Sveučilište u Zagrebu
University of Zagreb

IFRoS
Erasmus Mundus Joint Master in
INTELLIGENT FIELD ROBOTIC SYSTEMS

PALTECH

Efficient Coverage Path Planning for Autonomous Robots: A Behavioral Approach with Obstacle Avoidance

Author:
SYED MAZHAR
Msc in Intelligent Field Robotic Systems.

Internal Supervisor:

Tamara Petrović
Associate professor,
University of Zagreb.

External Supervisor:

Felix Schiegg
Co-Founder & CEO,
Paltech GmbH.

Master's Thesis in the domain of Autonomous Navigation Systems, Robotics.

Germany, June 2024

Abstract

For centuries, agriculture has been fundamental to sustaining human life, particularly through the cultivation of grass to feed livestock. The quality of animal products is directly influenced by the quality of their feed, which is traditionally grown in agricultural fields. However, these fields are often infested with weeds, some of which are detrimental to livestock health and, consequently, the quality of animal products. Traditionally, farmers have used non-organic chemicals to control weeds, but this method is not conducive to producing organic feed. Manual weed removal is an alternative, though it is labor-intensive and inefficient.

To address this challenge, we propose an automated solution using robots equipped with an efficient path planning algorithm designed for comprehensive weed removal. This problem, known as coverage path planning (CPP), is critical for ensuring that all the weeds are covered efficiently. Our research introduces a novel behavioral algorithm tailored to perform complete CPP while adhering to the non-holonomic constraints of the robot. The algorithm incorporates three adaptive behaviors that evolve as coverage progresses, aiming to maximize coverage rate, reduce computation time, minimize field operation time, conserve energy, and shorten route length. Additionally, it prioritizes generating straight paths over curved ones to produce natural straight paths.

We developed two variants of the algorithm: one for fields without obstacles and another for fields with polygonal obstacles, such as houses and fences. Comparative analyses with the graph search and lawnmower approaches were conducted to evaluate the strengths and weaknesses of each method. Results demonstrate that our algorithm consistently maintains a high coverage rate, generates efficient paths, and minimizes energy consumption and route length. It also ensures hundred percent coverage of weeds. The algorithm's flexibility also allows customization for curved paths making it a versatile solution for other coverage path planning problems.

Keywords: Complete Coverage Path Planning, Obstacle Avoidance, Behavioral Algorithms, Non-Holonomic Constraints, Agricultural Robotics, Autonomous Navigation, Motion Planning, Continuous space, Discrete space, Occupancy grids, Dubins path, Travelling Salesman Problem (TSP), Dubins Open Travelling Salesman Problem (DOTSP).

Table of Contents

Lists of Figures

List of Tables

Introduction

1 Background and context

Have you ever gazed out over a vast agricultural expanse and pondered the potential for technology to transform farming practices? In recent years, the convergence of robotics and agriculture has garnered substantial attention for its capacity to revolutionize farming methodologies and confront various challenges in crop cultivation and management. Among these challenges, the proliferation of weeds in grass fields emerges as a critical concern for farmers worldwide. These weeds not only compete with grass for essential resources but also pose significant threats to livestock health and food safety. In particular, species like Rumex have been identified as major nuisances in grasslands, contaminating forage intended for grazing animals.

The nutritional quality of grassland vegetation directly influences the health and productivity of livestock. Contaminated forage, tainted with harmful weed species like Rumex, can induce adverse health effects in cattle, impacting both the quantity and quality of milk production, as well as overall animal welfare. Therefore, the imperative to effectively remove weeds from grasslands becomes apparent, ensuring the integrity and safety of livestock feed and the sustainability of agricultural operations.

Hence, to address this imperative and enhance animal well-being while obtaining organic products, the development of robots and techniques for comprehensive field coverage and weed removal becomes indispensable. This underscores the critical need for robot coverage motion planning techniques to systematically eradicate weeds and optimize agricultural productivity.

In response to the pressing agricultural challenge posed by weed infestation in grasslands, numerous robots have been developed for specialized tasks. Each robot is equipped with sensors and sophisticated systems tailored to its specific function. Utilizing data from these sensors, many researchers have proposed distinct coverage path planning (CPP) algorithms to guide the robots in performing their tasks safely and accurately.

However, there is no generic CPP algorithm applicable to all types of robots and tasks. This limitation highlights the need for an innovative CPP algorithm following te non-hoilonomic constraints of the robot while prioritizing linear trajectories fin the agricultural field. Such an algorithm aims to reduce energy consumption and maintain the quality of the grass, thereby enhancing the efficiency and sustainability of autonomous robotic operations in agriculture.

The complexity of this task increases when the robot must navigate environments with obstacles, further complicating the path planning problem. As a result, this master thesis aims to design and develop two complete coverage path planning algorithms that respect the non-holonomic constraints of the robot while prioritizing linear trajectories: one for path planning without obstacles and the other for path planning with obstacles.

2 Problem statement

The challenge of weed infestation in grasslands poses significant threats to agricultural productivity and the quality of organic produce. Current robotic solutions, although advanced, are highly specialized,

with each robot designed for specific tasks and equipped with tailored sensor and sophisticated systems. These robots rely on task specific coverage path planning (CPP) algorithms to navigate and perform tasks on agricultural fields. However, existing CPP algorithms are often limited in scope, designed to suit particular robots and specific operational conditions. No universal CPP algorithm exists that can accommodate the diverse range of robots and tasks encountered in agricultural settings.

The complexity of designing such an algorithm is further compounded by the need to consider non-holonomic constraints, which restrict the robot's movement capabilities, making navigation and task execution more challenging. Additionally, agricultural environments are frequently equipped with obstacles, adding another layer of difficulty to the path planning process. Without a robust, adaptable CPP algorithm, the efficiency and effectiveness of robotic weed removal are significantly compromised.

This lack of a generic and efficient CPP algorithm shows the need of an innovative, flexible and robust CPP algorithm that respects the non-holonomic constraints of the robot while optimizing linear trajectories. This advancement is crucial for improving the overall sustainability of autonomous robotic operations in agriculture, making it imperative to address this gap in current research and technology.

3 Research Questions

The central question of this research is how to develop and implement a coverage path planning (CPP) algorithm for scenarios both with and without obstacles. To address this, we must consider the following research questions, which will guide the development of the proposed CPP algorithm and its evaluation in real-world agricultural settings:

1. How should a coverage path planning (CPP) algorithm be designed to respect the non-holonomic constraints of agricultural robots while prioritizing linear trajectories, minimizing computation time and energy consumption, and maintaining a high coverage rate?
2. How to design the CPP algorithm that is robust, adaptable, and versatile enough to accommodate a wide range of agricultural and other coverage tasks, ensuring optimal performance in diverse operational conditions?
3. What are the specific non-holonomic constraints that need to be considered in the design of a CPP algorithm for agricultural robots?
4. How can the CPP algorithm be adapted to function effectively in environments with different shapes and sizes of the obstacles?
5. What metrics should be used to evaluate the efficiency and effectiveness of the proposed CPP algorithm in real-world agricultural settings?
6. How does the proposed CPP algorithm compare to existing algorithms for complete coverage path planning?

4 Methodology and Approach

This thesis endeavors to pioneer the development and implementation of an innovative coverage path planning (CPP) algorithm. Unlike conventional approaches, this algorithm prioritizes straight paths to optimize energy efficiency, thereby offering versatility across a spectrum of coverage path

planning methodologies. While its primary application lies in weed removal within grasslands, its adaptability extends to diverse agricultural contexts, promising enhanced efficiency and sustainability in autonomous robotic operations. CPP plays a crucial role in guiding autonomous robotic systems to systematically traverse and cover an entire area of interest while minimizing overlap and maximizing efficiency.

The proposed CPP algorithm consists of two main parts: preprocessing the data to transform points with regions into centroids of the overlaps, and the main behavioral coverage path planning algorithm.

In the preprocessing stage, the algorithm transforms points with regions into centroids of the overlaps. Given the robot's specific width and its capability to remove weeds along this width, this transformation reduces the total number of points by at least 40 percent. This allows the robot to move along the centroids of the overlaps and efficiently remove weeds along its width.

The main algorithm employs a vision cone strategy to prioritize linear trajectories, limiting the next navigation point to be in front of the robot and within a certain angle difference from the current orientation, while adhering to the robot's non-holonomic constraints. The behavioral algorithm consists of three behaviors: Centroid-Seeking Behavior, Circumferential-Traversal Behavior, and Dubins Open Travelling Salesman Problem (DOTSP).

- Centroid-Seeking Behavior: This behavior navigates the robot using the vision cone to cover the area near the centroid of the dataset.
- Circumferential-Traversal Behavior: This behavior focuses on covering points near the boundary of the dataset, using the vision cone to prioritize linear trajectories. These two behaviors cover at least 95 percent of the points with a high coverage rate.
- Dubins Open Travelling Salesman Problem (DOTSP): For the remaining 5 percent of points, this behavior is invoked to cover the points efficiently, minimizing energy consumption and computation time.

The second algorithm for path planning with obstacles extends the first algorithm with two additional steps: setting up the obstacle and generating a dynamic grid for each obstacle, creating a hybrid space for continuous space path planning and discrete space obstacle avoidance. The second step is path validity checking, ensuring that if the path intersects with an obstacle, the algorithm provides a feasible shortest path to avoid the obstacle and continue with the main algorithm. In this scenario, the third behavior is replaced with a new behavior to efficiently cover the remaining points around the obstacles.

The proposed CPP algorithm will be evaluated using a series of metrics to assess its efficiency and effectiveness in real-world agricultural settings. These metrics include coverage rate, computation time, energy consumption, field operation time, and path length. The algorithm will be compared to existing CPP algorithms to determine its performance and potential for widespread adoption in autonomous robotic operations.

5 Thesis Contributions

The contributions of this thesis are multifaceted, spanning both theoretical advancements and practical applications in the field of coverage path planning (CPP) for agricultural and other fields:

1. **Innovative CPP Algorithm Development:** The primary contribution lies in the development of an innovative CPP algorithm that respects the non-holonomic constraints while prioritizing linear trajectories, minimizing computation time, route length and energy consumption, and maintaining a high coverage rate. By incorporating novel behavioral strategies such as centroid-seeking behavior, Circumferential-traversal behavior and DOTSP, the algorithm offers a versatile solution adaptable to various agricultural contexts.

2. **Innovative CPP with Obstacles Development:** Another major contribution is the design, development and the integration of obstacle setup and checking mechanism to avoid obstacles efficiently.

3. **Use of Hybrid Space:** The proposed algorithm uses a hybrid space for continuous space path planning and discrete space obstacle avoidance, ensuring efficient and obstacle-free traversal in agricultural environments.

4. **Dynamic Grid Generation based on non-holonomic constraints:** The proposed algorithm generates a dynamic grid for each obstacle, ensuring that the robot can navigate around obstacles while adhering to its non-holonomic constraints.

5. **Evaluation Metrics:** The thesis introduces a set of metrics to evaluate the efficiency and effectiveness of the proposed CPP algorithm in real-world agricultural settings. These metrics include coverage rate, computation time, energy consumption, field operation time, number of turns and path length, providing a comprehensive assessment of the algorithm's performance in various scenarios.

6 Thesis Structure

The thesis consists of six chapters, each focusing on a specific aspect of the research and development process. The chapters are structured as follows:

1. **Introduction:** This chapter provides an overview of the background context, research question, methodology and contributions of the thesis.

2. **Literature Review:** This chapter provides an overview of existing coverage path planning algorithms. It also discusses the challenges and limitations of current CPP algorithms and highlights the need for an innovative, adaptable, and efficient algorithm to optimize agricultural operations.

3. **Methodology:** This chapter outlines the proposed coverage path planning algorithm, detailing its design, implementation, and evaluation. It also describes the preprocessing and main behavioral algorithms, as well as the metrics used to assess the algorithm's performance.

4. **Results:** This chapter presents the results of the evaluation of the proposed CPP algorithm in real-world agricultural settings. It compares the algorithm's performance to existing CPP algorithms and discusses its potential for widespread adoption in autonomous robotic operations.

5. **Discussion:** This chapter analyzes the results of the evaluation and discusses the implications of the proposed CPP algorithm. It also identifies areas of development to further enhance the algorithm's efficiency and effectiveness.

6. **Future Directions and Conclusion:** This chapter outlines future research directions and concludes the thesis by summarizing the key findings and contributions of the proposed CPP algorithm.

Literature Review

1 Literature Review for CPP Algorithms.

Path planning algorithms play a crucial role in the field of robotics and autonomous navigation, enabling vehicles to navigate efficiently through complex environments while adhering to various constraints. Coverage path Planning (CPP) is an open problem in robotics in improving the efficiency of planning an optimal path to cover the target area, as well as generating a collision-free pathway with less computation. An overview of CPP problems with the objective, challenges, and design features as mention in the paper [2] is shown in (Figure 1)

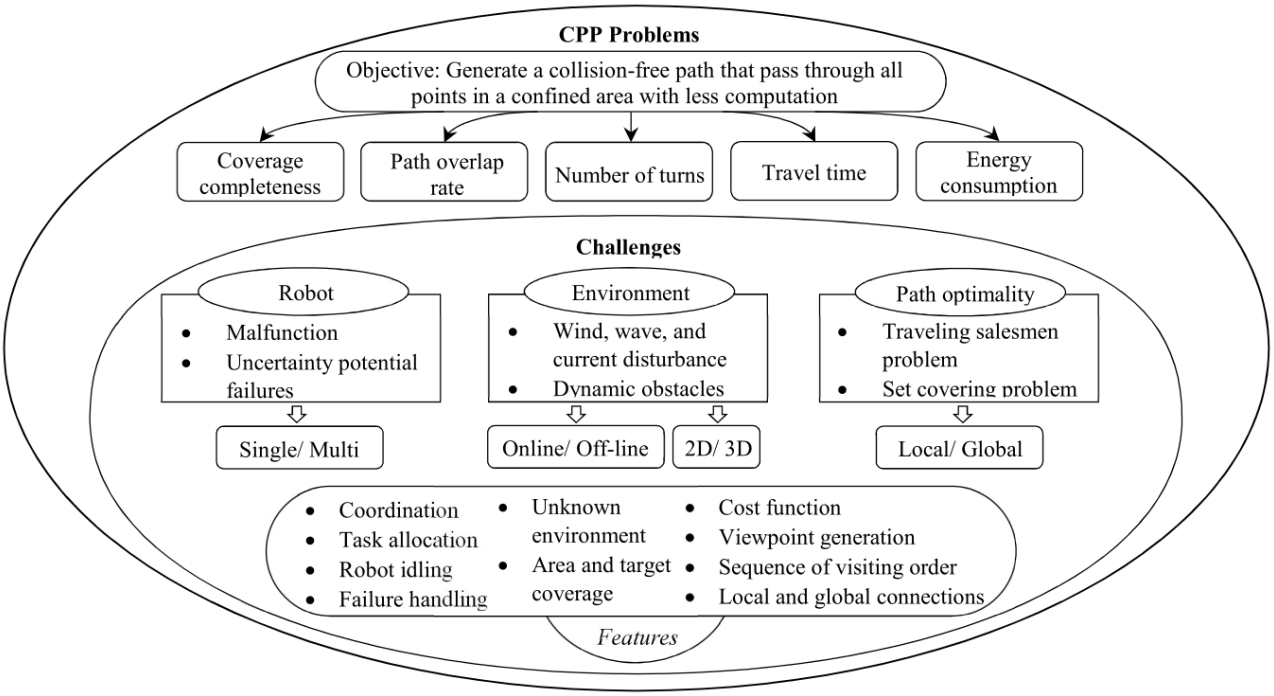


Figure 1: The objective and challenges in coverage path planning (CPP) problems.

Over the years, researchers have developed a myriad of algorithms to address different aspects of path planning, ranging from basic algorithms to more advanced methodologies. CPP algorithms can be categorized into two approaches, classical algorithms, and heuristic-based algorithms. The summarized details of CPP algorithms according to the characteristics of the algorithms as stated in the paper [2] are classified as shown in (Figure 2).

Several algorithms relevant to coverage path planning for points or regions, which adhere to non-holonomic constraints with the objective of minimizing total route length, computational time, and energy consumption, are discussed below.

