الجمهورية الجزائرية الديمقراطية الشعبية

Democratic and Popular Algerian Republic

وزارة التعليم العالي و البحث العلمي

Ministry of Higher Education and Scientific Research

جامعة سعد دحلب البليدة

Saad Dahlab University of BLIDA

كلية التكنولوجيا

Faculty of Technology

قسم الإلكترونيك

Department of Electronics

Master's Thesis

Specialization: Electronics of Embedded Systems

Presented by

Yacine Belalia

Autonomous Navigation of a Robotic Wheelchair in an Indoor Environment

---

Jury Members

President: Mr. Amar Bounemri Examiner: Mr. Yacine Kabir

Supervisor: Mme. Sara Bouraine Co-Supervisor: Mme. Djamila Naceur

Academic Year 2022-2023

Acknowledgments

I would like to express my heartfelt gratitude to my esteemed supervisor, Mme. Sara Bouraine, for her unwavering trust, enthusiasm, and leadership throughout this adventurous research journey. Her unyielding support and engagement, combined with her extensive experience and knowledge, have been pivotal in shaping this project. Her high standards have constantly pushed me to improve, and for that, I am truly grateful.

Special thanks are extended to Mme. Djamila Naceur for her honesty, valuable advice, guidance, and insightful remarks, which have made a significant contribution to the production of this thesis.

I am deeply appreciative of the jury members for their expertise and the dedicated effort they put into evaluating this thesis. Their advice, mentorship, and feedback have been invaluable. I would also like to extend my gratitude to all the teachers in the department who collectively made my experience at Saad Dahlab University an unforgettable milestone in my academic journey. I wish them all the best.

I express my sincere appreciation to my older brother, Abderraouf, for his invaluable insights and thought-provoking ideas. Our exchanges of perspectives on the various challenges encountered during this endeavor have been immensely enlightening.

To my parents and my family as a whole, I extend my profound thanks for their consistent support and understanding throughout the course of my research and thesis writing.

I would also like to express my gratitude to my friends for their valuable comments, as well as to the staff at CDTA for providing a warm and welcoming environment that fostered an atmosphere conducive to learning and growth.

Finally, I offer my gratitude to God for granting me the strength to overcome the challenges that arose during this academic pursuit and enabling me to complete my degree. I humbly seek continued guidance and blessings for my future endeavors.

Abstract

Autonomous Mobile Robots (AMR) are robotic systems capable of navigating in environments without human intervention. Their growing popularity and practical applications have led to a rapid expansion, driven by increasing interest and research. However, a major challenge faced by these systems is the generation and execution of movements required for efficient trajectory planning, which remains a persistent problem in autonomous systems. In this study, our objective is to contribute to the field of motion planning by introducing two new variants of the Rapidly-exploring Random Tree Star (RRT*) algorithm that integrate the Whale Optimization Algorithm (WOA) to generate near-optimal trajectories. To validate the proposed variants, we implemented them in a simulation environment. Then, we explored the parameter space of WOA for both variants in order to identify optimal parameters and deepen our understanding of behavior with different configurations. The results obtained from the two variants demonstrate significant improvements in trajectory quality, surpassing the performance of the original RRT* algorithm. These promising results highlight the untapped potential of using this optimization technique and also pave the way for further research to explore and exploit the benefits of parallelization aiming to enhance the efficiency and effectiveness of these variants.

Keywords: Autonomous Mobile Robots, Motion Planning, RRT*, Optimization Technique, Whale Optimization Algorithm

Résumé

Les robots mobiles autonomes (AMR, pour Autonomous Mobile Robots en anglais) sont des systèmes robotiques capables de naviguer dans des environnements sans intervention humaine. Leur popularité croissante et leurs applications concrètes ont entraîné une expansion rapide, propulsée par un intérêt et une recherche croissants. Cependant, un défi majeur auquel ces systèmes sont confrontés est la génération et l'exécution des mouvements nécessaires pour une planification de trajectoire efficace, ce qui reste un problème persistant dans les systèmes autonomes. Dans cette étude, notre objectif est de contribuer au domaine de la planification de mouvement en introduisant deux nouvelles variantes de l'algorithme Rapidly-exploring Random Tree Star (RRT*) qui intègrent l'algorithme d'optimisation des baleines (WOA) pour générer des trajectoires quasi-optimales. Pour valider les variantes proposées, nous les avons implémentées dans un environnement de simulation. Ensuite, nous avons exploré l'espace des paramètres de WOA pour les deux variantes, dans le but d'identifier les paramètres optimaux et d'approfondir notre compréhension du comportement avec différentes configurations. Les résultats obtenus à partir des deux variantes démontrent des améliorations significatives de la qualité de la trajectoire, surpassant les performances de l'algorithme RRT* d'origine. Ces résultats prometteurs mettent en évidence le potentiel inexploité de l'utilisation de cette technique d'optimisation et ouvrent également la voie à de nouvelles recherches pour explorer et exploiter les avantages de la parallélisation visant à améliorer l'efficacité et l'efficience de ces variantes.

Mots-clés: Robots Mobiles Autonomes, Planification de Mouvement, RRT*, Technique d'Optimisation, WOA

ملخص

الروبوتات المتنقلة الذاتية (AMR) هي أنظمة روبوتية تستطيع التنقل في البيئات دون تدخل بشري. ارتفاع شعبيتها وتطبيقاتها العملية أدى إلى توسع سريع، مدفوعًا بالاهتمام والبحث المتزايد. ومع ذلك، تواجه هذه الأنظمة تحديًا كبيرًا في إنشاء وتنفيذ الحركات المطلوبة لتخطيط المسار الفعال، وهذه مشكلة مستمرة في الأنظمة الذاتية. في هذه الدراسة، هدفنا هو المساهمة في مجال تخطيط الحركة من خلال تقديم اثنين من الإصدارات الجديدة لخوارزمية Rapidly-exploring Random Tree Star (RRT*) التي تدمج خوارزمية تحسين الحوت (WOA) لإنشاء مسارات تقترب من الأمثل. لتحقيق هذا الهدف، قمنا بتنفيذ الإصدارات المقترحة في بيئة محاكاة. ثم، استكشفنا مجال العوامل لخوارزمية تحسين الحوت لكلا الإصدارين لتحديد العوامل المثلى وتعميق فهمنا للسلوك مع اعدادات مختلفة. أظهرت النتائج المستخلصة من الإصدارين تحسينات كبيرة في جودة المسارات، وتفوقاً على أداء خوارزمية RRT* الأصلية. تسلط هذه النتائج الواعدة الضوء على الإمكانات غير المستغلة لاستخدام هذه التقنية التحسينية وتمهد الطريق أيضًا لمزيد من البحث لاستكشاف واستغلال فوائد التوازي مع التركيز على تعزيز كفاءة وفعالية هذه الإصدارات.

الكلمات الرئيسية: الروبوتات المتنقلة الذاتية، تخطيط الحركة، *RRT، تقنية التحسين، WOA

Acronyms and Abbreviations

AMRs	Autonomous Mobile Robots
PRM	Probabilistic Road Map
RRT	Rapidly-exploring Random Tree
EA	Evolutionary Algorithm
GA	Genetic Algorithm
PBIL	Population-Based Incremental Learning
GP	Genetic Programming
DE	Differential Evolution
GSA	Gravitational Search System
CSS	Charged System Search
CFO	Central Force Optimization
BBBC	Big-Bang Big-Crunch
SI	Swarm Intelligence
PSO	Particle Swarm Optimization
ACO	Ant Colony Optimization
AS	Ant System
ACS	Ant Colony System
FA	Firefly Algorithm
WOA	Whale Optimization Algorithm
SEOA	Social Emotional Optimization Algorithm
ICA	Imperialist Competitive Algorithm
TLBO	Teaching Learning Based Optimization
SLC	Soccer League Competition

ROS	Robot Operating System
OpenCV	Open Computer Vision
JSON	JavaScript Object Notation
GIMP	GNU Image Manipulation Program
SLAM	Simultaneous Localization And Mapping

Table of Contents

List of Figures

List of Tables

General Introduction

Autonomous mobile robots (AMRs) are robotic systems capable of operating and navigating in various environments without human intervention. With the help of sensors, computing capabilities, and decision-making algorithms, AMRs can autonomously perceive their surroundings, plan actions, and carry out tasks. AMRs have significant potential to enhance medical care and personal assistance for individuals with impaired mobility. In the medical field, they can automate tasks such as transporting supplies and patients, which allow healthcare professionals to dedicate more time to direct patient care. Equipped with sensors, AMRs can monitor vital signs and collect health data, facilitating timely decision-making. As personal assistants, AMRs can assist with daily tasks and improve social interactions, enabling individuals with disabilities to lead more independent and fulfilling lives [1].

The effectiveness of AMRs heavily relies on the integration of motion planning and optimization techniques, especially when navigating complex and dynamic environments. Motion planning addresses the challenge of generating efficient and feasible trajectories for AMRs, ensuring their movement from an initial state to a desired goal state while avoiding obstacles and adhering to constraints. Researchers have made significant progress in enabling AMRs to plan and execute precise and efficient motions by integrating algorithms, mathematical models, and optimization techniques.

Initially, traditional methods [5,6,7,8] faced limitations when dealing with complex scenarios, high-dimensional spaces, and dynamic obstacles. To overcome these challenges, optimization techniques [17,21,26] have been seamlessly incorporated into motion planning algorithms, effectively transforming the process into an optimization problem. Our project aims to contribute to the research field by providing a novel bio-inspired motion planning algorithm with two variants inspired by a swarm-based optimization technique.

The local variant incorporates the whale optimization algorithm (WOA) into the expansion process of RRT* to enhance the exploration of space towards promising

areas. On the other hand, the global variant only uses the path returned by RRT* as a population initializer and generates optimized paths using WOA. This document is organized as follows: Chapter 1 presents a literature review on motion planning and optimization techniques, providing a detailed analysis justifying our chosen approach. Chapter 2 explains RRT* and the whale optimization algorithm in detail, along with an explanation of our proposed variants. In Chapter 3, we clarify the approach used to implement these variants. We test the variants in simulation setups and benchmark them against the original algorithm. Additionally, we conduct a thorough parameter tuning process to analyze the behaviors of the two variants and identify optimal settings. The thesis concludes with our findings and aspirations for future endeavors in this domain.

Chapter 1

Motion Planning and Trajectory Optimization: State of the Art

1.1 Introduction

Motion planning and optimization techniques are crucial in robotics to generate feasible and optimal trajectories in complex and dynamic environments. By combining algorithms, mathematical models, and optimization principles, researchers have made significant progress in enabling precise and efficient motion execution. Motion planning involves finding actions or configurations to navigate from an initial state to a goal state while avoiding obstacles and adhering to constraints.

To overcome challenges in dealing with high-dimensional spaces and dynamic obstacles, optimization techniques have been integrated into motion planning algorithms. This approach formulates motion planning as an optimization problem, finding the best trajectory that satisfies constraints and optimization criteria. Mathematical optimization allows consideration of multiple objectives, ensuring safety and collision avoidance. By integrating motion planning with optimization techniques, collision-free trajectories can be generated while optimizing factors like distance traveled or task completion time. This leads to intelligent and efficient motion planners that adapt to dynamic environments, optimize multiple objectives simultaneously, and enhance autonomous system capabilities.

In the following sections, we will explore the primary categories of motion planning algorithms, analyzing their strengths, limitations, and selecting an appropriate algorithm for our application scenario. We will also discuss metaheuristic optimization algorithms, their operational mechanisms, and integrate a promising candidate into our chosen path planning method.

1.2 Motion Planning

Motion planning in robotics involves the generation of a collision-free motion from an initial to a goal position in a specified environment. However, this problem is known to be challenging to solve efficiently as it grows in difficulty with the complexity of the environment, the increase in degrees of freedom of the system, and the constraints imposed on it, such as kinematics constraints, dynamic constraints, sensor limitations, task-specific constraints, and real-time constraints [2]. Depending on the environment,

numerous motions may be possible in the space where the robot is able to move. Motion planning algorithms aim to find the best motion or at least an admissible approximation to it. The best motion here refers to the optimal one, in the sense that the resulting motion is obtained by minimizing the cost, time, or energy consumption through one or more objective optimization functions. Additionally, it's crucial to consider the aforementioned constraints to ensure that the resulting motion is feasible, safe, and efficient for the robot's specific task and environment. Figure 1.1 illustrates the planned trajectory, starting from the robot's initial position and leading to the goal, while effectively circumventing potential collision sources present in the environment.

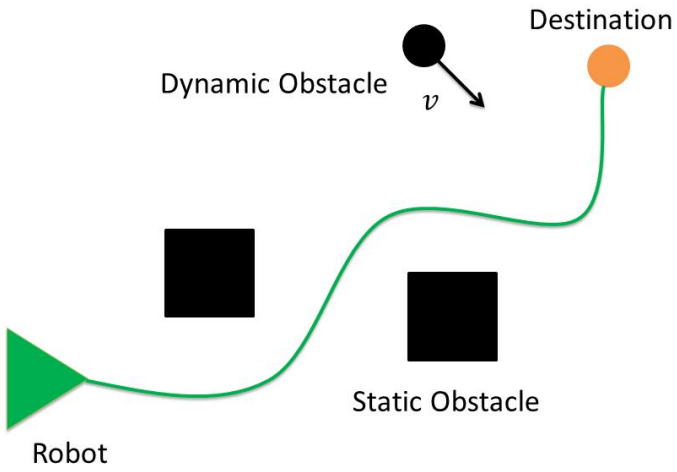


Figure 1.1 – Planned Path from the Robot's Initial to Goal Position with Collision Avoidance

Motion planning has been a fundamental topic in robotics for several decades. Early work in motion planning was focused on developing algorithms to plan motions for simple robots with limited degrees of freedom. In the 1980s, researchers began developing more sophisticated algorithms to handle more complex robot models and environments, and these algorithms have continued to evolve to this day [3]. One key development in motion planning was the introduction of probabilistic roadmap (PRM) algorithms in the 1990s. Other important developments include the use of visibility graphs, potential fields, and rapidly exploring random trees (RRTs). Today, motion planning continues to be an active area of research, with new algorithms and

techniques being developed to handle increasingly complex scenarios and real-time constraints.

Deterministic algorithms are one of the oldest and most traditional methods for motion planning. These algorithms are based on solving a set of equations or constraints to determine the optimal motion for a robot. Deterministic algorithms can be very accurate and efficient for simple scenarios. However, they often fail to produce feasible solutions for complex scenarios due to their inability to handle uncertain or noisy environments. In contrast, sampling-based algorithms are a more recent approach that uses random sampling to generate a set of potential motions. The advantage of sampling-based algorithms is their ability to handle complex and uncertain environments, making them suitable for real-world applications. However, sampling-based algorithms may require a significant amount of computational resources to generate a solution, and the solution may not always be optimal. Each approach has its own set of advantages and disadvantages, and the selection of the algorithm depends on the specific problem domain and implementation requirements [4]. Figure 1.2 below classifies motion planning algorithms into two families: deterministic and sampling-based algorithms.

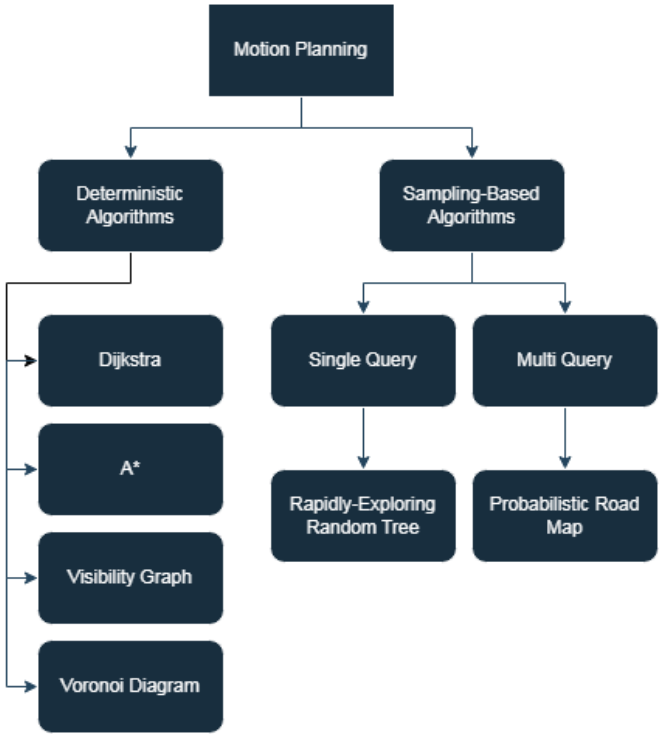


Figure 1.2 – Classification of Motion Planning Algorithms

1.2.1 Preliminaries

Before exploring the specifics of motion planning algorithms in the literature, it is important to establish some foundational terminology

Workspace. Refers to the physical environment in which a robot operates, encompassing all obstacles, boundaries, and other relevant features that may affect the robot's movement. The workspace is commonly represented by a geometric model, which can be either two-dimensional or three-dimensional such as figure 1.3, depending on the problem domain.

Configuration. Represents the complete set of parameters that describe the state of a robot within its workspace. It includes attributes such as position, orientation, and any additional degrees of freedom that are relevant to the task at hand.

Configuration Space. Also known as C-space, serves as a mathematical representation of all possible configurations that a robot can assume within its workspace. This high-dimensional space captures the robot's state in its environment, accounting for its degrees of freedom. The configuration space is instrumental in defining the feasible and obstacle-free regions through which the robot can navigate.

Initial configuration. Denotes the starting position and orientation of the robot within its workspace. It is determined by the values of the robot's degrees of freedom in the configuration space and serves as the initial state from which a motion planning algorithm can compute a path to the goal configuration.

Goal configuration. Represents the desired position and orientation of the robot within its workspace upon completion of a task. It serves as the final state to which the motion planning algorithm endeavors to guide the robot by planning a path from the initial configuration.

Heuristic. Refers to a function or technique used to estimate the cost or distance between a robot's current state and the goal configuration. Heuristics offer a quick and efficient means of evaluating different paths or actions to determine the most promising ones. The selection of an appropriate heuristic can significantly enhance the efficiency and effectiveness of the motion planning algorithm.

Optimality. Refers to the discovery of the path with the lowest cost or shortest distance between the start and goal configurations while avoiding collisions with obstacles.

Completeness. Pertains to the capability of a motion planning algorithm to find a solution if one exists. A complete algorithm guarantees the discovery of a feasible path between the start and goal configurations given sufficient time and resources.

Efficiency. Denotes the ability of a motion planning algorithm to generate a solution that is either close to optimal or optimal while minimizing computational resources and time required for computation.

Non-holonomy. Refers to the limitations on movement and turning options caused by mechanical constraints or design factors in mobile robots.

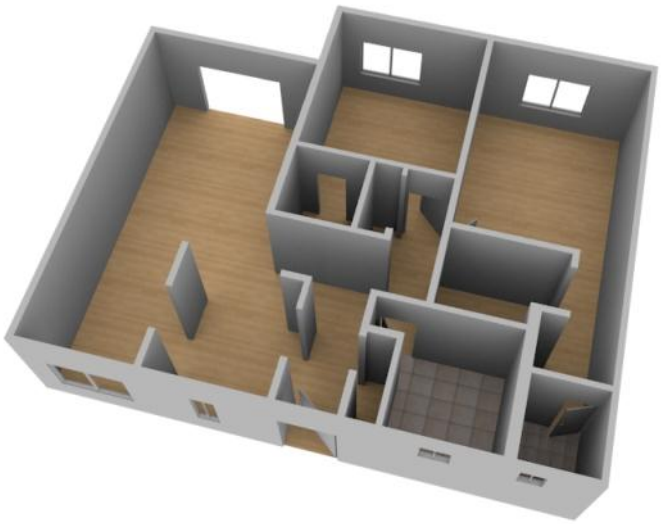


Figure 1.3 – Example of a Workspace

In the following subsections, we will delve into the most prominent deterministic and sampling-based algorithms in motion planning, outlining their distinctive characteristics.

1.2.2 Deterministic Algorithms

Deterministic algorithms are a class of algorithms used in motion planning that consistently produce the same output for a given input each time they are executed. These algorithms find widespread application in various fields, particularly robotics, where precise and repeatable outcomes are critical. In motion planning, deterministic algorithms ensure that the generated output is consistent, offering a reliable and dependable path for robots, vehicles, or other entities to follow while prioritizing safety and efficiency. Examples of such algorithms include Dijkstra, A*, Visibility Graph, and Voronoi Diagram.

A. Dijkstra's Algorithm

Dijkstra's algorithm is a pathfinding algorithm invented by Dutch computer scientist Edsger W. Dijkstra in 1956. It was originally designed to find the shortest path between two nodes in a graph with non-negative edge weights [5].

The algorithm works by maintaining a set of visited nodes and a set of unvisited nodes. It begins by setting the distance of the starting node to 0 and the distances of all other nodes to infinity. At each step of the algorithm, the unvisited node with the lowest tentative distance is selected, and its neighbors are examined. For each neighbor, if the distance to that neighbor through the current node is less than the neighbor's current tentative distance, the neighbor's distance is updated to the new lower value. This process is repeated until the destination node is reached or all reachable nodes have been visited [5].

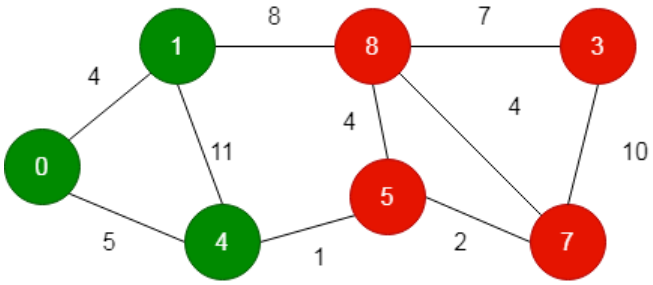


Figure 1.4 – Dijkstra's Pathfinding Graph: Visualization of Visited and Unvisited Nodes

Dijkstra's algorithm is a widely-used path planning approach that is simple to understand. However, it can be slow and memory-intensive for large graphs and is not suitable for graphs with negative edge weights. It also does not consider other environmental factors such as obstacles or dynamic changes to the environment that may affect pathfinding. Despite these limitations, it remains a valuable tool in path planning and related applications.

B. A* Algorithm

The A* algorithm is a pathfinding algorithm that was first proposed in 1968 by Peter Hart, Nils Nilsson, and Bertram Raphael of Stanford Research Institute. The algorithm builds upon Dijkstra's algorithm by incorporating a heuristic function that estimates the distance from each node to the goal, allowing it to search more efficiently and find the shortest path faster [6].

It works by maintaining two lists of nodes: an open list of nodes to be evaluated, and a closed list of nodes that have already been evaluated. The algorithm begins at the start node and adds it to the open list. Then, it selects the node with the lowest estimated cost to the goal (based on a heuristic function) and adds its neighbors to the open list. After that, as the algorithm explores the nodes, it evaluates and moves them from the open list to the closed list once they have been considered [6].

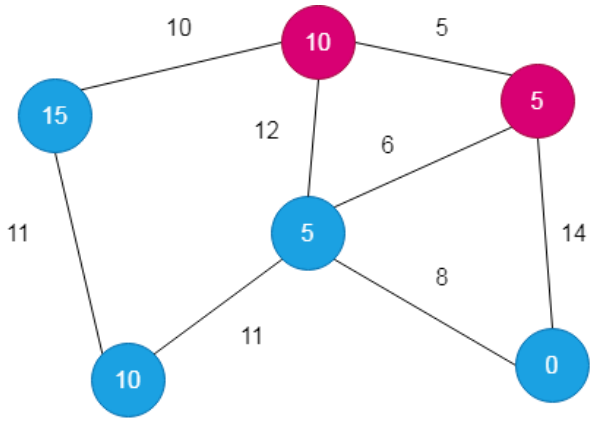


Figure 1.5 – A* Pathfinding Graph: Visualization of Open and Closed Nodes

A* algorithm has advantages such as its efficiency in large graphs, ability to find optimal paths with a well-designed heuristic function, and adaptability to various terrain and cost functions. However, A* may not always find the optimal path if the

heuristic function is not admissible, can be slower if the heuristic function is not well-designed, requires more memory compared to Dijkstra's algorithm.

C. Visibility Graph

The visibility graph algorithm was introduced by Lozano-Pérez, Tomás, and Michael A. Wesley in their 1979 paper titled "An algorithm for planning collision-free paths among polyhedral obstacles". The algorithm was originally designed for robot path planning in terrain with obstacles, where the robot's visibility was restricted by the terrain [7].

