{R}^3$ column vectors with respect to basis elements of *SO(3)* using the function *log*. The velocity calculation and particle update equations for t_i can be represented by the ordinary PSO algorithm as follows:

$$\begin{cases} v^t{}_i\,(t+1) = w\, v^t{}_i\,(t) + c_1 r^t{}_1\,(t^{ib}{}_i - t_i) + c_2 r^t{}_2\,(t^{gb} - t_i)) & \text{III.32} \\ t^{t+1}{}_i = t^t{}_i + v^t{}_i\,(t+1) & \text{III.34} \end{cases}$$

Where $t^{ib}{}_i$ and t^{gb} are the translation parts of the individual and global best particles, respectively. $r_1{}^t$ and $r_2{}^t$ represent uniform random vectors on $\mathbb{R}^3$. After particle update, the fitness function evaluation is performed at the new position determined by merging the newly updated R_i and t_i into g_i. Where body transformation g is:

$$g = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \qquad \text{III.35}$$

We can calculate ξ using:

$$\xi = \log g \qquad \text{III.36}$$

And we conclude the error Function II.2.

We present the key steps that PSO takes to determine the optimal motion in the following Overall algorithm:

1. Initialization

a. Set the number of particles as N and k = 0.

b. Set the number of PSO iterations as M.

c. For i = 1,…,N, set $\xi^{0}{}_{i} = rand(1)$ *, and* $v^{0}{}_{i} = 0$.

d. Set c_1 , c_2, r_1, r_2

2. Main

for j=1:M iterations.

k = 0

for i=1:N

- Update the particles via Eqs. III.24 and III.25

- Calculate $E(\xi^{t+1}{}_{i})$ *and update* P^{gb} *and* $P^{ib}{}_{i}$.

k = k + 1.

end

end

Return P^{gb}

The particle swarm optimization (PSO) method is a metaheuristic optimization technique that uses particles to search for an optimal solution within a defined 3D search space. These particles are attracted to their own or personal best solution (*Pbest*) and to the best global solution (*Gbest*) of all particles by the Function III.24.

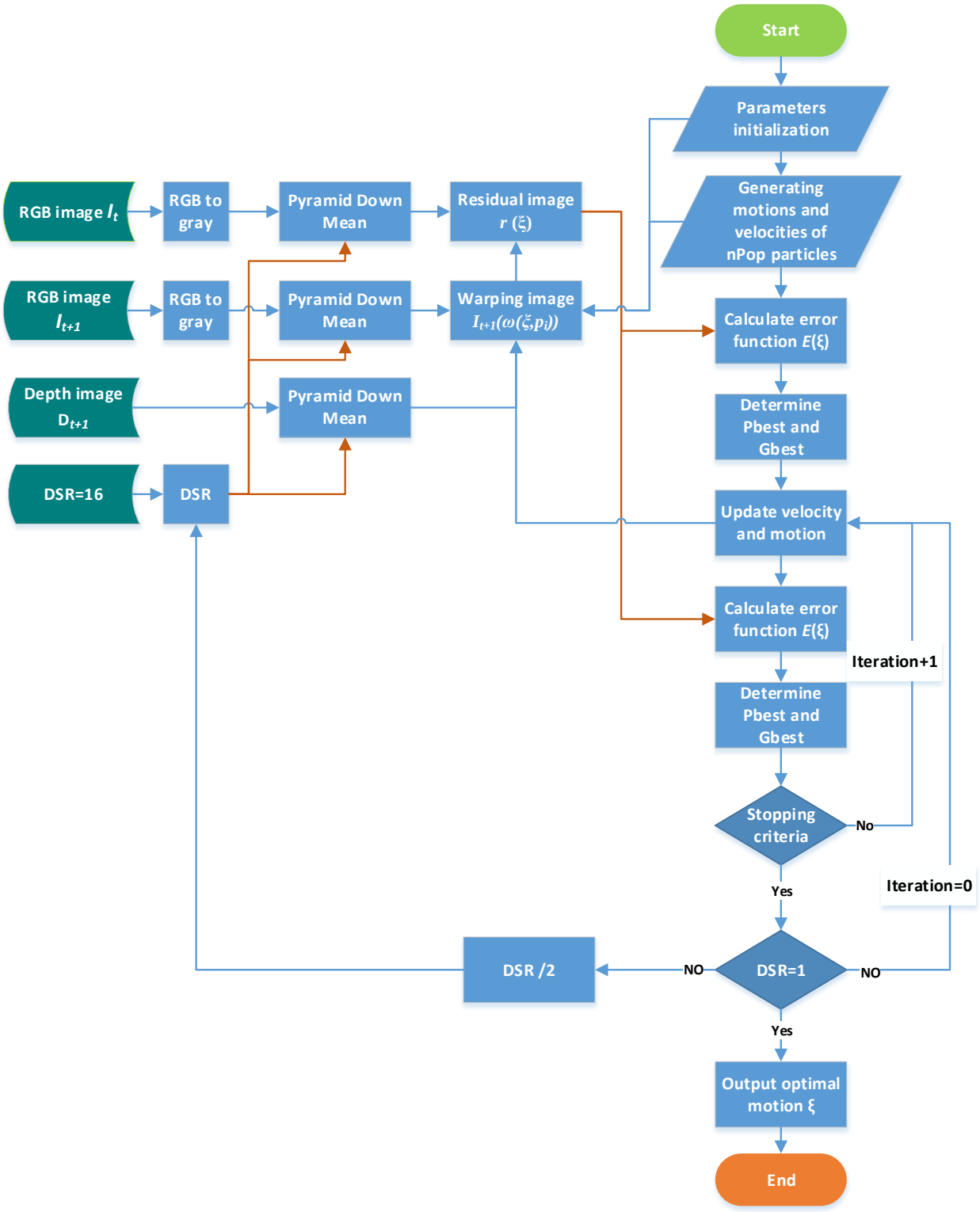

Figure III.8: Flowchart of the motion estimation using the PSO method

Initially, we need to define the motion and initial velocity of the particles, as indicated at the beginning of the PSO algorithm flowchart in Figure III.8. Additionally, the initial parameters, including the focal length in both the x and y directions (f_x, f_y) and the pixel coordinates of the

principal point (c_x, c_y) are defined as the algorithm's input. These parameters, along with the motion of each particle and the RGB-D image I_{t+1}, are used to calculate the warping image using Equation II.5. Subsequently, the residual image $r_i(\xi)$ is deduced by Function II.2.

The residual image of each particle is used to calculate the error $E(\xi)$ linked to each motion. This process is crucial for evaluating the motions of the particles and ranking them from least to greatest error. In the subsequent process of the PSO flowchart, the algorithm determines the best motion achieved by each particle (*Pbest* or the personal best motion) and the best motion achieved by all particles (*Gbest* or the global best motion) during the execution of the algorithm, both with the same *DSR* value.

These two motion values (*Pbest* and *Gbest*) are used in the next step to update the motions and velocities of the particles using Equations III.24 and III.25. This last process, along with calculating the errors of the particles' motion using Equation II.2 and the process of determining *Pbest* and *Gbest* is repeated over several iterations, as the main loop of the algorithm, until the stopping criteria are met.

Afterward, the *DSR* value is divided by 2, the iteration value is reset, and the main loop is repeated until the *DSR* equals 1, corresponding to the use of the original image at full resolution. An optimal solution is produced at the end of the execution of the PSO algorithm.

III.5.4. PSO coefficients used for convergence

The particle swarm optimization is an algorithm for finding optimal regions of complex search spaces through the interaction of individuals in a population of particles. The present section analyzes a particle's trajectory as it moves. These analyses lead to a generalized model of the algorithm, containing a set of coefficients to control the system's convergence tendencies.

Clerc and Kennedy define in [97] the construction coefficients by

$$\chi = \frac{2k}{|2-\varphi-\sqrt{\varphi(\varphi-4)}|} \qquad \text{III.37}$$

Where $\varphi = \varphi_1+\varphi_2 \geq 4$

After conducting several experiments and analyzing the results, they arrived at the following conclusions:

$$\begin{cases} k = 1 \\ \varphi 1 = 2.05 \\ \varphi 2 = 2.05 \end{cases} \quad \text{III.38}$$

Therefore, the values of the PSO coefficients for achieving the best convergence are as follows

$$\begin{cases} w = \chi \\ c1 = \chi\varphi 1 \\ c2 = \chi\varphi 2 \end{cases} \quad \text{III.39}$$

This part explores how the particle swarm algorithm works from the inside, i.e., from the individual particle's point of view. The coefficients can be chosen to guarantee convergence and encourage the particles to explore the entire search space. We remind that the real strength of the particle swarm derives from the interactions among particles as they search the space collaboratively. Effectively, the particles keep moving as they find better and better points in the search space with each iteration. As a particle swarm population searches over time, individuals are drawn toward one another's successes, with the usual result being the clustering of individuals in optimal regions of the space, thus achieving the optimal solution.

III.6. Firefly algorithm for motion estimation

In mathematical optimization, the firefly algorithm is a metaheuristic method proposed by Xin-She Yang and inspired by the flashing behavior of fireflies as described in [98], [99] and [91].

III.6.1. Material and Methods

In essence, the firefly algorithm (FA) uses the following idealized rules.

The brightness of a firefly is inversely proportional to the calculated error value using Equation II.2. Brightness that relies on light intensity determines attractiveness. The attractiveness of a firefly is inversely proportional to the distance between that firefly and the best firefly (i.e., the one with the highest brightness). The variation of attractiveness β, with respect to the distance r, is defined by:

$$\beta = \beta_0 e^{-\gamma r^2} \quad \text{III.40}$$

Where β_0 is the base value of the attraction coefficient, and γ is the light absorption coefficient. The motion of a firefly i is attracted to the best firefly j and is determined by:

$$\xi^{t+1}{}_i = \xi^t{}_i + \beta_0 e^{-\gamma r_{ij}^2}(\xi^t{}_j - \xi^t{}_i) + \alpha_t \epsilon_i \quad \text{III.41}$$

With

$$r_{ij} = || \xi^t{}_j - \xi^t{}_i || \quad \text{III.42}$$

The second part of Equation III.41 refers to the attraction between the particles i and the best particle j. The last part refers to the randomness factor, with α_t being the randomization parameter, and ϵ_i being a vector of randomness from uniform distribution. The value of the constants used to execute the algorithms for the equation III.41 are mentioned in Annex B.

III.6.2. The proposed motion estimation algorithm

The algorithm FA uses interactions between particles sharing information among themselves to achieve the optimal solution. Let it be a population as $\xi \triangleq \{\xi_1, \cdots, \xi_M\}$ randomly chosen in 3D space. Each particle $\xi_i \in \mathbb{R}^6$ moves in 3D space to achieve an optimal solution using the equation of this algorithm.

Each particle of the population calculates its velocity $v^{t+1}{}_i \in \mathbb{R}^6$, by the following equation:

$$v^{t+1}{}_i = \beta (\xi^t{}_j - \xi^t{}_i) + \alpha_t \epsilon_i \quad \text{III.43}$$

With

$$\beta = \beta_0 e^{-\gamma r_{ij}^2} \quad \text{III.44}$$

Then, this velocity will be used to update the motion ξ_i as

$$\xi^{t+1}{}_i = \xi^t{}_i + v^{t+1}{}_i \quad \text{III.45}$$

Finally, $\xi^t{}_j$ is newly updated as:

$$\xi^t{}_j \leftarrow \xi^t{}_i \quad if \quad f(\xi^t{}_i) > f(\xi^t{}_j) \quad \text{III.46}$$

Where f is an objective function. With:

$$f(p_i) = min\,(E(\xi)) \quad \text{III.47}$$

By repeating Equations III.43, III.45, and III.46 the optimal solution can be achieved.

III.6.3. Firefly algorithm on *SE*(3)

The coordinates of movement ξ defined by six degrees of freedom do not allow us to perform addition and subtraction operations contained in Equations III.43 and III.45 using the traditional way, in this part we will explain the method of calculating the previous equations, following the method of Baik in [53].

First, we note that:

$$P_{t+1} = g \times P_t \quad \text{III.48}$$

And

$$P_{t+1} = R \times P_t + T \quad \text{III.49}$$

The rigid body transformation matrices are formulated with matrix Lie groups in III.35.

The difference between two motions, as required in Equation III.41, was made using the Riemannian manifold (exponential and logarithmic maps in Riemannian manifolds) as described in [95].

First, we formulate a geometric attractiveness β on *SE(3)*, we suggest the difference:

$$d = \xi^t{}_j - \xi^t{}_i \quad \text{III.50}$$

The *log* map is used to compute the difference as follows:

$$h = log\ R_i^T R_j \quad \text{III.51}$$

We find:

$$R_d = exp(h) \quad \text{III.52}$$

And

$$t_d = t_j - t_i \quad \text{III.53}$$

We compute the distance by:

$$r_{ij} = \| d \|, \quad \text{III.54}$$

The attractiveness is:

$$\beta = \beta_0 e^{-\gamma r i j^2} \quad \text{III.55}$$

The main Equation III.41 of FA can be decomposed into III.43 and III.45. The difference in Equation III.43 with mentioned in III.50 cannot be done element by element, but through splitting $\xi_i \in SE(3)$ into $R_i \in$ SO(3) (special orthogonal group) and $t_i \in \mathbb{R}^3$.

The rotation part of velocity ($V_R^i \in so(3)$) and motion (R^i) are given by:

$$\begin{cases} V_i^R = \beta\,(\log R_i^T R_j) + \alpha_t \epsilon_i^R & \text{III.56} \\ R_i = R_i \cdot \exp(V_i^R) & \text{III.57} \end{cases}$$

Where ϵ_i^R represents $\mathbb{R}^3$ uniform random vectors, R_i and R_j are the rotation parts of the current and the brighter particle respectively, and:

$$R_i^T R = R R_i^T \quad \text{III.58}$$

Note that the differences are calculated by taking the *log* after multiplying R_i^T with R_j, and we have the condition

$$R^T R = I \quad \text{III.59}$$

Which implies that

$$R^{-1} = R^T \quad \text{III.60}$$

This clearly exists for every R. Since

$$\det R^T = \det R = 1 \quad \text{III.61}$$

R^T is also a rotation matrix.

The translation part of velocity and motion can be updated as:

$$\begin{cases} V_i^t = \beta\,(t_j - t_i) + \alpha_t \epsilon_i^t & \text{III.62} \\ t_i = t_i + V_i^t & \text{III.63} \end{cases}$$

Where t_i and t_j are the translation components of the current particle and the brighter particle respectively. ϵ_i^t represents uniform random vectors on $\mathbb{R}^3$.

After updating R_i and t_i and merging them into body transformation g using Equation III.35, ξ is defined by III.36, and after that, the cost function is calculated with II.2, which corresponds to the new motion of the current particle i.

After that, we calculate the error Function III.28, and then we conclude the fitness function:

$$f(p^i) = \min(E(\xi)) \quad \text{III.64}$$

This function is used to update the motion ξ of the best particle P_j as mentioned in the overall algorithm.

III.6.4. Overall algorithm

In the following, we present the most important steps that the algorithm FA goes through to reach the best solution:

1. Initialization

a. Choose N for the count of the number of particles.

b. Choose M for the number of FA iterations.

c. For i = 1,…,N, set $\xi^0_i = rand(1)$.

d. Set $\beta_0, \gamma, \alpha^t, \epsilon_j$,

2. FA Loop

for j=1:M iterations.

for i=1:N

- Update the particles via Eqs. III.43 *and* III.45

- Evaluate $E(\xi^j_i)$ *and update* ξ^j.

end

end

Return ξ^j

The Firefly Algorithm (FA) is a metaheuristic method used to solve various optimization problems, including visual odometry equations. This method relies on a population-based approach to find the optimal solution. The behavior of firefly motion is influenced by two key factors: attraction to brighter fireflies, which exhibit better motion, and a randomness factor. The movement of these fireflies, or particles, occurs within a limited search space.