1.1 Dubin's Path

In the realm of geometric analysis and constrained path planning, the pioneering work of L.E. Dubins stands as a cornerstone, providing profound insights into the properties and characteristics of paths subject to curvature constraints. Dubins' seminal exploration, outlined in the paper [1], lays a robust

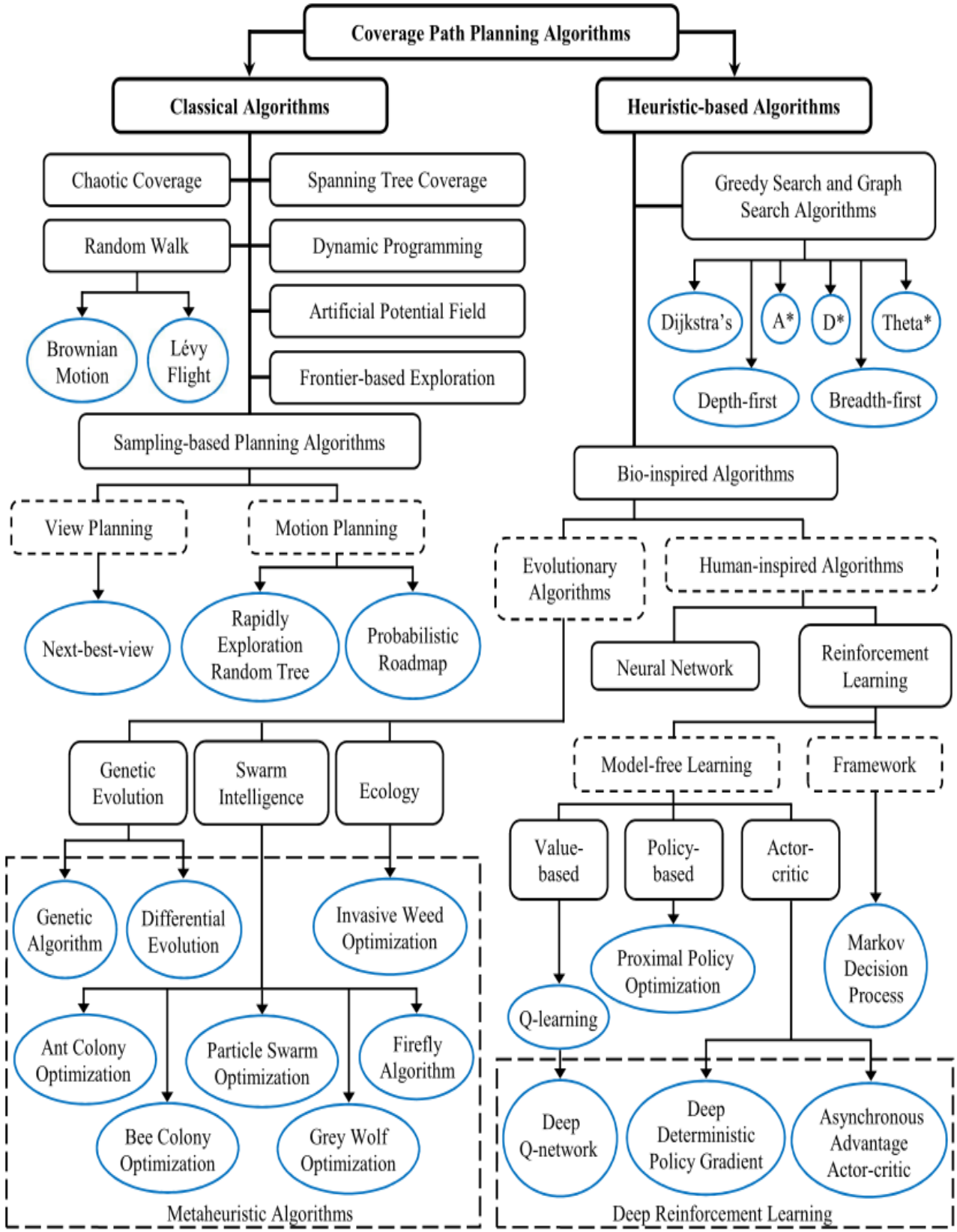


Figure 2: The classification of coverage path planning (CPP) algorithms.

foundation for understanding the fundamental principles governing constrained path planning and provides critical insights into the properties of paths constrained by curvature.

At the core of Dubins' research lies the concept of R-geodesics, representing paths of minimal length under specified curvature constraints. This notion encapsulates the geometric essence of constrained paths, defining them as combinations of straight lines and circular arcs with a minimum radius of curvature, denoted as R. Dubins' theorem regarding the structure of R-geodesics in two dimensions provides a clear geometric understanding, asserting that such paths consist of no more than three segments, each comprising either a straight line or an arc of a circle with radius R. This theorem delineates the structure of minimal paths and imposes precise constraints on their composition, revealing the inherent simplicity of paths subject to curvature constraints.

Dubins rigorously proved the existence of R-geodesics using mathematical tools like Ascoli's theorem and concepts from E. Schmidt's proof of A. Schur's Lemma. These proofs confirm the theoretical existence of such paths and illuminate their analytical and geometric properties.

Dubins' work represents a significant milestone in the study of geometric analysis and constrained path planning, providing not only a solution to a specific geometric problem but also a methodological framework applicable to a broader class of problems in path planning and optimization. Therefore, many extensions of Dubins have been studied since then, integrating with other approaches to solve complex path planning problems. Due to its effectiveness towards curvature constraints and path length minimization, the Dubins path serves as a potential technique to inspire and inform further research into path planning algorithms for agricultural robots.

1.2 Reeds-Shepp Paths

Reeds-Shepp paths, introduced by J.A. Reeds and L.A. Shepp in the paper [3], offers a flexible solution to optimal path planning for vehicles capable of moving forwards and backwards. Unlike Dubins paths, which only accommodate forward movement, Reeds-Shepp paths enhance maneuverability by incorporating reverse movements, doubling the range of possible maneuvers. This flexibility enables efficient navigation in complex environments, making them ideal for applications like robotics and autonomous vehicle navigation.

In agricultural robotics, Reeds-Shepp paths are particularly beneficial for tasks like weed removal, where precise navigation is essential. Their flexibility allows for efficient navigation through narrow spaces and around obstacles, reducing time and energy consumption. Reeds and Shepp's comprehensive analysis of Reeds-Shepp paths laid the foundation for subsequent research in optimal control and path planning, emphasizing the importance of considering a vehicle's full range of capabilities.

1.3 Travelling Salesman Problem (TSP) with Neighborhoods

The Traveling Salesman Problem (TSP) represents a classic conundrum in optimization, challenging researchers to find the most efficient route for a salesman to visit a set of locations and return to the starting point while minimizing the total distance traveled. Renowned for its computational complexity and practical applications in logistics and route planning, the TSP has spurred numerous investigations into variants that more accurately reflect real-world scenarios.

One such variant, explored in the paper [4], introduces the concept of TSP with neighborhoods (TSPN). In this formulation, destinations are not singular points but rather areas or neighborhoods, complicating the problem by requiring the salesman to visit each neighborhood at least once without

specifying exact points for each visit. TSPN is particularly relevant as it mirrors the challenge of navigating through regions (fields with weeds) rather than fixed points.

To address the challenges posed by TSPN, the paper introduces innovative approximation algorithms tailored to different types of neighborhoods, such as line segments or complex shapes described as "fat" regions. A notable contribution is the development of a constant factor approximation algorithm for neighborhoods represented as line segments, signifying a significant advancement in the field of geometric optimization.

A significant contribution of the paper is the m-guillotine method, which recursively subdivides the plane to approach a near-optimal solution. This method, along with key theoretical insights, underpins the algorithms' effectiveness in solving TSPN. However, it is essential to note that the TSPN problem is NP-hard, and the proposed algorithms provide only approximation solutions. While TSPN works with regions instead of points and strives to find the optimal solution, it does not explicitly consider the curvature constraints inherent in robotic path planning scenarios.

1.4 Traveling Salesperson Problems for Dubins' vehicle.

After TSPN evolved from TSP, the challenges associated with optimal path planning for Dubins' vehicles, constrained by their minimum turning radius, have garnered significant attention. The paper [5] offers a thorough examination of these challenges and introduces innovative methodologies to overcome them.

At the core of this paper are two fundamental problems: the point-to-point shortest path problem (PTP) and the traveling salesperson problem (TSP) tailored for Dubins' vehicles. These problems are crucial for developing algorithms capable of computing optimal or near-optimal paths for vehicles subject to nonholonomic constraints. the paper tackles the traveling salesperson problem (TSP) for Dubins' vehicles, wherein the vehicle must visit a predefined set of locations in the most efficient route without revisiting any point. Acknowledging the computational complexity of exact solutions for larger point sets, the authors explore heuristic and approximation algorithms to efficiently approximate solutions.

The methodology outlined in the paper integrates rigorous mathematical frameworks and computational geometry to address the complexities of path planning for Dubins' vehicles. By synthesizing Dubins constraints with the Traveling Salesperson Problem (TSP), the study offers a cohesive approach that bridges theoretical insights with practical solutions. this paper constitutes a significant contribution to the literature on path planning for nonholonomically constrained vehicles, paving the way for further advancements in navigating Dubins' vehicles efficiently and effectively.

1.5 Dubins Traveling Salesman Problem with Neighborhoods

To address the challenges posed by regions in the Dubins Traveling Salesman Problem (DTSP), the paper [6] introduces a tailored formulation termed the Dubins Touring Regions Problem (DTRP). This novel approach aims to determine an optimal sequence of configurations for entering and exiting regions, all while minimizing the total path length. Unlike the conventional Traveling Salesman Problem (TSP), which focuses on finding the shortest route visiting a set of points, the DTSPN involves regions or neighborhoods, necessitating the vehicle to access each region at least once. This problem holds significant relevance in various applications, including autonomous drone surveillance, delivery systems, and robotic exploration, where vehicles must efficiently traverse specified areas while adhering to nonholonomic constraints.

The proposed algorithm for solving the DTRP leverages local iterative optimization techniques, taking into account the unique dynamics of Dubins vehicles. It begins with an initial sequence of region visits derived from solving an Euclidean TSP (ETSP), which serves as a proxy for the region centers. The algorithm then iteratively refines the entry and exit configurations to minimize the total tour length.

A key aspect of the solution method is its decoupled approach, optimizing the heading and position of each entry point independently. This simplification, facilitated by mathematical techniques that project the problem into a more manageable form, allows for efficient local optimization. The algorithm iterates until no further improvements can be made or a termination criterion is met, incorporating strategies to escape local minima by adjusting vehicle headings and repositioning at region boundaries.

Empirical validation of the algorithm demonstrates its effectiveness and efficiency across various scenarios, including different region shapes and configurations, particularly in dense environments where regions are close together. Comparative analysis against existing evolutionary algorithms showcases the proposed method's ability to produce high-quality solutions with significantly reduced computational time, making it suitable for real-time applications on modest hardware, such as onboard computers in UAVs.

Despite its capability to generate near-optimal paths while adhering to the non-holonomic constraints, the algorithm exhibits notable drawbacks in terms of computational efficiency. For instance, computing a near-optimal path for 30 regions may require over a minute, and the algorithm's performance further deteriorates with an increasing number of points, especially when they are closely positioned. Moreover, the algorithm's limitation in considering the overlap of regions represents a critical drawback, as it fails to account for this aspect in real-world scenarios, thereby impacting the path planning accuracy and efficiency.

1.6 DTSPN with Overlappped Regions

The paper [7] introduces the Intersecting Regions Algorithm (IRA) to address the challenges posed by intersecting regions in DTSPN. By explicitly considering the overlap among regions, IRA offers a more comprehensive solution compared to traditional methods. It leverages sampling within intersecting regions to construct feasible tours for autonomous vehicles, resulting in optimized route selection and more efficient path planning.

Furthermore, IRA reduces the computational complexity associated with path planning by adopting a polynomially scalable algorithmic structure. This scalability ensures IRA's viability even in scenarios with numerous regions of interest, where computational resources are limited. Monte Carlo simulations validate IRA's practical utility, showcasing significant performance enhancements, particularly in scenarios with high degrees of region overlap. However, despite its advancements, IRA still encounters challenges when dealing with a large number of points and close proximity points. These limitations highlight avenues for further research and development to enhance the algorithm's robustness and efficiency in addressing complex DTSPN scenarios.

1.7 Path planning in confined spaces

Navigating narrow and complex environments presents challenges for non-holonomic vehicles. In the paper [8], the authors propose a path planning method tailored to address these challenges, focusing on generating paths with continuous curvature to ensure smooth operation in confined spaces.

The proposed method adopts a two-phase planning approach combining global and local strategies for efficient navigation. In the first phase, the authors employ the RTR (Rotate-Translate-Rotate) planner, a variation of the Rapidly Exploring Random Tree (RRT) method. This planner generates paths consisting of straight movements and in-place turning, simplifying complex maneuvering for navigating constrained spaces.

In the second phase, the global path is refined using the TTS local planning procedure. This planner approximates the initial path with a sequence of paths adhering to the vehicle's curvature constraints, ensuring smooth and feasible trajectories. The TTS planner can generate paths with continuous curvature turns (CC-turns) and straight segments, offering flexibility adaptable to various environmental constraints.

The paper also emphasizes maintaining similarity between global and local trajectories to avoid sudden changes in the path and ensure smooth passage through narrow regions. Ultimately, the objective is to enhance the capabilities of autonomous vehicles to operate safely and efficiently in diverse scenarios, thereby improving their practicality for everyday use.

2 Literature Review for Obstacle Avoidance.

2.1 Random walk (RW) algorithms

Random walk (RW) algorithms, commonly employed in path planning (CPP) tasks, offer a stochastic approach to obstacle avoidance in robotics. By mimicking the random movements observed in natural phenomena, RW algorithms enable robots to navigate environments while adapting obstacles. Although primarily designed for exploration and coverage, RW algorithms can effectively avoid obstacles by dynamically adjusting movement directions based on environmental cues. However, their effectiveness and efficiency in obstacle-rich environments is limited, as random movements could lead to inefficient navigation.

2.2 Dynamic Programming

Dynamic Programming (DP) offers a robust approach to Coverage Path Planning (CPP), efficiently optimizing coverage paths while considering factors like obstacles and turns. DP's advantage lies in its ability to handle overlapping sub-problems and exploit optimal substructure, making it suitable for generating globally coverage paths. By optimizing the sequence of segments and connections, DP algorithms construct shorter and more efficient coverage paths, addressing challenges like coverage overlaps. However, scalability concerns arise with large-scale CPP problems, as DP algorithms require extensive computational resources and time to generate complete paths.

2.3 Artificial Potential Field (APF) Algorithm

The artificial potential field (APF) algorithm is widely used for obstacle avoidance and navigation in Coverage Path Planning (CPP). By employing virtual repulsive forces around obstacles and attractive forces toward the goal, the APF algorithm guides robots while ensuring a safe distance from obstacles. The susceptibility of the artificial potential field (APF) algorithm to local optima presents a notable challenge, potentially leading to the entrapment of robots in specific environmental regions. Additionally, the algorithm's computation is resource-intensive, particularly in generating potential fields for the entire environment. Moreover, robots will experience repulsive forces not only from the front but also from the sides and behind obstacles, further complicating navigation.

2.4 Spanning Tree Coverage Algorithms in coverage path planning

Spanning Tree Coverage (STC) algorithms as mentioned in the paper [12] offer a systematic approach to coverage path planning by dividing the workspace into disjoint cells and constructing spanning trees within them. This method allows robots to navigate around or through obstacles for comprehensive coverage. Initially, the workspace is partitioned into cells, and spanning trees are formed within these cells to guide robot movement. However, challenges arise when obstacles obstruct sub-cells within mega-cells, impeding complete coverage.

To mitigate this issue, researchers have proposed extensions like the full-STC algorithm, enabling robots to maximize area coverage by addressing free sub-cells. Optimization efforts focus on improving cell assignment and task distribution. Auction-based algorithms and wall-following approaches facilitate obstacle navigation within mega-cells. Yet, minimizing backtracking and increasing coverage rates remain priorities, especially in scenarios with partially occupied mega-cells.

Recent advancements aim to enhance energy efficiency and fault tolerance in real-world applications. Hybrid approaches combining frontier-based exploration with STC algorithms reduce

energy consumption. Nonetheless, challenges persist, such as workload distribution imbalance. Despite these challenges, STC algorithms offer a systematic and efficient solution, particularly in environments with static obstacles. However, for large fields and complete coverage, discretization of the complete field is not feasible and optimal due to computational constraints and scalability issues.

2.5 Sampling-Based Planning Algorithms for Obstacle Avoidance

Sampling-based planning algorithms have become prominent tools for solving complex path planning problems in robotics, particularly in environments with numerous obstacles. These algorithms use random sampling to explore the configuration space, offering both heuristic and optimal solutions. Their probabilistic nature allows them to effectively handle the uncertainty and complexity inherent in real-world scenarios, making them suitable for applications requiring efficient navigation and obstacle avoidance. An overview of sampling-based planning algorithms is available in the paper cited as [14].

2.5.1 Probabilistic Roadmap (PRM)

The Probabilistic Roadmap (PRM) algorithm constructs a roadmap by randomly sampling configurations within the robot's environment and connecting them with collision-free paths. This roadmap serves as a global map, enabling efficient path planning between start and goal configurations. While PRM is effective for static environments and provides a comprehensive exploration of the space, its random node placement can limit coverage near boundaries and obstacles, and it may incur high computational costs in densely populated environments.

2.5.2 Randomized Potential Field (RPF)

The Rapidly Exploring Random Tree (RRT) algorithm incrementally builds a tree by randomly sampling configurations and expanding the tree toward unexplored regions or towards the goal. RRT is suited for static environments due to its ability to rapidly explore the configuration space without needing a precomputed roadmap. However, while RRT efficiently finds feasible paths, the paths may not always be optimal. Variants like RRT* have been developed to improve path quality by providing asymptotically optimal solutions, addressing challenges in navigating through narrow passages and cluttered spaces.