The visibility graph algorithm works by representing obstacles as vertices in a graph, and creating edges between vertices that are visible to each other. The start and goal points are added as vertices, and edges are added between them and any visible vertices. Then, a search algorithm is used to find the shortest path between the start and goal points in the graph [7].

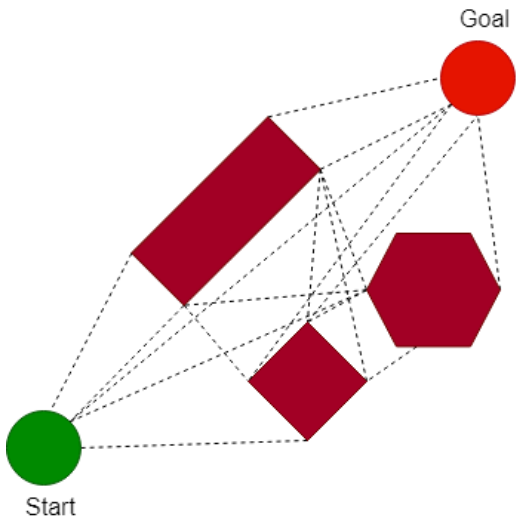


Figure 1.6 – Visibility Graph: The Dashed Lines Represent Candidate Paths and the Polygons Represent Obstacles.

The visibility graph algorithm offers advantages such as simplicity, efficiency, and optimality when solving path planning problems. However, it is computationally expensive for complex environments and does not account for non-holonomic constraints or dynamic/unknown environments.

D. Voronoi Diagram

The Voronoi diagram was first introduced by Georgy Voronoy in 1908, a Russian mathematician, who was interested in the theory of numbers and algebraic topology. In the 1980s, robotics researchers realized that Voronoi diagrams could be used for path planning in a range of applications [8]. Since then, many researchers have worked on improving the efficiency and accuracy of Voronoi diagram-based path planning algorithms.

The Voronoi diagram algorithm works by dividing the environment into regions based on the distance to the obstacles. Specifically, it creates a graph where the nodes represent the obstacles, and the edges are the lines of equal distance between adjacent obstacles. These lines form the boundaries of the regions, known as Voronoi cells. The Voronoi diagram algorithm then uses these cells to plan a path by connecting the start and goal locations to the cell boundaries and finding the path that passes through the fewest number of cells.

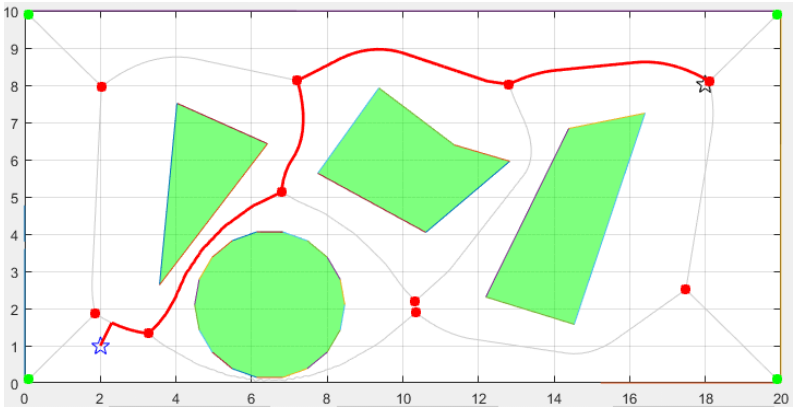


Figure 1.7 – Voronoi Diagram: Generated Path Through Cells to the Goal

Advantages of the Voronoi diagram algorithm in path planning include its ability to efficiently calculate paths for multiple agents and handle complex environments with a large number of obstacles. It also considers the size and shape of the agent, resulting in more feasible paths. However, the Voronoi diagram algorithm may not always find the shortest path and can be sensitive to the location of obstacles. It also assumes that the agent moves at a constant speed and does not take into account dynamic environments or other non-holonomic constraints.

1.2.3 Sampling Based Algorithms

Sampling-based algorithms are a class of motion planning algorithms that are extensively employed in robotics applications to discover a feasible path for a robot from its initial position to its desired goal position. These algorithms operate on the principle of randomly sampling the configuration space of the robot and constructing a graph or tree structure that captures the connectivity among the sampled configurations. The objective is to identify a path that connects the initial and goal configurations while effectively circumventing obstacles present within the environment.

A. Single Query

Single query sampling-based algorithms use random sampling to construct a graph and search for a path through it for a single start and goal pair. These algorithms are designed to work in continuous, high-dimensional state spaces where it is difficult to construct a deterministic path from a start to a goal location.

Rapidly-exploring Random Tree

The Rapidly-exploring Random Tree (RRT) algorithm was first introduced by Steven LaValle in 1998 [9]. The algorithm was designed to efficiently plan paths in high-dimensional configuration spaces, where traditional path planning algorithms struggle due to the problem of exponential growth of computational complexity with increasing dimensionality.

The algorithm works by constructing a tree-like structure through the exploration of the configuration space. The algorithm starts with an initial configuration and then iteratively grows the tree by randomly selecting a new configuration and attempting to connect it to the existing tree. The algorithm continues to do this until either a specified number of iterations has been reached or a feasible path from the start to the goal configuration has been found [9].

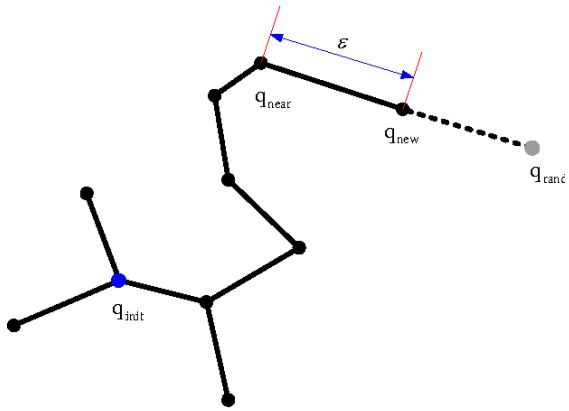


Figure 1.8 – RRT Tree Extension Process

The Rapidly-exploring Random Tree (RRT) algorithm offers several advantages for path planning in complex and high-dimensional environments, including the ability to handle non-holonomic constraints and changing environments. However, it may not always generate an optimal path and can struggle in environments with narrow passages or difficult-to-navigate areas.

B. Multi Query

Multi query sampling-based algorithms construct a graph that can be used to search for multiple start and goal pairs efficiently. These algorithms are particularly useful for planning paths for groups of robots, where multiple start and goal configurations need to be considered simultaneously.

Probabilistic Road Map

The Probabilistic Road Map (PRM) algorithm was introduced by Kavraki, Lydia E., Petr Svestka, J-C. Latombe, and Mark H. Overmars in 1996 as a probabilistic method for robot path planning. It was designed to efficiently plan paths in high-dimensional spaces with complex and changing environments [10].

The algorithm works by building a graph of nodes and edges that represent the configuration space of the environment. The nodes are generated by randomly sampling the configuration space, while the edges are formed by connecting the nodes that are within a certain distance of each other and that can be connected without colliding with any obstacles. Once the graph has been constructed, a search algorithm

such as Dijkstra's algorithm or A* is used to find a path from the start node to the goal node.

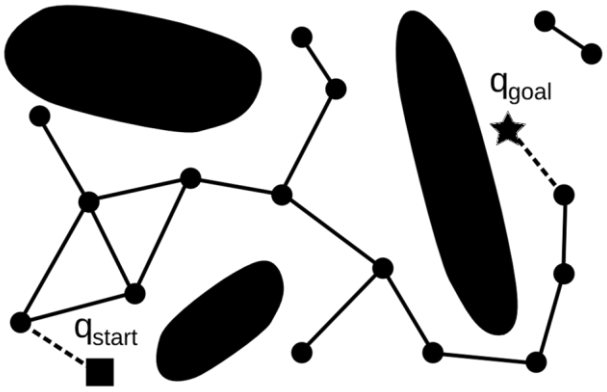


Figure 1.9 – Probabilistic Road Map: Graph Connecting Initial and Goal Points

The Probabilistic Road Map (PRM) algorithm in path planning has several advantages including its ability to handle complex and changing environments, generate paths that are moderately optimal, and work well in narrow passages. The computational complexity can also be lower than some other sampling-based algorithms. However, the algorithm may require a large number of samples to achieve good performance in certain scenarios, struggle with non-holonomic constraints, and the generated paths may not always be feasible.

1.2.4 Comparison

In this subsection, we delve deeper into the characteristics of the six previously cited algorithms, considering various factors such as completeness, optimality, memory usage, handling of dynamic environments, and the ability to handle non-holonomic constraints.

Dijkstra's algorithm is widely recognized for its completeness, guaranteeing that it will always find a path if one exists. It also offers optimality, ensuring that the path found is the shortest [11]. However, Dijkstra's algorithm has moderate memory usage, which can become a limitation in larger environments where memory resources are constrained. Furthermore, Dijkstra's algorithm does not consider dynamic environments, meaning it cannot adapt its path planning in real-time to changes in the

environment. It also does not handle non-holonomic constraints, such as limitations on the turning radius or differential drive constraints for certain robotic systems.

The A* algorithm, like Dijkstra's algorithm, is both complete and optimal, ensuring that it will always find the shortest path [12]. However, the A* algorithm typically exhibits higher memory usage compared to Dijkstra's algorithm, as it employs heuristics to guide the search process. This higher memory requirement can pose challenges in resource-constrained scenarios. On the positive side, the A* algorithm excels in handling dynamic environments due to its ability to incorporate heuristics that estimate the remaining cost to the goal. By dynamically updating these heuristics, the A* algorithm can adapt its path planning to changing circumstances. Nonetheless, similar to Dijkstra's algorithm, A* does not inherently handle non-holonomic constraints.

The Visibility Graph algorithm shares similarities with Dijkstra and A* algorithms in terms of completeness and optimality. It guarantees finding a path if one exists and ensures it is the shortest [13]. However, the Visibility Graph algorithm has high memory usage, which can be a drawback in memory-limited environments. Similar to Dijkstra and A*, it does not consider dynamic environments or non-holonomic constraints. As a result, the Visibility Graph algorithm is often more suitable for static environments where the map remains unchanged during the path planning process.

In contrast to the previous algorithms, the Voronoi Diagram approach is not complete or optimal. It cannot guarantee finding a solution even if one exists, and the path it generates may not be the shortest. Moreover, unlike the A* and Visibility Graph algorithms, the Voronoi Diagram algorithm typically exhibits low memory usage [14]. Like the previous algorithms, it does not account for dynamic environments or non-holonomic constraints. However, Voronoi Diagrams offer advantages in situations where an approximate solution is sufficient, such as scenarios where real-time path planning is not required, or where optimality is not the primary concern.

The RRT (Rapidly-exploring Random Tree) algorithm, while complete, is not optimal. It offers a trade-off between optimality and computational efficiency. One of its advantages is its low memory usage, which makes it suitable for resource-constrained

systems. RRT is particularly well-suited for handling non-holonomic constraints, as it can explore the configuration space of non-holonomic vehicles efficiently. Furthermore, RRT can be adapted to handle dynamic environments by incorporating techniques such as incremental planning or replanning. By periodically updating the tree structure and adapting the paths based on changes in the environment, RRT can address dynamic scenarios to some extent.

PRM (Probabilistic Roadmap) algorithm, similar to RRT, is complete but not optimal. It also boasts low memory usage, which is advantageous in memory-limited environments. PRM can handle dynamic environments; however, it struggles with non-holonomic constraints.

Algorithm	Completeness	Optimality	Memory Usage	Handling Dynamic Environment	Handling Non-holonomic Constraints
Dijkstra	Yes	Yes	Moderate	No	No
A*	Yes	Yes	High	Yes	No
Visibility Graph	Yes	Yes	High	No	No
Voronoi Diagram	No	No	Low	No	No
RRT	Yes	No	Low	Yes	Yes
PRM	Yes	Yes	Low	Yes	No

Table 1.1 – Comparison of Motion Planning Algorithms

1.2.5 Discussion

In the previous section, we compared various path planning algorithms. Based on our analysis, we have determined that certain criteria are crucial for our study: completeness, handling dynamic environments, and addressing the non-holonomic constraints imposed by the limited movement of the wheelchair. Considering these factors, we have selected the Rapidly-exploring Random Tree (RRT) as the most suitable path planning algorithm. While RRT may not guarantee optimality, we can overcome this limitation by exploring its variants in the following chapter.

To further support our choice for RRT, we would like to highlight a few reasons:

Probabilistic Completeness: RRT is probabilistically complete, meaning that given enough time, it is guaranteed to find a solution if one exists. This property ensures that the robotic wheelchair can always find a feasible path in the environment, providing reliable navigation capabilities.

Adaptability: RRT can be easily extended and modified to incorporate additional constraints and optimize the generated paths through post-processing techniques. This adaptability allows us to enhance the quality of the paths according to specific requirements.

Exploration and Coverage: The inherent randomness and exploratory nature of RRT make it well-suited for tasks involving exploration and coverage of unknown environments. For example, if the robotic wheelchair needs to navigate through a new environment or perform tasks like mapping or inspecting an area, RRT can efficiently explore the space and generate paths that cover the entire region of interest.

Dynamic Environment Handling: RRT is able to handle dynamic environments where obstacles can move or appear/disappear over time. By continuously updating the tree structure, RRT can adapt to changes in the environment and generate new paths on the fly, ensuring safe and efficient navigation of the wheelchair.

Potential for Real-World Deployment: The efficiency, adaptability, and extensive research on RRT make it a promising choice for the real-world deployment. Its

practicality and proven effectiveness in various applications strengthen its suitability for our study.

By considering these factors, we are confident in our decision to utilize the RRT algorithm as the path planning approach for our research.

1.3 Optimization Techniques

Optimization techniques in motion planning are mathematical and computational tools used to find the best trajectory that satisfies specific criteria, such as the shortest or safest path, among all possible paths from a starting configuration to a goal configuration. These techniques aim to optimize the complete trajectory or assist in creating an effective path towards the goal.

In motion planning, optimization techniques can be broadly categorized into approximate and exact methods. Approximate methods focus on finding a near-optimal solution to the motion planning problem, while exact methods aim to find the optimal solution. Approximate methods can be further divided into heuristic and metaheuristic methods.

Heuristic methods employ simple rules or heuristics to guide the search for a good trajectory. These methods are fast and easy to implement but do not guarantee the optimality of the found path. Heuristic methods are particularly useful when finding the exact solution is impractical or when a fast solution is required. Examples of heuristic methods include the Greedy algorithm, Simulated Annealing, and Tabu Search.

On the other hand, metaheuristic methods are more powerful than heuristic methods and are capable of finding better solutions to motion planning problems. These methods utilize stochastic techniques inspired by natural phenomena or biological systems to explore the search space for the best trajectory. While metaheuristic methods do not guarantee optimality, they are able to find good solutions within a reasonable amount of time. Examples of metaheuristic methods used in motion planning include the Genetic Algorithm, Particle Swarm Optimization, and Ant Colony Optimization.

1.3.1 Metaheuristic Approaches

The term “metaheuristic” was coined by Fred Glover in 1986 to describe a non-problem-specific heuristic method. Metaheuristics combine exploration (diversification) and exploitation (intensification) to form a robust searching mechanism. Exploitation involves searching in the vicinity of the best solution, while exploration involves exploring new search areas [15].

In optimization and problem-solving, a metaheuristic approach refers to a general strategy or framework that can be applied to various problems without relying on specific problem knowledge. These iterative, heuristic-based algorithms intelligently explore the solution space to search for near-optimal solutions. They are particularly useful when traditional optimization techniques are impractical or inefficient due to complex, large-scale, or multi-objective problems.

Metaheuristics are algorithmic structures that can be easily adapted to different optimization problems with minimal modifications. They possess fundamental characteristics such as applicability to multiple problems, approximate nature, exploration of the search space to find “good enough” solutions, and straightforward parallel implementation. Metaheuristics encompass a range of techniques, from basic local search methods to advanced learning techniques, incorporating mechanisms to prevent premature convergence.

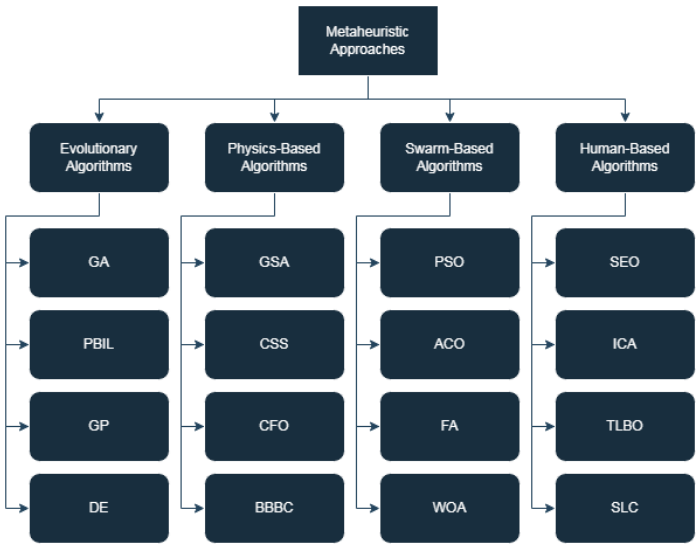


Figure 1.10 – Classification of Metaheuristic Approaches

A. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are population-based, fitness-oriented, and variation-driven algorithms inspired by natural evolution. They mimic the process of natural selection and adaptation to solve optimization and learning problems. Eas maintain a population of potential solutions, where each individual has its own genetic representation and fitness value. By introducing variations through genetic operators, Eas explore the solution space to find better solutions. Eas have been proposed since the 1960s and have been used in various domains. They are distinguished from related concepts such as soft computing and computational intelligence. The number of research papers on Eas has been increasing over time, indicating ongoing interest and development in the field. [16]

Genetic Algorithm

John Henry Holland proposed the genetic algorithm for optimization and problem solving in his 1975 book “Adaptation in Natural and Artificial Systems”. He introduced genetic algorithms as a method to solve complex problems [17]. This marked the foundation of applying evolutionary computation techniques to optimization and search problems. Today, genetic algorithms are widely utilized across diverse fields to find optimal solutions.

The Genetic Algorithm (GA) is an optimization technique inspired by natural selection. It begins with a population of potential solutions, called chromosomes, representing different possible solutions to the problem. The GA evaluates the fitness of each chromosome and selects two parents based on their fitness. Through crossover, genetic information is exchanged between parents, generating offspring with combined traits. Mutation introduces random changes, exploring new solutions. Offspring, including mutated individuals, form the next generation. This process repeats over multiple generations until a satisfactory solution is found or a termination criterion is met. The GA explores diverse solutions, converging towards an optimal or near-optimal solution by mimicking natural selection’s principles. [18]

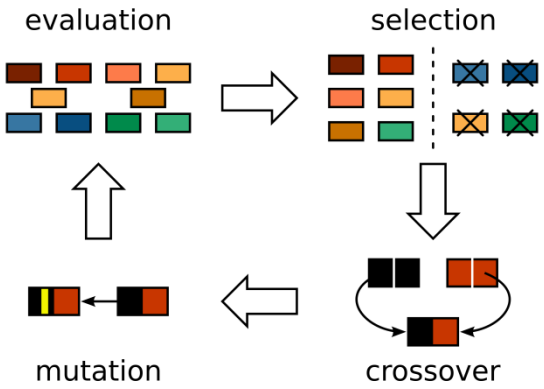


Figure 1.11 – Genetic Algorithm Cycle: Evaluation, Selection, Crossover, and Mutation Phases

Population-Based Incremental Learning

Shumeet Baluja introduced the Population-Based Incremental Learning (PBIL) method in 1994, aiming to integrate genetic search-based function optimization and competitive learning. PBIL combines the advantages of genetic algorithms with simple competitive learning, offering a more efficient and straightforward approach compared to traditional genetic algorithms [19].

Population-Based Incremental Learning (PBIL) is an optimization algorithm that addresses the limitations of Genetic Algorithms (Gas) in solving deceptive problems. PBIL maintains a population of individuals representing potential solutions and evolves them over generations. Instead of using crossover and mutation, PBIL estimates and samples the joint probability distribution of selected individuals.

PBIL maintains a probability vector representing the likelihood of each component in the solution space. At each iteration, the probability vector is used to generate new individuals, which are evaluated and ranked. The best individuals are selected to update the probability vector. The algorithm consists of initializing the probability vector and repeating the following steps until convergence: sampling a population, evaluating and ranking individuals, selecting the best ones, and updating the probability vector based on the selected individuals [20].

Genetic Programming

John Koza, an American computer scientist, proposed genetic programming as a method for automatically creating computer programs to solve complex problems. His work started in the 1980s and led to influential books like “Genetic Programming: On the Programming of Computers by Means of Natural Selection” in 1994 [21]. Genetic programming extends genetic algorithms to evolve programs using operators like crossover and mutation. This approach has gained prominence in the field of evolutionary computation for automated program synthesis and optimization.

Genetic programming (GP) is an evolutionary algorithm inspired by Darwinian principles. It uses a population of candidate solutions represented as complex trees. GP applies genetic operators like crossover and mutation to create a new generation of individuals. Fitness functions evaluate individual quality, influencing their selection for reproduction.

For successful GP, two conditions must be met: sufficiency, where the representation can solve the problem, and closure, where functions handle all possible input values. These conditions can be challenging when evolving programs for diverse value types [22].

Differential Evolution

Differential Evolution (DE) for optimization and problem solving was proposed by Rainer Storn and Kenneth Price in the late 1990s. In their seminal work titled “Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces” published in 1997, Storn and Price introduced DE as an evolutionary algorithm [23]. DE utilizes vector differences to generate new candidate solutions, making it highly efficient for optimization in continuous search spaces. This approach has gained widespread popularity as a robust and effective method for solving optimization problems across various domains.

Differential Evolution (DE) is an evolutionary algorithm used for searching in a solution space. It involves initialization, mutation, crossover, and selection steps. In initialization, a population of individuals represented by D-dimensional vectors is

created within the search space. Mutation generates a diverse population by adding the difference between two randomly selected individuals to a third individual. Crossover combines the target and mutation vectors, exchanging information between individuals using a binomial crossover. Selection evaluates the objective function for each test vector and replaces the target vector if it performs better.

The DE algorithm iterates through these steps, evolving the population over multiple generations. By differentiating, scaling, and applying mutation, crossover, and selection, it explores the solution space and enhances individual quality in each generation [24].

B. Physics Based Algorithms

Physics-based optimization algorithms are a category of metaheuristic optimization methods that draw inspiration from physical principles and laws to solve complex problems. These algorithms mimic physical processes found in nature and utilize concepts from physics to develop effective optimization techniques. They differ from biology-based algorithms and have been proposed as alternatives to solve various challenging problems [25].

Gravitational Search Algorithm

The Gravitational Search Algorithm (GSA) for optimization and problem solving was proposed by Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi in 2009. Rashedi, Nezamabadi-Pour, and Saryazdi, who are researchers in the field of computer science and optimization, introduced GSA as a novel metaheuristic algorithm inspired by the laws of gravity and motion [26]. Their work on GSA aimed to develop an efficient optimization algorithm for solving complex problems by simulating the interactions between celestial bodies. Since its proposal, GSA has been applied to various optimization problems and has shown promising results.

The Gravitational Search Algorithm (GSA) is an optimization algorithm inspired by the law of gravity. It represents the problem as a system of interacting masses, where each mass corresponds to a potential solution. The algorithm simulates the gravitational attraction between the masses, causing a global movement towards better solutions.

Each mass has position, inertial mass, active gravitational mass, and passive gravitational mass. The algorithm follows Newtonian laws of gravitation and motion to update the positions and velocities of the masses. Stochastic behavior is introduced through randomly weighted sum of forces. The algorithm balances exploration and exploitation by reducing the number of forces applied over time. GSA benefits from communication between masses, adaptive learning rate, and control over motion and attraction. It shows good convergence rates in experiments and allows for the adjustment of search accuracy [27]. Figure 1.12 demonstrates the force-based position update of a mass [28].

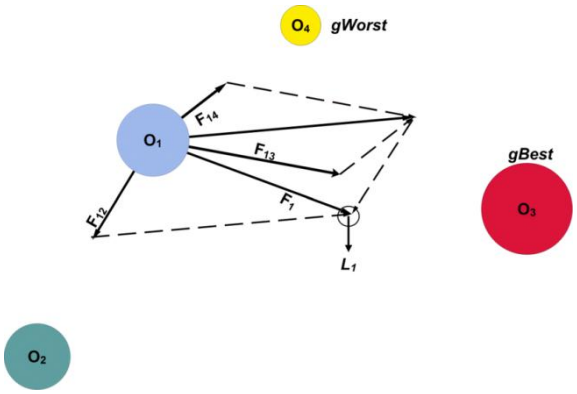


Figure 1.12 – Demonstration of Position Update in the GSA

Charged System Search

The Charged System Search (CSS) algorithm, proposed by A. Kaveh and S. Talatahari, is an optimization technique that draws inspiration from physics and mechanics. It utilizes principles from electrostatics and Newtonian laws to create a multi-agent approach. In CSS, each agent represents a charged particle (CP) and interacts with others based on fitness values and separation distances. Electrostatics laws determine the resultant force, while Newtonian mechanics laws govern the movement quality [29].