Figure III.9 presents a flowchart describing the execution process of the Firefly Algorithm. The initial parameters include the focal lengths in both the *x* and *y* directions (f_x, f_y) and the pixel coordinates of the principal point (c_x, c_y), which are defined as the input to the algorithm. Random initial motions are assigned to the fireflies.

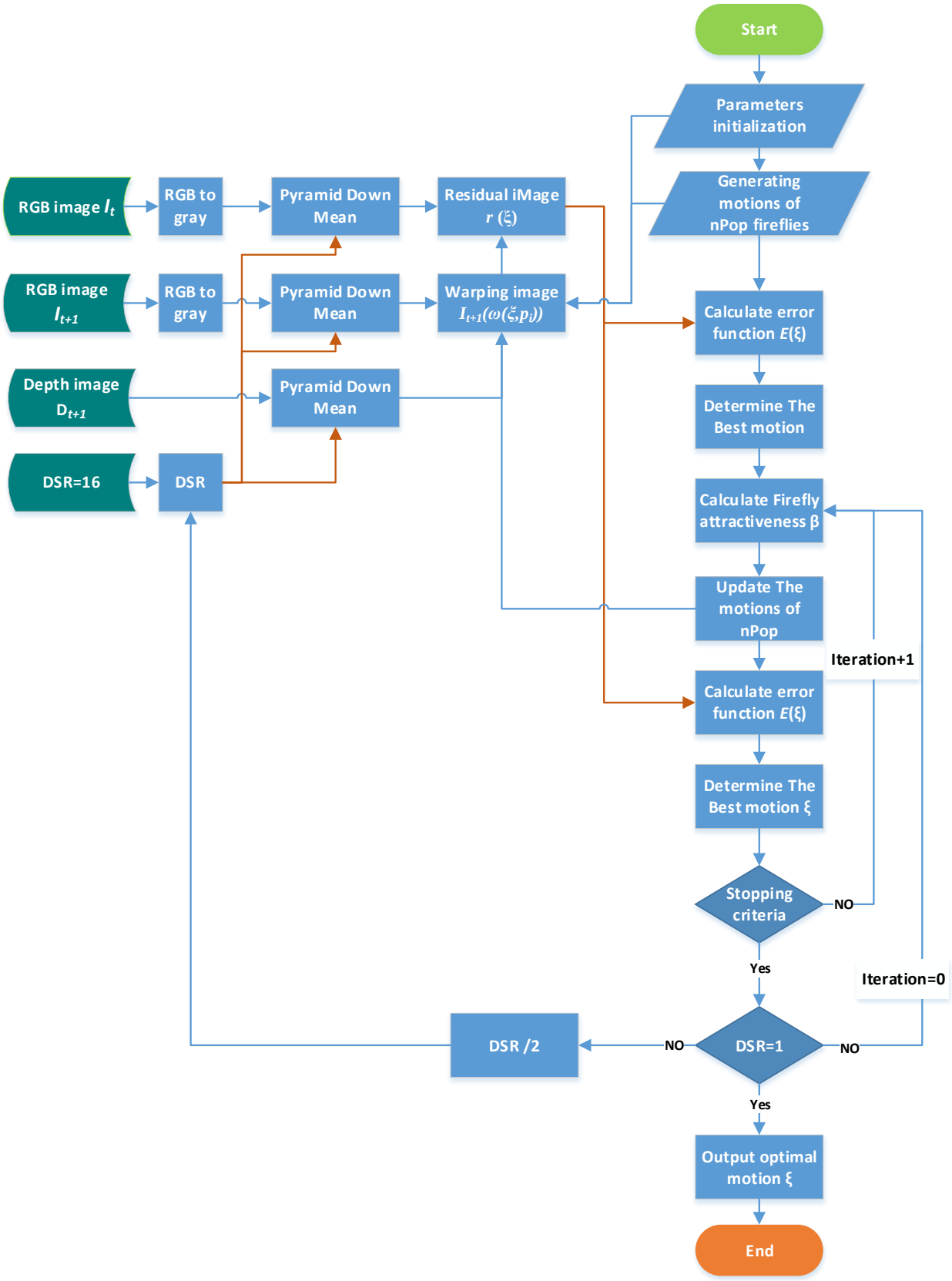

Figure III.9: Flowchart of the motion estimation using the Firefly algorithm

These initial parameters, along with the motion of each firefly and the RGB-D image I_{t+1}, are used to calculate the warping image using Equation II.5. This subsequently deduces the composition of the residual image $r_i(\xi)$ by Function II.2. The residual image of each particle is then used to calculate the error $E(\xi)$ associated with each firefly's motion. This step is crucial for evaluating the motions of the fireflies and ranking them in order of increasing error $E(\xi)$. The subsequent process in the FA flowchart identifies the best motion achieved by all the fireflies during the algorithm's execution, using the same *DSR* value.

The main loop of FA begins by using the previous best motion to calculate attractiveness β. The next step is to update the motions of the *nPop* fireflies using Equation III.41, where these fireflies are attracted to the brighter firefly *j*. This process helps calculate the error E(ξ) corresponding to each firefly's motion using Function II.2. The best motion ξ for the current iteration is then determined.

This main loop repeats over several iterations until the stopping criteria are met. Subsequently, the *DSR* value is halved, the iteration value is reset, and the main loop is repeated until the *DSR* equals 1, which corresponds to using the original image at full resolution. An optimal solution is produced at the end of the Firefly Algorithm's execution.

Our algorithms are made intelligent to find the optimal motion as fast as possible and to avoid the divergence of our algorithms towards a local minimum, which corresponds to non-optimal and false solutions. In practice, the future motion for robots or cars is often close to or equal to the previous one, and the variation of motion between two consecutive small instants of time is very small or zero. Therefore, when we generate random initial motions for an algorithm, it is necessary to equip a particle with the optimal motion found in the previous transition. Even better, it is better to give the motions of the ten best previous particles as initial motions to the current particles, while the other motions are generated randomly. With this operation, the optimal solution will be ensured.

III.7. Conclusion

In this section, four of the most important optimization methods in computer vision were explained. At the beginning of the section, the energy-based method is presented, which is considered an exact method. After that, three metaheuristic methods were explained, where GA is an evolutionary algorithm while PSO and FA are swarm intelligence methods.

We point out that GA and FA are innovative methods. With these four methods, most branches of optimization techniques have been addressed, as presented in Figure III.1. To show the role and effectiveness of these methods, we need to evaluate them after conducting several experiments, and this is what we will see below.

Chapter IV

Performance evaluation metrics

IV. Performance evaluation metrics

IV.1. Introduction

In this section, we evaluate our methods for motion estimation on a static scene using RGB-D frames that are available in [3]. For this, we use the relative pose error (RPE), root mean square error (RMSE), and 3D trajectory to compare our innovative methods (GA and FA) to particle swarm optimization (PSO) and energy-based method as a classic method, which are mentioned respectively in [53] and [26].

IV.2. Real-time graphical user interfaces

Multicriteria optimization involves choosing the most favorable option from a range of possible alternatives. Key factors in a metaheuristic approach, such as the number of particles, probability values, and other parameters, serve as criteria that need to be adjusted to obtain the best outcomes. This necessity led us to develop a graphical user interface (GUI) in MATLAB, as depicted in Figure IV.1. This interface allows for real-time monitoring and assessment of code execution results, enabling the selection of the most effective criteria values for achieving an optimal solution.

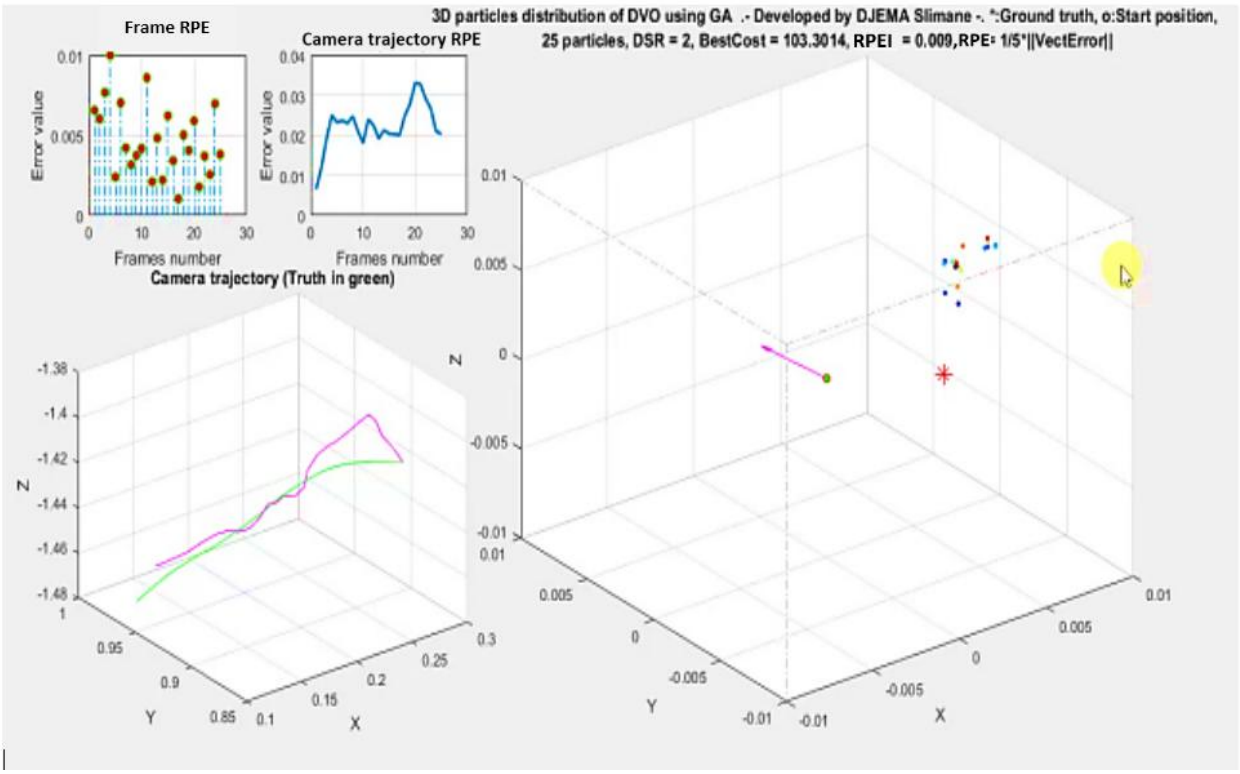

Figure IV.1: The real-time graphical user interfaces view code execution in MATLAB

The graphical user interfaces illustrated in Figure IV.1 consists of four windows. On the right, there is a three-dimensional space depicted as a parallelogram, with its dimensions representing the field of motion of the particles. This space shows how the particles search for the optimal motion value by seeking the lowest error value in Equation II.2. At the center of this finite space, a green disk indicates the three-dimensional position where the previous image was taken. The red star marks the true location where the subsequent image was captured, representing the target area where the colored particles are searching. The color of the particles indicates the error value, with blue signifying the lowest error value or the best cost. This visualization helps us observe particle behavior and assess the proposed method during the execution of the MATLAB code.

The red arrow indicates the direction of accumulated error (RPE) during the image sequence processing, extending as a three-dimensional ray with its length specified. We have displayed the actual value of this error at the top of the *3D* scene as $RPE = 1/5 * ||VectError||$, where $||VectError||$ is the length of the red vector represented in the 3D scene, but its real length is RPE. The best value of relative pose error achieved between the current and previous images is also shown as RPEI. Displaying the value and direction of the RPE as a ray in the same 3D scene with the behavior of particles and RPEI helps to identify whether the behavior of the particles is in favor of reducing the RPEI, the accumulated error (RPE), or both, which helps in choosing the best initial parameters such as the number of particles and others.

Additionally, the number of particles, Down Sampled Resolution (*DSR*), and RPE for the best particle are shown. This image clearly demonstrates the particles' behavior in detecting motion between two consecutive photos. The final RPE between these images for the best particle is represented as impulses in the upper graphical example. On the right, there is a graph showing the camera trajectory RPE, while the bottom graph compares the true and estimated trajectory within the same 3D scene.

IV.3. Relative pose error

Relative pose error (RPE) in the evaluation of visual odometry methods refers to a metric used to quantify the accuracy of estimating the relative pose or motion between consecutive frames in a sequence of images captured by a camera. Visual odometry is a crucial aspect of computer vision and robotics that aims to determine the camera's movement and position by analyzing image sequences.

The relative pose error measures the disparity between the estimated relative pose, typically represented by translation and rotation parameters, and the ground truth or reference pose. It

provides a numerical assessment of how well the visual odometry algorithm performs in terms of capturing the actual movement between frames.

The relative pose error (RPE) between two consecutive images corresponds to the best particle and is represented in the form of impulses, as shown in Figure III.1.

RPE calculates the drift of the trajectory estimated relative to the true trajectory as described in [7], [100], and [101] for the time interval Δ at step i as

$$E_i=(Q_i^{-1}Q_{i+\Delta})^{-1}(P_i^{-1}P_{i+\Delta}). \quad \text{IV.1}$$

Where $P_{i+\Delta}$ is the position of the camera deduced from motion estimation g and the previous position P_i using the following equation:

$$P_{i+\Delta} = g \times P_i \quad \text{IV.2}$$

Thus, the second part of Equation VI.1 is the estimated motion, written in the following form:

$$g = P_i^{-1} P_{i+\Delta} \quad \text{IV.3}$$

Q_i represents the pose of ground truth at time i, and Δ denotes a temporal displacement or the time interval between the two frames. Q_i^{-1} *and* $Q_{i+\Delta}$ denote the 3D positions (translations and rotations with respect to a principal point in 3D space) of the camera at time steps i and i+Δ respectively. As in the previous demonstration, $Q_i^{-1} Q_{i+\Delta}$ represents the difference between the two consecutive poses of ground truth, thus the real motion carried out by the camera.

Equation VI.1 represents the difference between the estimated and the true motion of the camera; this difference is represented in the calculation by the multiplication of motion using the Riemannian manifold (exponential and logarithmic maps in Riemannian manifolds), as described in [95].

Lower RPE values signify better performance, indicating that the visual odometry method provides more accurate relative pose estimates.

IV.4. Root mean square error

Root Mean Square Error (RMSE) is a common metric used in the evaluation of visual odometry methods. In the context of visual odometry, RMSE provides a measure of the average deviation between the estimated trajectory and the ground truth trajectory.

The RMSE is calculated by taking the square root of the mean of the relative pose error. In the case of visual odometry, these values represent the positional differences between the estimated camera trajectory and the true camera trajectory over time.

From a sequence of n images, $m=n-\Delta$ is the individual relative pose errors. We define the root mean square error (RMSE) as

$$RMSE(\boldsymbol{E}_{1:n},\Delta)=\left(\frac{1}{m}\sum_{i=1}^{m}||trans(Ei)||^2\right)^{1/2} \qquad \text{IV.4}$$

Where E_i is the relative pose error and $trans(E_i)$ is the translational component of the RPE. $E_{1:n}$ indicates the relative pose error of n images, in the experiments carried out, we used a frequency of 30 images. Thus, to determine RMSE, first, calculate the accumulated RPE of 30 images, and then compute the absolute value of the translational component (without the rotational component). After that, calculate the mean of the $RMSE_i$ for each 30 consecutive images.