However, sampling-based algorithms are not ideally suited for complete coverage path planning, as they are primarily designed for point-to-point navigation rather than comprehensive area coverage. Modeling non-holonomic constraints in coverage path planning presents additional challenges that necessitate alternative approaches. Implementing a sampling-based method for complete coverage by generating a roadmap for the entire environment is computationally intensive and impractical for large-scale environments.

2.6 Greedy Search Algorithms in Coverage Path Planning

Greedy search algorithms, such as Dijkstra's algorithm, make decisions based on the local optimal choice at each step, without considering global implications. For these approaches to find the goal point, a graph has to be generated before hand and shortest path will be ensured from that graph. This heuristic method is simple and fast but does not guarantee globally optimal solutions due to its short-term focus. In robotics, graph search algorithms like A*, D*, and Theta* are commonly used to plan and optimize coverage paths, employing strategies like boustrophedon motion or spiral patterns. These algorithms are crucial for obstacle avoidance, dynamically re-planning paths when encountering obstacles or blind spots to ensure continuous coverage. However, path searching in large grid maps

poses computational challenges due to the vast search space, necessitating ongoing advancements to improve efficiency and reduce computation costs.

2.6.1 Dijkstra's Algorithm in Coverage Path Planning

Dijkstra's algorithm is used to find the shortest path from a single source node to the goal node in a graph with non-negative edge costs. In coverage path planning (CPP), it helps optimize path sequences and minimize traversal costs, ensuring thorough coverage while avoiding obstacles. Applications include indoor navigation and optimizing coverage paths with minimal resource consumption. Despite its effectiveness. It will find the shortest path and is computationally expensive as compared to other graph search approaches.

2.6.2 A* Algorithm in Coverage Path Planning

The A* algorithm combines actual and estimated costs to determine the shortest path from a start node to a goal node, making it effective for CPP where minimizing cost is crucial. It has been used to optimize path sequences, reduce processing time, and ensure comprehensive coverage while avoiding obstacles. A* is valuable in CPP applications, balancing efficient pathfinding with comprehensive coverage, especially when tailored to address specific challenges in complex environments.

2.6.3 Theta* Algorithm in Coverage Path Planning

Theta* allows pathfinding in the graph, enabling more flexible and efficient navigation compared to other methods like A*. It is particularly useful for CPP in environments where precise path planning is essential. Practical applications include cleaning robots, where Theta* optimizes local backtracking paths to improve coverage time.

The aforementioned graph search approaches are effective when the graph is predefined and the objective is to find the shortest path from source to goal. However, in coverage path planning, the objective shifts to covering the entire area. Prioritizing straight paths makes generating a comprehensive graph for the entire environment computationally expensive and does not guarantee the generation of an optimal or near-optimal graph over the complete region that ensure the near optimal path.

2.7 Combinatorial Planning Techniques

Combinatorial planning techniques emerged as a response to the need for efficient and systematic methods to navigate robots through complex environments cluttered with obstacles. As robotics applications expanded into domains such as manufacturing, logistics, and exploration, the demand for reliable path planning algorithms became increasingly pronounced. Traditional approaches often struggled to cope with the intricacies of real-world environments, leading to the development of combinatorial planning techniques.

These techniques were invented to provide a structured framework for path planning by discretizing the continuous configuration space into a graph-based representation. By breaking down the problem into manageable components, combinatorial planning methods aimed to overcome the challenges posed by obstacles and non-trivial workspace geometries. They offered a systematic way to explore the configuration space, enabling robots to navigate from an initial pose to a desired goal while avoiding collisions. Key algorithms in this field include Visibility Graphs, Voronoi Diagrams, Exact Cell Decomposition, and Approximate Cell Decomposition, each tailored to address specific challenges and requirements.

2.7.1 Visibility Graphs

Visibility Graphs are a fundamental combinatorial planning technique used to navigate robots through environments cluttered with obstacles. This method constructs a graph by connecting the initial and goal configurations through vertices representing the obstacles. By leveraging visibility between vertices, Visibility Graphs systematically derive collision-free paths that optimize for efficiency and optimality. The paths generated inherently minimize distance and traversal time, leading to efficient navigation. While Visibility Graphs are robust and produce optimal paths, their computational overhead in complex environments with numerous obstacles remains a limitation.

2.7.2 Voronoi Diagrams

Voronoi Diagrams offer a distinct approach in combinatorial planning by generating collision-free paths through a tessellation of space into regions based on proximity to a set of points. Each region, or Voronoi cell, encompasses locations closer to its defining point than to any other. This partitioning provides valuable insights into potential navigation paths, offering robustness to complex environments with irregular obstacle geometries. Voronoi Diagrams prioritize paths that maintain safe distances from obstacles, enhancing safety. However, they may produce sub-optimal paths in intricate environments and involve significant computational costs in construction.

2.7.3 Exact Cell Decomposition

Exact Cell Decomposition decomposes the free configuration space into trapezoidal cells using vertical side segments from polygon vertices, simplifying the path planning process by focusing on individual regions. This approach ensures comprehensive exploration of the environment, making it suitable for applications like surveillance, mapping, and search-and-rescue missions. It offers increased clearance from obstacles and scalability to varying environment sizes. However, it tends to produce sub-optimal paths in complex obstacle configurations and is limited to polygonal obstacles, which restricts its utility in non-polygonal environments.

2.7.4 Approximate Cell Decomposition

Approximate cell decomposition offers a grid-based approach to path planning, dividing the configuration space into fixed or variable-sized grids and labeling each cell as free or occupied to represent free space and obstacles. The advantage of this method is its flexibility in handling obstacles of various shapes and sizes, allowing for greater versatility in navigating complex environments. Pre-generated grids reduce computational time during path planning, resulting in faster planning times compared to methods that require graph generation and search.

However, approximate cell decomposition has limitations. It will struggle with dynamic obstacles, needing frequent grid updates to accommodate environmental changes, which increases complexity and decreases efficiency. Fixed resolution constraints can limit its ability to represent fine-grained details, affecting accuracy and precision in intricate environments. Additionally, representing a very large region with high-resolution grids will be computationally expensive, as maintaining detailed obstacle and free space information increases computational overhead.

2.8 State Lattice Planning

State lattice planning as stated in [15] is a systematic method designed to generate feasible paths for robots navigating continuous environments with non-holonomic constraints. This approach involves

discretization, where the continuous space is divided into a finite number of nodes, each representing a possible robot pose, accounting for both position and orientation. The process continues with node expansion, ensuring that each node adheres to the robot's physical constraints, making the generated states traversable. This is followed by connectivity, where expanded states are connected to form a graph that represents possible paths through the environment. Path generation utilizes a graph search algorithm to identify the optimal path from the start to the goal state following the non-holonomic constraints.

State lattice planning offers several key advantages for robotic navigation. It ensures completeness, guaranteeing that if a feasible path exists, the algorithm will find it. This method also provides an optimal solution by identifying the shortest path, minimizing travel distance and time. Additionally, state lattice planning can dynamically replan paths in real-time in response to environmental changes or new constraints, enhancing the robot's adaptability and safety in dynamic scenarios.

Despite its advantages, state lattice planning has the limitation that generating an approximate straight path often necessitates a dense tree of nodes. This significantly increases computational time, particularly in environments with numerous obstacles. Thus, while state lattice planning provides a robust framework for path planning, its effectiveness can vary based on the specific requirements and constraints of the robotic application.

In addition to these classical techniques, several heuristic algorithms address exploration and coverage problems. These include boustrophedon motion, internal spiral algorithms, Voronoi partition approaches, Brick and Mortar algorithms, to name a few. Each algorithm offers unique strategies and approaches to address specific challenges in exploration and coverage, contributing to the diverse landscape of combinatorial planning in robotics.

2.9 Other Classical and Heuristic Algorithms.

In the realm of path planning for robotics, evolutionary algorithms (EAs) and human-inspired approaches have garnered considerable attention for their ability to find optimal or near-optimal solutions to complex optimization problems. One prominent example is Genetic Algorithms (GA), a metaheuristic inspired by natural genetic evolution, which has been extensively employed in solving various path planning problems. GA operates by iteratively evolving a population of potential solutions through mechanisms like crossover and mutation, eventually converging towards a solution that meets predefined criteria.

While GA offers the advantage of global search capability, it often suffers from poor stability and high computation time, particularly in scenarios with large search space complexity. To address these limitations, researchers have proposed enhancements such as multi-objective GA and hybrid approaches combining GA with other techniques like Dynamic Programming (DP) or simulated annealing. These adaptations aim to improve convergence speed and solution quality while mitigating the computational burden.

Another noteworthy EA is Differential Evolution (DE), which offers advantages such as quick convergence and robustness. DE operates by iteratively generating trial vectors through mutation, recombination, and selection processes, making it particularly suitable for optimization problems with complex search spaces. Researchers have explored various modifications to DE, such as combining it with roulette and multi-neighborhood operations, to enhance its performance in path planning tasks.

In addition to EAs, swarm intelligence algorithms have gained prominence for their ability to emulate collective behavior observed in natural systems. Particle Swarm Optimization (PSO), Ant

Colony Optimization (ACO), and Bee Colony Optimization (BCO) are notable examples of swarm intelligence algorithms applied to path planning. These algorithms leverage the collective intelligence of swarm agents to efficiently explore and optimize paths in complex environments. However, they may face challenges such as local optima trapping and slow convergence rates, prompting researchers to propose enhancements like distributed algorithms and improved pheromone updating rules.

On the other hand, human-inspired algorithms, such as neural networks and reinforcement learning (RL), draw inspiration from the workings of the human brain to optimize decision-making processes in path planning. Neural networks, including feedforward and convolutional architectures, have been utilized to learn complex mappings between sensory inputs and actions, enabling robots to navigate and plan paths in dynamic environments. Reinforcement learning, a subset of machine learning, allows agents to learn optimal behaviors through trial-and-error interactions with the environment. RL algorithms like Q-learning and Deep Q-Networks (DQN) have shown promise in optimizing path planning tasks, albeit with challenges related to convergence speed and scalability.

Despite their potential, evolutionary and human-inspired approaches will not be suitable for scenarios requiring real-time path planning or where computational efficiency is paramount. These methods typically involve iterative optimization processes that will incur significant computation time, making them less practical for time-sensitive applications. Thus, while these approaches offer valuable insights into path planning optimization, their adoption may depend on the specific requirements and constraints of the robotics task at hand.

2.10 Inspiration drawn from existing approaches

From the approaches discussed in the literature review, some are efficient in computation time using heuristics but do not follow non-holonomic constraints. Others follow non-holonomic constraints but do not prioritize naturally straight paths, while some algorithms focus solely on path optimization, resulting in high computational times for generating the optimal path. Hence, it is clear that none of the above approaches can be used directly for our problem statement. A new approach was needed that reduces computational time using heuristics, adheres to non-holonomic constraints, and prioritizes straight paths.

To develop an efficient and effective path planning algorithm for coverage path planning, inspiration was drawn from state-of-the-art approaches to build and enhance the algorithm. They are listed as follows:

- Implementation of a vision cone approach to prioritize linear paths.
- Application of a combined criterion to select the optimal point within the vision cone.
- Computing the centroids from the intersection of the union of multiple regions to derive points from regions.
- Modeling the non-holonomic constraints in the global path to closely resemble the local path.
- Utilization of the Dubins open traveling salesman problem to determine the shortest path across multiple points.
- Implementation of decoupled Dubins constraints following the generation of linear paths.
- Adoption of a grid-based methodology for obstacle representation, while employing continuous space for free space.
- Leveraging state lattice planning to construct a graph that adheres to non-holonomic constraints.

- Implementation of the A* algorithm to ascertain the optimal path from the generated graph.
- Application of Reeds-Shepp paths to determine the shortest route between two points in the local path for efficient weed coverage.

Problem Formulation

We consider a robotic scenario wherein a four-wheel robot equipped with a mechanical extraction system that operates within a defined rectangular area. Due to its non-holonomic nature, the robot imposes kinematic constraints on turning, enforcing a minimum turning radius of 2 meters.

The operational environment comprises grass fields assumed to be uniform without any slopes (z=0), covering a real area of 120x90 square meters. Weed distribution within this area is heterogeneous, with approximately 60 percent of points clustered following a Gaussian distribution with varying variances. Weed positions are obtained via drone-based data collection. This dataset serves as the basis for complete coverage path planning.

Given the robot's mechanical extraction implementation width of 60 cm, the operational region of the robot is modeled into points with each point representing an area of 30 cm, facilitating path planning optimization.

The primary objective is to develop a path planning algorithm capable of covering all weed points within the designated area while adhering to the robot's non-holonomic constraints. Additionally, the algorithm aims to generate paths that approximate straight lines where feasible, ensuring comprehensive coverage of all points. The algorithm should prioritize finding the shortest path distance to cover all weed points effectively while meeting the objectives as stated above.

The objectives of the proposed algorithm include:

- **Realistic Path Generation:** Develop paths that mimic natural movement patterns, enhancing operational realism.
- **Computational Efficiency:** Minimize processing time and resources required for path planning.
- **Energy Conservation:** Optimize energy consumption during path execution, enhancing overall operational efficiency.
- **Coverage Rate:** Maintain a high coverage rate of weed points across the dataset, ensuring comprehensive weed removal.

Experimental Setup

1 Environment

The environment for our coverage motion planning algorithm is situated in a grass field characterized by small grass uniformly distributed across the area. The grass field is representative of typical agricultural settings, where weed management is essential for maintaining crop quality and productivity. As the grass matures, it is common for various unwanted plants, particularly weeds, to grow sporadically throughout the field. Our primary focus in this study is on the removal of Rumex (commonly known as dock weeds), which pose a significant challenge to farmers. However, the algorithm can be adapted to target other types of weeds based on specific requirements, the only that change would be the detection system to identify the target weed.

2 Importance of Weed Removal

We are concentrating on Rumex plants due to their detrimental impact on agricultural productivity and livestock health. Although Rumex plants are not inherently toxic, their presence in cattle feed can adversely affect the quality of milk production. Ensuring the purity of grass fed to cattle is crucial for producing high-quality, bio milk, which is not only more beneficial for human consumption but also promotes better health and well-being of the cattle. Pure grass feed leads to higher nutritional value in milk, contributing to improved dairy products. Therefore, effective weed management, particularly the removal of Rumex plants, is essential for maintaining the quality and productivity of grass fields.

Traditionally, farmers have resorted to manually removing these weeds, a labor-intensive and time-consuming process. Manual removal becomes particularly arduous in large fields, imposing significant physical strain on farmers and limiting the efficiency of weed management. Automating this process with robotic systems offers a promising solution to enhance agricultural practices and improve farmers' quality of life.

3 Constraints

With a comprehensive understanding of the robot's capabilities and features, it is essential to delve into the constraints that arise due to its mechanical system and the nature of the field in which it operates. These constraints significantly influence the development of an effective motion planning algorithm.

Non-Holonomic Nature: The robot is equipped with four regular wheels, limiting its movement capabilities. Unlike holonomic robots, which can move in any direction, our robot cannot move sideways. It is constrained to forward and backward movements and must make turns to change its direction. This non-holonomic nature adds a layer of complexity to the motion planning algorithm, as it must account for the robot's inability to change its heading instantaneously.

Turning Radius: The robot has a minimum turning radius of 2 meters, meaning it cannot make sharp turns. This constraint necessitates careful planning to ensure the robot can navigate around obstacles and reach all designated weed points without making turns that exceed its turning capabilities.

Speed Variations: The robot's speed must be carefully regulated to prevent damage to the grass and ensure precise weed removal. When moving in a straight path, the robot operates at a velocity of 0.8 m/s. However, when making a turn, the velocity is reduced to 0.4 m/s to maintain the 2-meter turning radius. This adjustment helps in maintaining stability and accuracy during turns, which is critical for avoiding collateral damage to the grass and ensuring efficient weed extraction.

With a clear understanding of the robot's constraints, we can now proceed to develop a motion planning algorithm that leverages these constraints to ensure comprehensive and efficient weed removal. By integrating considerations for non-holonomic movement, adaptive speed control, optimized coverage, obstacle avoidance, and environmental adaptability, the algorithm will facilitate the robot's autonomous operation, making the weed removal process more efficient and reducing the burden on farmers. This strategic approach not only enhances the robot's performance but also contributes to maintaining the health of the grass and improving the overall quality of the field.

Methodology

The main algorithm is comprised of two primary steps: data preprocessing and the behavior algorithm. In the data preprocessing phase, regions are converted into points by computing the centroids of their overlaps. The behavior algorithm phase utilizes these computed points to generate the complete path.

1 Data Preprocessing

Data preprocessing is a pivotal step in the research methodology, especially in the context of coverage path planning for weed removal in agricultural fields. This section delineates the comprehensive preprocessing procedures employed to ensure the precision and efficacy of the data used in the subsequent analysis.

Data Acquisition: The initial phase of preprocessing involves the acquisition of data pertaining to the weed positions within the field. This data collection is facilitated through the use of a drone for pinpointing the locations of weeds. The drone performs a systematic survey of the field, capturing high-resolution images to detect weed positions. The resultant data is then utilized as input for the coverage path planning module. Due to the developmental status of the drone, the current implementation involves manual data acquisition using a real-time kinematic GPS (RTK) system. The RTK system offers centimeter-level accuracy in pinpointing weed locations, which, for the purpose of this research, is considered sufficiently precise.