In CSS, each potential solution is represented as a charged particle, and their interactions are governed by electric forces analogous to Coulomb's law. The charges on the particles reflect the quality of their solutions, and the algorithm employs attractive and repulsive forces between particles based on their charges and distances. This encourages exploration and exploitation of the search space. The resultant force

acting on each particle is calculated considering these factors. To update the particles' positions and velocities, Newtonian mechanics principles come into play. The new position depends on the previous position, velocity, and the resultant force acting on the particle. The velocity is adjusted based on the change in position and the time step.

CSS effectively balances exploration and exploitation by considering solution quality, attractive and repulsive forces, and Newtonian mechanics. This enables efficient search and convergence towards the optimal solution in the solution space. [29]

C. Central Force Optimization

Central Force Optimization (CFO) is a metaheuristic search algorithm proposed by R. A. Formato in 2007. Inspired by gravitational kinematics, CFO employs a metaphor of "probes" navigating a multidimensional decision space, mirroring the motion of masses under gravity. Equations derived from particle motion govern the positions and accelerations of the probes [30].

Central Force Optimization (CFO) is an optimization algorithm that utilizes the principles of gravitation to guide particles towards better solutions. Each particle represents an object or solution, and its movement is driven by gravitational attraction. The goal is to maximize a given fitness function that measures performance.

The algorithm starts with initializing a population of particles in a multi-dimensional space, distributed uniformly along "probe lines." Particle acceleration is initially set to zero. The next step involves calculating the acceleration of each particle based on Newton's law of gravity. The particle's mass, derived from the objective function, influences its gravitational attraction to other particles.

The motion procedure updates the positions and velocities of particles using Newton's motion laws. Particle movement is restricted within a feasible region, and positions are updated deterministically using a gradient algorithm. The fitness function is evaluated at the new positions, and the algorithm iterates until a desired or global best solution is found. Convergence conditions ensure the algorithm converges to the best solutions discovered [31].

Big-Bang Big-Crunch

The “Big Bang-Big Crunch” optimization method was proposed by Osman K. Erol and Ibrahim Eksin [32]. It is a physics-based algorithm that draws inspiration from the Big Bang and Big Crunch theory of the evolution of the universe. The method involves two phases: the Big Bang phase, where random points are generated, and the Big Crunch phase, where these points are converged to a representative point using a center of mass or minimal cost approach.

In the Big Bang phase, the algorithm creates an initial population of candidates randomly spread across the search space. This randomness represents energy dissipation in nature. The population size is fixed, and candidates are bounded within the search space. The algorithm then enters the Big Crunch phase, applying a convergence operator called the Big Crunch. It finds the center of mass of the population, representing the highest fitness value, using a formula that considers fitness values. This convergence process resembles gravitational attraction and eliminates the need for pairwise combinations. Next, the algorithm generates new candidate solutions for the next iteration by spreading offsprings around the center of mass using a normal distribution operation. The standard deviation decreases as iterations progress, converging towards an optimal point while exploring the search space.

The algorithm alternates between the Big Bang and Big Crunch phases until a stopping criterion is met. It involves generating candidates, evaluating fitness values, finding the center of mass, and creating new candidates around it. The goal is to converge to an optimal point while maintaining a diverse population that decreases the probability of being far from the center of mass [32].

C. Swarm Based Algorithms

Swarm-based optimization algorithms are a type of metaheuristic algorithms that draw inspiration from swarm intelligence (SI) observed in nature. These algorithms mimic the collective behavior of swarms or groups of individuals to solve optimization problems.

In swarm-based optimization algorithms, a population of candidate solutions (often called particles or agents) iteratively explores the search space to find an optimal or near-optimal solution. The individuals within the swarm interact with each other and exchange information to collectively improve their search performance [33].

Particle Swarm Optimization

Particle Swarm Optimization (PSO) was developed by James Kennedy and Russell Eberhart in 1995. Inspired by the collective behavior of birds and fish, they introduced PSO as an optimization algorithm. PSO employs a population of particles that dynamically move and interact with one another to explore and find optimal solutions within a given problem space [34]. Since its inception, PSO has gained significant popularity and has been extensively utilized across diverse domains for effective optimization problem-solving.

The Algorithm simulates a swarm of particles moving through a problem space to find the best solution. Each particle represents a potential solution and adjusts its position based on a velocity vector. Personal experience and social influence guide the particle's movement. It remembers its best position and learns from the swarm's collective knowledge.

The algorithm starts with randomly distributed particles. At each iteration, the velocity and position of each particle are updated based on its current and best positions. This process continues until a stopping criterion is met.

PSO balances personal experience and swarm knowledge to explore the problem space efficiently. Its stochastic nature and particle memory enable it to adapt and find good solutions in complex search spaces. [35]

Ant Colony Optimization

Ant Colony Optimization (ACO) was proposed by Marco Dorigo and his colleagues in the early 1990s [36]. Marco Dorigo, an Italian computer scientist, introduced the concept of ACO as an optimization algorithm inspired by the foraging behavior of ants. ACO simulates the behavior of ant colonies to solve optimization problems by iteratively constructing solutions and refining them based on pheromone trails. The

algorithm has gained popularity and has been successfully applied to various problem domains.

In ACO, artificial ants construct solutions by traversing a construction graph and depositing pheromone on the components they visit. The choice of components is influenced by pheromone levels and heuristic information. A local search can be applied to refine the solutions, and pheromone values are updated based on solution quality. ACO has different variations, such as Ant System (AS) and Ant Colony System (ACS), each with specific rules for pheromone update and solution construction. Overall, ACO is an effective algorithm for finding optimal solutions by mimicking the behavior of ants [36].

Firefly Algorithm

The Firefly Algorithm (FA) was proposed by Xin-She Yang in 2008. It is an optimization algorithm inspired by the flashing behavior of fireflies in their natural environment. The algorithm gained attention for its ability to effectively solve complex optimization problems [37].

The algorithm mimics how fireflies use their bioluminescent flashes to attract mates and communicate. Each firefly represents a potential solution, with its light intensity indicating solution quality.

The algorithm starts with a random population of fireflies. Fireflies move towards more attractive individuals based on their light intensity. Their movement considers attractiveness, distance, and a randomization parameter.

Iterations involve evaluating fitness, sorting fireflies, and updating positions. Termination occurs when a criterion is met. The algorithm achieves a balance between exploration and exploitation, mirroring the collective intelligence exhibited by social insects [38].

Whale Optimization Algorithm

The Whale Optimization Algorithm (WOA) was proposed by Seyedali Mirjalili in 2016 as a nature-inspired optimization algorithm. Mirjalili introduced the WOA in his research

paper titled “Whale Optimization Algorithm,” which aimed to develop a novel approach for solving optimization problems [39].

The Whale Optimization Algorithm (WOA) is an optimization algorithm inspired by the hunting behavior of humpback whales. It mimics their bubble-net feeding method, where whales work together to encircle and trap prey. The algorithm uses a population of search agents representing potential solutions. It combines exploration and exploitation phases, with agents moving randomly and towards the current best solution. The algorithm iterates, updating agent positions and evaluating fitness until a termination criterion is met. The WOA algorithm leverages whale behavior to efficiently search for optimal solutions in various optimization problems [39].

D. Human Based Algorithms

Human-based optimization algorithms, also known as bio-inspired or nature-inspired algorithms, are metaheuristic methods that draw inspiration from human behaviors, social interactions, or natural phenomena to solve complex optimization problems. These algorithms attempt to mimic the problem-solving strategies employed by humans or observe patterns in natural systems to find optimal solutions.

Social Emotional Optimization

The Social Emotional Optimization Algorithm (SEOA) was proposed by Yuechun Xu, Zhihua Cui, and Jianchao Zeng in 2010. This algorithm is a swarm intelligent technique that simulates human behavior guided by emotions to solve nonlinear constrained optimization problems. It aims to address the challenges of nonlinear programming problems by incorporating social and emotional aspects into the optimization process [40].

The Social Emotional Optimization Algorithm (SEOA) is inspired by human behavior in society, where individuals strive to increase their social status. It employs virtual individuals who make choices based on their emotional index, which is evaluated by society. Initially, all individuals have an emotion index of 1 and choose their behavior using a set of manners and random factors. The algorithm incorporates thresholds and

parameters to simulate human behavior and control the behavior selection process [41].

Imperialist Competitive Algorithm

The Imperialist Competitive Algorithm (ICA) was proposed by Esmaeil Atashpaz-Gargari and Caro Lucas in 2007. The algorithm is inspired by imperialistic competition and aims to solve optimization problems. It starts with an initial population divided into colonies and imperialists, forming empires. Through competition, powerful empires take over weaker ones, leading to convergence towards a state with a single empire. [42]

The Imperialist Competitive Algorithm (ICA) is an evolutionary algorithm that simulates the competition and assimilation processes among countries. Individuals in the population represent countries, divided into imperialist countries and colonies. Imperialists are selected based on lower costs, and colonies are assigned to imperialists based on their power. Colonies move towards their respective imperialists, representing assimilation. Empires compete and weaker ones collapse, leading to convergence. The algorithm includes revolution to prevent early convergence and position exchanges. Weaker empires lose colonies to stronger ones. Competition among empires determines colony distribution [43].

Teaching Learning Based Optimization

Teaching Learning Based Optimization (TLBO) is an optimization method proposed by R.V. Rao, V.J. Savsani, and D.P. Vakharia in 2011. It was specifically developed for constrained mechanical design optimization problems. Inspired by the relationship between a teacher and learners, TLBO is a population-based technique consisting of two phases: the Teacher Phase and the Learner Phase. In the Teacher Phase, learners acquire knowledge from the teacher, while in the Learner Phase, learners interact and improve collectively [44].

Teaching-Learning-Based Optimization (TLBO) is an optimization technique that emulates the teaching and learning process in a classroom. It simulates the interaction between a teacher and a group of learners to solve optimization problems.

TLBO operates in two phases: the Teacher Phase and the Learner Phase. In the Teacher Phase, the teacher guides the learners towards better performance by modifying the existing solutions using a teaching factor and random numbers. During the Learner Phase, learners interact with each other, comparing their performances and adjusting their solutions based on the difference between them and random numbers. TLBO is a population-based method where the population represents the group of learners. The goal is to find the global solution that maximizes or minimizes the fitness function, corresponding to the learners' performance.

By drawing inspiration from the dynamics of a classroom, TLBO offers a unique approach to optimization, fostering interactions and knowledge exchange to improve performance iteratively [45].

Soccer League Competition

The Soccer League Competition (SLC) algorithm, proposed by Naser Moosavian and Babak Kasaee Roodsari in 2014, is a meta-heuristic optimization technique inspired by the competitive nature of soccer leagues. It incorporates the concept of teams and players competing for top positions in the league table to solve various optimal design problems. By dividing the population into teams and simulating their competition, the SLC algorithm aims to find the global optimum.

In soccer league competitions, teams compete against each other over a season to achieve top positions in the league table. Matches are played, and teams earn points based on their performance. At the end of the season, the team with the most points becomes the champion, while the bottom two teams are relegated to a lower-level league, making room for new talent. Each team consists of fixed players and substitutes, with internal competitions to improve performance. Key players, such as the Star Player and Super Star Player, play crucial roles. Strategies and competitions within teams lead to overall performance improvement. The Soccer League Competition (SLC) algorithm simulates this process to solve optimization problems. It follows steps such as initializing parameters, generating samples, assessing teams, updating standings, and handling relegation and promotion. SLC algorithm utilizes the dynamics of soccer leagues to optimize solutions and improve performance [46].

1.3.2 Discussion

WOA (Whale Optimization Algorithm) is a recent approach that has shown promising performances in multiple engineering disciplines. In our research, we aim to pioneer its application in path planning, specifically in conjunction with the Rapidly-exploring Random Tree (RRT) algorithm, as we anticipate it to yield highly positive results. To the best of our knowledge, RRT has not yet been implemented with WOA for path planning.

There are several key reasons why WOA holds great potential for path planning:

Low computational complexity: WOA exhibits relatively low computational complexity when compared to other optimization algorithms. This characteristic is particularly advantageous for real-time applications like autonomous navigation, where quick decision-making is crucial.

Ability to handle nonlinear and non-convex problems: Path planning in autonomous navigation often involves nonlinear and non-convex optimization problems which may contain local minima. WOA has demonstrated effectiveness in dealing with these problem characteristics [39]. By leveraging its exploration and exploitation mechanisms, WOA can efficiently explore the search space, even in the presence of complex constraints and irregular landscapes. Consequently, it leads to better path planning outcomes.

Potential for parallelization: The characteristics exhibited by WOA make it well-suited for parallelization, which in turn allows for effective utilization of modern hardware architectures. By updating the agents individually in a parallel manner, the exploration of the search space is accelerated.

Our research aims to investigate and validate the potential benefits of leveraging WOA (Whale Optimization Algorithm) in conjunction with the RRT (Rapidly-exploring Random Tree) algorithm for path planning. The focus of our investigation lies in exploring the specific advantages offered by WOA and how they can be effectively utilized in this context.

1.4 Conclusion

In this chapter, we conducted a detailed examination of motion planning, encompassing its definition, a brief overview of its historical background, the fundamental concepts associated with it, and an introduction to the main classes of motion planning algorithms, namely deterministic and sampling-based approaches. We explored prominent algorithms within these classes, including Dijkstra's algorithm, A*, Visibility Graph, Voronoi Diagram, Rapidly-exploring Random Trees (RRT), and Probabilistic Road Maps.

Furthermore, we concluded the motion planning section by comparing the aforementioned algorithms in terms of completeness, optimality, memory usage, handling dynamic environments, and accommodating non-holonomic constraints. Additionally, we provided an overview of metaheuristic optimization approaches, specifically evolutionary-based, physics-based, swarm-based, and human-based methods, along with some of their corresponding algorithms.

We also justified our selection of the RRT algorithm as the primary path planning algorithm for our study, considering its advantages for our application scenario. We acknowledged the weaknesses of RRT and expressed our intention to address them by exploring its variants. One of the main reasons behind choosing RRT was its adaptability, allowing for extension through optimization algorithms. Consequently, we opted for the integration of the Whale Optimization Algorithm (WOA) as the optimization algorithm in conjunction with RRT, which was supported by several reasons specific to this choice.

In the upcoming chapter, we will introduce RRT and briefly discuss its variants. We will then provide a rationale for our selection of a particular variant. Additionally, we will present the formulation of WOA and outline our proposed variants, along with the type of control generation we have opted for.

Chapter 2

Optimal Motion Planners: Proposed Approaches

2.1 Introduction

Rapidly-exploring Random Tree (RRT) is a popular algorithm used for motion planning in robotics and computer graphics. It efficiently explores the search space to find feasible paths between a start and goal configuration. RRT is well-suited for high-dimensional and complex motion planning problems, overcoming the curse of dimensionality.

Many variants of RRT exist to address specific challenges and improve performance. Variants like RRT* [47], RRT-Connect [48], and Goal Biased-RRT [49] introduce modifications to tackle issues such as narrow passages, high-dimensional spaces, and dynamic obstacles.

Optimization techniques like the Whale Optimization Algorithm (WOA) can enhance the performance of RRT and its variants, discovering feasible and optimized paths in complex scenarios. WOA is versatile and has been successfully applied to various optimization problems, overcoming local optima and converging to high-quality solutions [39].

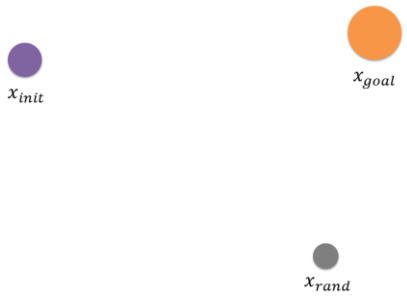
The chapter provides an overview of RRT, its limitations, and the development of variants. It explores the integration of WOA and its advantages. The sections cover RRT's operational principles, renowned variants, justification for choosing RRT*, its enhancements and limitations, and the proposed local and global variants of RRT* combined with WOA. The nature of the robot platform and the chosen approach for generating controls are also discussed.

2.2 Rapidly-exploring Random Tree

The Rapidly Exploring Random Tree (RRT) algorithm is a path planning technique introduced by Steven M. LaValle in June 1998. It was developed as a simple and iterative algorithm to efficiently search complex and high-dimensional spaces for feasible paths. The RRT algorithm has since become widely used in various robotic systems and other fields.

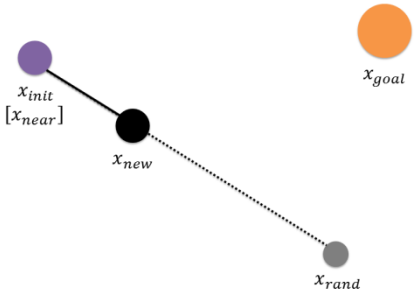
The algorithm begins with an initial configuration or point in the space, which acts as the root of the tree. This initial configuration could represent the starting position of a robot or any other desired location. The tree is then iteratively expanded by randomly sampling points in the space. In each iteration, a new random sample is generated in the space. The nearest point in the existing tree is identified, and a connection is established between this point and the new sample if it lies within the free space. The connection is usually made by following a straight line or any other suitable trajectory. This process is repeated for a predefined number of iterations or until a specific condition is met. As the algorithm progresses, the tree structure gradually expands and covers the space more uniformly. Eventually, if the initial and goal regions are reachable within the space, the growing tree will establish a connection between them. This connection indicates the discovery of a feasible path from the initial configuration to the goal configuration.

Initialization of the tree with x_{init}



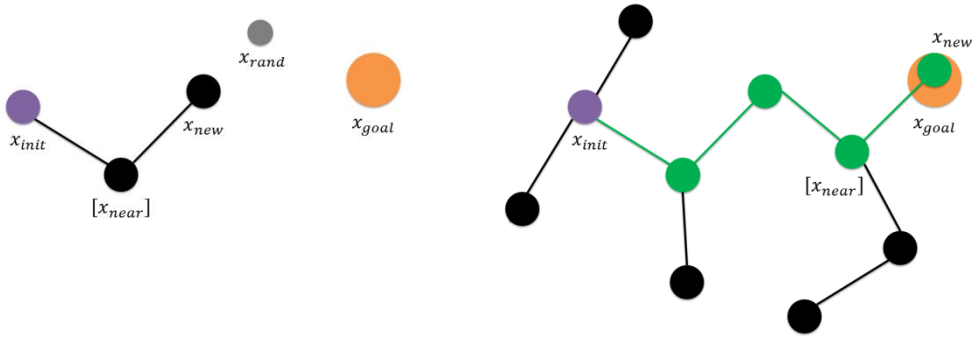
Generation of a random sample x_{rand}

The nearest node (x_{near}) to x_{rand} from



Insertion of x_{new} through steering from

the current tree is x_{init} x_{near} towards x_{rand}



Second expansion After 9 Added nodes, x_{new} falls within the goal region and the path is returned

Figure 2.1 – Building Process of the Rapidly-exploring Random Tree

Presented below is the Rapidly-exploring Random Tree algorithm.

Algorithm 1: $Path \leftarrow \mathrm{RRT}(x_{init}, x_{goal})$
1 $T \leftarrow \mathrm{InitializeTree}(x_{init}, T)$
2 ***for*** $i = 1$ *to* K ***do***:
3 $x_{rand} \leftarrow \mathrm{SampleFreeSpace}(\,)$
4 $x_{near} \leftarrow \mathrm{FindNearest}(x_{rand}, T)$
5 $x_{new} \leftarrow \mathrm{Steer}(x_{near}, x_{rand})$
6 ***if*** $\mathrm{ObstacleFree}(x_{new})$ ***then***:
7 $T \leftarrow \mathrm{InsertNode}(x_{new})$
8 ***if*** $\mathrm{GoalReached}(x_{new}, x_{goal})$ ***then***:
9 ***return*** $Path$
10 ***return*** $NoPathFound$

The Rapidly Exploring Random Tree (RRT) algorithm, although powerful, does have certain limitations. While RRTs are primarily designed for static environments and can be effective in many cases, they do not guarantee optimal paths and may converge slowly in certain scenarios. Additionally, they do not inherently handle dynamic obstacles or account for changes in the environment during the planning process.

However, researchers have developed variants of the RRT algorithm to address some of these limitations. These variants aim to enhance the efficiency and performance of the algorithm by considering dynamic obstacles and adapting to changes in the

environment. By incorporating these improvements, the modified versions of RRT strive to provide better planning results and overcome some of the original algorithm's shortcomings.

2.3 The RRT Variants

In this section, we will discuss briefly the main contributions of various RRT (Rapidly-exploring Random Tree) variants.

S-RRT. Introduces a greedy approach to shorten the total length of the path, Incorporates B-spline curves to provide smoother paths, and reduces the path complexity by removing redundant nodes from the tree [50].

RRT*. During the node insertion process. First, it selects the parent node that minimizes the overall cost to the root resulting in shorter paths. Second, dynamic edge rewiring is implemented to provide efficient exploration of the configuration space [51].

RJ-RRT. Incorporates a greedy-based sampling strategy that progressively narrows down the sampling space to the goal area. Additionally, it employs an enhanced environment judgment method that swiftly detects and explores narrow passage. This method utilizes a simple calculation to determine the environment at each sampling point [52].

RRT-Connect. Introduces the Connect heuristic, enabling longer-distance movements and rapid convergence to a solution. Moreover, the simultaneous maintenance of two trees facilitates quick and uniform exploration of the configuration space [53].

IRRT-Connect. Makes use of a third node based on the idea of dichotomous points, allowing the algorithm to be extended with four trees. In addition, biased spanning of the tree towards the goal point effectively addresses the blind search problem of RRT-Connect [54].

RRT-A*. Integrates the A* cost function, allowing for informed decision-making during the planning process and improved generated paths [55].

ORRT-A*. Employs morphological dilation to inflate obstacles before path generation, preventing collisions and improving safety. Furthermore, cubic spline interpolation is used to smoothen the generated path, ensuring seamless transitions and a visually appealing trajectory [56].

PG-RRT. Incorporates Gaussian models to enhance goal adaptation and faster convergence in RRT node generation. Additionally, feasible Gaussian model samples are included to ensure kinematic compatibility. Moreover, a specialized node has been introduced to facilitate intelligent tree expansion towards the goal [57].

pRRT. Integrates uncertainty by simulating multiple search tree extensions as a stochastic process. The variant also implements probabilistic selection of extensions and entire paths based on the expected probability of successful execution, considering costs and likelihood of success. In addition, it considers various cost metrics beyond path length, such as energy consumption and execution time, enabling paths with different cumulative costs. To enhance robustness in uncertain environments, the variant propagates uncertainty to planned paths and accounts for factors like terrain characteristics, sensor accuracy, and coefficients of friction [58].

NRRT*. Uses a trained CNN model to guide the sampling process and predict optimal path probabilities [59].

FG-RRT. Incorporates a fuzzy logic-based approach to allow for an intelligent decision-making process during tree expansion [60].

MRRT. Uses artificial guided points to explore narrow passages. Moreover, the incorporation of trajectory primitives respecting the robot's dynamic constraints enables generation of feasible trajectories [61].

GB-RRT. Integrates potential fields representation of the environment to significantly increase the sampling efficiency. Furthermore, the use of cubic B-splines to generate smoother paths and eliminate sharp turns, results in feasible paths [62].

Bi-RRT. Allows comprehensive exploration of the search space by simultaneously generating two trees, one from the starting point and the other from the target point [63].

RRT-Rope. Uses an optimized version of RRT-Connect enabling rapid computation of feasible paths. By simultaneously growing two trees towards each other, it significantly reduces path finding time. To further enhance the computed path, a deterministic shortcutting technique is employed. This technique efficiently shortens the path after computation by leveraging intermediate nodes without compromising resolution [64].

RRT*-Smart. Optimizes the initial path by directly connecting visible nodes, reducing the number of nodes compared to the original RRT* path. Additionally, it biases the sampling towards beacons, which are nodes in the optimized path, helping the algorithm approach optimality faster [65].