A lower RMSE value indicates better performance, as it signifies a smaller average deviation between the estimated and ground truth trajectories, reflecting greater accuracy in the visual odometry method.

IV.5. Conclusion

In this chapter, three evaluation methods are explained. RMSE is the most well-known method to test and compare the effectiveness of visual odometry methods because each method will have a numerical value, making it easy to use for evaluation and comparison. The equation of this method is based on the relative pose error, which is the distance between the ground truth and the estimated trajectory. This distance is tracked in real time during the execution of our code thanks to a platform that also contains the 3D real and estimated paths. In addition, a 3D distribution of particles helps to properly supervise their behavior during code execution and determine if they are converging towards the true solution, which thereby helps in the development of algorithms.

All of these methods help to evaluate the algorithm's execution during and after the experiment. These evaluation methods will be used in the experiments below.

Chapter V

Experimental setup

V. Experimental setup

V.1. Introduction

Visual odometry is an estimation of camera motion based on vision or captured images. The experiments performed in this chapter used data provided by a website that publishes research and datasets in this field. The following section is dedicated to explaining the composition of the dataset and methods for utilizing it. In addition, several experiments were conducted to better demonstrate the effectiveness of the visual odometry methods mentioned previously. An evaluation will be carried out to compare these methods using different techniques. In subsequent experiments, the focus will be on proving the effectiveness of two of our innovative visual odometry methods, the GA and FA algorithms, and comparing them with two existing techniques, the BE and PSO methods.

V.2. Dataset and camera

In recent decades, extensive research in computer vision has focused on RGB images [102], [103]. Despite their widespread use, RGB images primarily convey only the visual appearance of objects within a scene. This limited scope makes it exceedingly challenging, if not impossible, to address certain issues, such as differentiating between foreground and background when they share similar colors and textures, or recognizing objects. Moreover, the visual data captured by RGB images are not robust against common variations such as changes in lighting, which significantly hinders the effectiveness of RGB-based vision algorithms in practical applications. RGB-D images, which incorporate both visual (RGB) and depth information, have emerged as a powerful data representation. This combination of depth and visual data addresses fundamental problems more effectively. Additionally, integrating RGB and depth information has been shown to significantly enhance the accuracy of high-level tasks such as image and video classification [104], [105].

The essence of an RGB-D image or video lies in the depth image, typically produced by a range sensor. Unlike a 2D intensity image, a range image is more resilient to changes in color, lighting, rotation, and scale [2]. With the introduction of the affordable 3D Microsoft Kinect sensor on November 4, 2010, capturing RGB-D data has become more accessible and cost-effective. Consequently, research into computer vision algorithms utilizing RGB-D data has gained significant traction in recent years. RGB-D images support a broad array of applications, including computer vision, robotics, construction, and medical imaging [106]. As numerous algorithms have been developed to address technological challenges in these fields, a growing number of RGB-D datasets have been established to validate and evaluate these algorithms. The availability of public

RGB-D datasets not only conserve time and resources for researchers but also allows for the fair comparison of various algorithms.

RGB-D data has proven to be an invaluable representation of indoor scenes for addressing key computer vision challenges. This data combines the benefits of color images, which provide appearance information, with depth images, which are resistant to variations in color, lighting, rotation angle, and scale. The introduction of the low-cost Microsoft Kinect sensor, initially intended for gaming [2] but later widely adopted in computer vision, has made it easy to obtain high-quality RGB-D data. Recently, an increasing number of RGB-D image and video datasets have been created for various applications, playing a crucial role in benchmarking state-of-the-art methods. This thesis systematically employs popular RGB-D datasets for motion estimation using different approaches. The primary objective of this section is to thoroughly describe the available RGB-D datasets and the Kinect camera used to capture this data, thereby facilitating the selection of appropriate datasets for evaluating our algorithms.

V.2.1. Kinect camera

Over the past few years, RGB-D data obtained from the consumer-grade Kinect sensor has emerged as a new type of scene representation, demonstrating its potential for addressing complex computer vision challenges. Microsoft launched both the hardware sensor and the accompanying software package in November 2010, and they have achieved significant sales since then [107]. This device, coupled with advanced algorithms, has been utilized in a range of applications, including 3D simultaneous localization and mapping (SLAM) [108], [109], people tracking [110], object recognition [111], visual ego-motion, and human activity analysis [112], among others.

The Kinect sensor includes a near-infrared laser that projects a refracted pattern on the environment, an infrared camera that captures this pattern, and a color camera situated between them. Since the projected pattern is predetermined, the Kinect can collect a large dataset of sequences that include RGB-D data as well as ground truth positions from an external system.

A. Kinect hardware configuration

Generally, the basic version of Microsoft Kinect consists of an RGB camera, an infrared camera, an IR projector, a multi-array microphone, and a motorized tilt mechanism. Figure V.1 illustrates the components of the Kinect and two example images captured by the RGB and depth sensors, respectively.

The technology used for generating the depth map is based on analyzing the speckle patterns of infrared (IR) laser light [113].

Figure V.1: Hardware Kinect sensor, and two captured frames using RGB camera and depth camera.

This illustrates the setup of a Kinect sensor, which includes a depth sensor and an RGB camera. The depth sensor consists of the IR projector and the IR camera. The IR projector emits an IR speckle dot pattern onto the scene, which is then captured by the IR camera. Thus, the Kinect camera functions as a structured light-depth sensor. The geometric relationship between the IR projector and the IR camera is determined through an offline calibration process. The IR projector projects a specific light speckle pattern into the environment, which is not visible to the RGB camera but detectable by the IR camera. It is possible to match the calibrated projector dot patterns with the observed local dot patterns in the image because every local pattern of projected dots is distinct. The dot pattern's relative left-to-right translation can be used to determine a point's depth. The object's distance from the camera-projector plane affects this translation. A process like this is depicted in Figure V.2. Further information on the 3-D imaging method using structured light is available in [113].

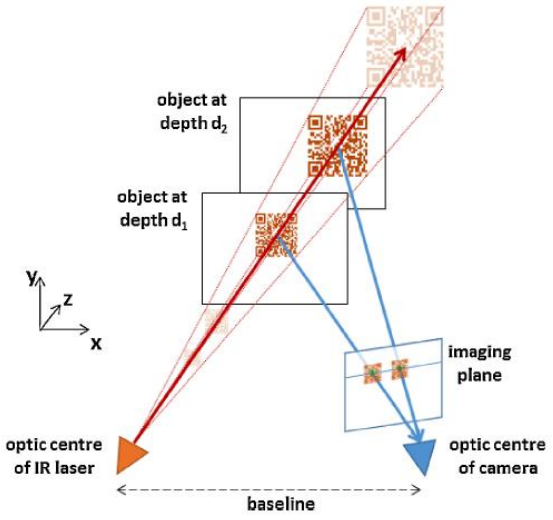

Figure V.2: Measurement of Kinect camera depth

The following explains each Kinect hardware part:

1) RGB Camera: Provides the video's three primary color components: red, green, and blue. This VGA camera can capture images at a resolution of 640×480 pixels with 8 bits per channel. It runs at 30 Hz.

2) 3-D Depth Sensor: It is made up of an infrared camera and an infrared laser projector. A depth map that shows the distance between an object and the camera is produced by the projector and camera working together. The sensor can detect objects at distances ranging from 0.8m to 3.5m and can output video with a resolution of 640×480 pixels of video at a frame rate of 30 frames per second. 43◦ vertically and 57◦ horizontally is the angular field of vision.

3) The Motorized Tilt: This pivot allows you to adjust the sensor. The sensor has a 27◦ tilt range in both the upward and downward directions. The two interconnected and crucial internal systems of Kinect are an accelerometer and a mechanism for tilting the Kinect head up and down. A motor that has gears to move the head up and down is used to tilt the head. The accelerometer is the method by which Kinect ascertains the head's location.

A tool for measuring acceleration is an accelerometer. By detecting the acceleration brought on by gravity, the accelerometer informs the system which way is down. This enables the device to calibrate to a value and position its head precisely so that it may be moved at particular angles.

4) The microphone array has four microphone capsules and processes 16-bit audio at a sampling frequency of 16 kHz for each channel.

B. Intrinsic Camera Calibration of the Kinect

The process of determining internal camera parameters, such as the image center *c*, focal length *f*, and lens distortion parameters, that have an impact on imaging is known as camera calibration. Because of poor lens quality and manufacturing flaws in cameras, accurate camera calibration is necessary. This is crucial for 3D image interpretation, reconstructing world models, and enabling robot interaction with their environments. A flat checkerboard pattern with a defined 3D geometry is used in the most commonly used technique. The user must capture many shots of the checkerboard in various positions, making sure to fill the camera's field of vision as much as possible. By resolving a least squares minimization problem where the input data consist of the 3D positions of the square corners on the checkerboard pattern and the associated 2D image

coordinates, the parameters are calculated. A variety of open-source tools are available for estimating camera parameters, including the C/C++ OpenCV calibration toolbox [114], as detailed in [115], [116], and [117], as well as the MATLAB camera calibration toolbox [118] and [119].

V.2.2. RGB-D image

The dataset utilized in these experiments contains RGB-D frames and ground-truth information, which serve to evaluate visual ego-motion algorithms. This dataset comprises RGB and depth frames captured by a Microsoft Xbox Kinect camera connected to a laptop, as detailed in [7], along with the true trajectory of the camera. The recordings of these images were taken at a rate of 30 Hz with a camera resolution of 640×480.

We used RGB-D datasets available on the website [3] for our experiments. A Kinect camera was used to capture the images in an office environment. An infrared (IR) camera captured the pattern that creates the depth image, as shown in Figure V.3, while a near-infrared laser projects a refraction pattern onto the surroundings as part of the Kinect sensor. The depth of the scene in this region is represented by the pixels' value.

RGB images, as represented in Figure V.4, reconstruct color by additive synthesis from three primary colors: red, green, and blue; abbreviated as RGB, this is the computer color coding system closest to hardware. For each of the primary colors, the value is expressed in an interval ranging from 0 to 255, which represents the maximum value. This applies to each pixel of the RGB image.

Additionally, for simplicity, we only use the grayscale values of the color, i.e., we define

$$I = (I_R + I_G + I_B)/3 \qquad \text{V.1}$$

And, for this, the residual image of the difference between two consecutive images appears in gray (or what we call a black and white image).

Using these two images, we have a dataset of a 3D scene. A large set of data sequences was acquired, containing both RGB-D data from the Kinect and ground truth poses delivered from the motion capture system.

Figure V.3: Depth image from the "*fr2_desk*" sequence.

Each sequence is provided as a single compressed TGZ archive, which consists of the following files and folders:

- " rgb/ ": a folder containing all color images that are stored in PNG format at 640×480 resolution with 3 channels, 8 bits per channel (capable of representing 256 colors in each one).
- " depth/ ": the same for the depth images, which are stored in PNG format at 640×480 resolution with 1 channel, 16 bits per channel (but transfers the data as integers with 11 bits to save space. Kinect quantizes the depth measurements in a range from 1 to 10.000 values), distance in meters scaled by a factor of 5000.
- " rgb.txt " : a text file containing a sequential list of all color image names (format: timestamp filename).
- " depth.txt " : a text file containing a sequential list of all depth image names (format: timestamp filename).
- " imu.txt " : a text file containing the timestamped accelerometer data (format: timestamp a_x a_y a_z).
- " kinect_params.txt " : a text file that includes focal length in both the *x* and *y* directions (f_x, f_y) and the pixel coordinate of the principal point (c_x, c_y), containing the camera parameter data (f_x c_x f_y c_y).
- " groundtruth.txt " : a text file containing the ground truth trajectory, which is stored as a timestamped translation vector and unit quaternion.

Figure V.4: RGB image from the "*fr2_desk*" sequence.

The format of each line of "groundtruth.txt" is '*timestamp* t_x t_y t_z q_x q_y q_z q_w', We explain the significance of each part below:

- timestamp (float) gives the number of seconds since the Unix epoch. Unix time (also known as POSIX time or UNIX Epoch time) is a system for describing a point in time. It is the number of seconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970, excluding leap seconds.
- t_x t_y t_z (3 floats) give the position of the optical center of the color camera relative to the world origin as defined by the motion capture system. The difference between two consecutive positions represents t in Equation II.8.
- q_x q_y q_z q_w (4 floats) give the orientation of the optical center of the color camera in the form of a unit quaternion relative to the world origin as defined by the motion capture system. The difference between two consecutive orientations can be converted into matrix R using the MATLAB function *quat2dcm* as represented in Equation II.8.

V.2.3. Data acquisition

Visual odometry is usually based on RGB-D images provided by a stereo camera. In our case, we do not have this type of camera, so we retrieved these images from the website [3]. This website provides the information needed to perform the experiments, such as the camera parameters used and the actual path traveled by this camera to capture these images.

The RGB-D data consists of two different folders: the first contains the RGB images, and the second contains the depth images. These images are named by Unix time. This data also contains three text documents .txt files: one contains the names of the RGB images, another contains the

names of the depth images, and the third contains the information about the actual journey traveled to capture these images in the form: '*timestamp* t_x t_y t_z q_x q_y q_z q_w', where the first piece of information '*timestamp*' represents Unix time. To link each RGB image to its depth image and the pose where it was captured, we exploit the information they have in common, which is Unix time. If we do not have an image depth *timestamp* equivalent to that of the RGB image, we take the closest depth image time, and apply the same approach with the real pose of the ground truth. Each pixel of the RGB image provided by the website contains three pieces of information representing the three colors. The first step after reading these images is to transform these three pieces of information into one for each pixel, which we call the grayscale image, according to Equation V.1, because the input image must be a single channel one. Then this frame will be subject to down mean sampling one or more times, depending on the *DSR* required as represented in Figure II.5.

The pose delivered with respect to the real trajectory is in the form of a 1×4 quaternion as *'q_x q_y q_z q_w'*, and therefore we are required to transform it into a direction cosine matrix 3×3 using the MATLAB function *quat2dcm* to make the evaluation using Equation IV.I. The direction cosine matrix *R*, which is mentioned in Equation II.8, performs the coordinate transformation of a vector in inertial axes to a vector in body axes.

Thus, we have explained, in this part, how we use the data delivered by the website to conduct our experiments.

Figure V.5: RGB image from the "*fr1/ xyz*" sequence.

V.3. Experimental and discussion

We evaluated our algorithms through various experiments in a static environment using RGB-D images of dimensions 640×480 with a frame rate of 30Hz. These images and their corresponding ground truths are available on the website [3].

The experiments were conducted on a personal computer of the marque Pavilion dv6, with a processor specification of Intel(R) Core(TM) i5 CPU M430 @ 2.27GHz (2.27GHz). The memory (RAM) is 3.80 Go usable, and the operating system is 64-bit. This PC uses Windows 7 Ultimate Edition.

In the first part, we compare our innovative genetic algorithm with PSO and the classic method (Energy-Based (EB)) as mentioned in [120].

In our first experiment, we used 90 consecutive frames of *rgbd_dataset_freiburg1_xyz*. These sequences are intended to facilitate the development of novel algorithms with separate motions along and around the principal axes of the Kinect in a typical office environment (6×6 m^2) as represented in Figure V.5. In the "*xyz*" sequences, the camera was moved approximately along the *X*-, *Y*- and *Z*-axis (left/right, up/down, forward/backward) with little rotational components.

The Function IV.1 gives the distance error between estimated motion and ground truth. Therefore, using this function, we compute the camera trajectory error of different methods through 90 consecutive images, and we represent these results in the same graph in Figure V.6. The evolution of distance error indicates that the accumulated error of 90 frames related to the classic method is the least, but the quasi-stabilization of the error of the PSO method in the last 60 frames allowed it to outperform in RMSE, as we can see in Table V.1, which is considered the most important evaluation criterion in visual odometry.