Handling Data Uncertainty: In a practical scenario, the drone's output would include positional data of weeds along with an associated uncertainty measure, reflecting the inherent imprecision of the system. However, given the reliance on RTK data for this research phase, we assume perfect positional accuracy. This uncertainty measure will be used to define the regions of influence for each weed, ensuring that the coverage path planning algorithm accounts for the imprecision in the data.

Region Definition and Overlap Management: When data points from the drone indicate weed positions, these points often come with overlapping regions due to the growth patterns of weeds like Rumex, which tend to cluster.

Overlap Reduction: To address overlaps, the preprocessing algorithm calculates the centroids of overlapping regions. Points within overlapping regions are consolidated to avoid duplication, ensuring that each weed cluster is represented by a single centroid. This consolidation reduces redundancy and enhances the efficiency of the coverage path planning.

Non-Overlapping Weeds: For weeds whose regions do not overlap with others, the center of each weed is directly considered as a centroid. This step ensures that isolated weeds are accurately accounted for in the data set. The centroids of both overlapping and non-overlapping regions can be visualized in the Figure 1. The green points represent the weeds, green circles represent the region of the weeds, and the red points represent the centroids of the regions.

Data Integration and Optimization: By processing the data to identify centroids and manage overlaps, we achieve a significant reduction in the total number of points. Preliminary results indicate a reduction of at least 40% in the number of data points, optimizing the dataset for coverage path

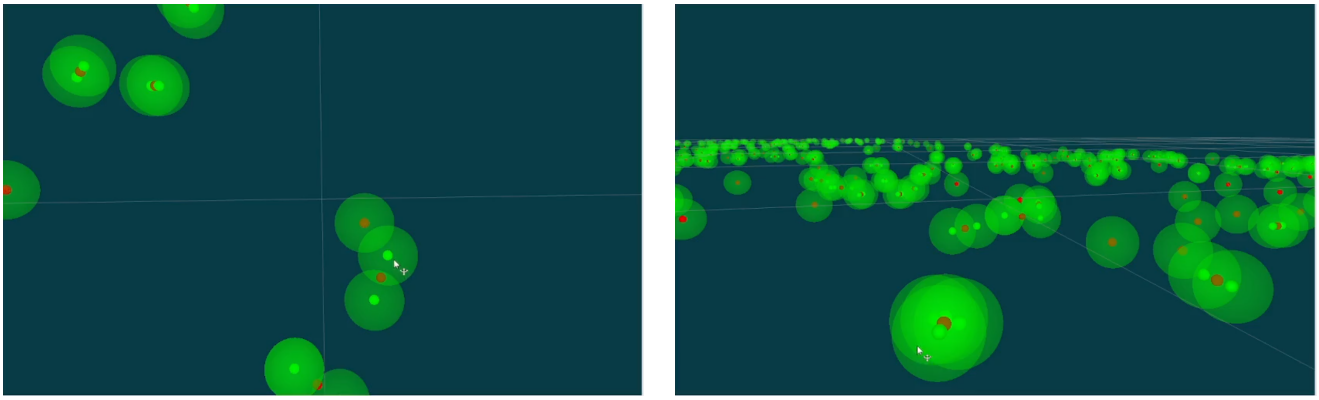


Figure 1: Centroids of overlapped regions.

planning. This reduction not only minimizes the total traversal length required for weed removal but also conserves energy and reduces redundant revisits.

The processed data is then fed into the coverage path planning module, which utilizes the optimized set of points to devise an efficient path for weed removal, ensuring comprehensive coverage with minimal resource expenditure. The following figure illustrates the data preprocessing workflow, highlighting the steps taken to transform raw positional data into an optimized dataset ready for coverage path planning.

This preprocessing approach ensures that the data fed into the coverage path planning algorithm is both accurate and efficient, laying a robust foundation for effective weed removal operations in agricultural fields.

2 Behavioral CPP Algorithm

Following the preprocessing phase aimed at resolving the regions associated with designated points, the subsequent imperative lies in formulating an algorithm capable of comprehensively covering all identified points. With the completion of preprocessing, the focus narrows down to ensuring the precise coverage of all points to effectively identify and extract weed infestations. Although the agricultural robot in question operates under non-holonomic constraints, the overarching objective transcends mere efficiency and the identification of the shortest path. Instead, the primary emphasis lies in achieving exhaustive point coverage while strategically favoring approximately linear trajectories.

The rationale behind prioritizing linear trajectories over strictly adhering to non-holonomic paths is multifaceted. Firstly, the adoption of more curved paths significantly heightens the risk of grass damage, thereby undermining the fundamental objective of preserving grass quality. Given the paramount importance of maintaining optimal grass conditions within agricultural fields, any approach that compromises this aspect inherently fails to align with the core objectives. Secondly, the energy consumption associated with traversing curved paths is substantially higher compared to linear trajectories. For the specific agricultural robot under consideration, empirical estimates suggest that traversing an equivalent path length via curved trajectories incurs an energy expenditure four times greater than that of linear paths. Consequently, the central objective of the algorithm resides in identifying the shortest path capable of encompassing all designated points while mitigating curvature and prioritizing linear trajectories.

The development of this behavioral approach necessitates a nuanced understanding of agricultural terrain dynamics, robot kinematics, and energy efficiency considerations. By integrating these facets into the algorithmic design process, the resultant solution seeks to strike a delicate balance between point coverage efficacy, grass preservation, and energy optimization.

2.1 Vision Cone Strategy:

At this stage, having acquired comprehensive global information about the points in the field, the next step involves prioritizing straight paths while minimizing computational complexity and time. An innovative approach is employed wherein the robot is equipped with a vision cone mechanism. This vision cone is defined by two lines extending from the robot at a fixed angle and distance. The angle of these lines is determined based on the robot's minimum turning radius. For instance, for a robot with a minimum turning radius of 2 meters, the angle of the cone on either side is set at 11 degrees. The distance to the end of the cone depends on the operational area of the robot but is set to a fixed distance of 100 meters for this scenario.

The vision cone allows the robot to consider only those points within this cone from its current position as potential next travel points. This selective consideration significantly reduces the computational effort required to determine the path, as it disregards points outside the cone. By narrowing the focus to relevant points within the vision cone, computational efficiency is enhanced, thus making the vision cone an intelligent and effective strategy. The vision cone can be visualized in (Figure Figure 2).

2.2 Algorithmic Framework:

The algorithmic framework is designed to facilitate comprehensive point coverage while minimizing curvature and prioritizing linear trajectories. The behavioral approach adopted for the algorithm is

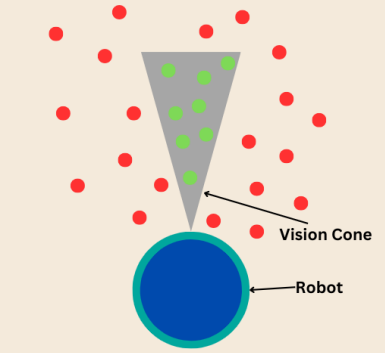


Figure 2: Vision Cone.

hierarchical, comprising three distinct behaviors that are sequentially activated. The transition from one behavior to the next is contingent upon the degree of point coverage achieved.

1. **Centroid-Seeking Behavior:** The algorithm commences with the first behavior, designed to initiate coverage from the starting point. This stage focuses on covering a substantial portion of the field, leveraging the vision cone to select the next travel points and maintaining the priority on straight paths.

2. **Circumferential-Traversal Behavior:** Upon achieving a certain threshold of point coverage, the algorithm transitions to the second behavior. This intermediate stage aims to further optimize coverage by adjusting the strategy based on the points that remain. The robot continues to utilize the vision cone but adopt a slightly more flexible criteria for point selection to ensure efficient coverage progression.

3. **Dubins Open Travelling Salesman Problem:** Once the intermediate behavior reaches its saturation point—where additional coverage gains diminish—the algorithm shifts to the final behavior. This stage is designed to ensure complete coverage of all remaining points. The final behavior will incorporate more refined strategies to target any residual areas, ensuring no point is left uncovered.

This hierarchical behavioral algorithm ensures a methodical and efficient approach to coverage path planning. By starting with broad coverage strategies and progressively refining the approach, the algorithm effectively balances the need for comprehensive point coverage with the constraints of the robot's kinematic capabilities and the operational goal of preserving grass quality. The pseudocode for the complete coverage path planning algorithm is provided in (Algorithm 1).

Convention to be followed for the algorithms:

- P: path.
- p: points.
- r: robot.
- R: Radius.
- VC: Vision cone.
- C: Centroid.
- N_s: Number of sample orientations.
- δ: Percentage.
- $step$: Step size.
- O: Orientation.

Notations for CompleteBehavioralAlgorithm:

- p_{cl}: clustered points.
- p_r: remaining points.
- δ_{bc}: behavior change percentage.
- δ_c: coverage percentage.
- P_c: completed path.
- P_s: straight path.
- R_{conc}: concurrent region radii.
- P_d: Dubins path.
- P_{dotsp}: DOTSP path.
- P_{cd}: Complete Dubins path.

Algorithm 1 CompleteBehavioralAlgorithm

Input: 2D points (p_{2d}), initial robot pose (r_{pos}), turning radius (R_{tu})
Output: Dubins path P_{cd}

$p_{cl}, \delta_{\text{bc}} \leftarrow$ CentroidsAndAutoBehaviorShift(P_{2d}, $R_{\min}$, $R_{\max}$, N)
$p_r \leftarrow p_{cl}$
$\delta_c \leftarrow 0$
$P_c \leftarrow []$
while True **do**
    if $\delta_c < \delta_{\text{bc}}$ **then**
        $P_s, p_r \leftarrow$ Behavior_1(p_r, p_{cl}, N_s, r_{pos}, VC, C, R_{conc}, $step$)
    else
        $P_s, p_r \leftarrow$ Behavior_2(p_r, p_{cl}, N_s, r_{pos}, VC, C, R_{conc}, $step$)
    end if
    $P_c \mathrel{+}= P_s$
    $r_{\text{pos}} \leftarrow P_c[-1]$
    $\delta_c \leftarrow$ UpdateCoveragePerc(P_c, p_r)
    if len$(P_s) == 0$ **then**
        break
    end if
end while
$P_d \leftarrow$ DubinsPath(P_c, R_{tu})
$P_{dotsp} \leftarrow$ DOTSPPath(p_r, r_{pos}, R_{tu})
$P_{cd} \leftarrow P_d + P_{dotsp}$
return P_{cd}

2.3 Rationale for Hierarchical Approach:

The hierarchical approach is adopted to improve convergence rate and coverage efficiency. If a single algorithm or behavior were followed throughout, the robot would cover many points initially but gradually cover fewer points over time, reducing the algorithm's accuracy and increasing the overall path length and operational time. By transitioning between different behaviors, the algorithm can adapt to the changing density and distribution of points, maintaining high efficiency throughout the coverage process.

This hierarchical strategy ensures that the algorithm remains effective even as the number of uncovered points decreases. By tailoring the approach to the specific conditions encountered at each stage, the robot can optimize its path, reduce unnecessary movements, and maintain high precision in point coverage. This not only conserves energy but also preserves the quality of the grass by minimizing excessive traversal. The adaptive nature of the hierarchical approach thus represents a robust and efficient solution for coverage path planning in agricultural fields.

2.4 Centroid-Seeking Behavior

The algorithm begins by receiving the centroids of the overlaps between the regions of the raw points as input. It takes the initial position and orientation of the robot, where the position remains fixed while the orientation is treated as a temporary orientation. From this temporary orientation, the algorithm considers an angular range of twenty-four degrees on either side. Within this range, it samples six orientations on each side, resulting in a total of thirteen orientations, including the initial orientation.

The algorithm designates the current orientation as the leftmost extreme orientation. Using this position and orientation, it then starts selecting the next point to navigate to. At this stage, the algorithm employs the vision cone mechanism, characterized by an 11-degree angle on either side and extending 100 meters forward. The primary objective at this juncture is to select the next most suitable point within the vision cone.

To determine the next best suitable point, the algorithm extracts the five closest points from the current position and orientation within the vision cone. These points are regarded as semi-potential points. For each of these semi-potential points, the algorithm computes the distribution of points on either side of the current orientation across the entire vision cone. A combined score is then calculated for each semi-potential point based on the distance from the robot and the total distribution of points in the vision cone. Both the distance and the distribution are normalized before calculating the combined score. The semi-potential point with the highest combined score is deemed the best potential candidate and is selected as the next point to navigate to. This scoring method is intelligent as it considers not only the distance but also the future distribution of points, thereby facilitating more efficient coverage as the robot progresses.

Once the best potential point is selected, it becomes the next point for navigation. The robot's orientation at this new point is updated based on the direction vector from the robot to the selected point. The robot then moves to this potential point with the updated orientation. This process is repeated, with the robot continually navigating to the next potential point until no points are visible within the vision cone. This pattern encourages the robot to move in an approximately straight line whenever possible. When the robot can no longer find any points within the vision cone, it indicates that the robot has reached the extreme end of the points and is considered as one temporary turn. At this point, the number of points covered in this orientation is recorded.

Returning to Initial Position and Orientation

After completing one temporary turn, the robot will return to its starting position and orientation. The robot then considers the next orientation in the set of thirteen sampled orientations and proceeds to complete an approximately straight path in that direction, recording the number of points covered along each path. This process is repeated for all thirteen orientations. Upon completing paths for all orientations, the orientation that results in the maximum number of points covered is selected as the optimal orientation for further navigation.

Incorporating Non-Holonomic Constraints for Turns

When making a turn, the robot must adhere to its non-holonomic constraints. To address this, the algorithm considers a circular region around the robot with defined minimum and maximum radii. The minimum radius is set to one and a half times the robot's minimum turning radius, ensuring feasible turns. The initial maximum radius is set to three times the minimum turning radius. If no points are found within this initial circular region, the outer radius is incrementally increased by two units until the maximum radius limit is reached.

Within this circular region, the algorithm identifies all potential points and filters them further. The selection process involves evaluating each point based on a combined score, considering both the distance from the robot and the angle differnce between the robot's orientation and the vector from the robot to the potential point. both the entities are normalized before comparison. A lower distance and a smaller orientation deviation result in a higher score for the point. The point with the highest score is selected as the next potential point for the robot to navigate to when making a turn.

Orientation Towards Centroid

At the beginning of the algorithm, the centroid of the entire set of points is computed. When selecting a point for the turn, the orientation is determined by the vector from the selected point towards this centroid. This selected point becomes the next navigation target, and the orientation towards the centroid becomes the new temporary orientation for the robot.

Continuing the Path and Making Turns

Upon reaching the end of a complete path, the robot needs to execute a turn. This turn is represented by a single completed path, after which the robot continues the same pattern of operation. From the point where the turn is made, the robot selects thirteen orientations and navigates along each one, completing a path and recording the number of points covered. The orientation that results in the maximum number of points covered is then selected as the optimal orientation for continued navigation. This process is iterative and continues throughout the first behavior.

Focus on Points Near the Centroid:

This first behavior of the robot emphasizes covering more points near the centroid of the data and not covering points on the outer sides. Initially, the robot covers a significant number of points with each path. However, as the number of turns increases, the coverage rate decreases. This reduction in efficiency occurs because the robot focuses on points near the centroid as after each turn it faces toward the centroid, resulting in majority of the points covered are in the central region. Consequently, while this behavior ensures substantial coverage, it becomes less efficient over time as fewer new points are covered with each turn.

The first behavior algorithm can be visualized in (Algorithm 2)

Notations for Behavior 1 and 2:

- P_{cs}: Complete straight path
- P_t: Temporary path
- p_{co}: Points covered
- O_{sa}: Sampled orientations
- p_v: Visible points
- p_{po}: Potential point
- O_n: New orientation
- p_{int}: Intermediate points
- O_{bi}: Best orientation index
- P_{tu}: Turn point
- O_b: Best orientation

Algorithm 2 Behavior_1

Input: 2D points (P_{2d}), clustered points (p_{cl}), number of sample orientations (N_s), robot pose (r_{pos}), vision cone (VC), centroid (C), concurrent region radii (R_{conc}), step size (S)
Output: Complete straight path P_{cs}, remaining points p_r

$P_{cs} \leftarrow []$
$P_t \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$p_{co} \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$O_{sa} \leftarrow \text{SampleTheOrientations}(R_{pos}[2], N_s, S)$
for i, O **in** O_{sa} **do**
    $R_{pos}[2] \leftarrow O$
    while no point is visible **do**
        $p_v \leftarrow \text{ComputeVisionConePoints}(r_{pos}, VC)$
        $p_{po} \leftarrow \text{FindPotentialPoint}(r_{pos}, p_v)$
        if p_{po} is None **then**
            break
        end if
        $O_n \leftarrow \text{FromCurrentPoseToPotentialPoint}(r_{pos}, p_{po})$
        $P_t[i].\text{append}([p_{po}, O_n])$
        $p_{int} \leftarrow \text{CheckIntermediatePoints}(p_r)$
        $P_t[i].\text{append}(p_{int})$
        $p_c[i].\text{append}(\text{len}(P_t))$
        $p_r \leftarrow p_{cl} - P_t$
        $r_{pos} \leftarrow [p_{po}, O_n]$
    end while
end for
$O_{bi} \leftarrow \text{argmax}(p_c[:])$
$P_{cs} \leftarrow P_t[O_{bi}]$
$r_{pos} \leftarrow P_{cs}[-1]$
$p_{tu} \leftarrow \text{PotentialPointToTurn}(p_r, R_{\text{conc}}, r_{pos})$
$O_b \leftarrow \text{TowardsCentroid}(p_{tu}, C)$
$P_{cs}.\text{append}([p_{tu}, O_b])$
$p_r.\text{remove}([p_{tu}])$
return P_{cs}, p_r

2.5 Circumferential-Traversal Behavior:

Circumferential-Traversal Behavior is designed to enhance the coverage rate and improve the convergence efficiency. Upon transitioning from the first to the second behavior, the algorithm takes all remaining points as input, considering the current position and orientation of the robot. At this stage, the density of uncovered points is lower near the centroid and higher towards the boundaries of the operational area. Therefore, the second algorithm focuses on covering points located along the periphery rather than near the centroid.