RRV. Explores narrow passages by utilizing dominant eigenvectors and precise sampling, allowing for the effective expansion of the tree. It is aware of tree nodes located near narrow passages, enabling appropriate expansion in those areas. Additionally, it employs a unique vine-like expansion strategy along obstacles, facilitating efficient identification and traversal of narrow passages [66].

RRT-Blossom. Enables local exploration while avoiding local minima, ensuring expansions move away from the target without regression. Additionally, it improves node generation efficiency and exploration rate by instantiating all eligible edges during node expansion. Furthermore, it detects deadlocks when all accessible paths are blocked, allowing the next expansion attempt to ignore regression constraints [67].

2.4 Discussion

The previous variants of the Rapidly Exploring Random Tree (RRT) algorithm have made significant contributions, each improving the algorithm in specific areas while potentially introducing limitations in others.

Our primary focus is to identify a variant that greatly prioritizes optimality, as obtaining a high-quality path enhances the comfort of wheelchair users and instills a sense of predictability among people around the device.

Among the variants, RRT* has emerged as a highly successful and widely recognized algorithm in the research field because of its asymptotic optimality property. This

means that given an infinite amount of time and samples, the algorithm will converge to the optimal solution if one exists. The aspect of RRT* achieving asymptotic optimality has been thoroughly studied, leading to the establishment of formal guarantees [68].

The upcoming section will explore the enhancements brought by RRT*.

2.5 The RRT* Algorithm

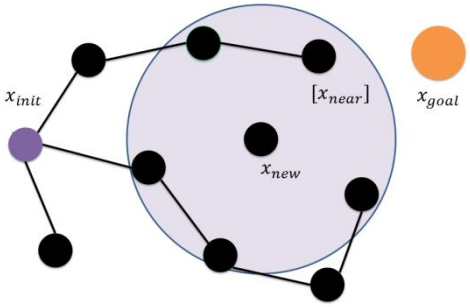
The RRT* algorithm, introduced in 2006 by Steven LaValle and James Kuffner, extends the Rapidly-exploring Random Tree (RRT) algorithm to overcome limitations of the original approach. RRT* improves upon RRT by introducing both parent choosing and edge rewiring steps to optimize the tree structure and reduce the overall cost of the path. These enhancements address the lack of optimality guarantees and sensitivity to algorithm parameters found in the original RRT. By combining parent choosing and edge rewiring, RRT* is able to generate near-optimal paths while efficiently exploring high-dimensional configuration spaces [51].

2.5.1 Enhancements

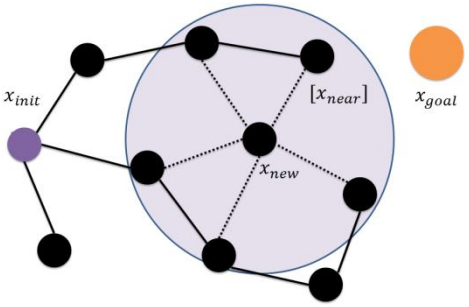
RRT* shares a similar approach with RRT in the initial phase, where random samples are incrementally connected to the existing tree structure to explore the configuration space. However, RRT* introduces two key modifications to enhance its performance.

A. Parent Choosing

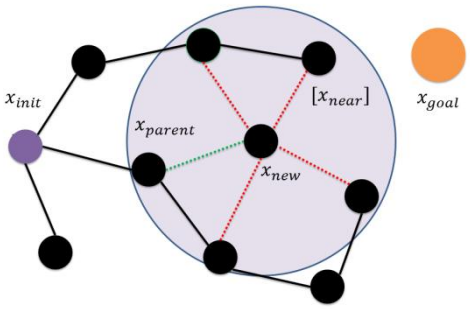
RRT* identifies nearby nodes when a new node is generated. Within this neighborhood, it selects the parent node that minimizes the path cost. This approach enhances path optimality and increases the probability of finding near-optimal paths.



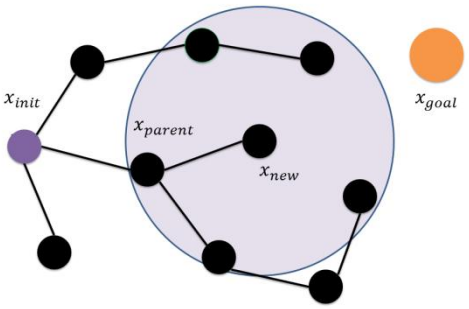
Generation of a new node x_{new}



Assessment of the cost of nearby nodes to the root x_{init}



Selection of the best parent node x_{parent}

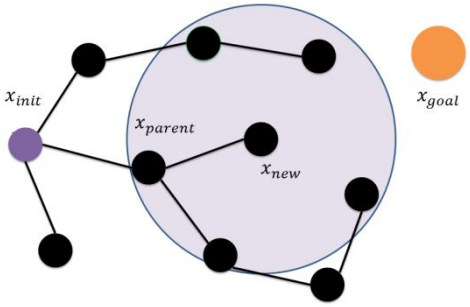


Insertion of the new node into the tree

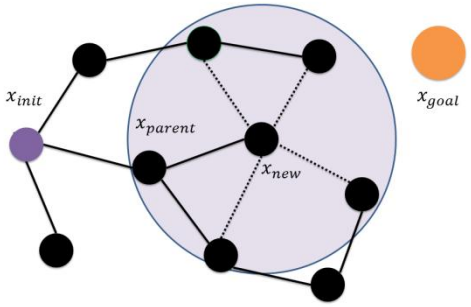
Figure 2.2 – Parent Choosing Process of RRT*

B. Edge Rewiring

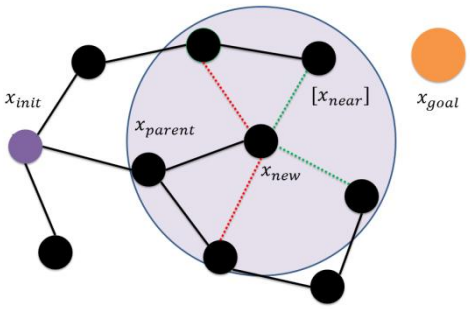
RRT* performs edge rewiring after adding a new node. It reevaluates the edges within the node's neighborhood and rewires them to achieve lower path costs. This process involves considering alternative connections and assessing the potential for reducing costs. By dynamically adjusting the edges, RRT* improves path efficiency and explores the configuration space more effectively.



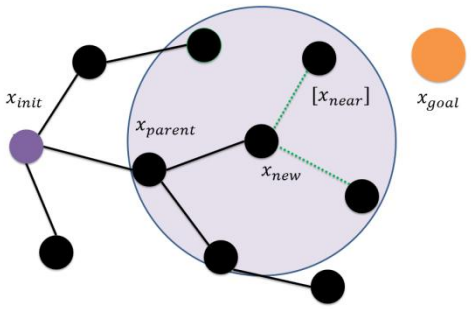
Insertion of the new node into the tree



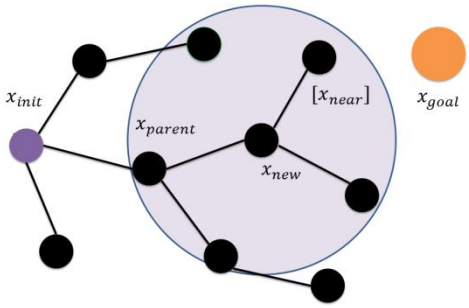
Assessment of the potential cost reduction of nearby nodes



Selection of candidate nodes (Green) for rewiring



The unselected nodes (Red) remain as they are, The candidate nodes have their current edges removed



The candidate nodes are rewired with x_{new} to achieve lower costs

Figure 2.3 – Edge Rewiring Process of RRT*

2.5.2 Tree Construction

The RRT* algorithm begins by taking the initial configuration x_{init} and goal configuration x_{goal} as parameters. It initializes the tree T with an empty set of nodes and edges. The algorithm inserts the root node x_{init} into the tree.

To build the tree, the algorithm samples a random configuration x_{rand} from the free space. It then finds the nearest node x_{near} in the tree based on the Euclidean distance metric. From x_{near}, a new node x_{new} is extended towards x_{rand} with a predefined step size.

If the new node x_{new} belongs to the free space, the algorithm considers its neighborhood $X_{neighbors}$. Within this neighborhood, it selects the parent node x_{parent} that minimizes the overall cost to the root node. The new node x_{new} is then inserted into the tree T.

Next, the algorithm reevaluates the neighboring nodes and rewires their connections to achieve lower path costs. This process improves the efficiency and optimality of the tree structure.

If the goal configuration x_{goal} is reached, the algorithm returns the path. Otherwise, the process of building the tree continues iteratively until the maximum number of iterations is reached. If no path is found within the maximum iterations, the algorithm does not return a path.

Presented below is the Rapidly-exploring Random Tree Star (RRT*) algorithm.

Algorithm 2: $Path \leftarrow \mathrm{RRT}^{*}(x_{init}, x_{goal})$
1 $T \leftarrow$ InitializeTree( )
2 $T \leftarrow \mathrm{InsertNode}(x_{init}, T)$
3 ***for*** $i = 1$ *to* K ***do***:
4 $x_{rand} \leftarrow$ SampleFreeSpace( )
5 $x_{near} \leftarrow \mathrm{FindNearest}(x_{rand}, T)$
6 $x_{new} \leftarrow \mathrm{Steer}(x_{near}, x_{rand})$
7 ***if*** $\mathrm{ObstacleFree}(x_{new})$ ***then***:
8 $X_{neighbors} \leftarrow \mathrm{GetNeighbors}(x_{new}, |R|, T)$
9 $x_{parent} \leftarrow \mathrm{ChooseParent}(x_{new}, X_{neighbors})$

$T \leftarrow \text{InsertNode}(x_{new}, x_{parent}, T)$
$T \leftarrow \text{Rewire}(X_{neighbors}, x_{new}, T)$
if $\text{GoalReached}(x_{new}, x_{goal})$ ***then***:
    return $Path$
return $NoPathFound$

It is important to note that RRT* is primarily applied in static environments, as it lacks inherent support for dynamic obstacles. Addressing this limitation is crucial for our application scenario, and it will be the subject of discussion in the upcoming chapter.

2.6 Whale Optimization Algorithm

The Whale Optimization Algorithm (WOA) is a metaheuristic optimization algorithm inspired by the hunting strategy of humpback whales. It was first proposed in 2016 by Seyedali Mirjalili, a researcher at Griffith University in Australia [39]. Since then, the WOA has gained significant attention and has been widely and recently used in various works of path planning due to its impressive performance [69,70,71].

2.6.1 Inspiration

The remarkable characteristics and behaviors of whales have served as a significant inspiration for the development of the Whale Optimization Algorithm. Whales, as the largest mammals on Earth, exhibit impressive traits such as intelligence and emotional capacity. This allows them to think, learn, communicate, and display emotions, albeit at a lower level of intelligence compared to humans.

Humpback whales, one of the largest baleen whale species, employ a distinctive hunting technique known as bubble-net feeding. It involves the creation of bubbles in a circular or spiral shaped path to enclose prey near the water's surface. Initially studied through surface observations, researchers later utilized tag sensors to capture and analyze 300 instances of bubble-net feeding in nine individual humpback whales. This research identified two maneuvers associated with bubble-net feeding: upward-spirals and double-loops. Humpback whales dive approximately 12 meters, create a spiral-shaped bubble net around their prey, and swim back up to the surface, strategically positioning themselves to capitalize on the trapped prey [39].

The exceptional bubble-net feeding behavior of humpback whales has provided inspiration for the authors of the algorithm, leading them to mathematically model and optimize this unique foraging strategy.

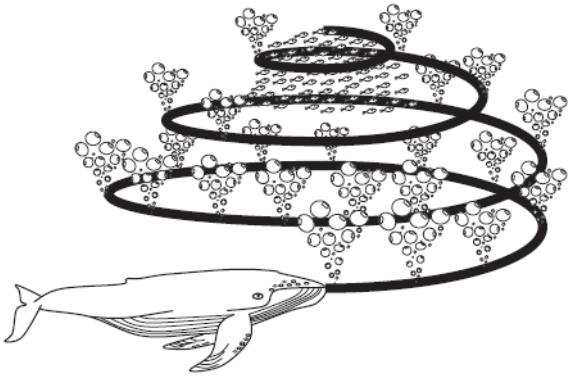


Figure 2.4 – Bubble-net Feeding Technique of Humpback Whales

2.6.2 Formulation

A. Initialization

The WOA algorithm represents whales as a population of n search agents, dynamically exploring the solution space. The best known solution, denoted as X^*, is associated with the concept of prey. The population size is predetermined, and the initial positions of whales are randomly assigned within the search space. To manage the algorithm's execution time, the maximum number of iterations is specified.

B. Choice of Maneuver

During the execution of the algorithm, each whale is required to make a choice between two distinct behaviors. The first option is to move in accordance with the "Shrinking Circle Mechanism" or engage in the "Search for Prey" behavior. Alternatively, the second behavior choice entails implementing the "Spiral Updating Position" strategy. This choice of strategies is determined by the values of two random parameters: p ranging in the interval [0, 1] and a A ranging in the interval [-2, 2]. Figure 2.5 depicts a diagram illustrating the behavior selection process within the Whale Optimization Algorithm.

- If p is less than 0.5 and $|A|$ is greater than 1, the whale selects a random agent to move away from, emphasizing exploration. However if $|A|$ is less than 1, it switches to the Shrinking Circle Mechanism, prioritizing exploitation.

- If p is greater than or equal to 0.5, the whale chooses to follow a spiral movement.

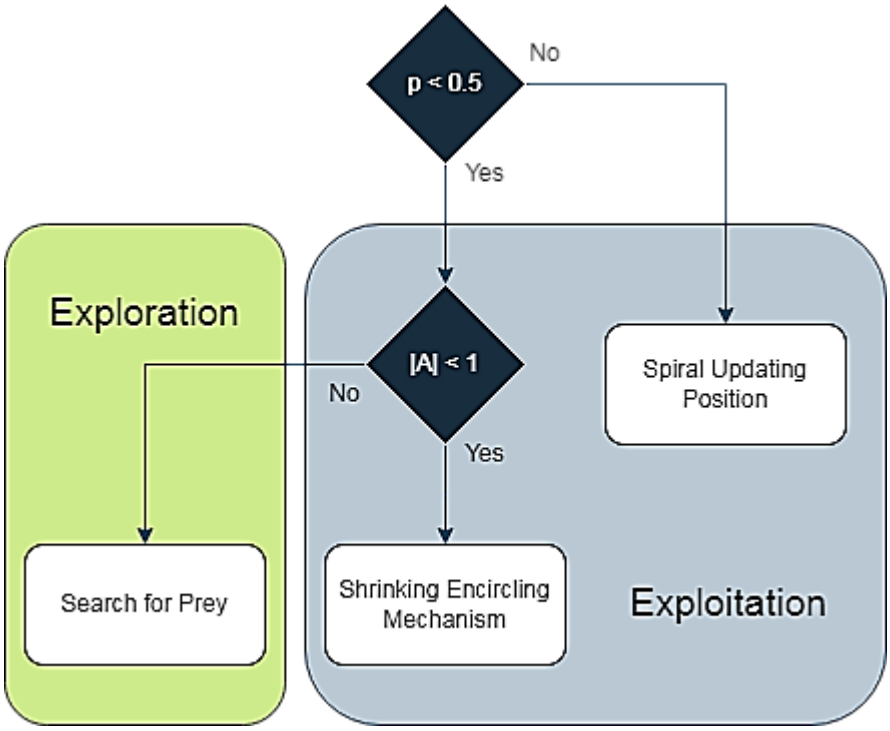


Figure 2.5 - Behavior Selection Diagram of the Whale Optimization Algorithm

C. Bubble-Net Attacking Method (Exploitation Phase)

Shrinking Encircling Mechanism enables each whale to navigate towards the optimal solution $\vec{X}^*$ using the following equation:

$$\vec{D} = \left|\vec{C} \cdot \vec{X}^*(t) - \vec{X}(t)\right| \quad (2.1)$$

$$\vec{X}(t+1) = \vec{X}^*(t) - \vec{A} \cdot \vec{D} \quad (2.2)$$

Where t indicates the current iteration, $\vec{X}^*$ is the position vector of the best solution obtained so far, $\vec{X}$ is the position vector, || is the absolute value, and $\cdot$ is an element-by-element multiplication. $\vec{A}$ and $\vec{C}$ are coefficient vectors and are calculated as follows:

$$\vec{A} = 2\vec{a} \cdot \vec{r} - \vec{a} \quad (2.3)$$

$$\vec{C} = 2 \cdot \vec{r} \quad (2.4)$$

Where $\vec{a}$ is linearly decreased from 2 to 0 over the course of iterations (in both exploration and exploitation phases) and $\vec{r}$ is a random vector in the interval [0, 1].

Spiral Updating Position enables each whale to navigate towards the optimal solution $\vec{X}^*$ using the following equation:

$$\overrightarrow{D'} = \left|\vec{X}^*(t) - \vec{X}(t)\right| \quad (2.5)$$

$$\vec{X}(t+1) = \overrightarrow{D'}.e^{bl}.\cos(2\pi l) + \vec{X}^*(t) \quad (2.6)$$

With l as a random variable varying within the interval [-1, 1] and b as a coefficient that defines the shape of the logarithmic spiral.

D. Search for Prey (Exploration Phase)

This strategy enables each whale to move far away from a random whale using the following equation:

$$\vec{D} = \left|\vec{C}.\overrightarrow{X_{rand}} - \vec{X}\right| \quad (2.7)$$

$$\vec{X}(t+1) = \overrightarrow{X_{rand}} - \vec{A}.\vec{D} \quad (2.8)$$

Where $\overrightarrow{X_{rand}}$ is a random agent's position vector.

2.6.3 Algorithm

The Whale Optimization algorithm starts by initializing a population of search agents with random positions, representing potential solutions. Their fitness values are calculated, indicating performance in terms of the objective function. The best fitness value X^* is initially set.

The algorithm proceeds to enter the main loop, where it iterates over each search agent and performs the following updates. The coefficients (a, A, C, l, p) are updated. If p is less than 0.5, the magnitude of A is examined. If $|A|$ is less than 1, an update is

performed using Shrinking Encircling Mechanism equation (2.2). Otherwise, Search for Prey equation (2.8) is used.

When p is greater than or equal to 0.5, agents are updated with Spiral Updating Position equation (2.6). After updating, a boundary check ensures positions remain within the search space. Fitness is recalculated, and the iteration counter t is incremented. The iterative process continues until the maximum number of iterations is reached.

Finally, the algorithm returns the best solution X^* found after the specified iterations, representing the optimal solution to the optimization problem.

Presented below is the Whale Optimization algorithm.

Algorithm 3: $X^* \leftarrow$ WOA( )
1 $Population \leftarrow$ InitializePopulation( )
2 $X^* \leftarrow$ CalculateFitness( )
3 ***while*** ($t <$ MaxIterations):
4 ***for*** *each search agent* ***do***:
5 CoefficientUpdate(a, A, C, l, p)
6 ***if*** $p < 0.5$ ***then***:
7 ***if*** $|A| < 1$ ***then***:
8 Update agent by Shrinking Encircling Mechanism Eq. (2.2)
9 ***if*** $|A| \geq 1$ ***then***:
10 Update agent by Search for Prey Eq. (2.8)
11 ***if*** $p \geq 0.5$ ***then***:
12 Update agent by Spiral Updating Position Eq. (2.6)
13 $Population \leftarrow BoundaryCheck( )$
14 $X^* \leftarrow CalculateFitness( )$
15 $t = t + 1$
16 ***return*** X^*

2.7 Proposed Local Variant

The proposed local variant optimizes the extension process of RRT* by leveraging the Whale Optimization Algorithm. This variant enhances the efficiency of exploring the search space by expanding towards promising areas. In this approach, each particle in the population is represented as a single configuration.

To guide the expansion process, biasing weights are employed in the variant, enabling a range of strategies from completely random to goal-oriented expansion. Additionally, the inclusion of visibility factors addresses the local minima problem. These factors prioritize solutions that have an unobstructed line of sight to both the starting node and the goal node. Essentially, the local variant favors solutions that can perceive both the initial and goal nodes without encountering any obstacles in between.

The proposed local variant incorporates the RRT* algorithm but replaces the steer function with algorithm 4. This modified algorithm takes into account three main parameters: the nearest configuration (x_{near}), the random configuration (x_{rand}), and the goal configuration (x_{goal}). Firstly, it uses x_{rand} as the center point and a predefined radius to initialize the population with random configurations that are located within that circle. Secondly, it utilizes x_{near}, x_{rand}, and x_{goal} to calculate the objective function, which is based on four factors. Two factors represent the distance to x_{rand} and x_{goal}, while the other two factors are binary and indicate the visibility of the solution to x_{near} and x_{goal}, the former one having higher priority.

Finally, the new node is generated by extending from x_{near} towards x^* with a specified step size, and this new node is returned as the result.

Algorithm 4: $x_{new} \leftarrow \text{SteerUsingWOA}(x_{near}, x_{rand}, x_{goal})$

$Population \leftarrow \text{InitializePopulation}(x_{rand})$
$x^* \leftarrow \text{CalculateFitness}(x_{rand}, x_{goal})$
while $(t < \text{MaxIterations})$:
    for *each search agent* ***do***:
        $\text{CoefficientUpdate}(a, A, C, l, p)$
        if $p < 0.5$ ***then***:
            if $|A| < 1$ ***then***:
                Update agent by Shrinking Encircling Mechanism Eq. (2.2)
            if $|A| \geq 1$ ***then***:
                Update agent by Search for Prey Eq. (2.8)
        if $p \geq 0.5$ ***then***:
            Update agent by Spiral Updating Position Eq. (2.6)
    $Population \leftarrow \text{BoundaryCheck}()$
    $x^* \leftarrow \text{CalculateFitness}(x_{rand}, x_{goal})$
    $t = t + 1$

16 $x_{new} \longleftarrow \text{Extend}(x_{near}, x^*)$
17 ***return*** x_{new}

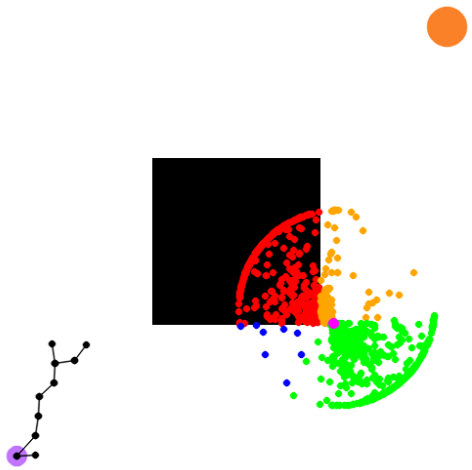


Figure 2.6 – Local Variant During Optimization Phase

The figure 2.6 above illustrates the population of solutions during the optimization phase. The red dots represent invalid solutions as they cross the obstacle, while the blue dots indicate solutions that possess a direct line of sight to the nearest configuration. On the other hand, the orange dots maintain a straight visibility line to the goal configuration. As for the green dots, they have visibility to both the nearest and goal configurations. Finally, the magenta node found in the center represents the current best solution achieved thus far.

2.8 Proposed Global Variant

The proposed global variant utilizes the Whale Optimization Algorithm to improve the quality of the path obtained from RRT*. In this variant, each particle in the population is encoded as an array of configurations, representing a specific path.

To optimize the path, the algorithm considers two essential factors. Firstly, it focuses on minimizing the length of the path, aiming to find a trajectory that is shorter and more efficient. Secondly, it employs a binary factor to indicate whether the path is collision-free or not, thereby ensuring safe traversal.

The global variant incorporates the path obtained from RRT* as the initial particle, serving as the population's initialization. To ensure diversity, randomization is applied

to each position along the path. This process involves assigning values within a circular region centered at the initial origin, with a radius of R, to every particle in the population. Afterward, the optimization algorithm refines the paths of all particles. Ultimately, the optimized solution is determined by selecting the best path among the particles. Figure 2.7, presented below, provides a visual representation of this variant.