Although the final accumulation error of GA is greater, the error almost maintained its value between the beginning and the end of the last thirty frames, which helped it improve the value of translational components of RPE and thus outperform the RMSE of the classic method (BE).

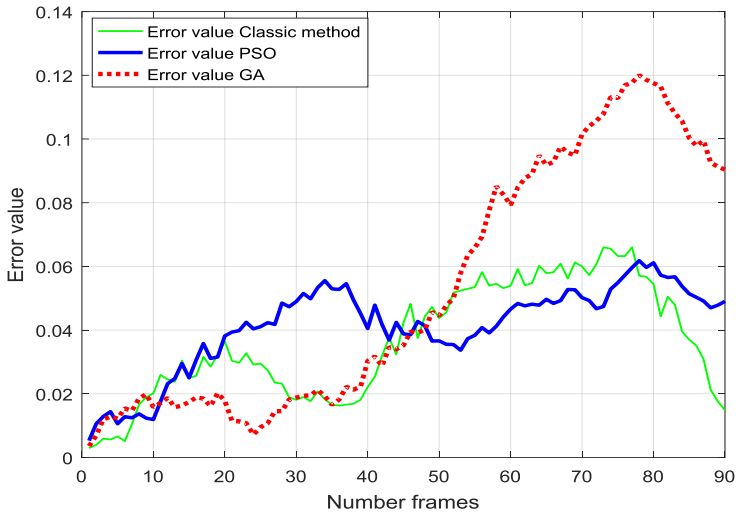

Figure V.6: Camera trajectory error of GA, PSO, and classic method (BE) using a part of the *fr1_xyz* dataset.

The representation in the *3D* scene of the truth and estimated camera trajectory clearly shows the effectiveness of the motion estimation methods. Figures V.7, V.8, and V.9 show the camera trajectory using the *fr1_xyz* dataset of ground truth and different motion estimation methods: based-energy and metaheuristic methods.

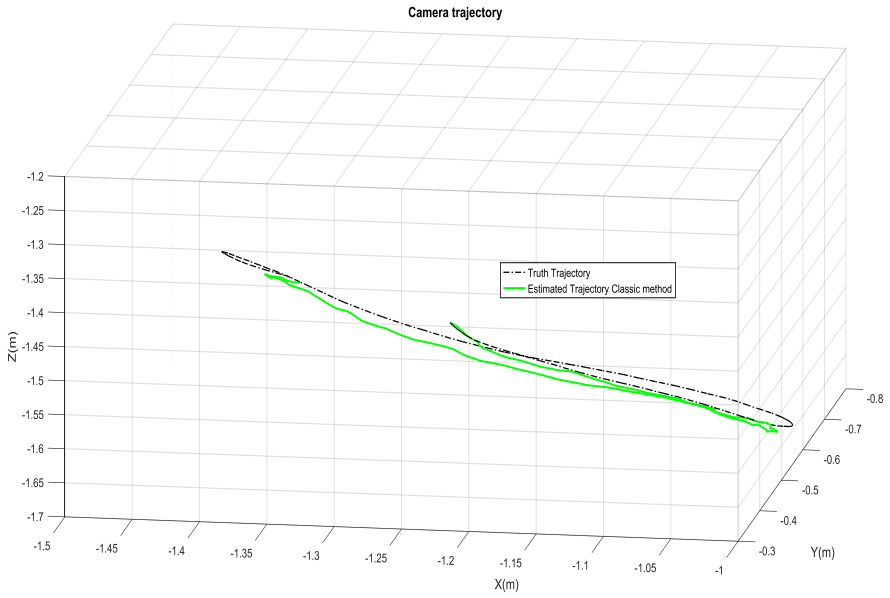

Figure V.7: trajectory of the true camera and the energy-based method using a part of the *fr1_xyz* dataset.

Through ninety RGB-D frames and following a back-and-forth path, the three methods produced very acceptable results, which confirm that the representation error in Figure V.6 is very small compared to the distance traveled, and there wasn't any deviation away from the true trajectory.

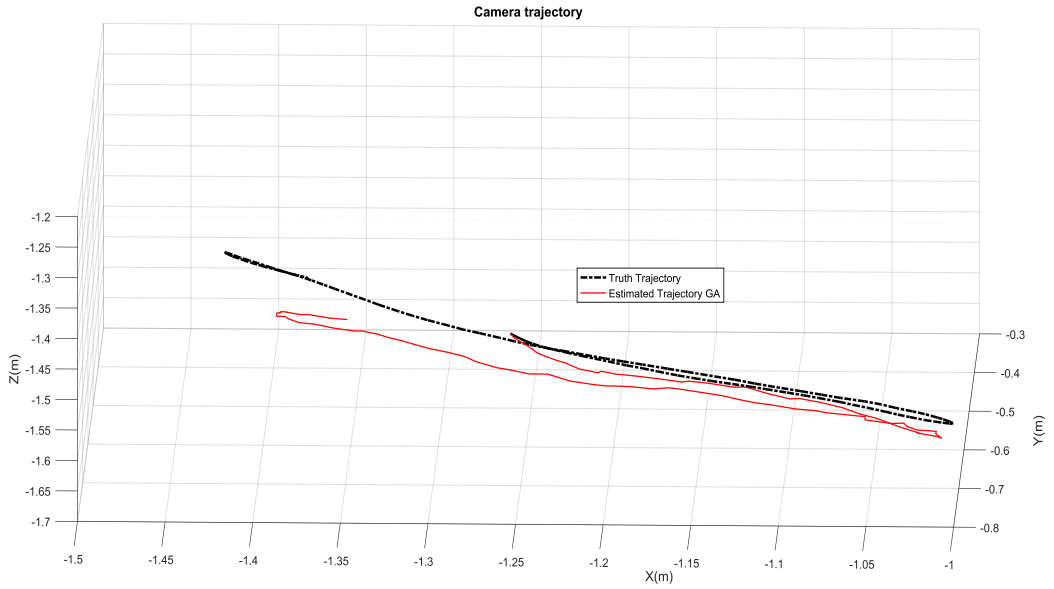

Figure V.8: True camera trajectory with the GA method using a part of the *fr1_xyz* dataset.

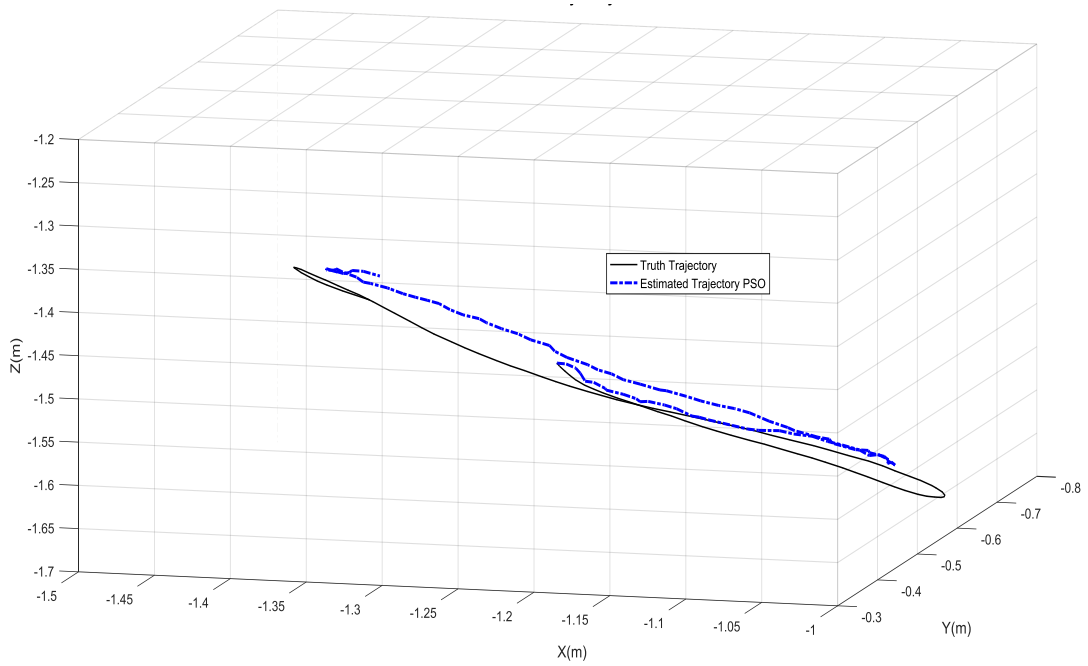

Figure V.9: The true trajectory of the camera and the PSO method trajectory using a part of the *fr1_xyz* dataset.

In the second experiment, we used a sequence of 60 consecutive frames from *rgbd_dataset_freiburg2_desk*, or *fr2_desk*. For this sequence, the scene is a large industrial hall (10×12 m^2), which contains an office environment in the middle of the motion capture area, consisting of two tables with various accessories like a monitor, a keyboard, and books. The distance error between the ground truth and the trajectory of GA, PSO, and the energy-based method is represented in Figure V.10. We notice that in the first thirty frames, the GA method

achieved the least distance error compared to the other methods. This is clearly shown in Table V.I, which demonstrates the superiority of the GA, although the final result of the GA was close to those of the classic method.

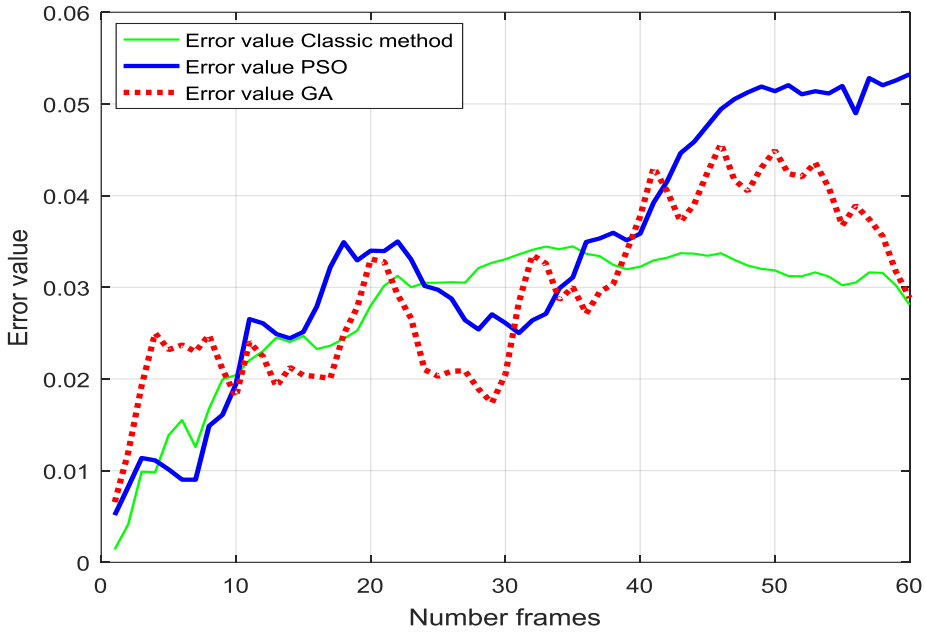

Figure V.10: The camera trajectory error of GA, PSO, and the classic method using a part of the *fr2_desk* dataset.

The representation in the same 3D scene of the true and estimated camera trajectory clearly shows the effectiveness of the motion estimation method. Figure V.11 shows the camera trajectory using the *fr2_desk* dataset of ground truth and different motion estimation methods: classic and meta-heuristic methods. The three methods gave acceptable results, and we observed that the corresponding trajectories are very close to the true trajectory. However, for the trajectory corresponding to the GA method, we notice that it is the closest to the true trajectory, which confirms through this experiment that this innovative method competes with the previous methods: classic and PSO, and may even be better. This confirms the results previously obtained from Figure V.11 and Table V.1.

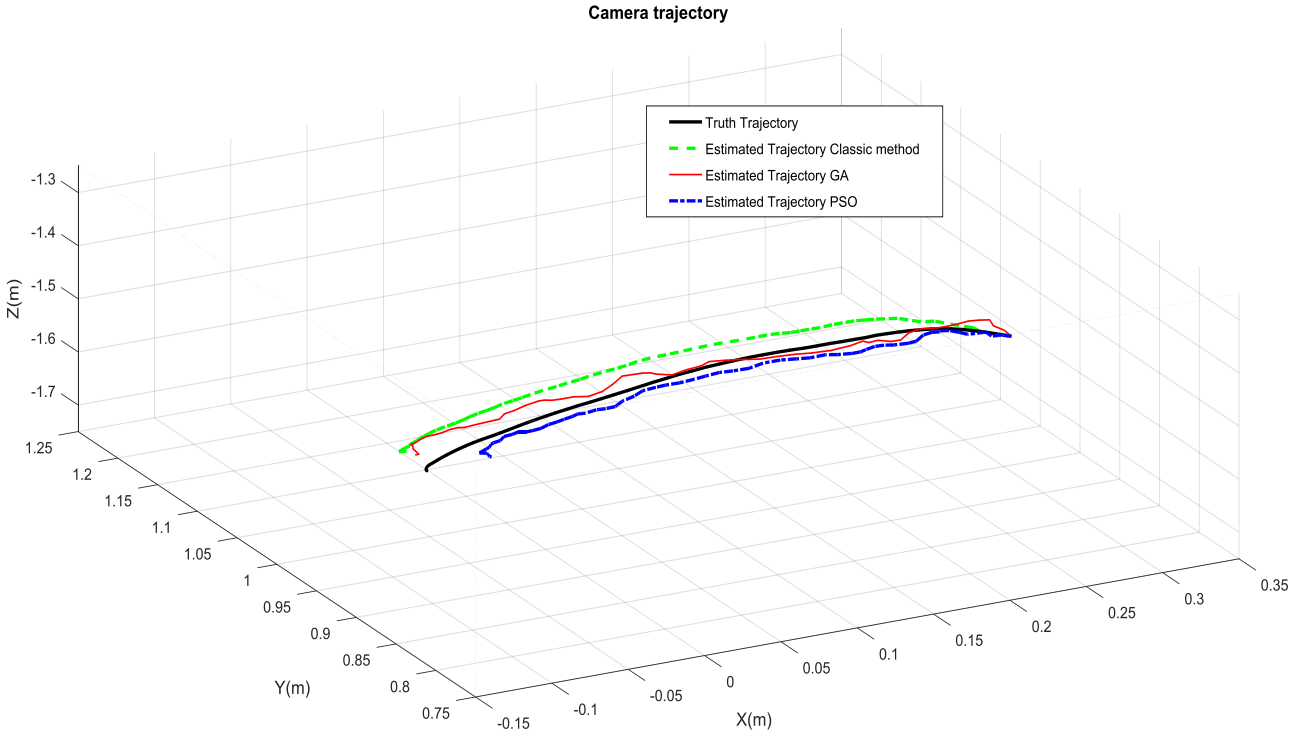

Figure V.11: The true trajectory of the camera and GA, PSO, and Classic method (BE) trajectories using a part of the *fr2_desk* dataset.

Table V.1 shows the root mean square error (RMSE) computed using the Function IV.4, for the two previous methods: GA, and PSO, as well as the results of the method classic presented in [5].

Table V.1: Root mean square error (RMSE) of drift in meters per second for different methods relative to ground truth.