Orientation Sampling and Path Selection:
The algorithm begins similarly by selecting thirteen orientations from the robot's current position and orientation. The robot navigates along each orientation, completing a path and recording the number of points covered. The orientation resulting in the highest number of points covered is selected

as the optimal direction for continued navigation. This ensures that the robot always moves in the direction that maximizes point coverage.

Refined Criteria for Selecting Turning Points:
A critical difference in the second behavior lies in the method for selecting the orientation of the next best point to make a turn. The algorithm considers a circular region around the robot with predefined minimum and maximum radii similarly as of first behavior and computes the potential points based on distance and orientation difference. By doing so, the algorithm ensures that the robot makes smooth, efficient turns that align with its non-holonomic constraints.

The point with the least orientation difference and the shortest distance is chosen as the next point for making a turn. Unlike the first behavior, where the orientation towards the centroid was prioritized, the orientation of this potential point is directed towards the next point along the same moving orientation. For example, if the potential point lies clockwise to the robot's current orientation, the robot's new orientation will aim towards the next nearest point with the smallest clockwise orientation difference, and the same applies for counterclockwise points. This new orientation becomes the temporary orientation, and the robot continues the process of orientation sampling and selection with the goal of maximizing point coverage.

Enhanced Coverage and Convergence: This behavior is particularly effective in covering points along the boundaries and in regions with higher point density, thereby increasing the overall coverage rate and convergence efficiency of the algorithm. While the first behavior predominantly focuses on points near the centroid, the second behavior shifts attention to the boundary points, ensuring a more balanced and comprehensive coverage pattern.

Enhancing Coverage with Intermediate Points: To further improve the coverage rate and convergence of the algorithm, an additional mechanism is incorporated into both the first and the second behaviors. This enhancement involves covering intermediate points along each path between two selected points. The algorithm checks for points that lie close to the current path within a certain threshold distance.

The orientation for these intermediate points is determined based on their position. If multiple points exist close to the path, the orientation is directed towards the next intermediate point in sequence. For the last intermediate point, the orientation is towards the final end point.

Another criterion for selecting intermediate points depends on the distance between the end points of the path. If this distance is substantial, points that are slightly farther but within the threshold are also considered. In such cases, the algorithm allows for a minor compromise on the straightness of the path to cover more points, thereby further enhancing the coverage rate and convergence efficiency.

Efficiency of Combined Behaviors: Both the first and second behaviors, along with the enhancement for intermediate points, ensure complete coverage of the points. These behaviors have been experimentally validated to provide good coverage rates and convergence for the algorithm. However, the efficiency in terms of coverage rate, convergence speed, and path length diminishes after approximately eighty-five to ninety percent of the points are covered. At this stage, the algorithm tends to cover fewer points per turn and makes multiple turns to cover just two or three points.

This inefficiency results in longer paths and higher energy consumption, which contradicts the goal of minimizing path length and conserving energy. This challenge highlights the necessity for a third behavior. The third behavior aims to optimize the coverage of remaining points by allowing

slight deviations from straight paths to cover more points efficiently, thus reducing the overall path length and energy consumption.

The pseudocode for the second behavior algorithm is shown in (Algorithm 3)

Algorithm 3 Behavior_2

Input: 2D points (P_{2d}), clustered points (p_{cl}), number of sample orientations (N_s), robot pose (r_{pos}), vision cone (VC), centroid (C), concurrent region radii (R_{conc}), step size (S)
Output: Complete straight path P_{cs}, remaining points p_r

$P_{cs} \leftarrow []$
$P_t \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$p_{co} \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$O_{sa} \leftarrow \text{SampleTheOrientations}(R_{pos}[2], N_s, S)$
for i, O **in** O_{sa} **do**
    $R_{pos}[2] \leftarrow O$
    while no point is visible **do**
        $p_v \leftarrow \text{ComputeVisionConePoints}(r_{pos}, VC)$
        $p_{po} \leftarrow \text{FindPotentialPoint}(r_{pos}, p_v)$
        if p_{po} is None **then**
            break
        end if
        $O_n \leftarrow \text{FromCurrentPoseToPotentialPoint}(r_{pos}, p_{po})$
        $P_t[i].\text{append}([p_{po}, O_n])$
        $p_{int} \leftarrow \text{CheckIntermediatePoints}(p_r)$
        $P_t[i].\text{append}(p_{int})$
        $p_c[i].\text{append}(\text{len}(P_t))$
        $p_r \leftarrow p_{cl} - P_t$
        $r_{pos} \leftarrow [p_{po}, O_n]$
    end while
end for
$O_{bi} \leftarrow \text{argmax}(p_c[:])$
$P_{cs} \leftarrow P_t[O_{bi}]$
$r_{pos} \leftarrow P_{cs}[-1]$
$p_{tu} \leftarrow \text{PotentialPointToTurnCCW}(p_r, R_{\text{conc}}, r_{pos})$
$O_b \leftarrow \text{VectorsTowardsNextCCWPoint}(p_{tu}, p_r)$
$P_{cs}.\text{append}([p_{tu}, O_b])$
$p_r.\text{remove}([p_{tu}])$
return P_{cs}, p_r

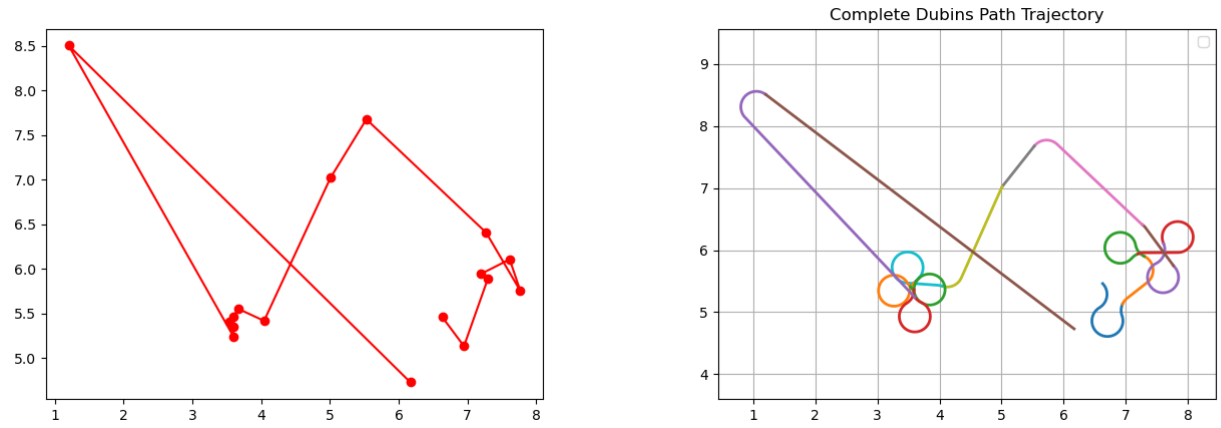
2.6 Dubins open travelling salesman problem (DOTSP):

The final behavior addresses the coverage of the remaining points, which are sparsely distributed across the area. At this stage, the concept of the vision cone is removed, as maintaining strict straightness is no longer prioritized. Although straight paths are typically more energy-efficient, the robot would consume more energy traveling long straight paths while covering fewer points. Instead, small circular turns with a radius of 3 meters are preferred, allowing the robot to avoid long straight paths of up to

100 meters. This approach optimizes energy usage by prioritizing the coverage of remaining points over path straightness.

To determine the optimal path for these remaining points, the algorithm employs the Dubins open traveling salesman problem (TSP). DOTSP extends the open TSP by accounting for the non-holonomic constraints of vehicles, such as turning radius and orientation. It seeks to find the shortest path that visits a set of points while respecting the vehicle's kinematic constraints.

Initially, the shortest path between the points is computed using the open TSP. Subsequently, Dubins constraints are applied to ensure the path is feasible for the robot. This method not only ensures complete coverage of the remaining points but also reduces the overall path length and energy consumption. By adopting this approach, the algorithm significantly improves the coverage rate and convergence speed, achieving comprehensive coverage in an efficient manner. The pseudocode for the DOTSP behavior algorithm is shown in (Algorithm 4). An example of the DOTSP path is shown in Figure 3.



(a) TSP path for remaining points.

(b) DOTSP path for remaining points.

Figure 3: Centroids of overlapped regions.

Notations for DOTSP:

- sn: Start node
- G: Graph
- P_{clo}: Closed path
- P_{op}: Open path
- P_{dotsp}: DOTSP path

Algorithm 4 DOTSP

Input: remaining points (p_r), initial robot position(r_{pos}), turning radius (R_{tu})
Output: Dubins path (P_{dotsp})

1: $sn \leftarrow r_{\text{pos}}$
2: $G \leftarrow$ GenerateGraph(p_r)
3: $P_{clo} \leftarrow$ SolveClosedTSP(G, sn)
4: $P_{op} \leftarrow$ RemoveEdgeWithMaxWeightFromStartNode(G, sn)
5: $P_{dotsp} \leftarrow$ DubinsPath(P_{op}, R_{tu})
6: **return** P_{dotsp}

This integrated strategy, comprising the first, second, and final behaviors, enables the robot to cover all points in the agricultural field effectively. Each behavior is tailored to optimize different stages of the coverage process, from focusing on dense central areas to sparsely populated boundaries and finally to the remaining isolated points. This hierarchical and adaptive approach ensures that the robot operates efficiently, minimizing both path length and energy consumption throughout the coverage process.

2.7 Automatic Shift Between Behaviors:

A critical aspect of this behavioral algorithm is the decision-making process for shifting from one behavior to the next. Different data distributions require varying coverage percentages to optimize this transition. Determining the optimal shift point is challenging and necessitates extensive experimentation and testing to achieve the best coverage rate and convergence.

The performance of the algorithm can significantly vary depending on when the shift between behaviors occurs. For the same dataset, different coverage percentages for behavior shifts can greatly influence the coverage rate and convergence efficiency. Therefore, it is essential to tailor the shift points for each specific dataset rather than relying on a fixed percentage.

One of the standout features of this behavioral algorithm is its ability to determine the near-optimal coverage percentage for shifting behaviors. Once the dataset is pre-processed, the algorithm computes the best percentage coverage for behavior shifts as follows:

- **Centroid and Concentric Circles:** The algorithm first computes the centroid of the dataset. Imaginary concentric circles are then centered at the centroid, with varying radii extending outward until all points are encompassed.
- **Point Distribution Analysis:** The percentage of points within each concentric region is calculated. The behavioral shift percentage is determined based on a threshold percentage found through experimentation and testing, which is 50 percent. The points are counted in each region, starting from the innermost circle, and the percentages are accumulated until the threshold is reached.
- **Threshold Determination:** The region where the threshold percentage is exceeded dictates the coverage percentage for the behavior shift. If the threshold is reached in the innermost region, indicating a dense point distribution near the centroid, the first behavior will be more effective, and the shift percentage will be automatically set to a higher percent (e.g., 60-70%). Conversely, if the threshold is reached in the outer regions, suggesting denser distribution near the boundaries, the second behavior becomes more pertinent, and the shift percentage is automatically set to a lower percent (e.g., 30-40%) by the algorithm.

This intellectually behavior shifting enables the algorithm to determine the optimal coverage percentage, thereby enhancing the overall coverage rate and convergence efficiency. By integrating these calculated shift points, the hierarchical behavioral algorithm adapts dynamically to different datasets, ensuring that the robot operates efficiently across various scenarios. The pseudocode for the Auto-Shift behavior algorithm is shown in (Algorithm 5). The concept of concentric circles, centroids and the points distribution considered for the automatic shift between behaviors can be visualized in the (Figure 4).

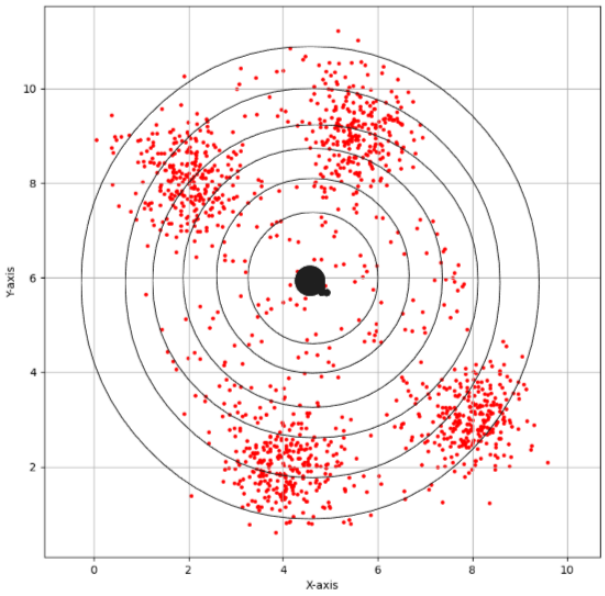


Figure 4: CCP Behavioral Algorithm flowchart.

Notations for AutoBehaviorShift:

- $\delta_t o$: Total percentage
- δ_{th}: Threshold percentage
- δ_{list}: List of percentages
- cc: Concentric circles
- R_{cu}: Current radius
- p_{cu_δ}: Percentage of points in subregion

Algorithm 5 AutoBehaviorShift

Input: *2D points* (P_{2d}), *min_radius* ($R_{\min}$), *max_radius* ($R_{\max}$), number of concurrent circles (N)
Output: *behavior_change_percent* δ_{bc}

$\delta_{to} \leftarrow 0$
$\delta_{th} \leftarrow 50$
$\delta_{list} \leftarrow \text{GeneratePercentageList}(30, 80, N)$ ▷ List from 30 to 80% with N steps.
$cc \leftarrow \text{ConcentricCircles}(P_{2d}, R_{\min}, R_{\max}, N)$
for i, R_{cu} **in** cc **do**
    $p_{cu_\delta} \leftarrow \text{PointsPercentageInSubregion}(R_{cu})$
    $\delta_{to} \leftarrow \delta_{to} + p_{cu_\delta}$
    if $\delta_{to} > \delta_{th}$ **then**
        $\delta_{\text{bc}} \leftarrow \delta_{list}[i]$
        break
    end if
end for
return δ_{bc}

The flowchart of the complete behavioal algorithm can be visualized in the (Figure 5)

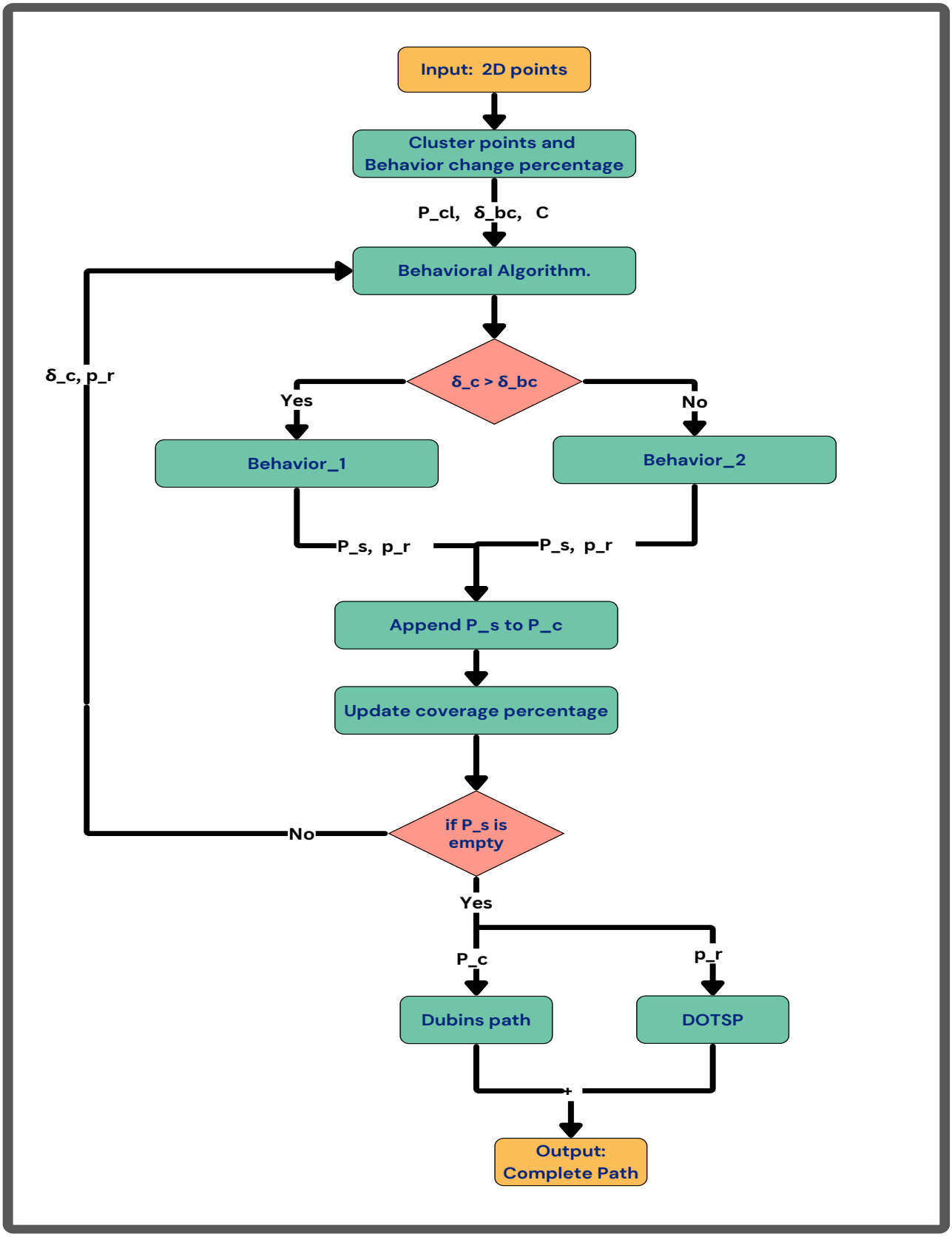


Figure 5: CCP Behavioral Algorithm flowchart.