Algorithm 5: $P^* \longleftarrow \text{Optimize}(Path)$
1 $Population \longleftarrow \text{InitializePopulation}(Path)$
2 $P^* \longleftarrow \text{CalculateFitness}()$
3 ***while*** $(t < \text{MaxIterations})$:
4 ***for*** *each search agent* ***do***:
5 $\text{CoefficientUpdate}(a, A, C, l, p)$
6 ***if*** $p < 0.5$ ***then***:
7 ***if*** $|A| < 1$ ***then***:
8 Update agent by Shrinking Encircling Mechanism Eq. (2.2)
9 ***if*** $|A| \geq 1$ ***then***:
10 Update agent by Seach for Prey Eq. (2.8)
11 ***if*** $p \geq 0.5$ ***then***:
12 Update agent by Spiral Updating Position Eq. (2.6)
13 $Population \longleftarrow BoundaryCheck()$
14 $P^* \longleftarrow CalculateFitness()$
15 $t = t + 1$
16 ***return*** P^*

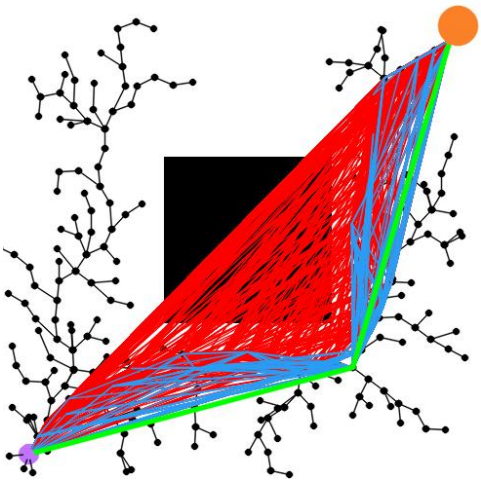


Figure 2.7 – Global Variant During Optimization Phase

The red paths denote invalid solutions as they traverse through the obstacle, while the blue paths indicate valid solutions. Among them, the green path represents the current best solution achieved thus far.

2.9 Robot Motion

In this section, we will discuss the nature of our robot platform and the approach we have chosen for control generation.

2.9.1 Robot Platform and Kinematic Modeling

Despite our proposed motion planning variants generating a path from the initial to the goal configuration, this high-level process that considers factors such as obstacles and optimality alone does not provide us with the necessary low-level information about the controls needed to effectively follow the trajectory.

The trajectories generated by our approach are non-linear in nature, and we also need to handle various constraints like collision avoidance and velocity limits while aiming for optimal control. This led us to opt for an optimization algorithm to generate the controls, rather than relying on analytical methods.

To begin, we will introduce our robot platform, emphasizing its key features and capabilities. Following that, we will present a model that approximates the robot's position based on its velocities. Finally, we will outline the approach we used to generate the appropriate controls for the robot, considering the trajectory and other relevant factors.

2.9.2 Differential Drive Robot and the Odometric Model

The motorized wheelchair, which serves as the mobile robot for our study, incorporates a differential drive system to control its movement. This system consists of two separate wheels, each driven by its own motor. Unlike traditional vehicles with a single motor driving both wheels, the differential drive system allows independent control of the two wheels.

By altering the speed and direction of each drive wheel, a differential drive robot can execute various types of maneuvers. For instance, if both wheels rotate at the same speed in opposite directions, the robot can pivot around a central point, enabling it to turn in place. If the wheels rotate at different speeds in the same direction, the robot will move along a curved path.

The differential drive configuration offers several advantages. Firstly, it simplifies the mechanical design by requiring fewer moving parts compared to other drive systems. This results in a more compact and lightweight robot. Secondly, it provides excellent maneuverability, as the robot can easily navigate tight spaces and perform agile movements.

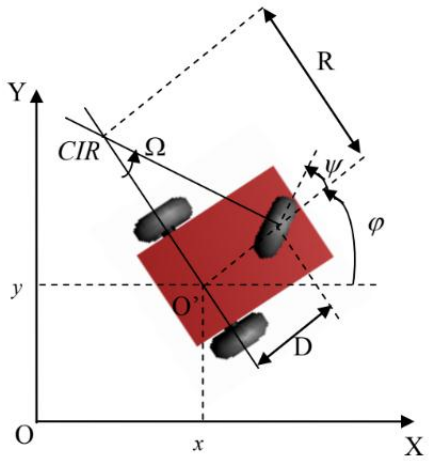


Figure 2.8 Representation of a Non-holonomic Differential Drive Mobile Robot

The cinematic model for a differential drive robot is a mathematical representation that describes the relationship between the robot's wheel velocities and its linear and angular motion. It allows us to predict the robot's position and orientation based on the inputs given to its wheels.

$$\dot{x} = v.\cos\varphi \tag{2.9}$$

$$\dot{y} = v.\sin\varphi \tag{2.10}$$

$$\dot{\varphi} = \Omega = \frac{v}{D}.\tan\psi \tag{2.11}$$

Where v is the speed of the caster wheel, φ The instantaneous rotation of the robot with respect to the reference frame (O, X, Y), Ω is the angular velocity of the caster wheel, D is the distance between the center of the two drive wheels and the caster wheel, and ψ is the instantaneous orientation of the caster wheel.

So, the position of the robot is given by:

$$x(t) = \int_0^t v(\sigma)\cos\big(\varphi(\sigma)\big)d\sigma \tag{2.12}$$

$$y(t) = \int_0^t v(\sigma)\sin(\varphi(\sigma))d\sigma \tag{2.13}$$

$$\varphi(t) = \int_0^t \Omega(\sigma)d\sigma \tag{2.14}$$

When considering the kinematic model of a differential drive system, the integration process to obtain the positions using equations (2.12), (2.13) and (2.14) can be troublesome and difficult for several reasons. Firstly, the complexity and accuracy involved in solving the integrals numerically or approximating them can be computationally expensive and introduce errors, particularly in real-time or complex trajectory scenarios. Additionally, the integration amplifies any measurement errors or inaccuracies in the input velocities and angular velocities, resulting in significant differences between the estimated position and the actual position of the robot. Instead, the odometric model, represented by equations (2.15), (2.16), and (2.17), offers a simpler alternative. By relying on incremental changes in position and orientation based on known wheel displacements, the odometric model eliminates the need for integration, reducing the accumulation of errors. Furthermore, the odometric model allows for real-time estimation, as it only requires information about the current and previous states, making it more practical for applications that demand quick updates and responsiveness. Due to its simplicity, reasonable accuracy for short-term estimates, and widespread use in robotics for tasks like localization and mapping, the odometric model is often preferred over the integration-based approach in differential drive systems.

$$x_k = x_{k-1} - \delta d.\sin\left(\varphi_{k-1} + \frac{\delta\varphi}{2}\right) \tag{2.15}$$

$$y_k = y_{k-1} + \delta d.\cos(\varphi_{k-1} + \frac{\delta\varphi}{2}) \tag{2.16}$$

$$\varphi_k = \varphi_{k-1} + \delta\varphi \tag{2.17}$$

2.9.3 Control Generation

The use of the Whale Optimization Algorithm (WOA) for control generation is straightforward. In this approach, each particle is represented as a variable array consisting of control pairs, namely linear (v) and angular (w) controls. Additionally, a time parameter (t) is included to specify the duration of the control execution.

This optimization technique is employed for each segment of the path. Consequently, between every successive waypoints on the path, a set of controls is determined to bridge the gap. The variable nature of these controls enables the execution of more complex maneuvers and facilitates non-linear behaviors between two consecutive nodes. To accomplish this, the objective function incorporates the odometric model formulas (2.15, 2.16, and 2.17) to accurately estimate the future positions of the robot. This estimation helps in achieving the appropriate set of controls to transition from one configuration to another, while also ensuring that the trajectory remains free from collisions.

The outcome of this problem-solving process is an array of velocity pairs, along with the corresponding time intervals during which these controls should be applied. It is worth noting that this approach has produced positive results.

Algorithm 6: $[(v_0, w_0, t_0), (v_1, w_1, t_1) \dots (v_n, w_n, t_n)] \longleftarrow \text{GenerateControls}(x_i, x_{i+1})$

$Population \longleftarrow \text{InitializePopulation}(Path)$
$C^* \longleftarrow \text{CalculateFitness}()$
while $(t < \text{MaxIterations})$:
    for *each search agent* ***do***:
        $\text{CoefficientUpdate}(a, A, C, l, p)$
        if $p < 0.5$ ***then***:
            if $|A| < 1$ ***then***:
                Update agent by Shrinking Encircling Mechanism Eq. (2.2)
            if $|A| \geq 1$ ***then***:
                Update agent by Seach for Prey Eq. (2.8)
        if $p \geq 0.5$ ***then***:
            Update agent by Spiral Updating Position Eq. (2.6)
    $Population \longleftarrow BoundaryCheck()$
    $C^* \longleftarrow CalculateFitness()$
    $t = t + 1$
return C^*

2.10 Conclusion

In this chapter, we delved into a detailed examination of the Rapidly Exploring Random Tree (RRT) algorithm and its variants. We also explored the notable improvements introduced by the RRT* variant and gained a comprehensive understanding of the Whale Optimization Algorithm.

Additionally, we proposed two variants that combine the Whale Optimization Algorithm with RRT*. The first variant is a local approach that focuses on the expansion process, while the second variant is a global approach that operates on the path generated by RRT*. By incorporating these variants, we aim to enhance the performance and efficiency of the overall algorithm.

To conclude the chapter, we discussed the kinematic modeling of our robot platform and emphasized the significance of the odometric model in aiding us in generating controls for the planned path. Understanding the robot's kinematics and leveraging the odometric model is crucial for accurately implementing and executing the planned path.

In the upcoming chapter, we will shift our focus to the simulation setup, where we will meticulously analyze the proposed variants. Our analysis will involve benchmarking these variants against the original RRT* algorithm, enabling us to assess their comparative performance and evaluate their effectiveness in achieving our objectives.

Chapter 3

Simulation, Comparative Analysis, and Implementation

3.1 Introduction

In this chapter, we will provide a comprehensive overview of the structure used to implement the algorithms, namely RRT* and the proposed variants (local and global). Additionally, we will discuss the image processing library utilized to retrieve obstacle information, as well as the various types of obstacles encountered in an environment and how our system responds to each of them.

Following this, we will present the experimental setup employed for simulation purposes, detailing the methodology utilized for conducting benchmarks. We will then proceed to conduct a comparison of the three algorithms: RRT*, the local variant, and the global variant. Subsequently, based on our initial findings, we will delve deeper into an extensive benchmark analysis of the two variants. This analysis will involve varying several parameters of the Whale Optimization Algorithm and meticulously examining the outcomes. We will discuss our discoveries and insights resulting from this process.

Lastly, we will address the hardware and software aspects of this project. We will introduce the underlying functioning of the Robot Operating System (ROS) and emphasize its pivotal role in our implementation. To conclude this chapter, we will present the results obtained and explore their implications.

3.2 Simulation

This section will focus on software implementation of the following algorithms: RRT*, the local proposed variant, and the global proposed variant. Additionally, we will evaluate the approach performances in static and dynamic environments. Finally, we will showcase the simulation results.

3.2.1 Algorithm Implementation

The algorithms were implemented in C++ using an object-oriented approach and a modularized structure. Since they share the RRT* algorithm, ensuring code reusability

was a crucial step in facilitating the software development. Therefore, we adopted a module framework throughout our project, which consists of the following modules:

Map Processing Module: It handles the processing of map data, such as loading and representing the environment in a suitable format for motion planning to take place.

Planning Module: It is responsible for generating paths or trajectories for the given map using RRT* or any of the variants.

Optimization Module: It utilizes the WOA (Whale Optimization Algorithm) to enhance space exploration in the local variant and to refine the generated trajectory in the global variant.

Control Generation Module: It focuses on converting the planned path into suitable control inputs for the robot ensuring successful execution of that motion.

We integrated JSON files for configuration and calibration, simplifying the process of storing and retrieving simulation parameters. The figure 3.1 illustrates an example of the configuration file for the global variant.

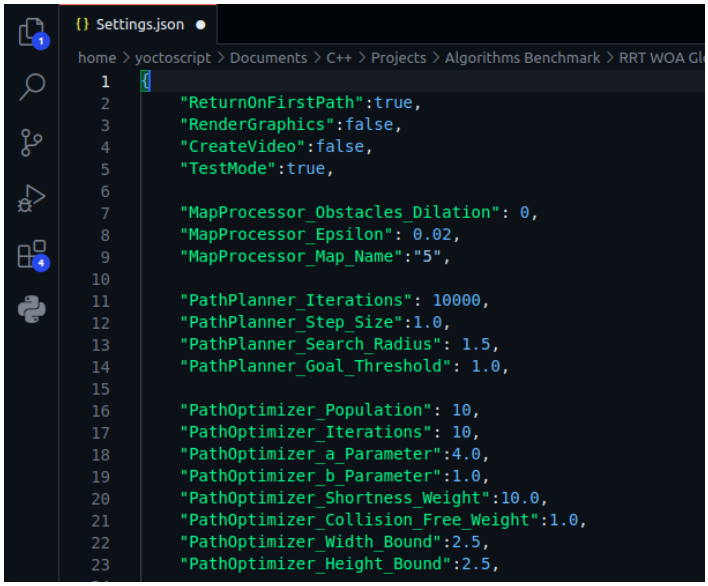


Figure 3.1 – Configuration File of the Global Variant

3.2.2 Obstacle Avoidance

To enable obstacle avoidance, we utilized OpenCV (Open Computer Vision) library to binarize the workspace once the map was acquired. In this process, white areas indicates free space, while black areas represent occupied obstacles. Additionally, the library was employed to transform all existing obstacles within the workspace into a set of polygons. During the expansion and optimization phases, the segments of these polygons were checked for intersection, ensuring the generation of collision-free trajectories.

To further enhance collision avoidance and prevent close proximity to obstacles, we implemented an additional measure known as obstacle inflation through a dilation function provided by OpenCV. This involves expanding the size of the obstacles, effectively creating a safety buffer around them. By increasing the size of the obstacles, we create a larger margin for the robot to navigate around them, reducing the risk of potential collisions. An example is depicted in figure 3.2 where obstacle boundaries are retrieved.

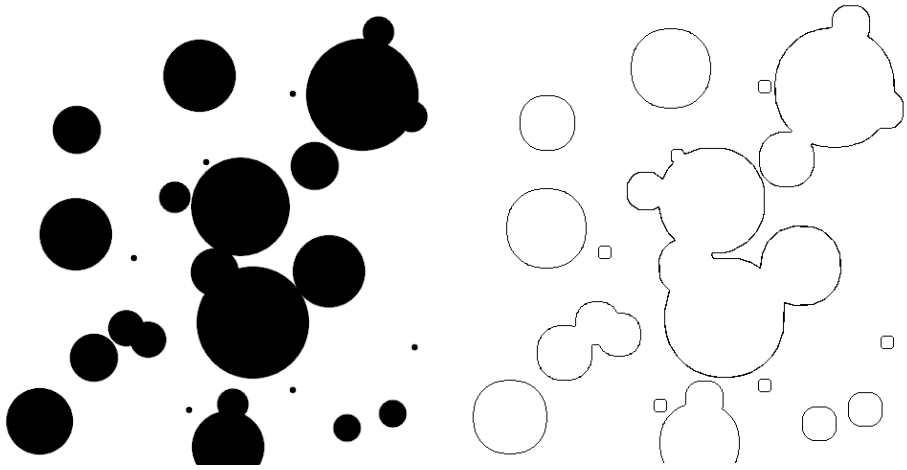


Figure 3.2 – Map Polygon Approximation with 3-Pixel Dilation

When it comes to dealing with obstacles, we have implemented various behaviors for our system based on the nature of the obstacles. They can be classified into three main types:

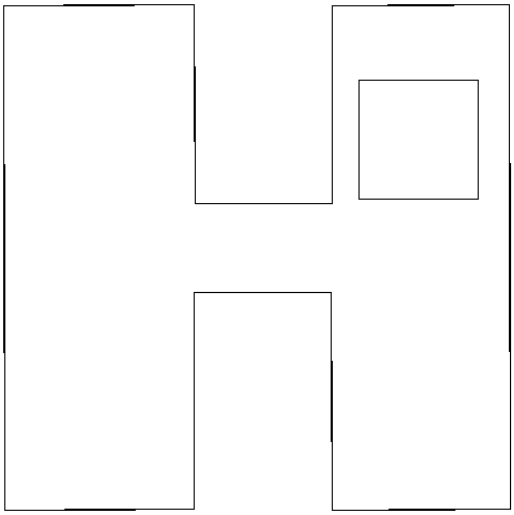
Static Obstacles: Static obstacles refer to stationary objects that remain in a fixed position. Our system takes them into account during the motion planning process, ensuring that they are avoided while determining the optimal trajectory to the goal.

Temporary Obstacles: Temporary obstacles are obstructions that are present for a limited period of time. Whenever our system encounters such an obstacle, it promptly reevaluates the entire trajectory to the goal. By incorporating the new obstacle into consideration, it recalculates a new motion that avoids this temporary obstacle and allows the system to continue towards the intended destination.

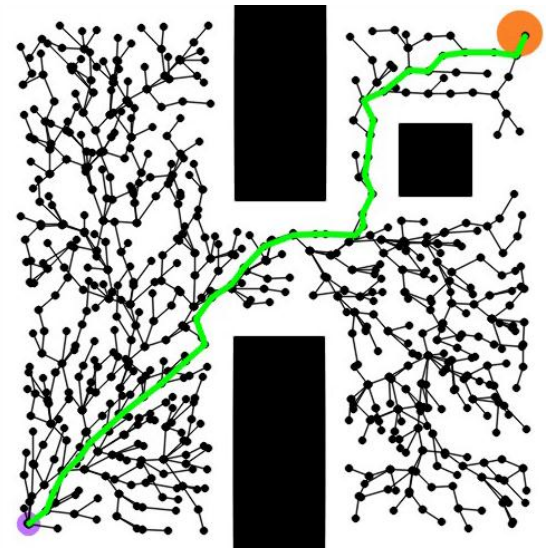
Dynamic Obstacles: Dynamic obstacles include objects or entities that are in motion and can change their position, speed, and direction. To address the uncertainty associated with the behavior of these objects, we have chosen to halt the movement of the robot altogether when approaching a dynamic obstacle. This precautionary measure ensures the safety of the system and prevents any potential collisions or undesired interactions.

3.2.3 Results

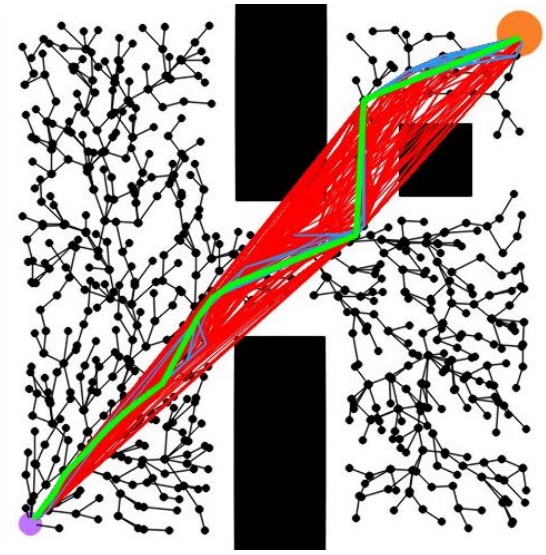
The Figure 3.3 illustrates the workflow of the global variant's simulation. Initially, the map is processed to identify all static obstacles present within the workspace. Subsequently, the Rapidly-exploring Random Tree Star (RRT*) algorithm is employed to find a collision-free motion from the starting point to the destination. In the third step, the Whale Optimization Algorithm (WOA) is utilized to optimize the path with respect to its length. Finally, the necessary controls for guiding the robot along the optimized trajectory are generated.



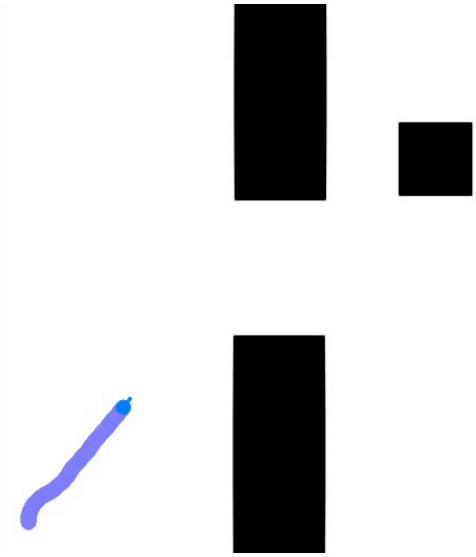
Map Processing: Retrieving Obstacle Information



Planning: Generating a Collision-free Motion from Initial to Goal Position



Optimization: Enhancing Path Length and Smoothness



Control Generation: Generating Necessary Velocities to Execute The Motion

Figure 3.3 – Simulation Workflow of the Proposed Global Variant

When faced with temporary obstacles along the path, the system chooses to fully replan the motion towards the goal from its current location. The replanning process depicted in figure 3.4 below, showcases the system's adaptive and dynamic nature in responding to real-time challenges.

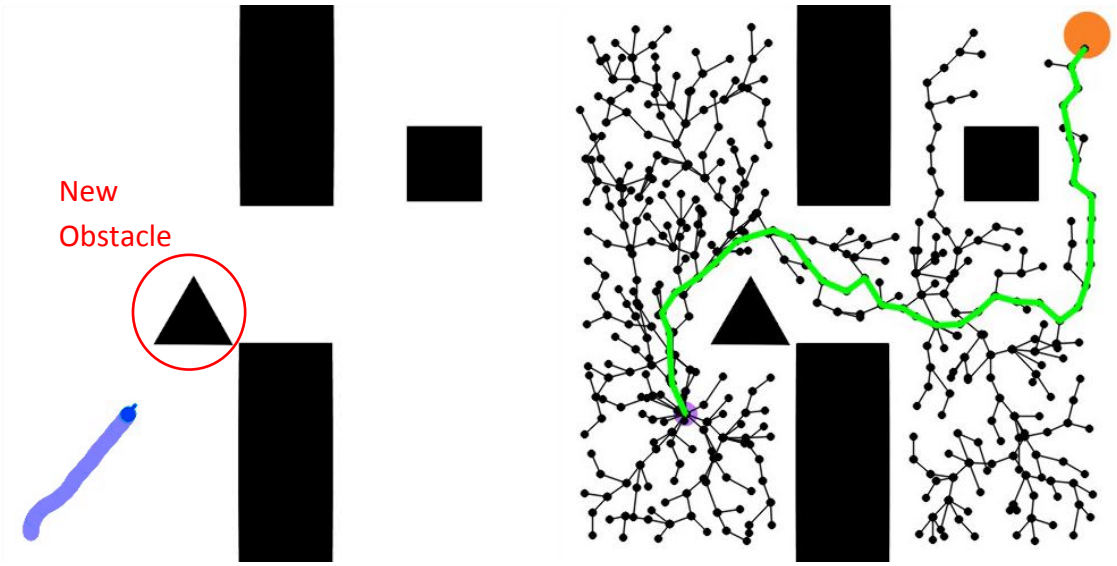


Figure 3.4 – Adaptive Replanning: Responding to Environmental Changes

3.3 Comparative Analysis

In this section, we will present the experimental setup utilized to perform the benchmarks. We will also provide a detailed comparison between the classical RRT* algorithm and the proposed variants. Additionally, we will examine the influence of varying the parameters of the Whale Optimization Algorithm (WOA) on the performance of these variants.

3.3.1 Experimental Setup

The experimental setup for the benchmarks conducted in this study involved running the motion planning algorithms on an i3-11th gen 3.00GHz Quad-Core laptop with 4 GB of memory. To ensure reliable results, each reported result is an average of 100 runs conducted on each map.

The experimental map set used in this study was created using the image manipulation program GIMP. The maps were designed to test the performance of the proposed algorithms in various environments. These environments encompassed both simple and complex scenarios.

The simple environments consisted of empty spaces and areas occupied by unobtrusive obstacles (map 1 and 2), allowing for a baseline evaluation of the

algorithms' basic path planning capabilities. On the other hand, the complex environments incorporated additional challenging features (map 3, 4 and 5). These features included narrow passages, local minima, and cluttered areas, which aimed to observe and analyze the algorithms' behaviors in more intricate and demanding scenarios.