Dataset	GA	PSO	Classic (BE)
fr1_xyz	0.04062 m	**0.03598 m**	0.04827 m
fr2_desk	**0.01856 m**	0.02836 m	0.02524 m

As a summary and synthesis of the results of the first phase of the experiments, we conclude that for the dataset (*freiburg1_xyz*), our method using GA has proven its efficacy in comparison to the classic method. Regarding the dataset (*freiburg2_desk*), our innovative method using GA has successfully proven its efficiency compared to both methods used in the experiments PSO and the classic method.

In the second phase, we evaluated our innovative FA method by conducting two experiments in a static environment and using RGB-D images as mentioned in [121]. These images and their corresponding ground truth are available on the website [3]. First, we used 60 images of *rgbd_dataset_freiburg2_desk*. Figure V.12 represents the distance between the trajectory of ground truth and the trajectory of the three methods used in our experiment to estimate the motion as given by Equation III.33.

Our method using FA has proven its efficacy compared to the classic method and PSO in the first 25 consecutive frames. After that, FA maintains a more valuable RPE but is close to other methods.

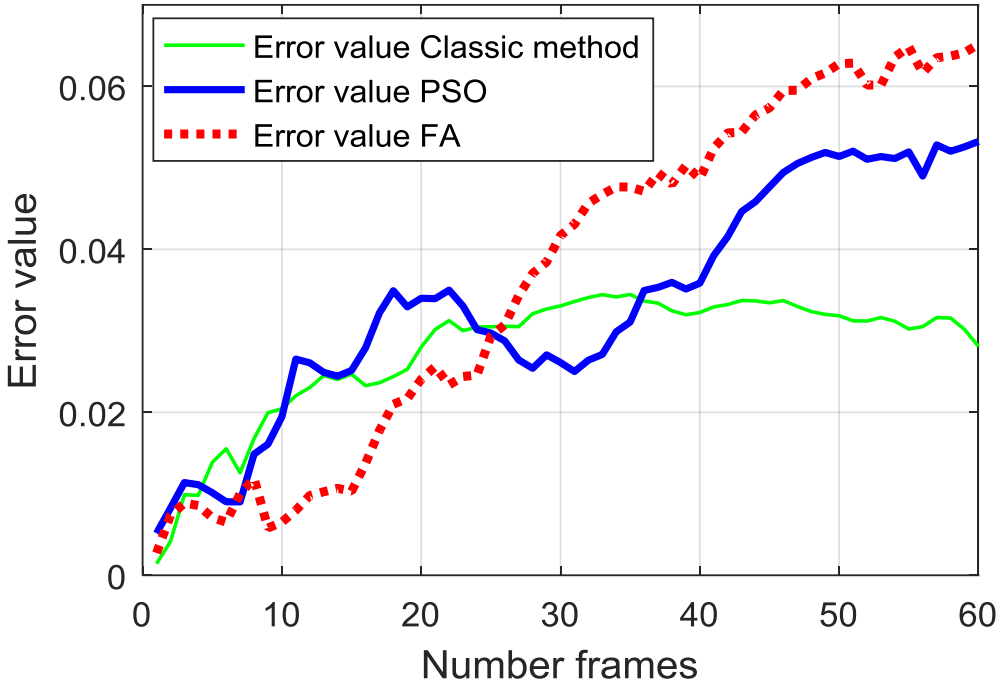

Figure V.12: the camera trajectory error of FA, PSO, and the Classic method (BE) using a part of the *fr2_desk dataset*.

The FA method produced acceptable results in this experiment, which are shown in Figure V.12. The trajectory in the 3D scene of the FA is parallel and close to the true trajectory, and the result is similar to the classic method (EB) and PSO.

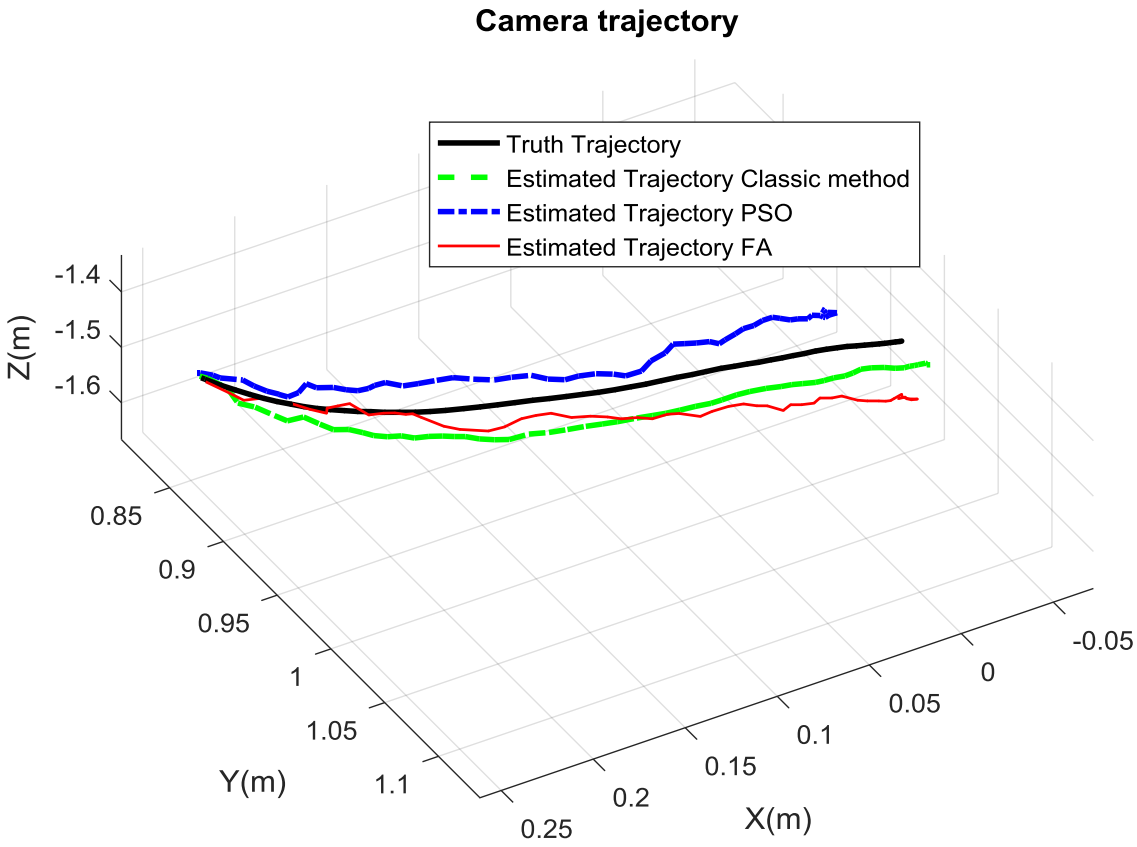

Figure V.13: The true camera trajectory with FA, PSO, and the classic method (BE) trajectories using a part of the *fr2_desk dataset*.

Table V.2 gives the RMSE of FA, PSO, and the classic method based on Function IV.4 as described by Dib [9]. We noticed the RMSE of FA in the previous experiment, which was slightly greater than the RMSE of the other methods, but it did not affect its trajectory in the 3D scene.

Table V.2: Root mean square error (RMSE) of drift in meters per second of different methods compared with ground truth.

dataset	FA	PSO	Classic (BE)
fr2_desk	0.03376 m	0.02836 m	**0.02524 m**
fr1_xyz	**0.03176m**	0.03598 m	0.04827 m

In the second experiment, we utilized *rgbd_dataset_freiburg1_xyz* for 90 consecutive images. We can evaluate our novel method more accurately as the camera's actual trajectory is one of back and forth.

The representation of the 3D landmarks of the true trajectory and the trajectories of previous methods in the same graph (Figures V.14, V.15, and V.16) shows the effectiveness of these methods. The three methods gave acceptable results, where we noticed the corresponding trajectories are very close to the true trajectory in black.

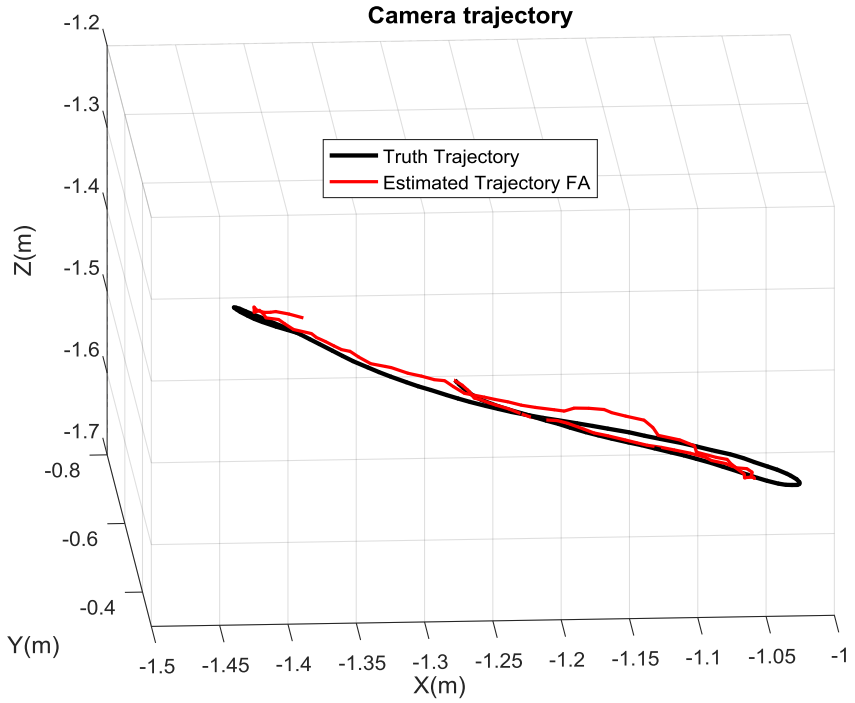

Figure V.14: The true trajectory and estimated trajectory using FA in a 3D scene.

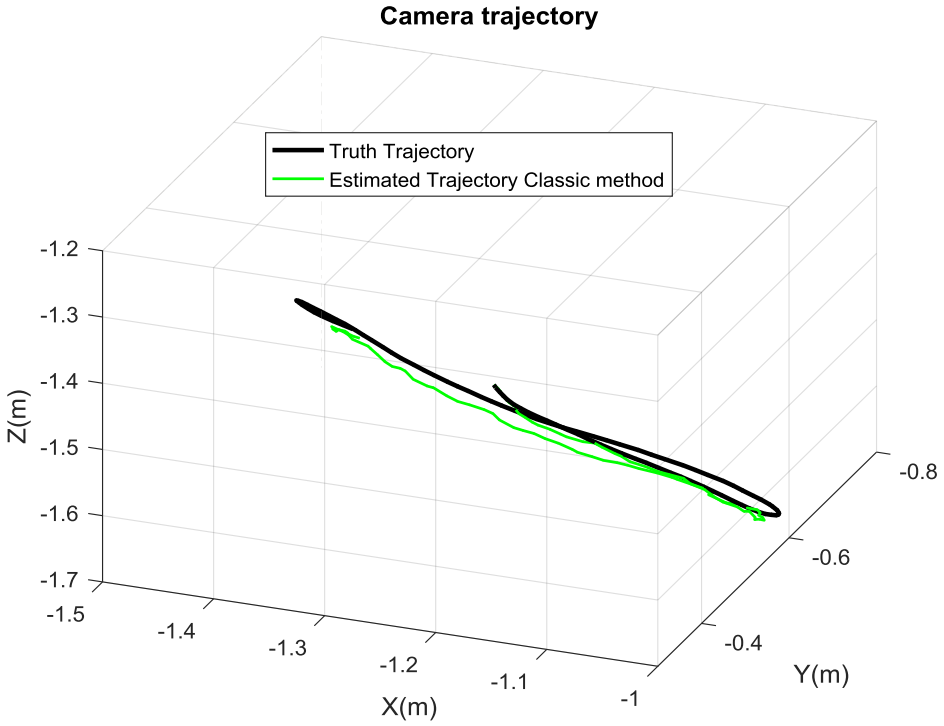

Figure V.15: The true trajectory and estimated trajectory using the classic method (EB) in a 3D scene.

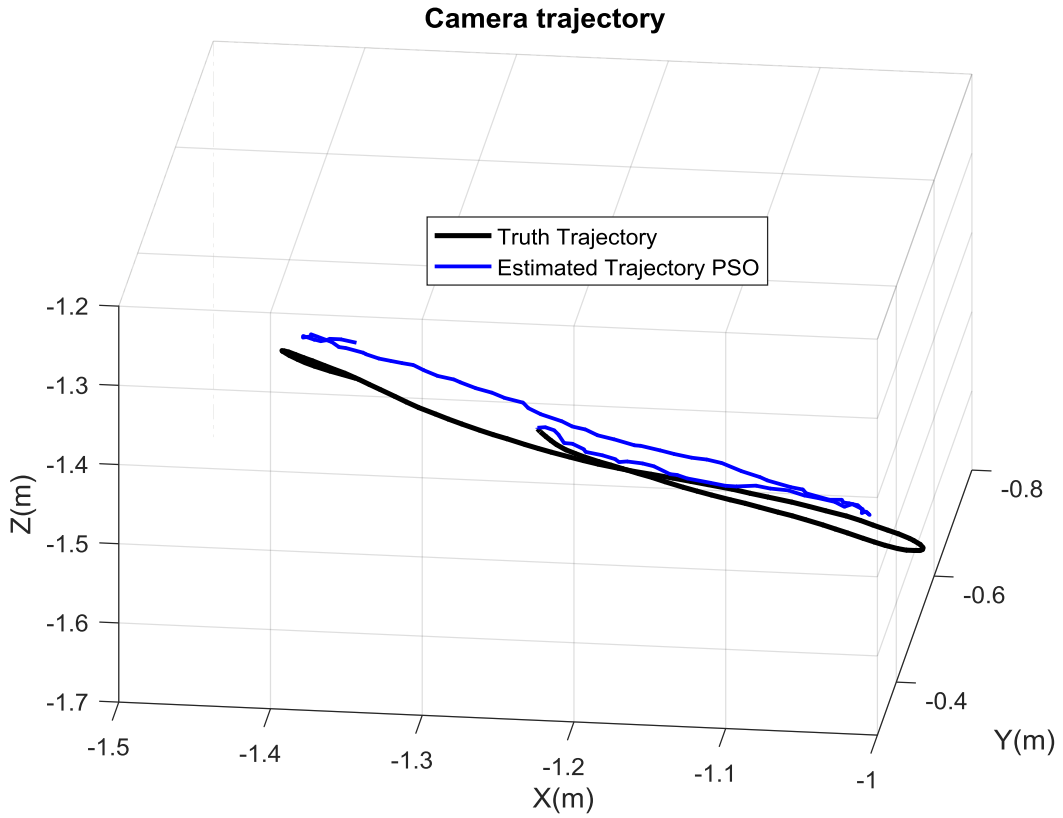

Figure V.16: The true trajectory and estimated trajectory using PSO on a 3D scene.

The evolution of the distance error RPE in Figure V.17 indicates that the FA method is more efficient in motion estimation than the PSO and classic method because the evolution of the error value is more stable for every sequence of 30 images. In contrast, the graphic curve of the classic method in the second stage (the second thirty of the image sequence) increased greatly, and in the third stage, there was a significant decrease, which made the RMSE value greater. The same note applies to the first stage of the PSO method. Additionally, Table V.2 confirmed that FA, in this experiment, gave excellent results and outperformed the other methods.