3 Obstacle Avoidance

Obstacle avoidance is a crucial component of coverage path planning (CPP), applicable in various contexts such as agricultural fields, warehouses, and more. In agricultural fields, static obstacles include trees, rocks, houses, and other structures, collectively referred to as keep-out zones. Obstacles are represented as polygons of various shapes and sizes, which the robot must avoid and all the obstacle information is available beforehand.

One of the critical aspects of the CPP algorithm is its efficiency in obstacle avoidance. Given that the robot will encounter obstacles multiple times during its operation, it is essential that the algorithm minimizes the computation time required for each avoidance maneuver. The algorithm should be optimized to quickly navigate around obstacles. Path adjustments must be computed swiftly to ensure minimal disruption to the overall coverage path.

The obstacle avoidance algorithm in coverage path planning (CPP) consists of two main components:

1. Setup and dynamic grid generation.
2. Path validity checking and obstacle free path generation.

3.1 Setup and Dynamic Grid Generation

The initial setup begins once the algorithm receives all the data. The first crucial step is to account for the robot's physical dimensions by creating a configuration space. This ensures that the algorithm considers the robot's actual operating space, not just a point in the field.

CPP operates in continuous space for path planning, but representing obstacles solely in continuous space is computationally prohibitive, especially in obstacle-rich environments. On the other hand, completely using discrete space is impractical for large fields, such as agricultural areas, because methods like occupancy grids would be too expensive to generate and manage over extensive areas.

To address this, a hybrid approach is adopted. Continuous space is used for general path planning, while obstacles are represented in both continuous and discrete spaces. Discrete space involves creating separate occupancy grids for each obstacle. A critical challenge here is determining the size of the grid cells: they must be small enough to accurately represent the obstacle yet large enough to avoid excessive computational load. The algorithm addresses this by generating dynamic grids based on the size of the obstacle and the robot's non-holonomic constraints.

Dynamic Grid Generation: The initial objective is to dynamically generate an occupancy grid for each obstacle. The algorithm begins by extending the vertices of each polygonal obstacle to create a safe zone around it. This extended obstacle is then approximated as a rectangle using the minimum and maximum coordinates of the extended polygon.

To generate the grid for this approximated obstacle, the algorithm considers the robot's non-holonomic constraints. It calculates a curve using the two extreme angles of the vision cone and a step length, ensuring the robot can navigate this curve with the minimum turning radius until a 90-degree turn is achieved. At this point, the changes in the x and y coordinates (dx and dy) are recorded. These values are used to determine the grid's dimensions, with an allowance of 1.5 times the curve distance along the robot's heading and half the obstacle's width. This setup ensures that the robot can navigate around the obstacle even with its non-holonomic constraints.

The non-holonomic constraints used for grid generation and generated grids based on obstacle dimensions are illustrated in Figure 6. The pseudocode for the dynamic grid generation for each obstacle is provided in (Algorithm 6)

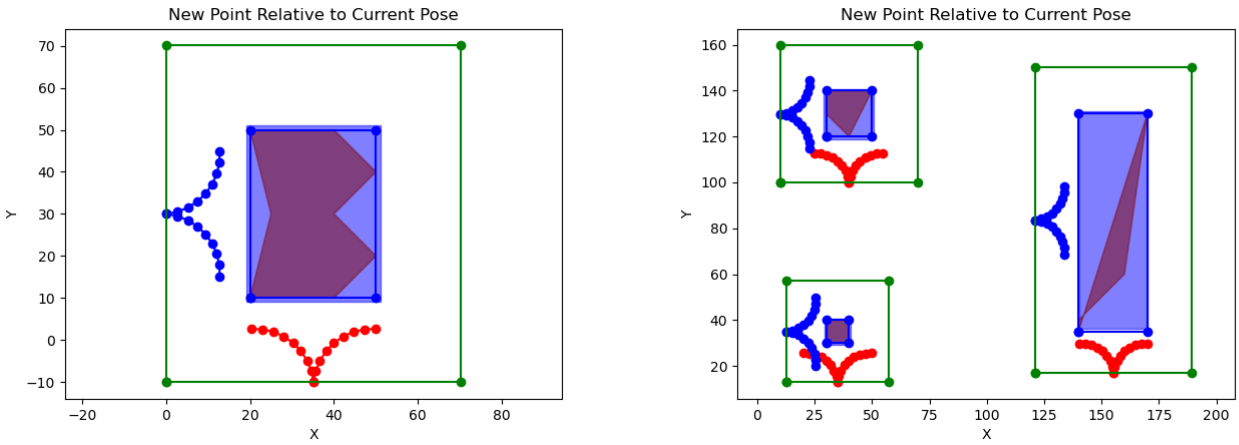


Figure 6: Dynamic Grid Generation.

In the figure, the filled shape represents an obstacle. The red and blue curves depict the robot's minimum turning radius. The blue rectangular region indicates the extended and approximated obstacle area, while the green rectangle represents the generated grid for the obstacle.

Convention to be followed for the Obstacle Avoidance Algorithm:

- OB: Obstacle.
- OB_p: Polygonal obstacles.
- OB_e: Extended obstacles.
- g: grid.
- g_s: grids.
- p_{bu}: Buffer points.
- SP: Salient Point.
- P_{re}: Remaining path.

Notations for the Dynamic Grid Generation Algorithm:
- dx: Distance in the x direction at 90degree turn.
- dy: Distance in the y direction at 90degree turn.
- fd: Free grid space from the obstacle edge.

Algorithm 6 DynamicGridAlgorithm

Input: polygonal obstacles (OB_p), grid diameter (g_d), vision cone (VC), safe margin (sm)
Output: Extended polygonal obstacles (OB_e), grids (g_s), buffer points (p_{bu})

1: $dx, dy \leftarrow$ ComputeDistancesFor90DegreeTurn(VC)
2: $fd \leftarrow 1.5 \times dy$
3: $OB_e \leftarrow$ ExtendPolygonForSafeMargin(OB_p, sm)
4: $g_s \leftarrow$ ComputeGrid(OB_e, g_d, fd)
5: $p_{bu} \leftarrow$ PointsSurroundingObstacles(OB_e)
6: **return** OB_e, g_s, p_{bu}

Once the grid size is determined, the grid cells are generated. The cell size is chosen based on the robot's dimensions: larger robots do not require fine grids as they cannot move cell by cell. Therefore,

an appropriate grid cell size is selected to suit the robot's size. For each obstacle, point data (centers of the complete grid) is generated over the occupancy grid. The algorithm extracts points around the obstacle's boundary, known as buffer points. These buffer points help in navigating around the obstacles efficiently.

3.1.1 Path validity checking and obstacle free path generation

This section outlines the second phase of the obstacle avoidance algorithm, focusing on the real-time aspects of navigation once the initial setup is complete. After setting up the obstacles and generating the dynamic grids, the algorithm proceeds with the regular behavioral approach for path planning. The pseudocode for the complete algorithm with obstacles is shown in (Algorithm 7)

Algorithm 7 CompleteBehavioralObstacles

Input: 2D points (p_{2d}), initial robot pose (r_{pos}), turning radius (R_{tu}), polygonal obstacles (OB_p), grid diameter (g_d), vision cone (VC)
Output: Dubins path P_{cd}

$OB_e, g_s, p_{bu} \leftarrow$ SetupObstacles(OB_p, g_d, VC, sm)
$p_{cl}, \delta_{\text{bc}} \leftarrow$ CentroidsAndAutoBehaviorShift(P_{2d}, R_{min}, R_{max}, N)
$p_r \leftarrow p_{cl}$
$\delta_c \leftarrow 0$
$P_c \leftarrow []$
while True **do**
    if $\delta_c < \delta_{\text{bc}}$ **then**
        $P_s, p_r \leftarrow$ Behavior_1(OB_e, g_s, p_{bu}, p_r, P_{cl}, N_s, r_{pos}, VC, C, R_{conc}, $step$)
    else
        $P_s, p_r \leftarrow$ Behavior_2(OB_e, g_s, p_{bu}, p_r, P_{cl}, N_s, r_{pos}, VC, C, R_{conc}, $step$)
    end if
    $P_c \mathrel{+}= P_s$
    $r_{\text{pos}} \leftarrow P_c[-1]$
    $\delta_c \leftarrow$ UpdateCoveragePerc(P_c, p_r)
    if len(P_s) $== 0$ **then**
        break
    end if
end while
$P_d \leftarrow$ DubinsPath(P_c, R_{tu})
$P_{re} \leftarrow$ PathAroundObstaclesAlgorithm(OB_e,P_r, r_{pos}, R_{tu})
$P_{cd} \leftarrow P_d + P_{re}$
return P_{cd}

The path generation process begins as described previously. However, this time, the algorithm includes a mechanism to detect path collisions with obstacles. If no obstacles are detected along the planned path, the behavioral algorithm operates normally. When an obstacle is detected, the algorithm initiates a sequence of steps to navigate around it. The pseudocode for the behavior 1 and behavior 2 with the inclusion of path validity check is shown in (Algorithm 8) and (Algorithm 9) respectively.

Description of the notations:
- P_{cu}: Current path

- P_{OBfree}: Obstacle free path

Algorithm 8 Behavior_1_with_Obstacles

Input: Extended polygonal obstacles (OB_e), grids (g_s), buffer points (p_{bu}), 2D points (P_{2d}), clustered points (p_{cl}), number of sample orientations (N_s), robot pose (r_{pos}), vision cone (VC), centroid (C), concurrent region radii (R_{conc}), step size (S)
Output: Complete straight path P_{cs}, remaining points p_r

$P_{cs} \leftarrow []$
$P_t \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$p_{co} \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$O_{sa} \leftarrow \text{SampleTheOrientations}(R_{pos}[2], N_s, S)$
for i, O **in** O_{sa} **do**
    $R_{pos}[2] \leftarrow O$
    while no point is visible **do**
        $p_v \leftarrow \text{ComputeVisionConePoints}(r_{pos}, VC)$
        $p_{po} \leftarrow \text{FindPotentialPoint}(r_{pos}, p_v)$
        if p_{po} is None **then**
            break
        end if
        $O_n \leftarrow \text{FromCurrentPoseToPotentialPoint}(r_{pos}, p_{po})$
        $P_{cu} \leftarrow [[r_{pos}, p_{po}]]$
        $P_{OBfree} \leftarrow \text{ComputeObstacleFreePath}(P_{cu}, O_n)$
        $P_t[i] \mathrel{+}= P_{OBfree}$
        $p_{int} \leftarrow \text{CheckIntermediatePoints}(p_r)$
        $P_t[i].\text{append}(p_{int})$
        $p_c[i].\text{append}(\text{len}(P_t))$
        $p_r \leftarrow p_{cl} - P_t$
        $r_{pos} \leftarrow P_{OBfree}[-1]$
    end while
end for
$O_{bi} \leftarrow \text{argmax}(p_c[:])$
$P_{cs} \leftarrow P_t[O_{bi}]$
$r_{pos} \leftarrow P_{cs}[-1]$
$p_{tu} \leftarrow \text{PotentialPointToTurn}(p_r, R_{\text{conc}}, r_{pos})$
$O_b \leftarrow \text{TowardsCentroid}(p_{tu}, C)$
$P_{cu} \leftarrow [[r_{pos}, p_{tu}]]$
$P_{OBfree} \leftarrow \text{ComputeObstacleFreePath}(OB_e, g_s, p_{bu}, P_{cs}, P_{cu}, O_b)$
$P_{cs} \mathrel{+}= P_{OBfree}$
$p_r.\text{remove}([P_{OBfree}])$
return P_{cs}, p_r

Algorithm 9 Behavioral_2_with_Obstacles

Input: Extended polygonal obstacles (OB_e), grids (g_s), buffer points (p_{bu}), 2D points (P_{2d}), clustered points (p_{cl}), number of sample orientations (N_s), robot pose (r_{pos}), vision cone (VC), centroid (C), concurrent region radii (R_{conc}), step size (S)
Output: Complete straight path P_{cs}, remaining points p_r

$P_{cs} \leftarrow []$
$P_t \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$p_{co} \leftarrow [[] \text{ for } \_ \text{ in range}(N_s)]$
$O_{sa} \leftarrow \text{SampleTheOrientations}(R_{pos}[2], N_s, S)$
for i, O **in** O_{sa} **do**
    $R_{pos}[2] \leftarrow O$
    while no point is visible **do**
        $p_v \leftarrow \text{ComputeVisionConePoints}(r_{pos}, VC)$
        $p_{po} \leftarrow \text{FindPotentialPoint}(r_{pos}, p_v)$
        if p_{po} is None **then**
            break
        end if
        $O_n \leftarrow \text{FromCurrentPoseToPotentialPoint}(r_{pos}, p_{po})$
        $P_{cu} \leftarrow [[r_{pos}, p_{po}]]$
        $P_{OBfree} \leftarrow \text{ComputeObstacleFreePath}(P_{cu}, O_n)$
        $P_t[i] \mathrel{+}= P_{OBfree}$
        $p_{int} \leftarrow \text{CheckIntermediatePoints}(p_r)$
        $P_t[i].\text{append}(p_{int})$
        $p_c[i].\text{append}(\text{len}(P_t))$
        $p_r \leftarrow p_{cl} - P_t$
        $r_{pos} \leftarrow P_{OBfree}[-1]$
    end while
end for
$O_{bi} \leftarrow \text{argmax}(p_c[:])$
$P_{cs} \leftarrow P_t[O_{bi}]$
$r_{pos} \leftarrow P_{cs}[-1]$
$p_{tu} \leftarrow \text{PotentialPointToTurnCCW}(p_r, R_{\text{conc}}, r_{pos})$
$O_b \leftarrow \text{VectorsTowardsNextCCWPoint}(p_{tu}, p_r)$
$P_{cu} \leftarrow [[r_{pos}, p_{tu}]]$
$P_{OBfree} \leftarrow \text{ComputeObstacleFreePath}(OB_e, g_s, p_{bu}, P_{cs}, P_{cu}, O_b)$
$P_{cs} \mathrel{+}= P_{OBfree}$
$p_r.\text{remove}([P_{OBfree}])$
return P_{cs}, p_r

3.2 Collision Detection and Salient Point Identification

Upon detecting an obstacle in the path, the algorithm first identifies a line along the robot's heading and checks for intersection points with the obstacle. At this stage, buffer points previously defined around the obstacle become critical. The algorithm identifies the furthest buffer point along the robot's heading, which marks the end of the line. It then calculates the perpendicular distances from this line to all buffer points, selecting two points on either side of the line at the maximum distance. These points are designated as salient points, serving as key navigational targets to avoid the obstacle.

Once the salient points are determined, they become the goal points for the robot to bypass the obstacle. The robot's current position is the starting point for this avoidance maneuver. The algorithm then checks whether the robot is inside the grid. If the robot is outside the grid, the salient points are directly used as goal points, and the robot navigates towards them following its non-holonomic constraints. The grid is designed such that if the robot is at the edge, it can avoid the obstacle by reaching these extreme points directly.