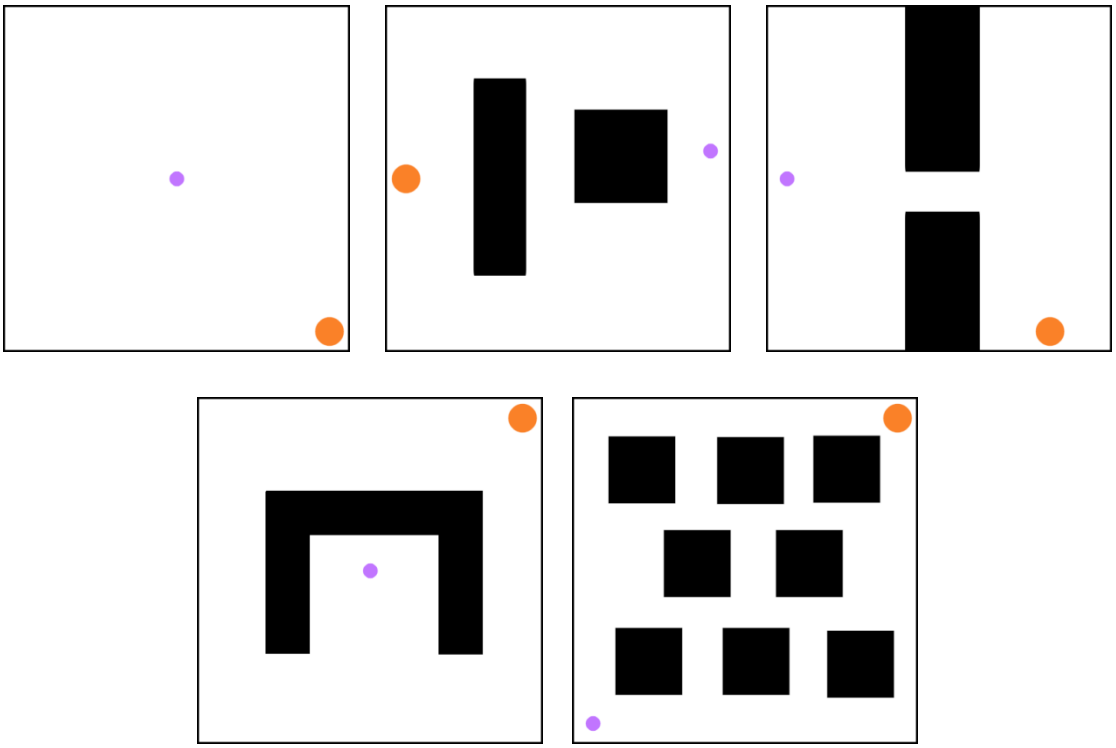


Figure 3.5 – Experimental Map Set: The Purple Dot Represents the Initial Position of the Robot, while the Orange Dot Indicates the Goal Position

3.3.2 Algorithms Assessment

The table below presents benchmarking results of the motion planning algorithms. It compares the original RRT* algorithm with two proposed variants.

Algorithm	Time (s)	Length (m)	Nodes
RRT*	0.026	35.268	409.258
Local Variant	1.440	27.132	167.378
Global Variant	0.072	30.656	410.598

Table 3.1 – Experimental Results: Analysis of RRT*, Local and Global Variants' Performance

Upon analysis, the local and global variants of the RRT* algorithm, incorporating the WOA technique, have demonstrated notable performance improvements compared to the original RRT* algorithm.

The local variant of the RRT* algorithm demonstrated superior performance compared to the original algorithm. It achieved a significantly shorter path length of 27.132 meters, compared to the path length of 35.268 meters obtained by the RRT* algorithm. Despite taking slightly longer to compute, with a return time of 1.440 seconds due to the execution of WOA on every expansion, the local variant exhibited improved efficiency by generating only 167.378 nodes. These findings highlight the effectiveness of the local variant in optimizing path length and enhancing space exploration.

Similarly, the global variant of the RRT* algorithm demonstrated significant improvements over the original algorithm. Although it took a slightly longer time of 0.072 to return the path, it managed to achieve a shorter path length of 30.656 meters, compared to the original algorithm's path length of 35.268 meters. Remarkably, the global variant accomplished this while generating a similar number of nodes. These results highlight the effectiveness of the global variant in optimizing path length while maintaining a comparable level of exploration in the search space.

These findings suggest that both variations of the WOA algorithm, namely the local and global variants, have demonstrated promising improvements in optimizing path length and efficiently exploring the search space. However, it is important to consider the trade-off associated with increased computation time. Figure 3.6 provides a visual representation of the improvements achieved by these variants.

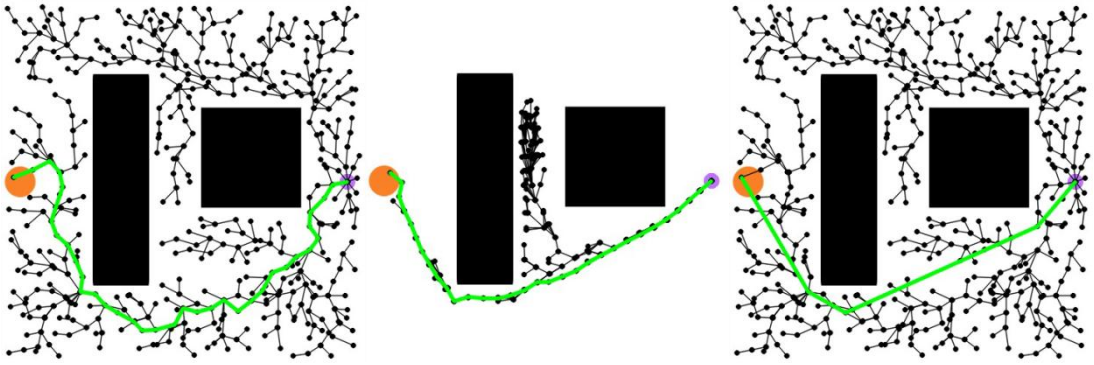


Figure 3.6 – Visual Comparison of Paths: RRT*, Local Variant and Global Variant

3.3.3 Parameter Tuning

In this segment, our objective is to thoroughly examine and analyze the influence of adjusting key parameters in the Whale Optimization Algorithm (WOA) on the performance of the proposed variants, both local and global, while focusing on metrics such as computation time, path length and nodes generated. Moreover, by exploring the impact of parameters like population size, number of iterations and other relevant factors, we aim to gain an understanding of the optimal configurations that yield the most favorable outcomes.

All parameters are fixed except for the parameter currently undergoing variation.

A. Varying Population

Varying the population size in the WOA algorithm impacts its performance. A larger population enables more extensive exploration of the solution space, potentially yielding better solutions. However, it comes with increased computation time.

Local Variant

Population	Time (s)	Length (m)	Nodes
10	0.4112	35.2668	329.9866
20	0.5120	34.9051	219.2681
30	0.6930	34.8108	201.9980
40	0.8151	34.6556	179.9042
50	0.9948	34.5616	175.8849

60	1.1826	34.5598	175.1680
70	1.4912	34.6016	173.3743
80	1.6152	34.5794	177.6069
90	1.8053	34.4544	174.1088
100	1.9976	34.4393	172.7764

Table 3.2 – Results of Population Size Variation in Local Variant

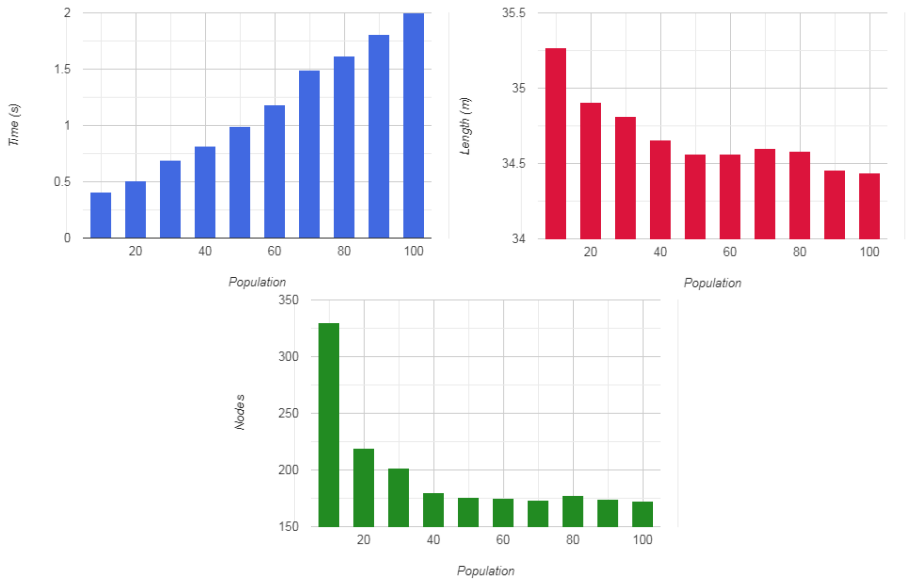


Figure 3.7 - Analysis of Population Variation in Local Variant

Based on Figure 3.7 and Table 3.2 above, it is evident that increasing the population size in the local variant leads to longer computation times. However, this increase in population size also results in decreased path lengths, indicating improved solution quality. Furthermore, a larger population size enables more efficient exploration of the search space with fewer generated nodes.

Global Variant

Population	Time (s)	Length (m)	Nodes
10	0.0391	35.7192	411.2046
20	0.0614	35.3398	409.0541
30	0.0694	35.1144	407.9880

40	0.0848	35.1648	403.1343
50	0.1052	35.0045	408.2663
60	0.1152	35.0506	407.1966
70	0.1300	34.9082	412.9768
80	0.1448	34.9326	402.4641
90	0.1606	34.8842	409.8182
100	0.1762	34.7814	409.1384

Table 3.3 – Results of Population Size Variation in Global Variant

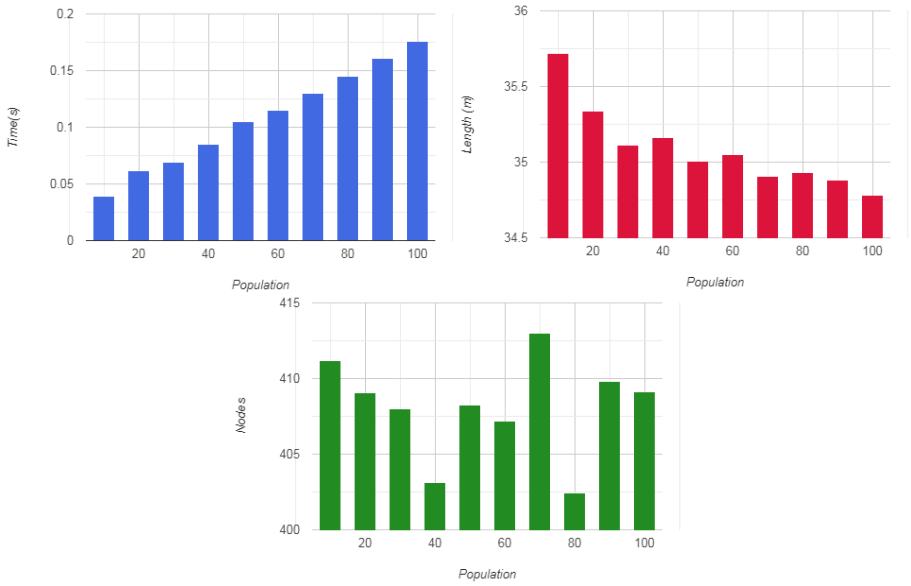


Figure 3.8 – Analysis of Population Variation in Global Variant

As demonstrated by Figure 3.8 and Table 3.3, it can be observed that increasing the population size in the global variant leads to slightly longer computation times. However, it also results in fluctuating but decreasing values of length, indicating a slight improvement in solution quality. On the other hand, a larger population size does not necessarily guarantee more efficient exploration of the search space, as the number of generated nodes fluctuates without a clear trend.

B. Varying Iterations

Varying iterations in the Whale Optimization Algorithm affects exploration. More iterations lead to better exploration but increase the computational time.

Local Variant

Iterations	Time (s)	Length (m)	Nodes
10	0.4110	35.2260	326.9866
20	0.5912	35.0804	247.3840
30	0.7836	35.1534	229.2912
40	1.0902	34.9312	237.5325
50	1.2914	34.9588	224.6928
60	1.4906	34.9626	215.1647
70	1.7073	34.9158	212.1142
80	1.9996	34.9718	217.5400
90	2.2538	34.8430	215.3588
100	2.5102	34.9368	216.6423

Table 3.4 – Results of Iterations Variation in Local Variant

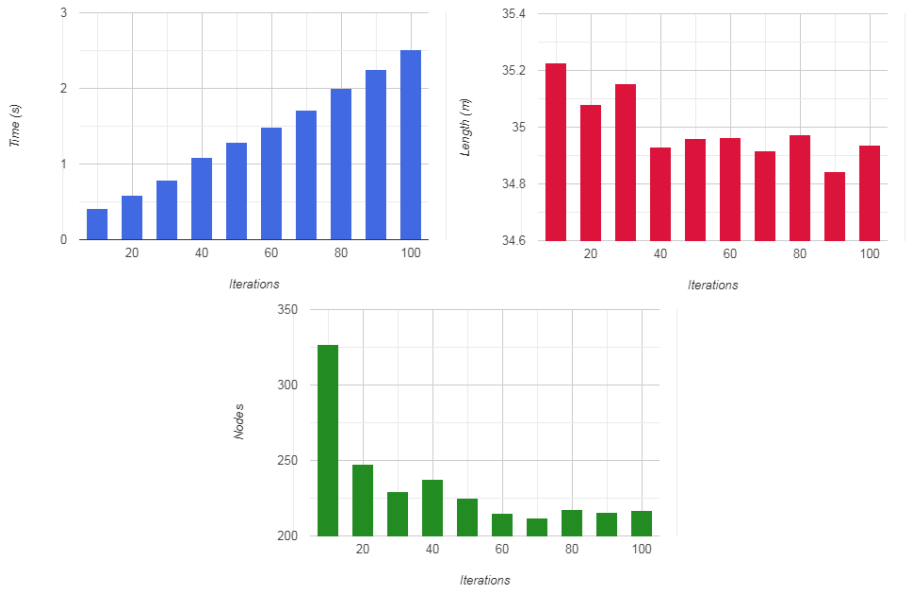


Figure 3.9 - Analysis of Iterations Variation in Local Variant

As demonstrated by Figure 3.9 and Table 3.4, it is evident that increasing the number of iterations in the local variant leads to significantly longer computation times. However, this increase in population size also results in fluctuating values of length, without a clear trend, yet an overall decrease can be observed. Interestingly, a larger number of iterations in the local variant lead to more efficient exploration of the

search space, as the number of generated nodes fluctuates but displays an exponential decrease.

Global Variant

Iterations	Time (s)	Length (m)	Nodes
10	0.0399	35.7192	411.2049
20	0.0510	35.0342	394.9081
30	0.0654	34.7646	411.1840
40	0.7683	34.6472	408.1522
50	0.8881	34.4450	407.1823
60	0.1014	34.4542	402.3282
70	0.1148	34.3808	416.7160
80	0.1278	34.3528	418.2768
90	0.1386	34.2745	406.1363
100	0.1518	34.3318	395.7544

Table 3.5 – Results of Iterations Variation in Global Variant

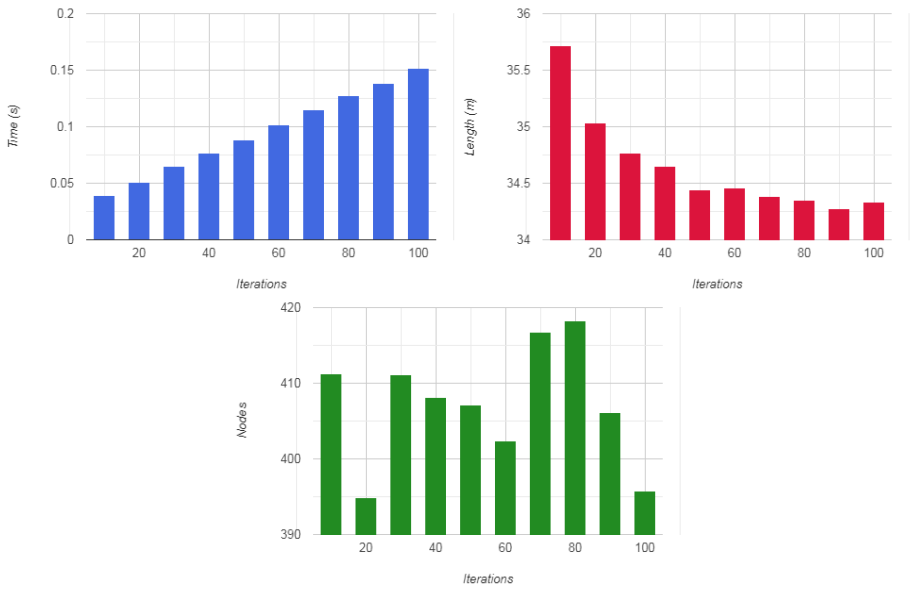


Figure 3.10 - Analysis of Iterations Variation in Global Variant

As demonstrated by Figure 3.10 and Table 3.5, increasing the iterations in the algorithm leads to consistently greater computation times. However, it also results in exponentially decreasing values of length with minimal fluctuations. Surprisingly,

despite the decreasing length values, more iterations do not necessarily lead to more efficient exploration of the search space, as the number of generated nodes fluctuates without a clear trend.

C. Varying μ

The variable μ serves as the threshold for testing the random parameter p, determining whether the algorithm should choose the first behavior, the Shrinking Encircling Mechanism or Search for Prey, or the second behavior, Spiral Updating Position. Lower values of this threshold provide a higher probability for the second behavior, while greater values favor the first behavior. The Figure 3.11 below provides a visual representation of this concept.

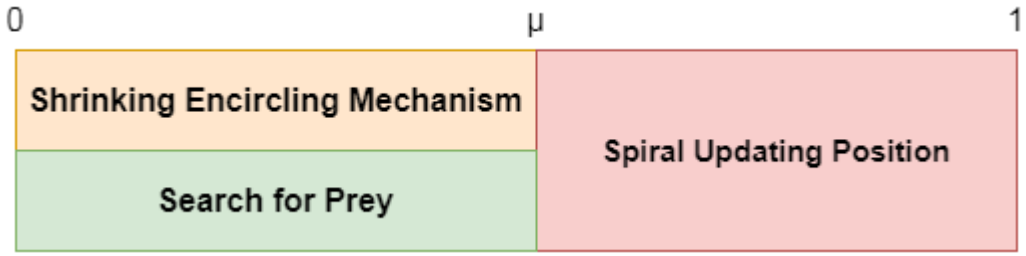


Figure 3.11 – The Behavior Associated with Parameter p Value

Local Variant

μ	Time (s)	Length (m)	Nodes
0.0	0.3731	35.6852	306.9522
0.1	0.3789	35.5608	310.7000
0.2	0.3902	35.5272	314.3561
0.3	0.4016	35.4266	323.1766
0.4	0.3968	35.4346	317.6481
0.5	0.4032	35.3592	318.4669
0.6	0.4042	35.3334	317.3785
0.7	0.4182	35.3378	325.9347
0.8	0.4022	35.3646	309.4812
0.9	0.4036	35.3190	310.8347
1.0	0.4098	35.2666	302.6145

Table 3.6 – Results of μ Threshold Variation in Local Variant

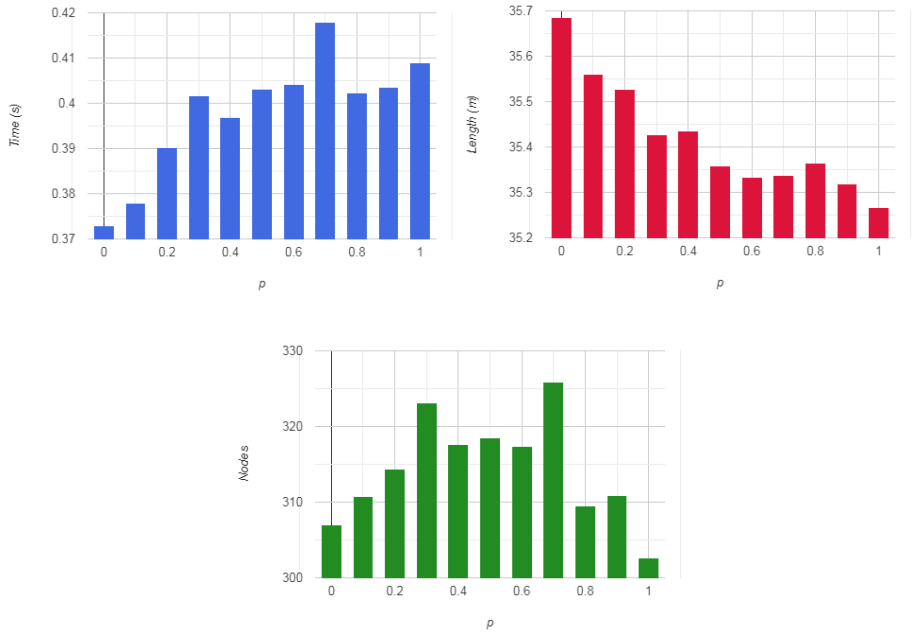


Figure 3.12 - Analysis of μ Threshold Variation in Local Variant

As Figure 3.12 and Table 3.6 demonstrate, the behavior of computation time does not exhibit a clear trend. However, it is noticeable that lower thresholds result in lower computation times, while higher thresholds lead to longer computation times. On the other hand, increasing the threshold in the algorithm results in exponentially decreasing values of length with minimal fluctuations. Interestingly, despite the decreasing length values, the increase in threshold does not lead to more efficient exploration of the search space, as the number of generated nodes fluctuates without a clear trend.

Global Variant

μ	Time (s)	Length (m)	Nodes
0.0	0.0396	38.4018	411.9044
0.1	0.0386	36.7648	407.1043
0.2	0.0382	36.4454	399.5581
0.3	0.0386	35.9325	400.7561

0.4	0.0400	35.8574	433.7714
0.5	0.0376	35.5156	319.9583
0.6	0.0381	35.4291	406.4640
0.7	0.0384	35.3558	408.0682
0.8	0.0378	35.2112	405.3261
0.9	0.0394	35.1938	417.4583
1.0	0.0376	35.0976	403.7628

Table 3.7 – Results of μ Threshold Variation in Global Variant

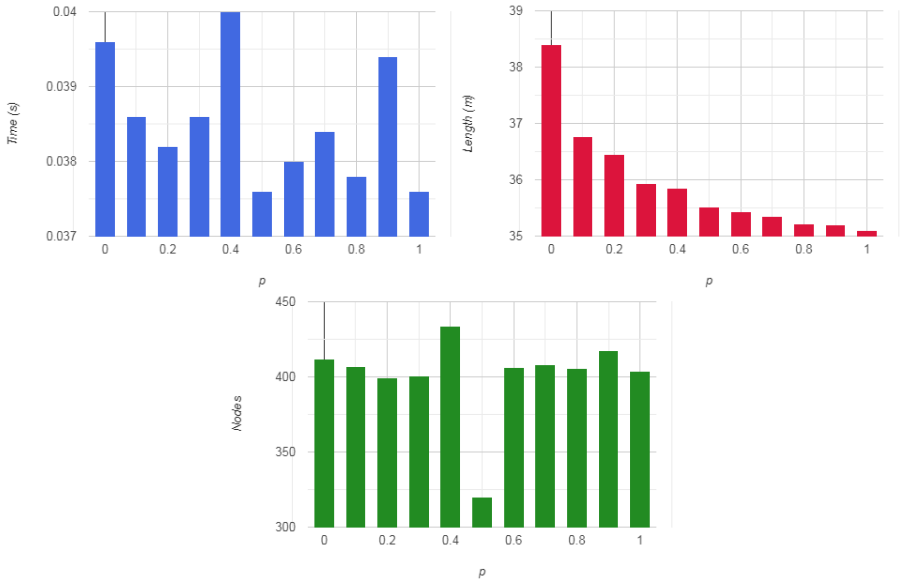


Figure 3.13 – Analysis of μ Threshold Variation in Global Variant

As demonstrated by Figure 3.13 and Table 3.7, varying the threshold does not have a clear impact on the behavior of computation time. However, it does result in exponentially decreasing values of length with an increase in the threshold. However, despite the increase in threshold, it does not lead to more efficient exploration of the search space. This is because the number of generated nodes fluctuates throughout the graph, with higher values observed throughout the entirety of the graph and a single instance of a significant drop at the center.

D. Varying b

The parameter b defines the shape of the logarithmic spiral in the Spiral Updating Position behavior [39].