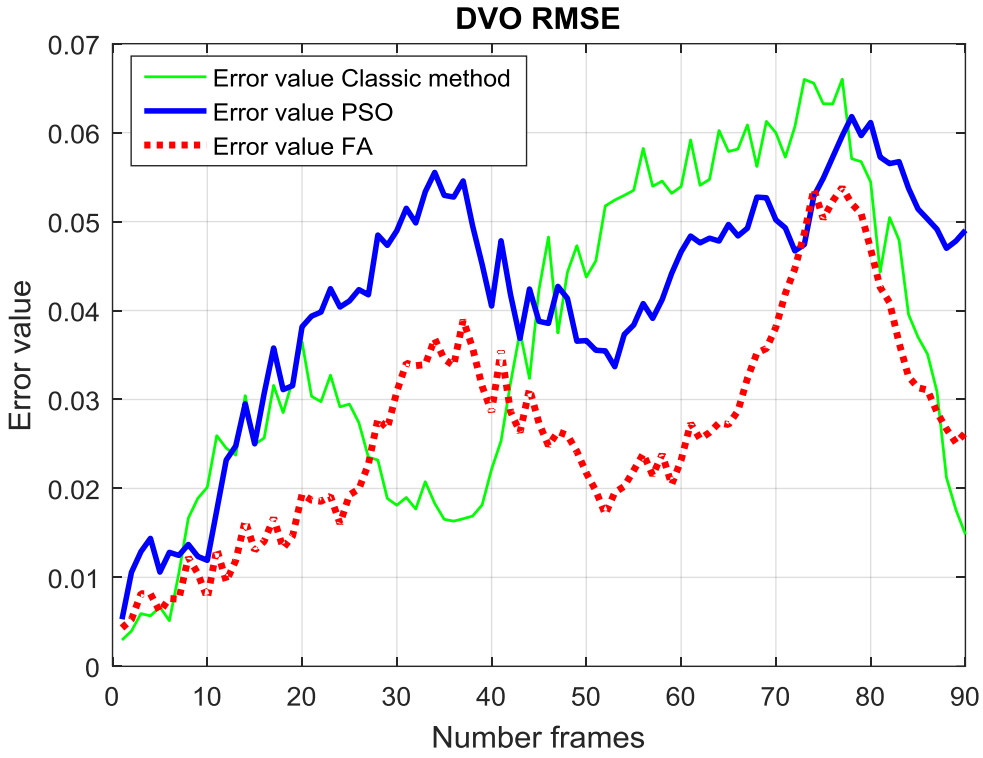

Figure V.17: Camera trajectory error of FA, PSO, and the Classic method using a part of the *fr1_xyz* dataset.

We conducted two important experiments to prove the efficiency of our innovative algorithm FA. Drawing the trajectory in a 3D scene shows that our method achieves the goal of estimating the trajectory using images coming from a mobile camera.

The second experiment, which used a large number of images and a complex true trajectory, clearly demonstrates the superiority of our method over other methods, as shown in Figure V.17 and Table V.2.

V.4. Conclusion

In this thesis, we have presented four optimization methods; among these methods, there are two newly created ones. Two types of experiments were conducted with two different groups of RGB-D images, *fr2_desk*, and *fr1_xyz*, to evaluate these methods. This evaluation was performed by comparing each new method, GA or FA, with the two existing methods, PSO and the classical EB method. All four methods produced acceptable results based on the RMSE values, demonstrating their effectiveness for use in real-world applications in visual odometry.

Conclusion and future work

Conclusion and future work

Visual odometry is a technique for estimating motion using images captured by a camera mounted on the head of a mobile robot or moving object. In this work, a comprehensive exploration and analysis of various optimization methods for dense visual odometry are presented in a static scene. The objective of this research is to enhance the accuracy and efficiency of visual odometry for use in crucial fields such as robotics, augmented reality, and autonomous navigation. The investigation involves the development and evaluation of four optimization methods, with a focus on their applicability and performance in dense visual odometry tasks.

This thesis commences with a review of related work, providing a foundation for understanding the existing landscape of visual odometry and optimization techniques. The review highlights the challenges associated with dense visual odometry and underscores the significance of optimization methods in addressing these challenges. It serves as the backdrop against which the novel contributions of this research are framed.

After that, a literature review of visual odometry is presented, along with an explanation of how this physical problem is modeled as a mathematical equation.

The third part of the thesis introduces the four optimization methods under consideration: the three metaheuristic methods are the Genetic Algorithm (GA), the Firefly Algorithm (FA), and the Particle Swarm Optimization (PSO) method, as well as the classical Energy-Based (EB) method, which is considered an exact method. Each method is carefully explained, detailing its underlying principles.

Following the methodological introduction of different optimization methods, the core of the thesis delves into experimental design and implementation. Two distinct sets of experiments are conducted using RGB-D image data from *fr2_desk* and *fr1_xyz*. These experiments aim to evaluate and compare the performance of each optimization method in the context of dense visual odometry.

The evaluation metrics employed in the experiments include the Root Mean Square Error (RMSE), a widely accepted measure for assessing the accuracy of dense visual odometry. The results were carefully analyzed, and comparisons were conducted between two novel methods (GA and FA) and two existing methods (PSO and EB). These results provide valuable insights into the strengths of each optimization approach.

One of the noteworthy findings is the superior performance of GA and FA in certain experiments, displaying their effectiveness in optimizing dense visual odometry. The adaptability of these novel methods to diverse datasets and their ability to handle complex optimization problems is demonstrated. In the future, we plan to extend this work to estimate body motion in a dynamic scene.

Visual odometry (VO) is a technology that plays a crucial role in various domains, leveraging computer vision and sensor fusion techniques to estimate the motion of a camera or sensor system in a given environment. Visual odometry is essential for autonomous robots to navigate and move through their surroundings. By continuously analyzing visual input from cameras, robots can determine their position and adjust their movements accordingly.

Visual odometry helps autonomous vehicles determine their precise location by tracking the movement of the vehicle through visual input. This is critical for safe and reliable navigation in real-world scenarios. It assists in recognizing obstacles and adjusting the robot's path accordingly. It enhances the vehicle's ability to respond dynamically to changes in the environment.

Visual odometry is used in AR and VR applications to enhance spatial awareness. It enables the accurate overlay of virtual objects onto the real-world environment, creating a seamless and immersive experience.

In summary, visual odometry plays a vital role across various technology domains by providing accurate and real-time information about the motion and position of devices or systems. It enhances navigation, control, and perception capabilities, contributing to the development of advanced and intelligent technologies.

Finally, we hope that this comprehensive work in the field of visual odometry serves as a support for future research by students and researchers, further enriching the domain and advancing its application in the local development of robots, drones, autonomous vehicles, and other innovations that will elevate our country to the forefront of technological advancement. We will devote ourselves and all our efforts and resources to realizing this goal.

Annexes

Annex A

Calculation of the Jacobian components J_i

From Equation III.4, the *i*-th row of Jacobian $J_i(\xi_k)$ is equal to the derivative of image residual r_i with respect to the camera pose parameters ξ_k. According to the chain rule, we can find this equation:

$$J_i(\xi_k) = \frac{\partial r_i}{\partial \xi} \Big|_{\xi=\xi_k} = \frac{\partial r_i}{\partial P(x,y)} \cdot \frac{\partial P(x,y)}{\partial \xi} \Big|_{\xi=\xi_k} \quad \text{A.1}$$

We can compute the derivative of image r_i with respect to frame coordinates $P(x, y)$ through the gradient of each pixel in the image in the direction of *x* and *y* as follows:

$$\frac{\partial r_i}{\partial P(x,y)} = gradient\,(r_i) = \nabla I = \left[\nabla I_x \;, \nabla I_y\right] \quad \text{A.2}$$

Where ∇I_x (resp. ∇I_y) denotes the gradient of image *I* in the direction *x* (resp. *y*).

Thus, we have the following equation

$$J_i(\xi_k) = \frac{\partial r_i}{\partial \xi} \Big|_{\xi=\xi_k} = gradient\,(r_i). \frac{\partial P(x,y)}{\partial \xi} \Big|_{\xi=\xi_k} = \nabla I \,.\, \frac{\partial P(x,y)}{\partial \xi} \Big|_{\xi=\xi_k} = \nabla I \,.\, J_w(\xi_k) \quad \text{A.3}$$

Using the decomposition theorem, we can decompose the warping Jacobian J_w as follows:

$$J_w(\xi_k) = \frac{\partial P(x,y)}{\partial \xi} \Big|_{\xi=\xi k} = \frac{\partial P(x,y)}{\partial P'(X',Y',Z')} \cdot \frac{\partial P'(X',Y',Z')}{\partial \xi} \Big|_{\xi=\xi k} = J_P \,.\, J_\xi(\xi_k) \quad \text{A.4}$$

According to the perspective projection function described in the Equations II.6, we can calculate J_P the derivative of image coordinates $P(x, y)$ with respect to the world point $P'(X', Y', Z')$ as below:

$$J_p = \frac{\partial P(x,y)}{\partial P'(X',Y',Z')} \quad \text{A.5}$$

The Jacobean J_p is a 2×3 matrix, and it is written as follows:

$$J_P = \begin{bmatrix} \frac{\partial P_x}{\partial X'} & \frac{\partial P_x}{\partial Y'} & \frac{\partial P_x}{\partial Z'} \\ \frac{\partial P_y}{\partial X'} & \frac{\partial P_y}{\partial Y'} & \frac{\partial P_y}{\partial Z'} \end{bmatrix} = \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x.X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y.Y'}{Z'^2} \end{bmatrix} \quad \text{A.6}$$

$J_\xi(\xi_k)$ is the Jacobian matrix 2×6 of the exponential map with respect to ξ, and it is calculated as:

$$J_\xi(\xi_k) = \frac{\partial P'(X',Y',Z')}{\partial \xi} \Big|_{\xi=\xi k} = \left[\, I \quad [P'(X',Y',Z')]_\times \,\right] \quad \text{A.7}$$

Where $P'(X', Y', Z') = [\, X', Y', Z' \,]$ A.8

And the operator $[P'(X', Y', Z')]_\times$ creates a 3×3 skew symmetric matrix from a 3×1 vector:

$$[P'(X', Y', Z')]_\times = \begin{bmatrix} 0 & -Z' & Y' \\ Z' & 0 & -X' \\ -Y' & X' & 0 \end{bmatrix} \quad \text{A.9}$$

We point out that *I* denotes a 3×3 identity matrix.

So, we conclude from the above that the equation J_ξ is

$$J_\xi(\xi_k) = \begin{bmatrix} 1 & 0 & 0 & 0 & -Z' & Y' \\ 0 & 1 & 0 & Z' & 0 & -X' \\ 0 & 0 & 1 & -Y' & X' & 0 \end{bmatrix} \quad \text{A.10}$$

The warping Jacobian matrix J_w is calculated as:

$$J_w(\xi_k) = J_P \,.\, J_\xi(\xi_k) = \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x.X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y.Y'}{Z'^2} \end{bmatrix} . \begin{bmatrix} 1 & 0 & 0 & 0 & -Z' & Y' \\ 0 & 1 & 0 & Z' & 0 & -X' \\ 0 & 0 & 1 & -Y' & X' & 0 \end{bmatrix} \quad \text{A.11}$$

Thus, the form of the matrix J_w is as shown below

$$J_w(\xi_k) = J_P \,.\, J_\xi(\xi_k) = \begin{bmatrix} \frac{fx}{Z\prime} & 0 & -\frac{fx.X\prime}{Z\prime^2} & -\frac{fx.X'.Y\prime}{Z\prime^2} & fx + \frac{fx.X\prime^2}{Z\prime^2} & -\frac{fx.Y\prime}{Z\prime} \\ 0 & \frac{fy}{Z\prime} & -\frac{fy.Y\prime}{Z\prime^2} & -fy - \frac{fy.Y\prime^2}{Z\prime^2} & \frac{fy.X'.Y\prime}{Z\prime^2} & \frac{fy.X\prime}{Z\prime} \end{bmatrix} \quad \text{A.12}$$

In the previous analysis, we had the following form:

$$J_i(\xi_k) = \frac{\partial\, r_i}{\partial\, \xi} \Big|_{\xi = \xi k} = \frac{\partial\, r_i}{\partial P(u,v)} \,.\, \frac{\partial P(u,v)}{\partial\, \xi} \Big|_{\xi = \xi k} = \textit{gradient}\,(r_i).\, J_w(\xi_k) = \nabla I_{t+1} \,.\, J_w(\xi_k) \quad \text{A.13}$$

The final Jacobian is written as follows:

$$J_i(\xi_k) = [\, \nabla It+1, x \;, \nabla It+1, y \,] . \begin{bmatrix} \frac{fx}{Z\prime} & 0 & -\frac{fx.X\prime}{Z\prime^2} & -\frac{fx.X'.Y\prime}{Z\prime^2} & fx + \frac{fx.X\prime^2}{Z\prime^2} & -\frac{fx.Y\prime}{Z\prime} \\ 0 & \frac{fy}{Z\prime} & -\frac{fy.Y\prime}{Z\prime^2} & -fy - \frac{fy.Y\prime^2}{Z\prime^2} & \frac{fy.X'.Y\prime}{Z\prime^2} & \frac{fy.X\prime}{Z\prime} \end{bmatrix} \quad \text{A.14}$$

Annex B

In this part, we have provided the values of the constants for the important equations used to execute the algorithms of the considered optimization methods.

Equation number	Constants value
II.6	f_x=525.0, f_y=525.0, c_x=319.5, c_y=239.5,
III.22	p_m = 0.1
III.24	w=0.72984, c_1=1.496172, c_2=1.496172,
III.37	k =1, φ =4.1, χ =0.72984,
III.41	β_0=2, γ=1, α_t=0.2,

Annex C

MATLAB code of Lie Algebra and Lie Group Mapping.

Here is the code for se3 -> SE3 where `ksi`= ξ

```
      function g = RBMotion(ksi)
% Project: Dense Visual Odometry
% Function: RBMotion
%
% Description:
%   Rigid Body Motion Calculation (Lie Algebra)
%  g: (SE3) Rigid body motion matrix (4*4), which describe the camera
%  motion
%   between the two successive snapshot.
%   ksi: (se3) A 6*1 matrix which includes camera motion parameters (ksi
%   = (v1 v2 v3 w1 w2 w3)').
%
%
%   [ SE3 ] = se3_SE3( se3 )
%   se3_SE3 Exponential Mapping from Lie Algebra to Lie Group
%  each of the six elements on multiplication with the generator matrices
%   as follows give the complete matrix:
%   se3 = v1*g1 + v2*g2 + v3*g3 + w1*g4 + w2*g5 + w3*g6
%   To map se3 to SE3 we need to perform e^(se3)
```

```
%   This can be done by following the algorithm:
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
g = eye(4);
v = ksi(1:3);
w = ksi(4:6);

len_w = sqrt(dot(w,w));
Wx = TwistMatrix(w);
if len_w < 1e-7
R = eye(3) + Wx + 0.5*Wx*Wx;

V = eye(3) + 0.5*Wx + Wx*Wx/3;
else
R = eye(3) + sin(len_w)/len_w*Wx + (1- cos(len_w))/len_w^2*(Wx*Wx);
V = eye(3) + (1-cos(len_w))/len_w^2*Wx + (len_w-...
sin(len_w))/len_w^3*(Wx*Wx);
end
t = V*v';

g(1,1:3) = R(1,:);
g(2,1:3) = R(2,:);
g(3,1:3) = R(3,:);
g(1:3,4) = t;
```

Here is the code for SE3 -> se3

```
function ksi = LieLogrithm(g)
% Project: Dense Visual Odometry
% Function: LieLogrithm
%
% Description:
%   Get the camera motion parameter ksi from its corresponding rigid
%   body
```

```
%   motion matrix.
%
% Example:
%   ksi = LieLogrithm(g)
%
%   ksi: (se3) The camera motion parameters (a 6*1 matrix
%   (v1,v2,v3,w1,w2,w3)')
%   g  : (SE3) The rigid body motion matrix (a 4*4 matrix)
%   [ se3] = SE3_se3_back( SE3 )
%   SE3_se3_back Logarithm Mapping from Lie Group to Lie Algebra
%   To map SE3 to se3 we need to perform log^(SE3)
%   This can be done by following the algorithm:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
R = g(1:3,1:3);
t = g(1:3,4);
theta = acos((trace(R)-1)/2);
if theta < 0.001
    w = (0.5*[R(3,2)-R(2,3),R(1,3)-R(3,1),R(2,1)-R(1,2)])';
else
    w =(0.5*theta/sin(theta)*[R(3,2)-R(2,3),R(1,3)-R(3,1),R(2,1)- ...
R(1,2)])';
end
len_w = sqrt(dot(w,w));
Wx = TwistMatrix(w);
if len_w < 0.001
    V = eye(3) + 0.5*Wx + (Wx*Wx)/3;
else
    V =eye(3) + (1-cos(len_w))/len_w^2*Wx + (len_w-...
sin(len_w))/len_w^3*(Wx*Wx);
end
v = V\t;
ksi = [v',w'];
```

References

[1] F. Cheng, C. Liu, H. Wu, and M. Ai, “DIRECT SPARSE VISUAL ODOMETRY WITH STRUCTURAL REGULARITIES FOR LONG CORRIDOR ENVIRONMENTS,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XLIII-B2-2020, pp. 757–763, Aug. 2020, doi: 10.5194/isprs-archives-XLIII-B2-2020-757-2020.