The Figure 7 illustrates the selection of salient points based on the robot's orientation when the path intersects with an obstacle. In the figure, dark blue points denote grid centers, red and black points mark grid centers within the obstacle, and green points indicate buffer points. The yellow point signifies the extreme end of the robot's orientation, while the purple direction represents the robot's orientation. Light blue points indicate the region of interest, and the two dark purple points at the extreme corners of the obstacle represent the salient points.

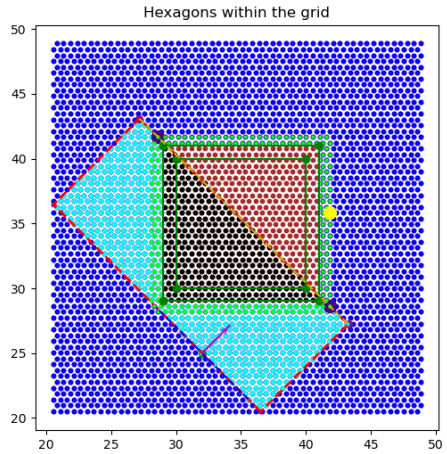


Figure 7: Selection of Salient Points.

3.3 Graph-Based Path Finding

If the robot is inside the grid, a more complex process ensures that the robot can reach the salient points while adhering to its motion constraints. A graph-based approach is utilized to find the shortest and feasible path from the robot's current position to the salient points using the grid cells.

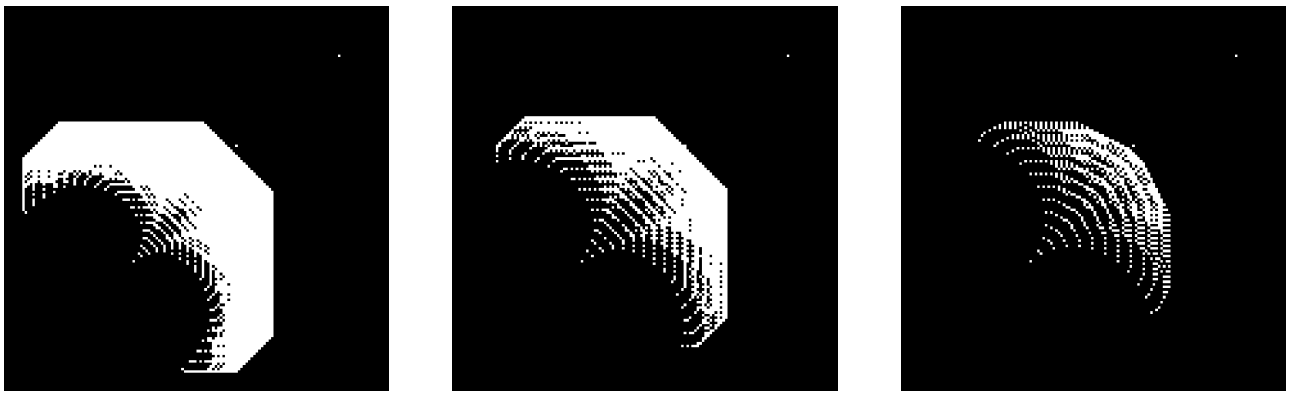
Efficient and rapid graph generation is crucial, as this process must occur each time an obstacle is encountered. The algorithm employs two extreme angles of the vision cone and a step length to create the graph. Each iteration produces a new generation of grid cells. For instance, if the robot occupies one cell, the next generation, based on the two extreme angles and step length, will occupy two cells. Subsequent generations expand similarly, covering more cells until the goal point is included in the graph.

One challenge in this graph-based approach is balancing the step length. If the step length is too short, the graph requires many generations to reach the goal point, increasing computational time. Conversely, if the step length is too long, the graph may become sparse, risking the possibility of not adequately covering the goal point and potentially passing through it.

To ensure an efficient and adaptive approach to obstacle avoidance, the algorithm dynamically determines the step length for graph generation. By allowing the algorithm to decide the step length, it

can find a near-optimal length that generates a sparse graph initially, gradually becoming denser as it approaches the goal point. This adaptive strategy optimizes computational efficiency while ensuring thorough coverage.

Automating the selection of step length involved an experimental analysis to understand its dependence on various parameters. Notably, the distance to the salient point emerged as a significant factor. By conducting experiments with different salient point positions and step lengths, a linear relationship between the distance and step length was identified. Consequently, the algorithm fits a line to this data at the outset, enabling it to automatically determine the dynamic step length based on the distance to the salient point. This near-optimal step length encourages the algorithm to generate a sparse graph initially, gradually densifying it as it approaches the goal point. This approach significantly enhances computational efficiency. The effect of step length on graph generation is illustrated in Figure 8.



(a) Step length 4.0 cm at Generation 20.

(b) Step length 4.96 cm at Generation 10.

(c) Step length 6.0 cm at Generation 8.

Figure 8: The Effect of Step Length on Graph Generation.

Once the graph is generated, multiple paths from the robot's position to the salient point are available. The algorithm employs an A* search over the graph to find the shortest path. The intermediate points along this path serve as the route to avoid the obstacle and reach the goal point efficiently and swiftly. Subsequently, the regular behavior resumes for further coverage of the designated points.

An example of graph generation and pathfinding is illustrated in Figure 9. In Figure 9a, the scenario with salient points is depicted. The while square represents the obstacle, two corner white points represent the salient points and the lower left white point denotes the robot oriented at a 45-degree angle. In Figure 9b, the graph generation is shown. Initially, the graph is sparse and as it approaches the goal point, it becomes denser due to automatic step length selection. In Figure 9c, the shortest path from the robot to the goal point is found using A* search over the generated graph.

(a) Scenario with salient points. (b) Graph Generation. (c) A* Search.

Figure 9: Graph Generation and Search.

If the generated graph does not include the goal point, indicating that the goal point is unreachable within the constraints, the algorithm retraces one step back in the path. It then repeats the process from the point where the obstacle was detected, ensuring that the robot can circumvent the obstacle and complete its coverage path planning seamlessly.

The pseudocode for path validity check and finding obstacle free path is shown in (Algorithm 10) . Description of the notations:

- obs_idx: obstacle index.
- S_l: Graph step length.
- gen_{max}: Maximum generation for graph.
- P_{sh}: Shortest path.

Algorithm 10 ComputeObstacleFreePath

Input: Extended obstacles (OB_e), grids (g_s), buffer points (p_{bu}), complete path (P_c), current path (P_{cu}), current orientation (O_{cu})
Output: Obstacle-free path (P_{OBfree})

1: $r_{pos} \leftarrow P_{cu}[-1]$
2: $is\_path\_in\_obstacle, obs\_idx \leftarrow$ CheckPath(P_{cu}, OB_e)
3: **if** $is\_path\_in\_obstacle$ **then**
4: $OB \leftarrow OB_e[obs\_idx]$
5: $g \leftarrow g_s[obs\_idx]$
6: $SP \leftarrow$ ExtractSalientPoints(OB, g, p_{bu}, r_{pos})
7: G.initialize()
8: $S_l \leftarrow$ AutoSelectStepLength(r_{pos}, SP)
9: G, $goal\_SP$, $is\_goal\_found \leftarrow$ GenerateGraph(G,r_{pos},SP,S_l,gen_{max})
10: **if** is_goal_found **then**
11: $P_{sh} \leftarrow$ AStarSearch(G,r_{pos}, $goal\_SP$)
12: $P_{OBfree} \leftarrow P_{sh}$
13: **return** P_{OBfree}
14: **else**
15: $r_{pos} \leftarrow P_c[-1]$ ▷ Move one step back in the path
16: **goto** GenerateGraph (step 9)
17: **end if**
18: **else**
19: **return** $[[P_{cu}[-1], O_{cu}]]$
20: **end if**

This iterative process repeats each time an obstacle is encountered, enabling the robot to efficiently navigate around obstacles and reach its designated goal points. This comprehensive approach ensures robust obstacle avoidance within the coverage path planning algorithm, facilitating efficient and timely completion of tasks.

One last change in the complete coverage with obstacles as compared to the regular coverage is replacing the DOTSP algorithm with another algorithm that can compute path for the remaining points left near the obstacles. Since, after the behavior 2, in this case there will be only some points left that will be close to the obstacles. This happens if the points are quite close to the obstacles. Path through this points cannot be computed through DOTSP algorithm as it collides with the obstacle and path validity is not available with DOTSP. Hence, a new algorithm is introduced to compute path through these points that are close to the obstacles.

This algorithm first associate each point to its nearest obstacle. Then, we will end up with each obstacles assciated with some points. Then, the algorithm first generates the visibility graph for each obstacle with points. Then, it will find the shortest path covering all the points of first obstacle. Then, it will move to the next obstacle and cover all the points of that obstacle efficiently and so on. Eventually, it will cover all the remaining points close to all the obstacles. However, When dubins constraints are applied to this straight obstacle free path, some of the crves intersects with the corners of the obstacles due to looping of the path caused by the closeness of the points. This can be visualized in the **??**.

The pseudocode for this path finding for remaining points is shown in (Algorithm 11).
Notations:

- P_{list}: Points list
- P_{cu}: Current points
- P_{cov}: path through all points close to one obstacle.

Algorithm 11 PathAroundObstaclesAlgorithm

Input: Extended obstacles (OB_e), 2D points (P_{2d}), robot pose (r_{pos}), turning radius (R_{tu})
Output: Path (P)

$p_{list}, robot\_obs\_idx \leftarrow$ PointsCloseToObstacle(OB_e, P_{2d})
$p_{cu} \leftarrow p_{list}[robot\_obs\_idx]$ ▷ Robot close to an obstacle will be the first.
$OB_{cu} \leftarrow OB_e[robot\_obs\_idx]$
$P \leftarrow []$
for i **in** range(len(OB_e)) **do**
    G.initialize()
    $G \leftarrow$ GenerateVisibilityGraph(OB_{cu}, p_{cu}, r_{pos})
    $P_{cov} \leftarrow$ CoverageGraphSearch(G, r_{pos})
    $P \mathrel{+}= P_{cov}$
    $r_{pos} \leftarrow P_{cov}[-1]$
    $OB_{cu}, obs\_idx \leftarrow$ FindNearestObstacle(r_{pos}, p_{list})
    $p_{cu} \leftarrow p_{list}[obs\_idx]$
end for
return P

The obstacle avoidance approach in coverage path planning addresses several critical challenges with innovative solutions. These include dynamic grid selection to balance computational complexity and accuracy, employing a hybrid space approach to represent obstacles, and optimizing computational time through dynamic step length determination. The algorithm dynamically adjusts step length based on the distance to the salient point, ensuring near-optimal path planning efficiency. Additionally, utilizing both continuous and discrete spaces allows for efficient representation of obstacles. Moreover, employing an A* search algorithm facilitates the determination of the optimal shortest path, enabling swift and effective obstacle avoidance. These integrated strategies ensure robust obstacle avoidance within the coverage path planning algorithm, enhancing overall efficiency and effectiveness.

The flowchart of the complete behavioal algorithm with obstacles can be visualized in the (Figure 10)

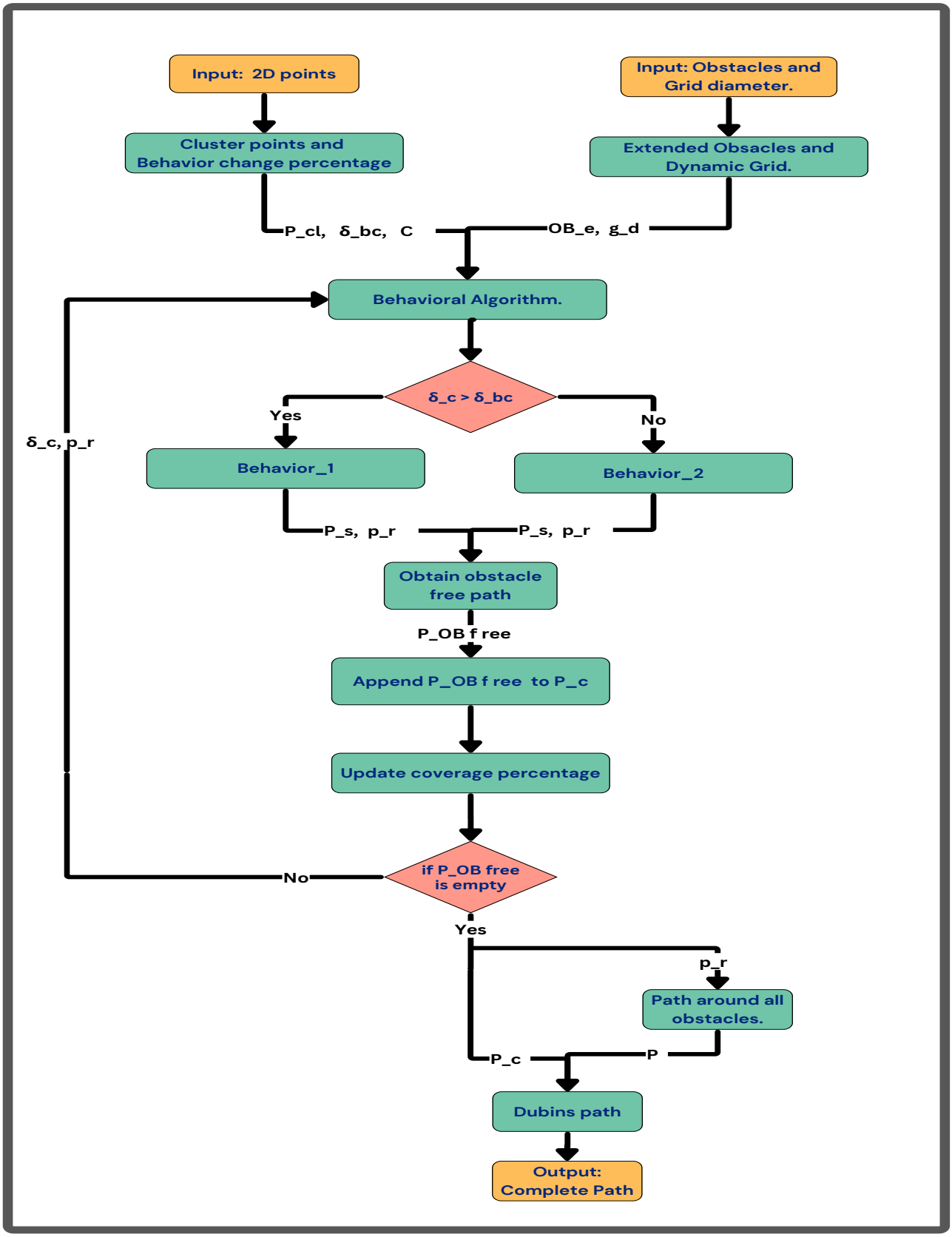


Figure 10: CCP Behavioral Algorithm with Obstacles flowchart.

Simulation Test and Analysis

This section presents a thorough exploration of the proposed complete coverage path planning algorithm tailored for agricultural field applications. It unfolds into two distinct sections:

Simulation Setup:
This section is dedicated to providing a detailed exposition of the simulation setup. Meticulous attention is directed towards delineating the experimental framework within which the simulation tests are conducted.

Simulation Results and Analysis:
Comprehensive empirical findings obtained from the meticulously designed simulation experiments within the aforementioned framework are detailed in this section. Key performance metrics are analyzed to provide the assessment of the algorithm's performance under diverse scenarios. we conduct a thorough analysis of the empirical results, meticulously scrutinizing each detail. Through this examination, we aim to discern the operational dynamics of the algorithm, identify its strengths, and pinpoint areas for potential refinement.

1 Simulation Setup

The simulation setup is meticulously designed to emulate the real-world operational environment of an agricultural field, ensuring that the algorithm's performance can be rigorously tested under realistic conditions. The global coverage path planning is initially performed using Matplotlib, followed by testing in a simulation environment leveraging the Robot Operating System (ROS) and the Gazebo simulator.

In the context of coverage path planning for agricultural fields, it is crucial to account for varying plant distributions. Therefore, different datasets are used to represent diverse scenarios, with the aim of assessing the algorithm's robustness under varying initial conditions and plant distributions. Specifically, the simulation setup incorporates different datasets with the same initial position and orientation of the robot to demonstrate the algorithm's adaptability.

Dataset Description:

To comprehensively evaluate the algorithm, six distinct datasets are employed within the flat field of 120m x 120m area, with each dataset containing different numbers of points: 250, 500, 2000, 4000, 6000, and 10,000. Each primary dataset is further subdivided into four sub-datasets, characterized by different spatial distributions:

- Random Distribution: Points are distributed randomly throughout the field.
- Clustered Distribution (4 Clusters): 20% of the points are distributed randomly, while 80% are distributed in four clusters with distinct Gaussian distribution variances.
- Clustered Distribution (6 Clusters): 20% of the points are distributed randomly, with the remaining 80% distributed in six clusters, each with distinct means and variances.
- Clustered Distribution (10 Clusters): 20% of the points are randomly distributed, and 80% are in ten clusters, each characterized by unique Gaussian distribution parameters.

This results in a total of 24 different datasets, meticulously crafted to test the algorithm's robustness across various scenarios, including variations in the number of points, their distribution patterns, the number of clusters, and the statistical properties of these clusters. Such comprehensive testing ensures that the algorithm is robust and adaptable to real-world agricultural environments, where plant distributions can significantly vary.