Local Variant

b	Time (s)	Length (m)	Nodes
1	0.3972	35.4062	312.4426
2	0.3904	35.4784	310.2246
3	0.3904	35.3436	313.6448
4	0.4076	35.4213	323.9785
5	0.4354	35.4862	346.2684
6	0.4200	35.5178	333.1662
7	0.4416	35.5286	347.6088
8	0.4257	35.5112	337.0482
9	0.4404	35.4542	347.3921
10	0.4466	35.4415	351.3527

Table 3.8 – Results of Parameter b Variation in Local Variant

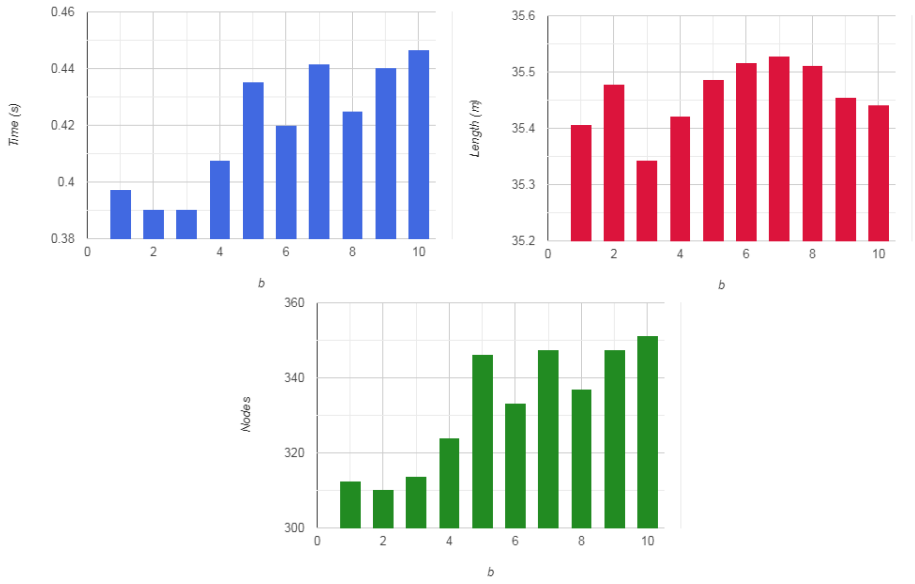


Figure 3.14 – Analysis of Parameter b Variation in Local Variant

As demonstrated by Figure 3.14 and Table 3.7, increasing the value of the b parameter does not exhibit a clear trend in computation time, but it does show an overall increase. However, it results in fluctuating values of length that are generally similar. Interestingly, the number of generated nodes exhibits the same behavior as computation time, indicating inefficient exploration of the search space.

Global Variant

b	Time (s)	Length (m)	Nodes
1	0.0400	35.5972	406.7246
2	0.0378	35.5724	400.8224
3	0.0394	35.4664	415.3000
4	0.0392	35.5938	412.7943
5	0.0388	35.5549	410.0640
6	0.0392	35.6272	414.5662
7	0.0388	35.5488	410.4026
8	0.0392	35.5436	410.5246
9	0.0381	35.5943	396.5448
10	0.0374	35.5213	393.2330

Table 3.9 – Results of Parameter b Variation in Global Variant

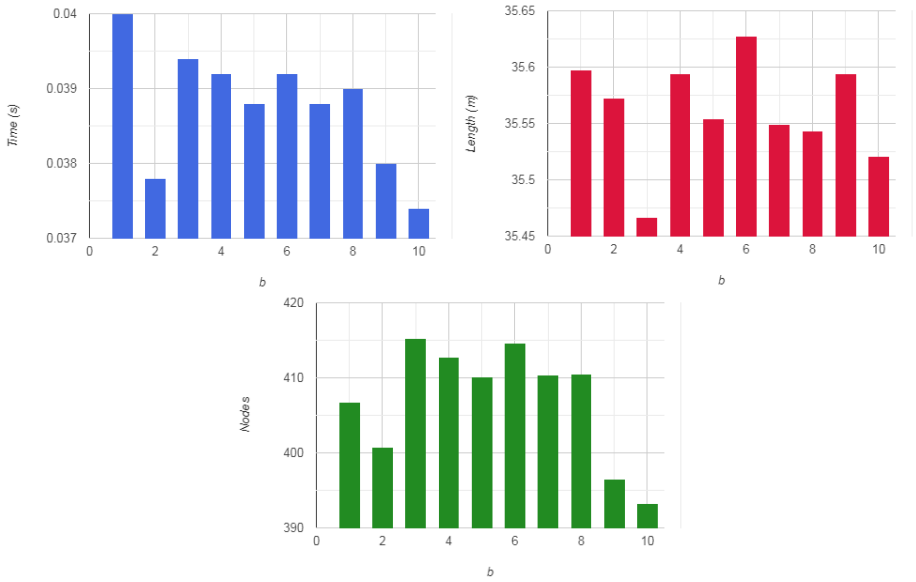


Figure 3.15 - Analysis of Parameter b Variation in Global Variant

As demonstrated by Figure 3.15 and Table 3.8, increasing the value of the b parameter in the global variant does not exhibit a clear trend in computation time, length of the path, or the number of generated nodes.

E. Varying a

The algorithm's behavior when $p < 0.5$ is contingent upon the random parameter A, which spans a range from $-a$ to $+a$. If the absolute value of A is less than 1 (|A| < 1), the algorithm adopts a Shrinking Encircling Mechanism, which signifies a form of solution exploitation. Conversely, if the absolute value of A exceeds 1 (|A| > 1), it opts for a Search for Prey approach that prioritizes exploration.

To visualize these behaviors, refer to Figure 3.16 below, which illustrates the zones of exploration and exploitation corresponding to different values of A.

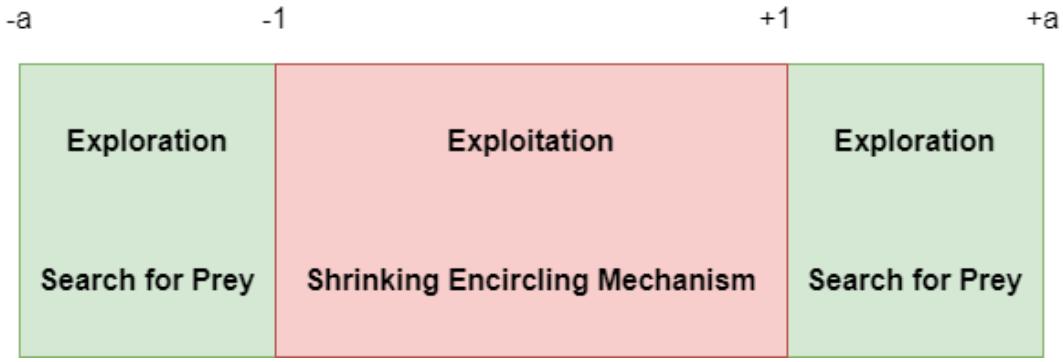


Figure 3.16 – The Behavior Associated with Absolute Value of A with Fixed Threshold

Local Variant

a	Time (s)	Length (m)	Nodes
1	0.4160	35.3606	354.5611
2	0.3972	35.4062	312.4426
3	0.3986	35.4462	313.3360
4	0.4100	35.4514	318.9224
5	0.4432	35.4678	346.8369
6	0.4338	35.5494	339.5965
7	0.4471	35.5688	347.8200
8	0.4752	35.5798	368.2943
9	0.4729	35.6766	366.5369
10	0.5058	35.6948	391.8987

Table 3.10 – Results of Parameter a Variation in Local Variant

Table 3.9 presents the results indicating the impact of increasing the parameter a. It can be observed that as a increases, the time required fluctuates but remains relatively constant. The length, however, exhibits a gradual but slight increase. On the other hand, no clear trend is discernible when it comes to the number of nodes generated.

Global Variant

A	Time (s)	Length (m)	Nodes
1	0.0400	35.5200	400.5411
2	0.0402	35.5972	406.7246
3	0.0398	35.6942	404.6527
4	0.0382	35.8951	405.6246
5	0.0382	35.9034	402.9347
6	0.0394	35.9527	415.8466
7	0.0386	36.0132	405.1663
8	0.0386	36.1188	408.2100
9	0.0382	36.1536	401.6641
10	0.0378	36.2312	399.0600

Table 3.11 – Results of Parameter a Variation in Global Variant

The findings presented in Table 3.10 highlight the influence of increasing the parameter a. It can be observed that as a increases, the time required shows minimal variation. The length, however, demonstrates a gradual but modest increase. However, no clear trend is discernible when it comes to the number of nodes generated.

We observed similar behavior in both variants when the parameter $\boldsymbol{a}$ was varied. Specifically, as the exploration to exploitation ratio increased, there was an increase in path length. To further investigate this phenomenon, we conducted additional experiments by varying both the $\boldsymbol{a}$ parameter and the threshold $\boldsymbol{\lambda}$ for the two behaviors. Our initial hypothesis suggests that achieving optimal path lengths involves striking a balance between the two behaviors or placing emphasis on the exploitation phase.

F. Varying a & A

The tables below demonstrate how the threshold value λ affects the performance of the variants. Increasing the threshold value signifies a higher exploitation to exploration ratio, while lower values indicate a greater emphasis on exploration. Figure 3.17 provides a visual representation of the relationship between these two parameters and behaviors.

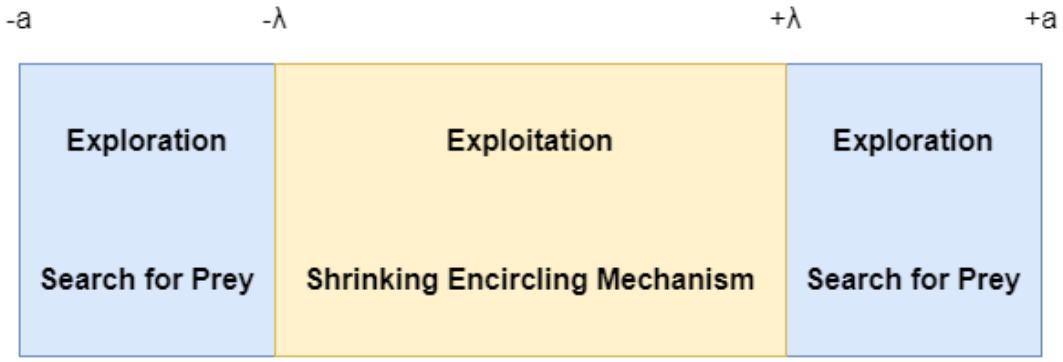


Figure 3.17 – The Behavior Associated with Absolute Value of A with Variable Threshold

Local Variant

a	λ	Time (s)	Length (m)	Nodes
2	**0.0**	0.5708	35.6828	427.7613
	0.2	0.4763	35.4884	357.1549
	0.4	0.4122	35.3508	319.9561
	0.6	0.4434	35.3592	323.8325
	0.8	0.3992	35.4044	313.1066
	1.0	0.4136	35.3394	321.5842
	1.2	0.4066	35.3314	324.2280
	1.4	0.4374	35.3896	326.3268
	1.6	0.4204	35.3222	320.2570
	1.8	0.4276	35.2944	314.5886
	2.0	0.4002	35.3158	319.4081
4	**0.0**	0.6516	35.7036	474.9743

	0.4	0.4648	35.4836	350.1941
	0.8	0.4352	35.4892	336.2662
	1.2	0.4108	35.3772	329.5914
	1.6	0.4000	35.8408	318.4549
	2.0	0.3946	35.3804	319.3347
	2.4	0.3978	35.4024	324.2684
	2.8	0.3988	35.4454	325.9918
	3.2	0.3985	35.2592	322.6309
	3.6	0.4118	35.3778	333.7981
	4.0	0.4028	35.2875	328.3861
6	**0.0**	0.6784	35.6741	505.6981
	0.6	0.4614	35.4664	353.5123
	1.2	0.4332	35.5052	337.9941
	1.8	0.4214	35.4208	331.8741
	2.4	0.4406	35.4168	337.4369
	3.0	0.4222	35.4032	337.156
	3.6	0.4542	35.3544	350.4381
	4.2	0.3944	35.4304	325.4681
	4.8	0.4156	35.3426	346.7881
	5.4	0.4004	35.3471	331.4628
	6.0	0.4256	35.4282	351.202

Table 3.12 – Results of Parameter α and Threshold λ Variation in Local Variant

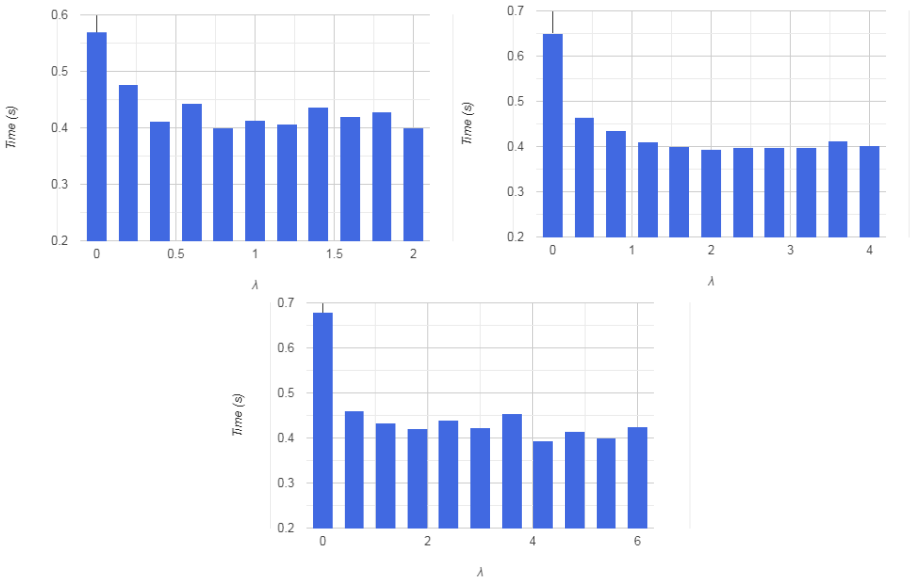


Figure 3.18 - Analysis of Parameter a and Threshold λ Variation in Local Variant in Relation to Computation Time

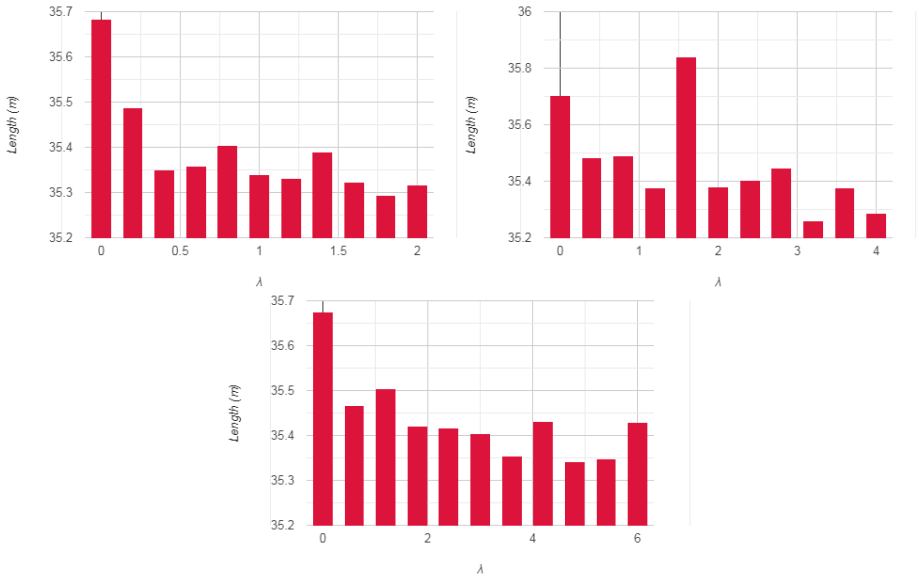


Figure 3.19 – Analysis of Parameter a and Threshold λ Variation in Local Variant in Relation to Path Length

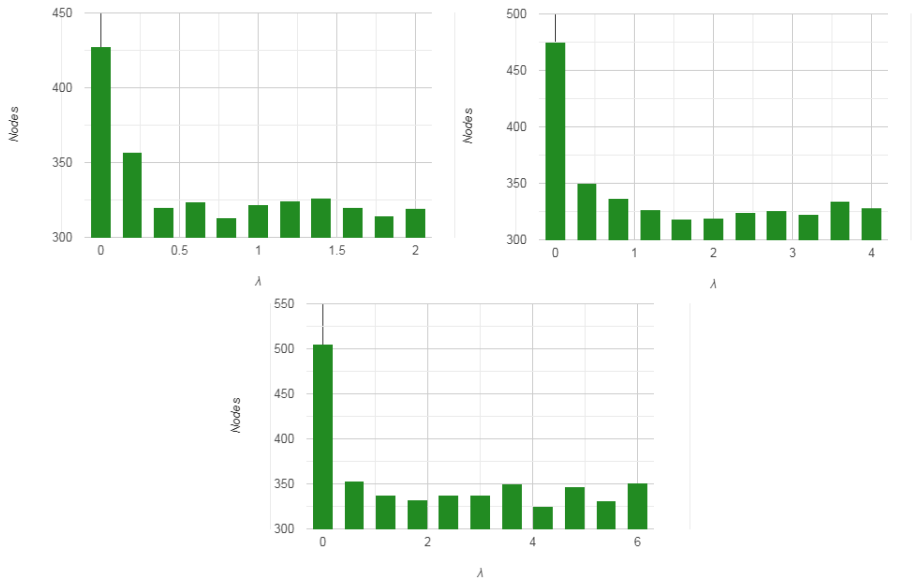


Figure 3.20 - Analysis of Parameter a and Threshold λ Variation in Local Variant in Relation to Nodes Generated

Based on the analysis of the three previous figures, namely 3.18, 3.19, and 3.20, along with Table 3.11, it is evident that achieving a balance between exploration and exploitation, or placing emphasis on exploitation, leads to improved performance across all three metrics: computation time, path length, and the number of nodes generated.

Global Variant

a	λ	Time (s)	Length (m)	Nodes
2	**0.0**	0.0396	37.6369	407.3921
	0.2	0.0392	36.4714	410.0369
	0.4	0.0382	35.9981	401.1300
	0.6	0.0396	35.7939	348.4921
	0.8	0.0388	35.6356	411.4260
	1.0	0.0378	35.3934	399.7000
	1.2	0.0382	35.4446	406.8921
	1.4	0.0382	35.3502	397.7022
	1.6	0.0388	35.3794	401.5001
	1.8	0.0392	35.2898	404.3123

	2.0	0.0384	35.4112	399.8246
4	**0.0**	0.0394	37.7156	405.9246
	0.4	0.0381	36.4264	401.5347
	0.8	0.0396	36.0464	414.9741
	1.2	0.0386	35.7956	406.8741
	1.6	0.0392	35.5626	414.3300
	2.0	0.0382	35.5028	407.0505
	2.4	0.0392	35.5106	410.4786
	2.8	0.0378	35.3182	403.0721
	3.2	0.0388	35.3594	413.2562
	3.6	0.0382	35.5040	404.9145
	4.0	0.0384	35.3800	406.444
6	**0.0**	0.0400	37.5474	419.8246
	0.6	0.0386	36.1036	407.9965
	1.2	0.0388	35.7194	410.7010
	1.8	0.0386	35.5134	408.7628
	2.4	0.0386	35.4098	406.3682
	3.0	0.0391	35.4896	408.4711
	3.6	0.0391	35.2736	412.9801
	4.2	0.0394	35.5206	410.4724
	4.8	0.0384	35.4252	400.4300
	5.4	0.0396	35.5066	403.7866
	6.0	0.0410	35.4574	405.5949

Table 3.13 – Results of Parameter a and Threshold λ Variation in Global Variant

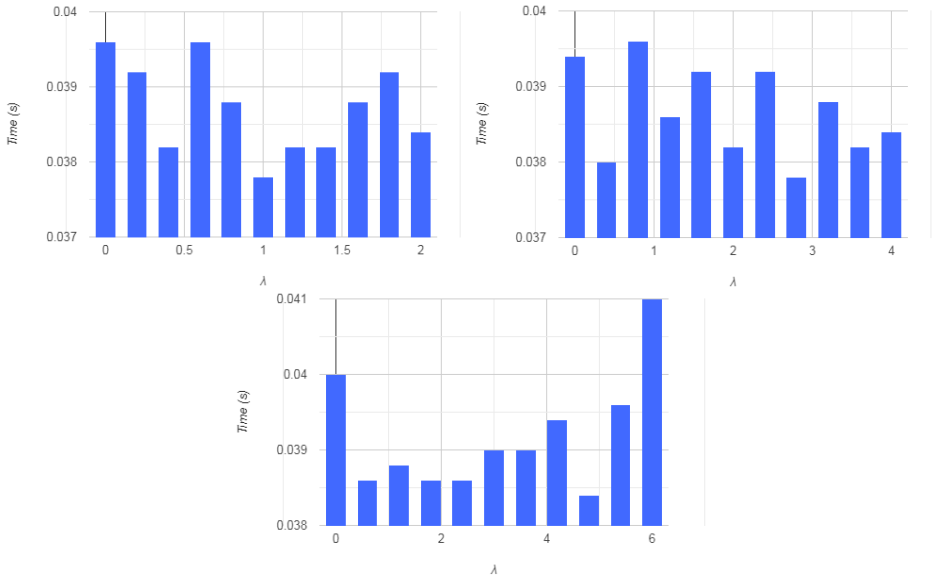


Figure 3.21 - Analysis of Parameter a and Threshold λ Variation in Global Variant in Relation to Computation Time

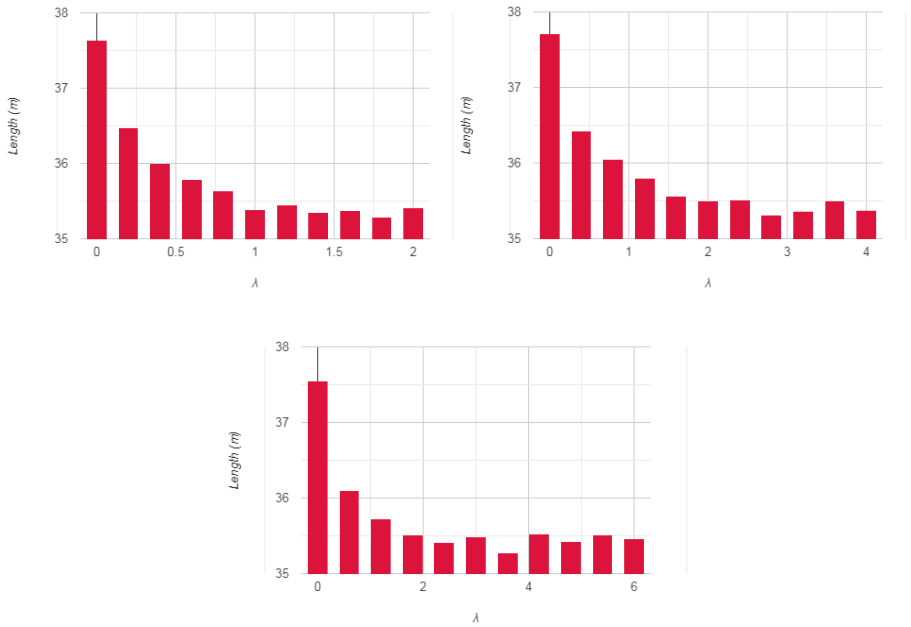


Figure 3.22 - Analysis of Parameter a and Threshold λ Variation in Global Variant in Relation to Path Length

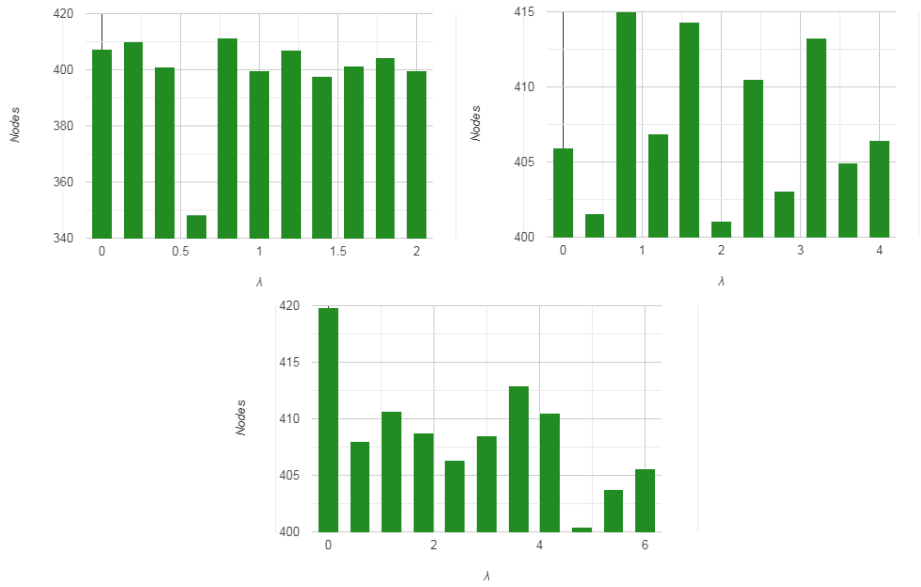


Figure 3.23 - Analysis of Parameter a and Threshold λ Variation in Global Variant in Relation to Nodes Generated

Based on the examination of the three preceding figures, namely 3.21, 3.22, and 3.23, along with Table 3.12, it is evident that finding a balance between exploration and exploitation or emphasizing exploitation yields varying results with no consistent trend in both computation time and the number of nodes generated. However, it undeniably demonstrates superior performance in terms of reducing the path length.