[2] L. Cruz, D. Lucio, and L. Velho, “Kinect and RGBD Images: Challenges and Applications,” in *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials*, Ouro Preto, Brazil, Aug. 2012, pp. 36–49. doi: 10.1109/SIBGRAPI-T.2012.13.

[3] “RGB-D SLAM Dataset and Benchmark,” TUM Department of Informatics, Technical University of Munich, Germany, 2011. Accessed: Jan. 01, 2021. [Online]. Available: http://vision.in.tum.de/data/datasets/rgbd-dataset.

[4] C. Patruno, V. Renò, M. Nitti, N. Mosca, M. Di Summa, and E. Stella, “Vision-based omnidirectional indoor robots for autonomous navigation and localization in manufacturing industry,” *Heliyon*, vol. 10, no. 4, p. e26042, Feb. 2024, doi: 10.1016/j.heliyon.2024.e26042.

[5] A. Dib and F. Charpillet, “Robust dense visual odometry for RGB-D cameras in a dynamic environment,” Istanbul, Jul. 2015, pp. 1–7. doi: 10.1109/ICAR.2015.7298210.

[6] A. Dib, “Vers un système de capture du mouvement humain en 3D pour un robot mobile évoluant dans un environnement encombré,” doctoral thesis, Universit e de Lorraine, Nancy, France, 2016. [Online]. Available: https://hal.science/tel-01752233v2

[7] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of RGB-D SLAM systems,” Vilamoura-Algarve, Portugal, Oct. 2012, pp. 573–580. doi: 10.1109/IROS.2012.6385773.

[8] S. T. Khawase, S. D. Kamble, N. V. Thakur, and A. S. Patharkar, “An Overview of Block Matching Algorithms for Motion Vector Estimation,” Jun. 2017, vol. 10, pp. 217–222. doi: 10.15439/2017R85.

[9] M. Ghaffari, W. Clark, A. Bloch, R. M. Eustice, and J. W. Grizzle, “Continuous Direct Sparse Visual Odometry from RGB-D Images,” no. arXiv:1904.02266. arXiv, Aug. 23, 2019. doi: 10.48550/ARXIV.1904.02266.

[10] N. Zhang and Y. Zhao, “Fast and Robust Monocular Visua-Inertial Odometry Using Points and Lines,” *Sensors*, vol. 19, no. 20, p. 4545, Oct. 2019, doi: 10.3390/s19204545.

[11] D. Nister, O. Naroditsky, and J. Bergen, “Visual odometry,” in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004.*

CVPR 2004., Washington, DC, USA, 2004, vol. 1, pp. 652–659. doi: 10.1109/CVPR.2004.1315094.

[12] K. Konolige, M. Agrawal, and J. Solà, "Large-Scale Visual Odometry for Rough Terrain," in *Robotics Research*, vol. 66, M. Kaneko and Y. Nakamura, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 201–212. doi: 10.1007/978-3-642-14743-2_18.

[13] A. S. Huang *et al.*, "Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera," Flagstaff, Arizona, USA, 2011. doi: 10.1007/978-3-319-29363-9_14.

[14] J. Engel, J. Sturm, and D. Cremers, "Camera-based navigation of a low-cost quadrocopter," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, Oct. 2012, pp. 2815–2821. doi: 10.1109/IROS.2012.6385458.

[15] S. Weiss, M. W. Achtelik, S. Lynen, M. Chli, and R. Siegwart, "Real-time onboard visual-inertial state estimation and self-calibration of MAVs in unknown environments," in *2012 IEEE International Conference on Robotics and Automation*, St Paul, MN, USA, May 2012, pp. 957–964. doi: 10.1109/ICRA.2012.6225147.

[16] G. Klein and D. Murray, "Parallel Tracking and Mapping for Small AR Workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, Nara, Japan, Nov. 2007, pp. 1–10. doi: 10.1109/ISMAR.2007.4538852.

[17] E. Rosten and T. Drummond, "Machine Learning for High-Speed Corner Detection," in *Computer Vision – ECCV 2006*, vol. 3951, A. Leonardis, H. Bischof, and A. Pinz, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 430–443. doi: 10.1007/11744023_34.

[18] C. Harris and M. Stephens, "A Combined Corner and Edge Detector," in *Procedings of the Alvey Vision Conference 1988*, Manchester, 1988, p. 23.1-23.6. doi: 10.5244/C.2.23.

[19] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Kerkyra, Greece, 1999, pp. 1150–1157 vol.2. doi: 10.1109/ICCV.1999.790410.

[20] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, Jun. 2008, doi: 10.1016/j.cviu.2007.09.014.

[21] H. Matsuki, R. Scona, J. Czarnowski, and A. J. Davison, "CodeMapping: Real-Time Dense Mapping for Sparse SLAM using Compact Scene Representations," *IEEE Robot. Autom. Lett.*, vol. 6, no. 4, pp. 7105–7112, Oct. 2021, doi: 10.1109/LRA.2021.3097258.

[22] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm,” in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, Quebec City, Que., Canada, 2001, pp. 145–152. doi: 10.1109/IM.2001.924423.

[23] G. Blais and M. D. Levine, “Registering multiview range data to create 3D computer objects,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 17, no. 8, pp. 820–824, Aug. 1995, doi: 10.1109/34.400574.

[24] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments,” in *Experimental Robotics*, vol. 79, O. Khatib, V. Kumar, and G. Sukhatme, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 477–491. doi: 10.1007/978-3-642-28572-1_33.

[25] J. Stückler and S. Behnke, “Model Learning and Real-Time Tracking Using Multi-Resolution Surfel Maps,” *Proc. AAAI Conf. Artif. Intell.*, vol. 26, no. 1, pp. 2081–2087, Sep. 2021, doi: 10.1609/aaai.v26i1.8388.

[26] F. Steinbrucker, J. Sturm, and D. Cremers, “Real-time visual odometry from dense RGB-D images,” in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Barcelona, Spain, Nov. 2011, pp. 719–722. doi: 10.1109/ICCVW.2011.6130321.

[27] T. Tykkala, C. Audras, and A. I. Comport, “Direct Iterative Closest Point for real-time visual odometry,” in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, Barcelona, Spain, Nov. 2011, pp. 2050–2056. doi: 10.1109/ICCVW.2011.6130500.

[28] B. D. Lucas and T. Kanade, “An Iterative Image Registration Technique with an Application to Stereo Vision,” Vancouver, BC, Canada, Aug. 1981, pp. 674–679.

[29] S. Baker and I. Matthews, “Lucas-Kanade 20 Years On: A Unifying Framework,” *Int. J. Comput. Vis.*, vol. 56, no. 3, pp. 221–255, Feb. 2004, doi: 10.1023/B:VISI.0000011205.11775.fd.

[30] R. Jiang, R. Klette, and S. Wang, “Modeling of Unbounded Long-Range Drift in Visual Odometry,” in *2010 Fourth Pacific-Rim Symposium on Image and Video Technology*, Singapore, Singapore, Nov. 2010, pp. 121–126. doi: 10.1109/PSIVT.2010.27.

[31] R. A. Newcombe *et al.*, “KinectFusion: Real-time dense surface mapping and tracking,” in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, Basel, Oct. 2011, pp. 127–136. doi: 10.1109/ISMAR.2011.6092378.

[32] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, “DTAM: Dense tracking and mapping in real-time,” in *2011 International Conference on Computer Vision*, Barcelona, Spain, Nov. 2011, pp. 2320–2327. doi: 10.1109/ICCV.2011.6126513.

[33] B. Canovas, M. Rombaut, A. Negre, D. Pellerin, and S. Olympieff, “Speed and Memory Efficient Dense RGB-D SLAM in Dynamic Scenes,” Las Vegas, NV, USA, Oct. 2020, pp. 4996–5001. doi: 10.1109/IROS45743.2020.9341542.

[34] A. I. Comport, E. Malis, and P. Rives, “Real-time Quadrifocal Visual Odometry,” *Int. J. Robot. Res.*, vol. 29, no. 2–3, Art. no. 2–3, Feb. 2010, doi: 10.1177/0278364909356601.

[35] P. J. Besl and N. D. McKay, “A method for registration of 3-D shapes,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 14, no. 2, Art. no. 2, Feb. 1992, doi: 10.1109/34.121791.

[36] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, “Robust real-time visual odometry for dense RGB-D mapping,” Karlsruhe, Germany, May 2013, pp. 5724–5731. doi: 10.1109/ICRA.2013.6631400.

[37] C. Kerl, J. Sturm, and D. Cremers, “Robust odometry estimation for RGB-D cameras,” Karlsruhe, Germany, May 2013, pp. 3748–3754. doi: 10.1109/ICRA.2013.6631104.

[38] C. Kerl, J. Sturm, and D. Cremers, “Dense visual SLAM for RGB-D cameras,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Nov. 2013, pp. 2100–2106. doi: 10.1109/IROS.2013.6696650.

[39] J. Wei and A. Yilmaz, “A Visual Odometry Pipeline for Real-Time UAS Geopositioning,” *Drones*, vol. 7, no. 9, p. 569, Sep. 2023, doi: 10.3390/drones7090569.

[40] Z. Han, “Multimodal intelligent logistics robot combining 3D CNN, LSTM, and visual SLAM for path planning and control,” *Front. Neurorobotics*, vol. 17, p. 1285673, Oct. 2023, doi: 10.3389/fnbot.2023.1285673.

[41] Kyung M. Kim, “Monocular Visual Odometry for Fixed-Wing Small Unmanned Aircraft Systems,” Degree of Master of Science in Computer Science, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, USA, 2019.

[42] JENNY NILSSON BOIJ, “Localization of Combat Aircraft at High Altitude using Visual Odometry,” Master’s Thesis in Engineering Physics, Department of Physics, Umea University, Ume, Sweden, 2022.

[43] M. Maimone, Y. Cheng, and L. Matthies, “Two years of Visual Odometry on the Mars Exploration Rovers,” *J. Field Robot.*, vol. 24, no. 3, pp. 169–186, Mar. 2007, doi: 10.1002/rob.20184.

[44] Yang Cheng, M. Maimone, and L. Matthies, “Visual Odometry on the Mars Exploration Rovers,” in *2005 IEEE International Conference on Systems, Man and Cybernetics*, Waikoloa, HI, USA, 2005, vol. 1, pp. 903–910. doi: 10.1109/ICSMC.2005.1571261.

[45] K. Yousif, A. Bab-Hadiashar, and R. Hoseinnezhad, “An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics,” *Intell. Ind. Syst.*, vol. 1, no. 4, pp. 289–311, Dec. 2015, doi: 10.1007/s40903-015-0032-7.

[46] K. Wang, S. Ma, J. Chen, F. Ren, and J. Lu, “Approaches, Challenges, and Applications for Deep Visual Odometry: Toward Complicated and Emerging Areas,” *IEEE Trans. Cogn. Dev. Syst.*, vol. 14, no. 1, pp. 35–49, Mar. 2022, doi: 10.1109/TCDS.2020.3038898.

[47] E. Cuevas, D. Zaldívar, M. Pérez-Cisneros, H. Sossa, and V. Osuna, “Block matching algorithm for motion estimation based on Artificial Bee Colony (ABC),” *Appl. Soft Comput.*, vol. 13, no. 6, pp. 3047–3059, Jun. 2013, doi: 10.1016/j.asoc.2012.09.020.

[48] Marco Tagliasacchi, “A genetic algorithm for optical flow estimation,” *Image Vis. Comput.*, vol. 25, no. 2, pp. 141–147, Feb. 2007, doi: 10.1016/j.imavis.2006.01.021.

[49] M. Shahbazi, G. Sohn, J. Théau, and P. Ménard, “ROBUST SPARSE MATCHING AND MOTION ESTIMATION USING GENETIC ALGORITHMS,” *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. XL-3/W2, pp. 197–204, Mar. 2015, doi: 10.5194/isprsarchives-XL-3-W2-197-2015.

[50] Adarsh Sehgal, Ashutosh Singandhupe, Hung Manh La, Alireza Tavakkoli, Sushil J. Louis, “Lidar-Monocular Visual Odometry with Genetic Algorithm for Parameter Optimization,” *Adv. Vis. Comput. Springer Int. Publ.*, vol. 11845, pp. 358–370, 2019, doi: 10.48550/arXiv.1903.02046.

[51] Y. K. Yu, K. H. Wong, and M. M. Y. Chang, “Pose Estimation for Augmented Reality Applications Using Genetic Algorithm,” *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 35, no. 6, pp. 1295–1301, Dec. 2005, doi: 10.1109/TSMCB.2005.850164.

[52] C.-F. Chao, M.-H. Horng, and Y.-C. Chen, “Motion Estimation Using the Firefly Algorithm in Ultrasonic Image Sequence of Soft Tissue,” *Comput. Math. Methods Med.*, vol. 2015, pp. 1–8, 2015, doi: 10.1155/2015/343217.

[53] Y. K. Baik, J. Kwon, H. S. Lee, and K. M. Lee, “Geometric particle swarm optimization for robust visual ego-motion estimation via particle filtering,” *Image Vis. Comput.*, vol. 31, no. 8, Art. no. 8, Aug. 2013, doi: 10.1016/j.imavis.2013.04.004.

[54] Aleksander Kostusiak and Piotr Skrzypczyński, “On the Efficiency of Population-Based Optimization in Finding Best Parameters for RGB-D Visual Odometry,” *J. Autom. Mob. Robot. Intell. Syst.*, vol. 11, no. 2, Art. no. 2, Jul. 2019, doi: 10.14313/JAMRIS/2-2019/13.

[55] H.-C. Lee, S.-K. Park, J.-S. Choi, and B.-H. Lee, “PSO-FastSLAM: An improved FastSLAM framework using particle swarm optimization,” in *2009 IEEE International Conference on Systems, Man and Cybernetics*, San Antonio, TX, USA, Oct. 2009, pp. 2763–2768. doi: 10.1109/ICSMC.2009.5346572.

[56] H. S. Lee and K. M. Lee, “Multiswarm Particle Filter for vision based SLAM,” in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, St. Louis, MO, USA, Oct. 2009, pp. 924–929. doi: 10.1109/IROS.2009.5354144.

[57] I. A. Sulistijono, N. Kubota, Dept. of Mechanical Engineering, Graduate School of Engineering, Tokyo Metropolitan University, 6-6 Asahigaoka, Hino, Tokyo 191-0065, Japan, Electronics Eng. Polytechnic Institute of Surabaya - ITS (EEPIS-ITS), Kampus ITS Sukolilo, Surabaya 60111, Indonesia, Dept. of System Design, Tokyo Metropolitan University, 6-6 Asahigaoka, Hino, Tokyo 191-0065, Japan, and SORST, Japan Science and Technology Agency (JST), “Human Head Tracking Based on Particle Swarm Optimization and Genetic Algorithm,” *J. Adv. Comput. Intell. Intell. Inform.*, vol. 11, no. 6, pp. 681–687, Jul. 2007, doi: 10.20965/jaciii.2007.p0681.

[58] Xiaoqin Zhang, Weiming Hu, S. Maybank, Xi Li, and Mingliang Zhu, “Sequential particle swarm optimization for visual tracking,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, AK, USA, Jun. 2008, pp. 1–8. doi: 10.1109/CVPR.2008.4587512.

[59] Christian Kerl, “Odometry from RGB-D Cameras for Autonomous Quadrocopters,” Master’s Thesis in Robotics, Cognition, Intelligence, FACULTY OF INFORMATICS OF THE TECHNICAL, UNIVERSITY OF MUNICH, Munich, Germany, 2012.

[60] Y. Ahmine, G. Caron, F. Chouireb, and E. M. Mouaddib, “Continuous Scale-Space Direct Image Alignment for Visual Odometry From RGB-D Images,” *IEEE Robot. Autom. Lett.*, vol. 6, no. 2, Art. no. 2, Apr. 2021, doi: 10.1109/LRA.2021.3061309.

[61] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Illustrée, Réimprimée. Cambridge University Press, 2004. [Online]. Available: https://books.google.dz/books?id=MHrYnQEACAAJ

[62] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 11, pp. 1330–1334, Nov. 2000, doi: 10.1109/34.888718.

[63] J. Smisek, M. Jancosek, and T. Pajdla, “3D with Kinect,” in *Consumer Depth Cameras for Computer Vision*, A. Fossati, J. Gall, H. Grabner, X. Ren, and K. Konolige, Eds. London: Springer London, 2013, pp. 3–25. doi: 10.1007/978-1-4471-4640-7_1.

[64] “Focal Length and Intrinsic Camera Parameters,” Tarnum Java SRL, Bucharest, Romania, Computer vision. Accessed: Jan. 01, 2022. [Online]. Available: https://www.baeldung.com/cs/focal-length-intrinsic-camera-parameters

[65] J. L. B. Claraco, “A tutorial on SE(3) transformation parameterizations and on-manifold optimization,” ETS Computer Engineering - University of Malaga, Systems and Automation Engineer 012010, Jul. 2022. [Online]. Available: https://w3.ual.es/personal/jlblanco/

[66] A. Sarthak, “Lie Algebra to Lie Group Mappin.html,” *math.stackexchange*, Jun. 27, 2015. https://math.stackexchange.com/questions/1312314/lie-algebra-to-lie-group-mapping

[67] Y. Ma, S. Soatto, J. Košecká, and S. S. Sastry, *An Invitation to 3-D Vision*, vol. 26. New York, NY: Springer New York, 2004. doi: 10.1007/978-0-387-21779-6.

[68] H. Strasdat, J. M. M. Montiel, and A. Davison, “Scale Drift-Aware Large Scale Monocular SLAM,” presented at the Robotics: Science and Systems 2010, Jun. 2010. doi: 10.15607/RSS.2010.VI.010.

[69] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, “Pyramid methods in image processing,” Perceptual Science Group, Department of Brain and Cognitive Sciences, 77 Massachusetts Ave MIT, 46-4115 Cambridge, MA 02139, 1984. [Online]. Available: http://persci.mit.edu/publications

[70] E. N. Eriksen, “Monocular Visual Odometry for Underwater Navigation,” Master thesis, Cybernetics and Robotics, Norwegian University of Science and Technology, Trondheim, Norvège, 2020. Accessed: Jan. 05, 2022. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2656718

[71] Jamal TOUTOUH, “Natural Computing for Vehicular Networks,” UNIVERSIDAD DE MÁLAGA, Avda. Cervantes, n.º 2 Málaga, España, 2015. [Online]. Available: http://hdl.handle.net/10630/13432

[72] S. Ruder, “An overview of gradient descent optimization algorithms.” arXiv, 2016. doi: 10.48550/ARXIV.1609.04747.

[73] I. STANIMIROVIC, *ADVANCES IN OPTIMIZATION AND LINEAR PROGRAMMING*. S.l.: APPLE ACADEMIC PRESS, 2024.

[74] Rabia Khan, Suriya Gharib, Syeda Roshana Ali, and Memoona khanam, “System Of Linear Equations , Guassian Elimination,” *Glob. J. Comput. Sci. Technol.*, vol. 15, no. C5, pp. 23–26, Mar. 2015.

[75] V. Kumar and S. M. Yadav, “A state-of-the-Art review of heuristic and metaheuristic optimization techniques for the management of water resources,” *Water Supply*, vol. 22, no. 4, pp. 3702–3728, Apr. 2022, doi: 10.2166/ws.2022.010.

[76] É. D. Taillard, *Design of Heuristic Algorithms for Hard Optimization: With Python Codes for the Travelling Salesman Problem*. Cham: Springer International Publishing, 2023. doi: 10.1007/978-3-031-13714-3.

[77] R. Jovanovic, A. Bousselham, and S. Voß, “A heuristic method for solving the problem of partitioning graphs with supply and demand,” *Ann. Oper. Res.*, vol. 235, no. 1, pp. 371–393, Dec. 2015, doi: 10.1007/s10479-015-1930-5.

[78] Fred Glover and Gary A. Kochenberger, *HANDBOOK OF METAHEURISTICS*. NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW: Kluwer Academic Publishers, 2003.

[79] M. Gendreau and J.-Y. Potvin, Eds., *Handbook of Metaheuristics*, vol. 272. Cham: Springer International Publishing, 2019. doi: 10.1007/978-3-319-91086-4.

[80] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM Comput. Surv.*, vol. 35, no. 3, pp. 268–308, Sep. 2003, doi: 10.1145/937503.937505.

[81] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, “Optimization by Simulated Annealing,” *Am. Assoc. Adv. Sci.*, vol. 220, no. 4598, pp. 671–680, May 1983.

[82] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999. doi: 10.1093/oso/9780195131581.001.0001.

[83] R. Eberhart and J. Kennedy, “A new optimizer using particle swarm theory,” in *MHS’95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, Nagoya, Japan, 1995, pp. 39–43. doi: 10.1109/MHS.1995.494215.

[84] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 1992. doi: 10.7551/mitpress/1090.001.0001.

[85] L. N. De Castro, *Fundamentals of Natural Computing: Basic Concepts, Algorithms, and Applications*, 0 ed. Chapman and Hall/CRC, 2006. doi: 10.1201/9781420011449.

[86] D. E. Goldberg and D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*, 30. print. Boston: Addison-Wesley, 2012.

[87] Å. Björck, *Numerical Methods for Least Squares Problems*. Society for Industrial and Applied Mathematics, 1996. doi: 10.1137/1.9781611971484.

[88] J. O. Ogundare, *Understanding Least Squares Estimation and Geomatics Data Analysis*, 1st ed. Wiley, 2018. doi: 10.1002/9781119501459.

[89] J. Zhu, "Image Gradient-based Joint Direct Visual Odometry for Stereo Camera," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, Melbourne, Australia, Aug. 2017, pp. 4558–4564. doi: 10.24963/ijcai.2017/636.

[90] A. Fraser and D. G. Burnell, *Computer models in genetics*. New York: McGraw-Hill, 1970.

[91] E. Talbi, *Metaheuristics: From Design to Implementation*, 1st ed. Wiley, 2009. doi: 10.1002/9780470496916.

[92] Walchand College of Engineering, U. A.J., S. P.D., and Government College of Engineering, Karad, "CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW," *ICTACT J. Soft Comput.*, vol. 06, no. 01, pp. 1083–1092, Oct. 2015, doi: 10.21917/ijsc.2015.0150.

[93] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization: An overview," *Swarm Intell.*, vol. 1, no. 1, pp. 33–57, Oct. 2007, doi: 10.1007/s11721-007-0002-0.

[94] Gregory G. Slabaugh, "Computing Euler angles from a rotation matrix," Queen Mary University, London, UK, 1999. [Online]. Available: https://eecs.qmul.ac.uk/~gslabaugh/publications/euler.pdf

[95] R. Subbarao and P. Meer, "Nonlinear Mean Shift over Riemannian Manifolds," *Int. J. Comput. Vis.*, vol. 84, no. 1, pp. 1–20, Aug. 2009, doi: 10.1007/s11263-008-0195-8.

[96] S. Gwak, Junggon Kim, and F. C. Park, "Numerical optimization on the euclidean group with applications to camera calibration," *IEEE Trans. Robot. Autom.*, vol. 19, no. 1, pp. 65–74, Feb. 2003, doi: 10.1109/TRA.2002.807530.

[97] M. Clerc and J. Kennedy, "The particle swarm - explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002, doi: 10.1109/4235.985692.

[98] X.-S. Yang, "Firefly Algorithms for Multimodal Optimization," no. arXiv:1003.1466. arXiv, Mar. 07, 2010. doi: 10.48550/arXiv.1003.1466.

[99] X.-S. Yang, *Nature-inspired optimization algorithms*, First edition. Amsterdam ; Boston: Elsevier, 2014. [Online]. Available: https://www.sciencedirect.com/book/9780124167438/nature-inspired-optimization-algorithms

[100] W. Chen *et al.*, "An Overview on Visual SLAM: From Tradition to Semantic," *Remote Sens.*, vol. 14, no. 13, Art. no. 13, Jun. 2022, doi: 10.3390/rs14133010.

[101] D. Prokhorov, D. Zhukov, O. Barinova, K. Anton, and A. Vorontsova, “Measuring robustness of Visual SLAM,” Tokyo, Japan, May 2019, pp. 1–6. doi: 10.23919/MVA.2019.8758020.

[102] J. K. Aggarwal and Q. Cai, “Human Motion Analysis: A Review,” *Comput. Vis. Image Underst.*, vol. 73, no. 3, pp. 428–440, Mar. 1999, doi: 10.1006/cviu.1998.0744.

[103] C.-S. Chua, H. Guan, and Y.-K. Ho, “Model-based 3D hand posture estimation from a single 2D image,” *Image Vis. Comput.*, vol. 20, no. 3, pp. 191–202, Mar. 2002, doi: 10.1016/S0262-8856(01)00094-4.

[104] D. Tao, L. Jin, Z. Yang, and X. Li, “Rank Preserving Sparse Learning for Kinect Based Scene Classification,” *IEEE Trans. Cybern.*, vol. 43, no. 5, pp. 1406–1417, Oct. 2013, doi: 10.1109/TCYB.2013.2264285.

[105] D. Tao, J. Cheng, X. Lin, and J. Yu, “Local structure preserving discriminative projections for RGB-D sensor-based scene classification,” *Inf. Sci.*, vol. 320, pp. 383–394, Nov. 2015, doi: 10.1016/j.ins.2015.03.031.

[106] Jungong Han, Ling Shao, Dong Xu, and J. Shotton, “Enhanced Computer Vision With Microsoft Kinect Sensor: A Review,” *IEEE Trans. Cybern.*, vol. 43, no. 5, pp. 1318–1334, Oct. 2013, doi: 10.1109/TCYB.2013.2265378.

[107] Z. Cai, J. Han, L. Liu, and L. Shao, “RGB-D datasets using microsoft kinect or similar sensors: a survey,” *Multimed. Tools Appl.*, vol. 76, no. 3, pp. 4313–4355, Feb. 2017, doi: 10.1007/s11042-016-3374-6.

[108] G. Hu, S. Huang, L. Zhao, A. Alempijevic, and G. Dissanayake, “A robust RGB-D SLAM algorithm,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, Oct. 2012, pp. 1714–1719. doi: 10.1109/IROS.2012.6386103.

[109] T. Lee, S. Lim, S. Lee, S. An, and S. Oh, “Indoor mapping using planes extracted from noisy RGB-D sensors,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura-Algarve, Portugal, Oct. 2012, pp. 1727–1733. doi: 10.1109/IROS.2012.6385909.

[110] I. Oikonomidis, N. Kyriazis, and A. Argyros, “Efficient model-based 3D tracking of hand articulations using Kinect,” in *Procedings of the British Machine Vision Conference 2011*, Dundee, 2011, p. 101.1-101.11. doi: 10.5244/C.25.101.

[111] L. Bo, X. Ren, and D. Fox, “Unsupervised Feature Learning for RGB-D Based Object Recognition,” in *Experimental Robotics*, vol. 88, J. P. Desai, G. Dudek, O. Khatib, and V.

Kumar, Eds. Heidelberg: Springer International Publishing, 2013, pp. 387–402. doi: 10.1007/978-3-319-00065-7_27.

[112] L. Chen, H. Wei, and J. Ferryman, "A survey of human motion analysis using depth imagery," *Pattern Recognit. Lett.*, vol. 34, no. 15, pp. 1995–2006, Nov. 2013, doi: 10.1016/j.patrec.2013.02.006.

[113] J. Geng, "Structured-light 3D surface imaging: a tutorial," *Adv. Opt. Photonics*, vol. 3, no. 2, p. 128, Jun. 2011, doi: 10.1364/AOP.3.000128.

[114] Lionel Heng, "camodocal." Accessed: Jan. 01, 2023. [Online]. Available: https://github.com/hengli/camodocal

[115] L. Heng, Bo Li, and M. Pollefeys, "CamOdoCal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Nov. 2013, pp. 1793–1800. doi: 10.1109/IROS.2013.6696592.

[116] G. R. Bradski and A. Kaehler, *Learning OpenCV: computer vision with the OpenCV library*, 1. ed., [Nachdr.]. Beijing: O'Reilly, 2011.

[117] Christian Kühling, "Fisheye Camera System Calibration for Automotive Applications," Masterarbeit am Institut für Informatik der Freien Universität Berlin, Dahlem Center for Machine Learning and Robotics, Berlin, Germany, 2017. [Online]. Available: https://www.mi.fu-berlin.de/inf/groups/ag-ki/Theses/Completed-theses/Master_Diploma-theses/2017/Kuehling/Master-Kuehling.pdf

[118] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab." [object Object], May 04, 2022. doi: 10.22002/D1.20164.

[119] D. Scaramuzza, A. Martinelli, and R. Siegwart, "A Flexible Technique for Accurate Omnidirectional Camera Calibration and Structure from Motion," in *Fourth IEEE International Conference on Computer Vision Systems (ICVS'06)*, New York, NY, USA, 2006, pp. 45–45. doi: 10.1109/ICVS.2006.3.

[120] S. Djema, Z. A. Benselama, R. Hedjar, and A. Krabi, "Dense Visual Odometry Using Genetic Algorithm," *Int. J. Intell. Syst. Appl. Eng.*, vol. 11, no. 3, pp. 611–619, Jul. 2023, doi: 10.48550/arXiv.2311.06149.

[121] Slimane Djema, Zoubir Abdeslem Benselama, Ramdane Hedjar, and Abdellah krabi, "Firefly Algorithm Based Visual Odometry," University of 20 August 1955 Skikda, Algeria, May 2023.