3.3.4 Discussion

In this section, we conducted multiple analyses to assess the impact of parameters on the performance of both the local and global variants. Initially, the local variant demonstrated exceptional performance in terms of path length and node generation, albeit at the expense of computation time. On the other hand, the global variant showed no reduction in the number of nodes generated but exhibited a decrease in path length, accompanied by a slight increase in computation time.

After thorough examination, we can confirm that the local variant responds positively to an increase in population size. Additionally, a slight improvement was observed when increasing the threshold, favoring the behaviors of shrinking encircling mechanism and search for prey over spiral updating position. Furthermore, achieving a balance between the shrinking encircling behavior and search for prey, or focusing

primarily on the shrinking encircling mechanism to emphasize exploitation, resulted in enhanced performance across all three metrics.

Similarly, the global variant responded well to both iterations and population increase. It demonstrated a significant exponential decrease in path length when favoring the behaviors of shrinking encircling and search for prey over spiral updating position. Likewise, just like the local variant, achieving a balance between the shrinking encircling mechanism and emphasizing exploitation led to improved performance in path length.

Overall, our analyses indicate that both variants exhibit promising results when specific parameters are adjusted and the exploitation to exploration ratio is appropriately calibrated.

3.4 Implementation

In this section, we will explore the hardware and software implementation aspects of this project.

3.5.1 Hardware

The robotic wheelchair showed in figure 2.24 is equipped with microcontrollers that perform low-level calculations. The first microcontroller handles tasks such as control and management of speeds, sense of rotation of brushless motors, and the retrieval and calculation of odometric data from encoders. The second microcontroller manages all the onboard ultrasonic sensors on the wheelchair.

To enable remote manual control of the wheelchair, a wireless joystick is incorporated. For obstacle detection, a laser rangefinder is installed at the front of the wheelchair. It scans the horizontal plane with a 270° field of view and a detection range of 10 meters. To enhance obstacle detection further, eight ultrasonic sensors of two different models are added and strategically placed at the front of the wheelchair, covering a wide field of vision.

To enable the wheelchair to navigate its environment, it is crucial to determine its position at all times. To achieve this, encoders are utilized to measure the motor

rotation speed for speed control purposes. Encoders are fixed parallel to the axis of each wheel.

The power supply for the system consists of two 36-volt, 4Ah lithium batteries. These batteries offer the advantage of being significantly lighter (1.6 kilograms) compared to the lead-acid batteries commonly used in most electric wheelchairs.

In this project, the chosen control method is the widely recognized Proportional Integral Derivative (PID) control. PID control is widely employed due to its effectiveness. The main objective of control is to achieve and sustain a specific target value by directly influencing the system based on the difference between the reference value and the measured value.

Figure 2.24 – Robotic Wheelchair

Overall, the combination of microcontrollers, sensors, joystick, encoders, and power supply enables the robotic wheelchair to perform tasks such as speed control, obstacle detection, localization, and precise movement using the PID control method.

3.4.2 Software

The project was developed using Visual Studio Code on Ubuntu 20.04. We chose to utilize the C++ language for its performance and robust object-oriented support. This version of Ubuntu provided compatibility with ROS (Robot Operating System) Noetic, a popular robotics framework. Despite its name, ROS is not an operating system but rather a framework built on top of Ubuntu. It played a vital role in our project development, offering features such as hardware abstraction, seamless communication between different components, and efficient sensor processing.

Figure 3.25 – Robot Operating System Logo

ROS operates based on a publish-subscribe model, where nodes communicate by publishing and subscribing to topics. Each node represents a basic building block in the robot system, performing specific tasks such as reading sensor data, controlling actuators, or performing computations. Communication between nodes occurs through messages, which define the structure and type of data exchanged. ROS provides a range of predefined message types, including sensor data, commands, and control messages.

Nodes communicate with each other by publishing messages to topics and subscribing to topics to receive messages. Topics serve as named channels that facilitate message exchange. Multiple nodes can publish or subscribe to the same topic, allowing for loose coupling and seamless integration of separate components. This flexibility enables the system to adapt and scale effectively.

To facilitate the registration and discovery of nodes and topics, ROS employs a centralized component called the ROS Master. The Master keeps track of active nodes and the topics they publish or subscribe to. It acts as a central hub for coordinating the

communication between nodes, ensuring efficient and reliable message exchange throughout the system.

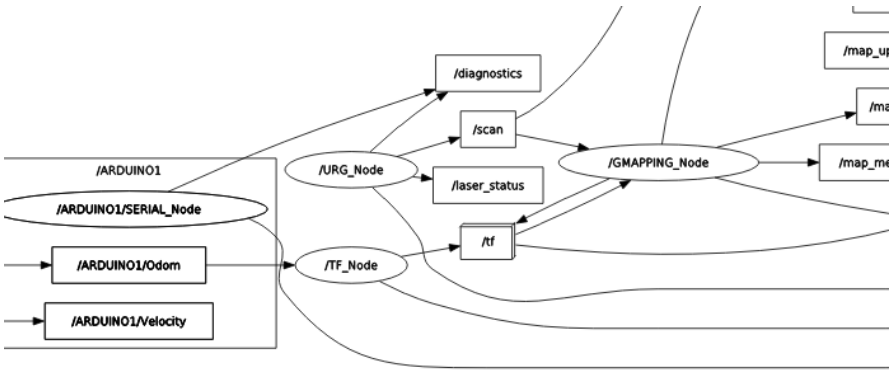


Figure 2.26 - Visual Representation of the Communication Architecture and Data Flow Between Different Nodes in a ROS System

Our project utilized several nodes to achieve successful results. These nodes played vital roles in various aspects of the project, including mapping the environment, establishing serial communication with the Arduino boards on the wheelchair, and transmitting control signals to the actuators, among other functions. We will briefly provide an overview of each node and its specific contribution to the project.

gmapping_node. Responsible for implementing simultaneous localization and mapping (SLAM) using a grid-based approach. It receives sensor data from the LiDAR and uses this information to create a 2D occupancy grid map of the surrounding environment. Simultaneously, it estimates the robot's pose within the map. To accomplish this, the gmapping_node subscribes to two topics: ”/scan” for laser scans and “/tf” for transforms that define the positions and orientations of the laser, base, and odometry frames. It then publishes two topics: “/map” containing the map of the environment and “/tf” providing the estimated pose of the robot [ZA].

tf_transform_node. Supplies the necessary transforms to the gmapping_node, enabling it to establish relationships between various frames. Specifically, it handles transformations from the frame associated with the incoming scan, known as "laser_link," to "base_link." Additionally, it manages the transformation from "base_link" to "odometry." To achieve this, the tf_transform_node subscribes to the "/ARDUINO1/Odom" topic, which is published by the Arduino board, and publishes

"/tf" with the appropriate frame_id and child_id to meet the required transform specifications.

urg_node. Serves as the driver responsible for establishing a connection and facilitating communication with the LiDAR sensor. Its primary function is to retrieve range measurements from the sensor, which correspond to the distances between the sensor and objects within its field of view. These range measurements are then published by the urg_node to the "/scan" topic, allowing other nodes in the system to access and utilize this data.

rviz_node. Launches and manages the RViz visualization tool. RViz is a 3D visualization tool widely used in robotics for visualizing sensor data, robot models, and other elements related to perception, planning, and navigation. With the rviz_node, users can customize the RViz environment to their needs. It subscribes to ROS topics like sensor data, robot pose, or map data to obtain the required information for visualization. The data is then rendered in a user-friendly graphical interface, enabling users to interact with and visualize sensor data, robot models, and other relevant visualizations.

navigation_node. Responsible for planning a motion from the intial to the goal configuration and providing the necessary controls to execute that motion. It subscribes to the topic "/map" in order to acquire the grid representation of the environment, allowing it to retrieve information about obstacles and perform its tasks. Additionally, it subscribes to the topic "/scan" to detect any new changes made to the environment. To enable the robot to move and rotate as required, the navigation_node publishes velocity commands to the robot's mobile base on the topic "/ARDUINO1/Velocity". This ensures that the robot can navigate through the environment and respond to the detected changes.

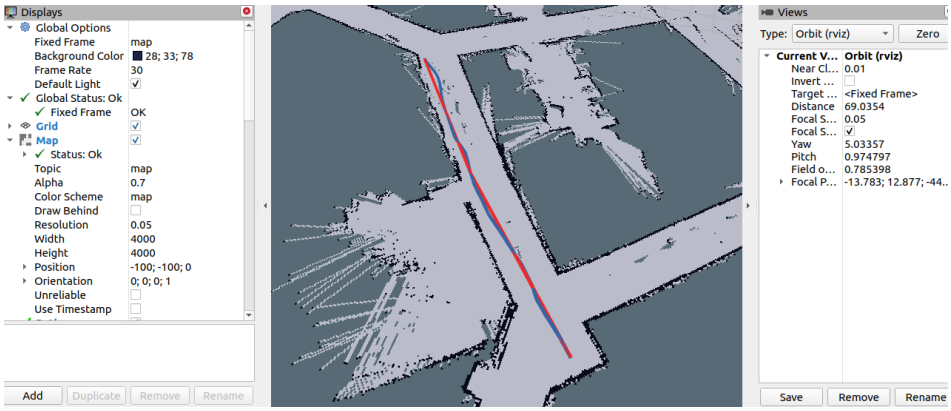


Figure 2.27 – Control Generation in Mapped Environment in RViz (Red Curve: Represents the path generated by the global variant, Blue Curve: The trajectory approximation of the robot's movement when executing the generated controls)

3.5 Conclusion

This chapter concludes the thesis by presenting simulation results for the proposed variants. The obtained results were highly positive and demonstrate the remarkable potential of the Whale Optimization Algorithm as a robust tool for optimizing path planners. Additionally, we have extensively examined the algorithm's response to various settings in both variants.

Moreover, this chapter provides an overview of the software and hardware aspects of the project. It is important to note that during the physical implementation, we encountered certain challenges related to communication and configuration. However, we are determined to address these issues and validate the efficacy of these variants in a real-world scenario.

General Conclusion

This project aimed to make a significant contribution to the field of robotics research by introducing a novel bio-inspired motion planning algorithm inspired by a swarm-based optimization technique. The proposed solution integrates the whale optimization algorithm with the RRT* algorithm, resulting in the development of two approaches: a local variant and a global variant of RRT*.

Both variants of the algorithm demonstrated substantial improvements in path quality compared to the original RRT* algorithm. In terms of path length, the proposed variants outperformed the original algorithm by 13 to 23 percent. Moreover, unlike RRT*, our approaches provide feasible trajectories by utilizing a WOA-based trajectory generator that takes into consideration the robot model.

From an optimization perspective, both the local variant and global variant of the algorithm responded favorably to increased population size and iterations number, as well as lower values of p and the lower half of values in the interval of the random parameter A.

The findings of this study demonstrate that the whale optimization algorithm is a suitable technique for optimizing path planners, yielding positive results and showcasing potential for further enhancements. For instance, future research could explore parallelization techniques and other performance-boosting tweaks to maximize the algorithm's capabilities.

In summary, this thesis introduced two variants of the RRT* algorithm that incorporate the whale optimization algorithm. Both variants exhibited promising results, and the comparative analysis provided valuable insights into their behavior under different settings. This study aspires to pave the way for future research that explores the untapped potential of the whale optimization algorithm in the context of path planning.

References

[1] Zghair, N.A.K. and Al-Araji, A.S., 2021. A one decade survey of autonomous mobile robot systems. International Journal of Electrical and Computer Engineering, 11(6), p.4891.

[2] Karur, K., Sharma, N., Dharmatti, C. and Siegel, J.E., 2021. A survey of path planning algorithms for mobile robots. Vehicles, 3(3), pp.448-468.

[3] Latombe, J.C., 2012. Robot motion planning (Vol. 124). Springer Science & Business Media.

[4] Khanmirza, E., Haghbeigi, M., Nazarahari, M. and Doostie, S., 2017, October. A comparative study of deterministic and probabilistic mobile robot path planning algorithms. In 2017 5th RSI international conference on robotics and mechatronics (ICRoM) (pp. 534-539). IEEE.

[5] Dijkstra, E.W., 2022. A note on two problems in connexion with graphs. In Edsger Wybe Dijkstra: His Life, Work, and Legacy (pp. 287-290).

[6] Hart, P.E., Nilsson, N.J. and Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), pp.100-107.

[7] Lozano-Pérez, T. and Wesley, M.A., 1979. An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM, 22(10), pp.560-570.

[8] Aurenhammer, F., 1991. Voronoi diagrams—a survey of a fundamental geometric data structure. ACM Computing Surveys (CSUR), 23(3), pp.345-405.

[9] LaValle, S.M., 1998. Rapidly-exploring random trees: A new tool for path planning.

[10] Kavraki, L.E., Svestka, P., Latombe, J.C. and Overmars, M.H., 1996. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE transactions on Robotics and Automation, 12(4), pp.566-580.

[11] Smith College. Dijkstra's Algorithm. Visited on June 28, 2023 https://www.science.smith.edu/~istreinu/Teaching/Courses/274/Spring98/Projects/Philip/fp/dijkstra.htm

[12] Russell Stuart J and Peter Norvig. 2020. Artificial Intelligence : A Modern Approach. 4th ed. Boston: Pearson.

[13] Smith College. Visiblity Graph. Visited on June 28, 2023 https://www.science.smith.edu/~istreinu/Teaching/Courses/274/Spring98/Projects/Philip/fp/visibility.htm

[14] UPC Research Group. Voronoi Diagram. Visited on June 28, 2023 https://dccg.upc.edu/wp-content/uploads/2020/06/GeoC-Voronoi-storing.pdf

[15] Abdel-Basset, M., Abdel-Fatah, L. and Sangaiah, A.K., 2018. Metaheuristic algorithms: A comprehensive review. Computational intelligence for multimedia big data on the cloud with engineering applications, pp.185-231.

[16] Yu, X. and Gen, M., 2010. Introduction to evolutionary algorithms. Springer Science & Business Media.

[17] Sampson, J.R., 1976. Adaptation in natural and artificial systems (John H. Holland).

[18] Katoch, S., Chauhan, S.S. and Kumar, V., 2021. A review on genetic algorithm: past, present, and future. Multimedia Tools and Applications, 80, pp.8091-8126.

[19] Baluja, S., 1994. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science.

[20] González Morgado, C., Lozano Alonso, J.A. and Larrañaga Múgica, P.M., 2000. Analyzing the population based incremental learning algorithm by means of discrete dynamical systems. Complex Systems, 12(4), pp.465-479.

[21] Koza, J.R., 1994. Genetic programming as a means for programming computers by natural selection. Statistics and computing, 4, pp.87-112.

[22] Espejo, P.G., Ventura, S. and Herrera, F., 2009. A survey on the application of genetic programming to classification. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 40(2), pp.121-144.

[23] Storn, R. and Price, K., 1997. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization, 11(4), p.341.

[24] Deng, W., Shang, S., Cai, X., Zhao, H., Song, Y. and Xu, J., 2021. An improved differential evolution algorithm and its application in optimization problem. Soft Computing, 25, pp.5277-5298.

[25] Can, Ü. and Alataş, B., 2015. Physics based metaheuristic algorithms for global optimization.

[26] Rashedi, Esmat, Hossein Nezamabadi-Pour, and Saeid Saryazdi. "GSA: a gravitational search algorithm." Information sciences 179, no. 13 (2009): 2232-2248.

[27] Rashedi, E., Nezamabadi-Pour, H. and Saryazdi, S., 2009. GSA: a gravitational search algorithm. Information sciences, 179(13), pp.2232-2248.

[28] Mittal, H., Tripathi, A., Pandey, A.C. and Pal, R., 2021. Gravitational search algorithm: A comprehensive analysis of recent variants. Multimedia Tools and Applications, 80, pp.7581-7608.

[29] Kaveh, A. and Talatahari, S., 2010. A novel heuristic optimization method: charged system search. Acta mechanica, 213(3-4), pp.267-289.

[30] Formato, R.A., 2007. Central force optimization. Prog Electromagn Res, 77(1), pp.425-491.

[31] Ding, D., Qi, D., Luo, X., Chen, J., Wang, X. and Du, P., 2012. Convergence analysis and performance of an extended central force optimization algorithm. Applied Mathematics and Computation, 219(4), pp.2246-2259.

[32] Erol, O.K. and Eksin, I., 2006. A new optimization method: big bang–big crunch. Advances in Engineering Software, 37(2), pp.106-111.

[33] Yang, X.S., 2014. Swarm intelligence based algorithms: a critical analysis. Evolutionary intelligence, 7, pp.17-28.

[34] Eberhart, R. and Kennedy, J., 1995, November. Particle swarm optimization. In Proceedings of the IEEE international conference on neural networks (Vol. 4, pp. 1942-1948). [35] Venter, G. and Sobieszczanski-Sobieski, J., 2003. Particle swarm optimization. AIAA journal, 41(8), pp.1583-1589.

[36] Dorigo, M., Birattari, M. and Stutzle, T., 2006. Ant colony optimization. IEEE computational intelligence magazine, 1(4), pp.28-39.

[37] Yang, X.S., 2010. Firefly algorithm, stochastic test functions and design optimisation. International journal of bio-inspired computation, 2(2), pp.78-84.

[38] Fister, I., Fister Jr, I., Yang, X.S. and Brest, J., 2013. Swarm and evolutionary computation. A Comprehensive Review of Firefly Algorithms.

[39] Mirjalili, S. and Lewis, A., 2016. The whale optimization algorithm. Advances in engineering software, 95, pp.51-67.

[40] Xu, Y., Cui, Z. and Zeng, J., 2010. Social emotional optimization algorithm for nonlinear constrained optimization problems. In Swarm, Evolutionary, and Memetic Computing: First International Conference on Swarm, Evolutionary, and Memetic

Computing, SEMCCO 2010, Chennai, India, December 16-18, 2010. Proceedings 1 (pp. 583-590). Springer Berlin Heidelberg.

[41] Xu, Y., Cui, Z. and Zeng, J., 2010. Social emotional optimization algorithm for nonlinear constrained optimization problems. In Swarm, Evolutionary, and Memetic Computing: First International Conference on Swarm, Evolutionary, and Memetic Computing, SEMCCO 2010, Chennai, India, December 16-18, 2010. Proceedings 1 (pp. 583-590). Springer Berlin Heidelberg.

[42] Atashpaz-Gargari, E. and Lucas, C., 2007, September. Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. In 2007 IEEE congress on evolutionary computation (pp. 4661-4667). Ieee.

[43] Hosseini, S. and Al Khaled, A., 2014. A survey on the imperialist competitive algorithm metaheuristic: implementation in engineering domain and directions for future research. Applied Soft Computing, 24, pp.1078-1094.

[44] Rao, R.V., Savsani, V.J. and Vakharia, D.P., 2011. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Computer-aided design, 43(3), pp.303-315.

[45] Rao, R.V., Savsani, V.J. and Vakharia, D.P., 2011. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Computer-aided design, 43(3), pp.303-315.

[46] Moosavian, N. and Roodsari, B.K., 2014. Soccer league competition algorithm: A novel meta-heuristic algorithm for optimal design of water distribution networks. Swarm and Evolutionary Computation, 17, pp.14-24.

[47] Karaman, S. and Frazzoli, E., 2011. Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7), pp.846-894.

[48] Kuffner, J.J. and LaValle, S.M., 2000, April. RRT-connect: An efficient approach to single-query path planning. In Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065) (Vol. 2, pp. 995-1001). IEEE.

[49] Jin, X., Yan, Z., Yang, H., Wang, Q. and Yin, G., 2020, December. A goal-biased RRT path planning approach for autonomous ground vehicle. In 2020 4th CAA International Conference on Vehicular Control and Intelligence (CVCI) (pp. 743-746). IEEE.

[50] Song, Q., Li, S., Yang, J., Bai, Q., Hu, J., Zhang, X. and Zhang, A., 2021. Intelligent optimization algorithm-based path planning for a mobile robot. Computational Intelligence and Neuroscience, 2021.

[51] Karaman, S. and Frazzoli, E., 2011. Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7), pp.846-894.

[52] Chai, Q. and Wang, Y., 2022. RJ-RRT: Improved RRT for Path Planning in Narrow Passages. Applied Sciences, 12(23), p.12033.

[53] Kuffner, J.J. and LaValle, S.M., 2000, April. RRT-connect: An efficient approach to single-query path planning. In Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065) (Vol. 2, pp. 995-1001). IEEE.

[54] Chen, J., Zhao, Y. and Xu, X., 2021. Improved RRT-Connect Based Path Planning Algorithm for Mobile Robots. IEEE Access, 9, pp.145988-145999.

[55] Li, J., Liu, S., Zhang, B. and Zhao, X., 2014, September. RRT-A* motion planning algorithm for non-holonomic mobile robot. In 2014 proceedings of the SICE annual conference (SICE) (pp. 1833-1838). IEEE.

[56] Ayawli, B.B.K., Mei, X., Shen, M., Appiah, A.Y. and Kyeremeh, F., 2019. Optimized RRT-A* path planning method for mobile robots in partially known environment. Information technology and control, 48(2), pp.179-194.

[57] Sharma, P., Gupta, A., Ghosh, D., Honkote, V., Nandakumar, G. and Ghose, D., 2021, September. Pg-rrt: A gaussian mixture model driven, kinematically constrained bi-directional rrt for robot path planning. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 3666-3673). IEEE.

[58] Melchior, N.A. and Simmons, R., 2007, April. Particle RRT for path planning with uncertainty. In Proceedings 2007 IEEE International Conference on Robotics and Automation (pp. 1617-1624). IEEE.

[59] Wang, J., Chi, W., Li, C., Wang, C. and Meng, M.Q.H., 2020. Neural RRT*: Learning-based optimal path planning. IEEE Transactions on Automation Science and Engineering, 17(4), pp.1748-1758.

[60] Taheri, E., Ferdowsi, M.H. and Danesh, M., 2018. Fuzzy greedy RRT path planning algorithm in a complex configuration space. International Journal of Control, Automation and Systems, 16, pp.3026-3035.

[61] Wang, Q., Wang, W. and Li, Y., 2012, November. A multi-RRT based hierarchical path planning method. In 2012 IEEE 14th International Conference on Communication Technology (pp. 971-975). IEEE.

[62] Jin, X., Yan, Z., Yang, H., Wang, Q. and Yin, G., 2020, December. A goal-biased RRT path planning approach for autonomous ground vehicle. In 2020 4th CAA International Conference on Vehicular Control and Intelligence (CVCI) (pp. 743-746). IEEE.

[63] LaValle, S.M. and Kuffner Jr, J.J., 2001. Randomized kinodynamic planning. The international journal of robotics research, 20(5), pp.378-400.

[64] Petit, L. and Desbiens, A.L., 2021, October. RRT-Rope: A deterministic shortening approach for fast near-optimal path planning in large-scale uncluttered 3D environments. In 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC) (pp. 1111-1118). IEEE.

[65] Islam, F., Nasir, J., Malik, U., Ayaz, Y. and Hasan, O., 2012, August. Rrt∗-smart: Rapid convergence implementation of rrt∗ towards optimal solution. In 2012 IEEE international conference on mechatronics and automation (pp. 1651-1656). IEEE.

[66] Tahirovic, A. and Ferizbegovic, M., 2018, May. Rapidly-exploring random vines (RRV) for motion planning in configuration spaces with narrow passages. In 2018 IEEE International Conference on Robotics and Automation (ICRA) (pp. 7055-7062). IEEE.

[67] Kalisiak, M. and van de Panne, M., 2006, May. RRT-blossom: RRT with a local flood-fill behavior. In Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. (pp. 1237-1242). IEEE.

[68] Solovey, K., Janson, L., Schmerling, E., Frazzoli, E. and Pavone, M., 2020, May. Revisiting the asymptotic optimality of RRT. In 2020 IEEE International Conference on Robotics and Automation (ICRA) (pp. 2189-2195). IEEE.

[69] Dao, T.K., Pan, T.S. and Pan, J.S., 2016, November. A multi-objective optimal mobile robot path planning based on whale optimization algorithm. In 2016 IEEE 13th international conference on signal processing (ICSP) (pp. 337-342). IEEE.

[70] Yan, Z., Zhang, J., Yang, Z. and Tang, J., 2021. Two-dimensional optimal path planning for autonomous underwater vehicle using a whale optimization algorithm. Concurrency and Computation: Practice and Experience, 33(9), p.e6140.

[71] Kumar, S.V., Jayaparvathy, R. and Priyanka, B.N., 2020. Efficient path planning of AUVs for container ship oil spill detection in coastal areas. Ocean Engineering, 217, p.107932.