Performance Metrics The performance of the algorithm is evaluated using several critical metrics across all datasets:

- Computational Time: The time required for the algorithm to compute the coverage path.
- Field Operation Time: The actual time taken for the robot to complete the coverage task.
- Total Path Length: The total distance traveled by the robot.
- Number of Turns: The total number of turns the robot makes during its operation.
- Energy Consumption: The energy used by the robot to complete the coverage task.
- Coverage Efficiency: The percentage of points covered by the robot within the operational time.

These metrics provide a comprehensive assessment of the algorithm's efficiency, effectiveness, and overall performance under different test scenarios.

Processor Specifications:
Processor: AMD® Ryzen 5 Pro 5675U with Radeon Graphics x 12.
Processing Speed: 2.3 GHz, Number of Cores: 6, and RAM: 16 GB.

Visual Representation of the Dataset
The distribution of points across the different datasets can be visualized in (Figure 1), illustrating the varying scenarios used to test the algorithm's robustness.

2 Simulation Results and Analysis

This section presents the detailed results and analysis of the simulation experiments conducted to evaluate the robustness of the proposed complete coverage path planning behavioral algorithm. The empirical findings are meticulously analyzed to provide insights into the algorithm's performance under diverse scenarios, enabling a comprehensive assessment of its efficiency and effectiveness.

Experimental Consistency To ensure the validity of comparisons and isolate the effects of different datasets on the algorithm's performance, the initial position and orientation of the robot were kept constant throughout all experiments. Additionally, the parameters for both the robot and the algorithm were consistently maintained across all trials.

Robot Constraints and Algorithm Parameters:
The robot's constraints and operational parameters are set to realistic values to ensure the validity of the simulation results. The parameters' data can be visualized in (Table 1), offering a comprehensive overview of the robot's physical and operational characteristics.

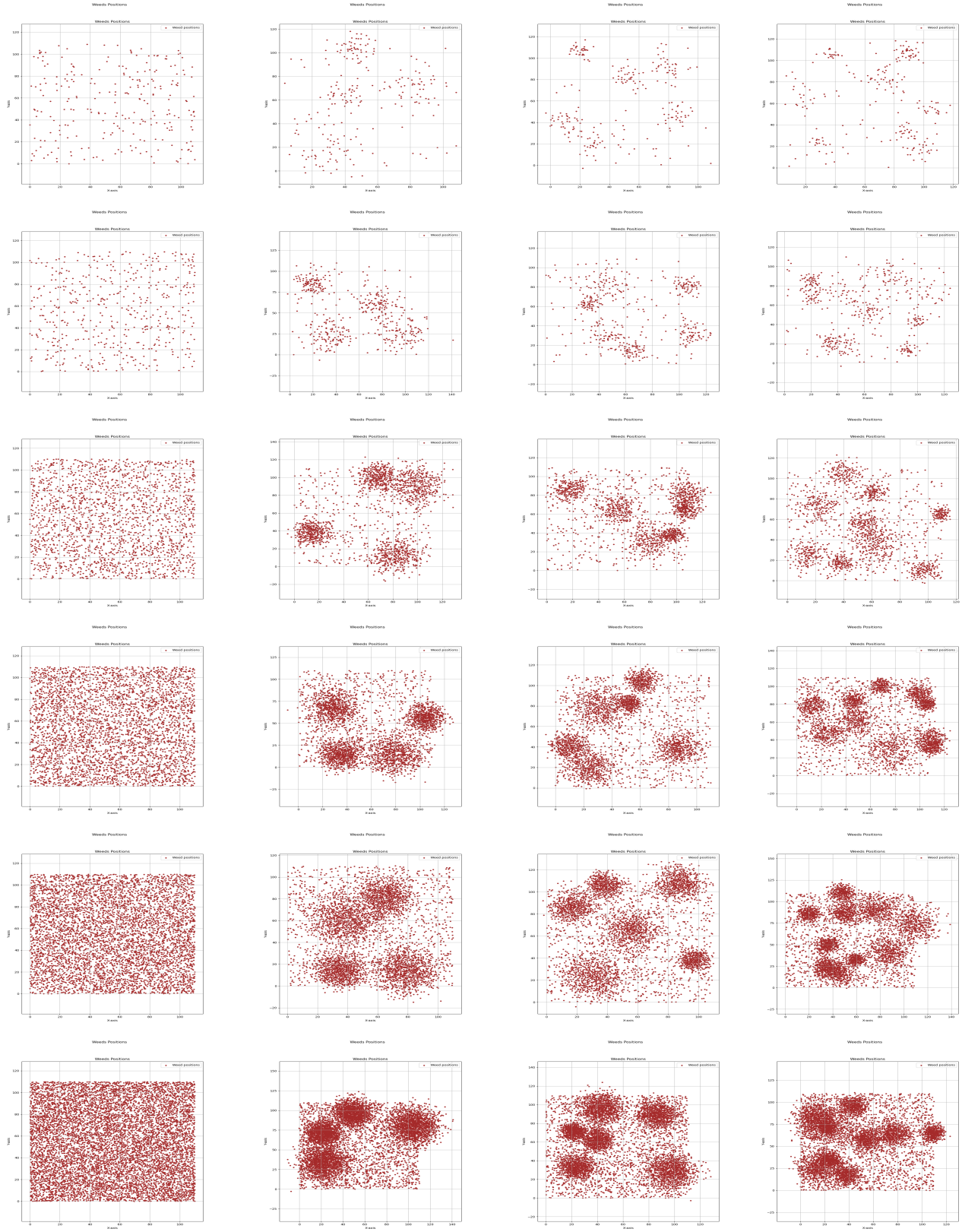


Figure 1: Dataset.

Table 1: Robot and algorithm Constraints and Parameters

Constraints and Parameters	***Values***
Robot initial position	[60.0, 1.0]
Robot initial orientation	120
Search angle of the vision cone	±11
Minimum search radius of the vision cone.	1.25
Maximum search radius of the vision cone.	100
Minimum turning radius allowed for the robot.	2.0
robot velocity on curved paths.	0.4
robot velocity on straight paths.	0.8
Distance used to compute field time taken on curved paths.	7.85
Number of points to consider in the vision cone.	5
Minimum distance between intermediate points, and start and end point.	2.7
Minimum distance between intermediate points.	2
Maximum distance of the point form the path.	0.6

The algorithm parameters are also meticulously defined to ensure consistent performance across all experiments. These parameters are set based on the robot's physical constraints and operational requirements, ensuring that the algorithm operates within realistic boundaries.

The algorithm was designed to initially compute the global coverage path using straight lines. Subsequently, Dubins paths are generated for each line segment to ensure the shortest feasible path between each pair of points, optimizing the overall path length. The (Figure 2) illustrate the straight paths generated by the algorithm for different datasets.

Once the Dubins paths are generated from the straight paths, the resultant paths can be visualized in the (Figure 3). These figures showcase the robot's traversal across different datasets, illustrating how the algorithm adapts to various point distributions.

The performance metrics for all datasets are summarized in the (Table 2). These metrics provide insights into the algorithm's computational efficiency, operational effectiveness, and overall robustness.

The coverage rate plot over the field time offers a visual representation of the algorithm's performance across the whole datasets. This plot demonstrates how effectively the algorithm achieves coverage over time, allowing for a comparative analysis of its efficiency under varying point distributions and densities. This plot can be visualized in (Figure 4).

The performance metrics plots provide a visual representation for better understanding of the algorithm's performance across different datasets. Therefore, separate plots for each performance metric are generated to analyze the algorithm's performance under varying scenarios. These plots can be visualized in the (Figure 5), (Figure 6), (Figure 7), and (Figure 8).

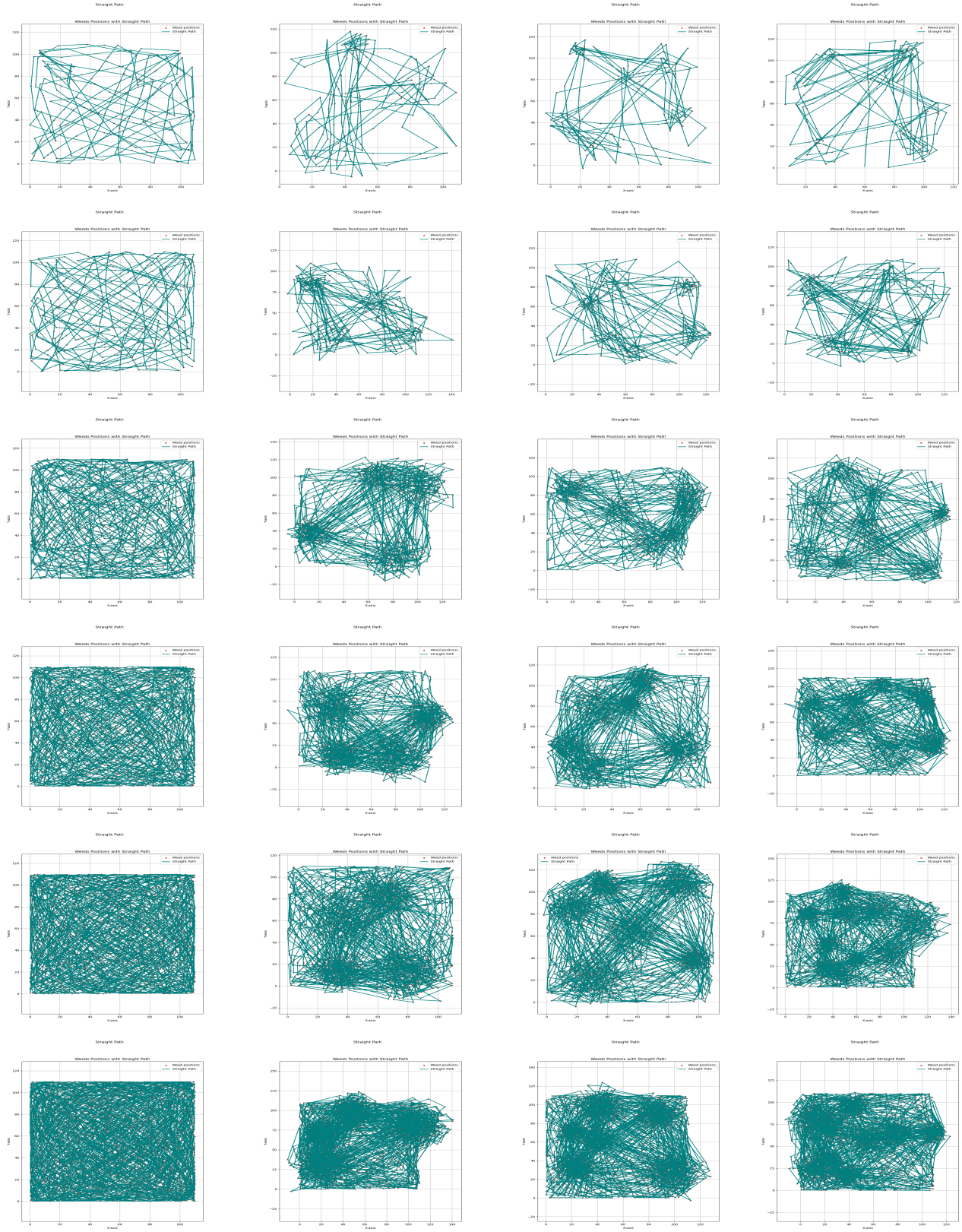


Figure 2: Straight Path.

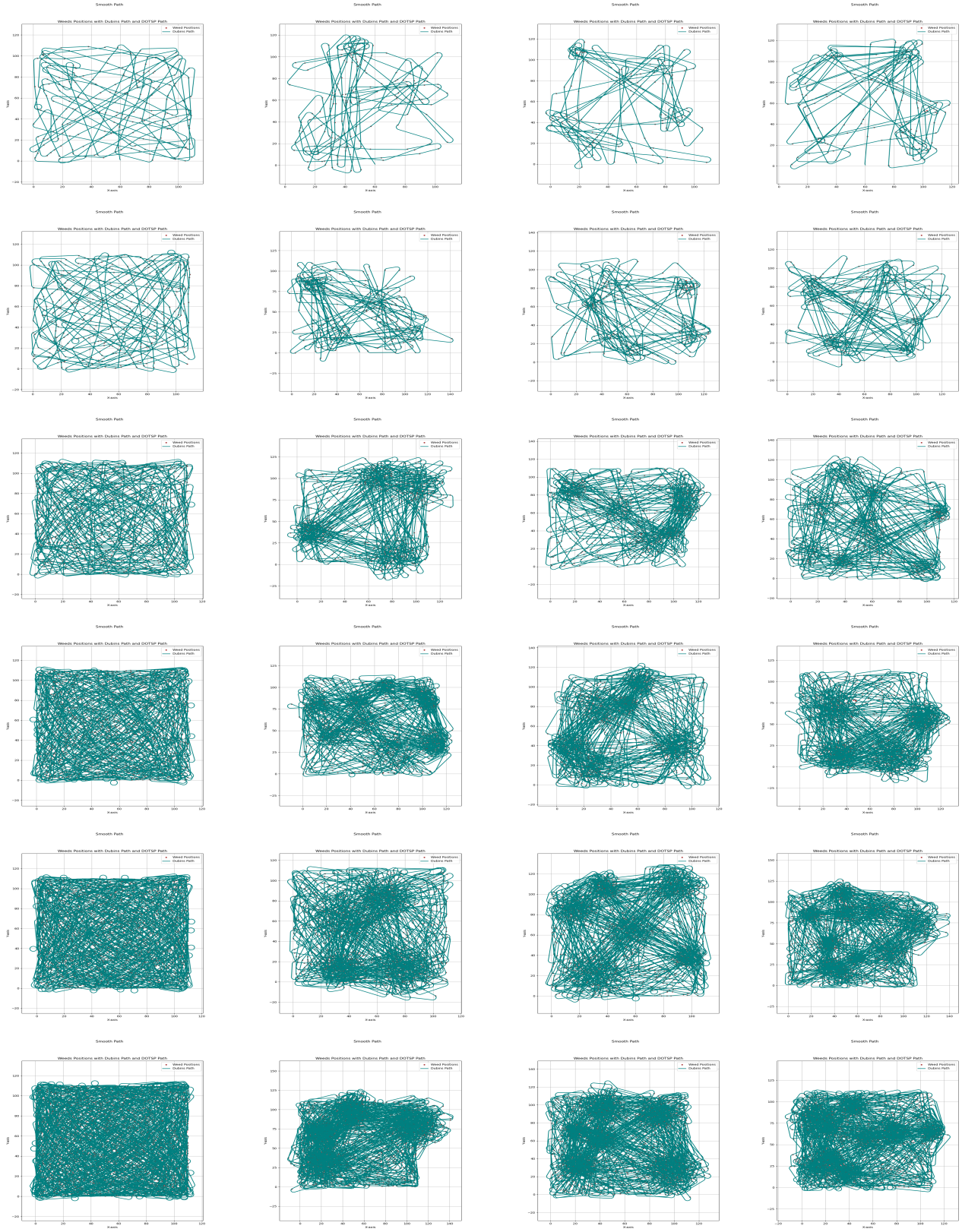


Figure 3: Dubin's Path.

Table 2: Results for the performance metrics.

No. of points	*Cases*	*Computation Time in secs.*	*Field Operation Time in sec.*	*Route length in meters*	*No. of Turns.*	*Energy Consumption in KWh.*
250	case 1	0.858553886	6282.400938	4652.945	48	5783.34459
	case 2	0.775789261	5125.168209	3769.005	41	4734.555207
	case 3	0.788151503	5413.822753	3985.978	44	5022.177564
	case 4	0.825942278	6216.01818	4620.695	44	5656.894864
500	case 1	1.842700005	9520.140257	7027.363	75	8793.612844
	case 2	1.696401358	9151.211911	6788.619	67	8366.468569
	case 3	1.940592766	8589.717644	6336.544	67	7914.394434
	case 4	1.609200478	9761.624722	7250.2	70	8898.700417
2000	case 1	9.04812026	21751.64778	16189.858	154	19816.55759
	case 2	8.591621161	22226.37676	16650.701	144	20041.90077
	case 3	7.65038085	18446.73883	13728.721	131	16813.77138
	case 4	8.155776024	20839.67217	15547.469	139	18820.91894
4000	case 1	24.18394971	32301.69933	24191.259	210	29136.75914
	case 2	20.85941482	29871.08874	22356.011	198	27018.91147
	case 3	18.23963118	27659.4681	20620.054	192	25141.65448
	case 4	19.22349644	29701.58872	22222.31	200	26932.31049
6000	case 1	45.23673987	39851.80224	29777.382	262	35947.48179
	case 2	33.23154497	34821.16749	26039.854	236	31597.65399
	case 3	30.80121374	35035.83592	26223.169	230	31639.66906
	case 4	32.34737062	35496.36509	26570.832	226	31893.13207
10000	case 1	122.6808474	50681.89334	38128.675	308	45382.07468
	case 2	74.37674475	45367.87779	34118.252	277	40641.60223
	case 3	65.54510021	42291.24514	31650.696	278	38197.59611
	case 4	62.76576424	42801.16425	32154.031	274	38606.73138

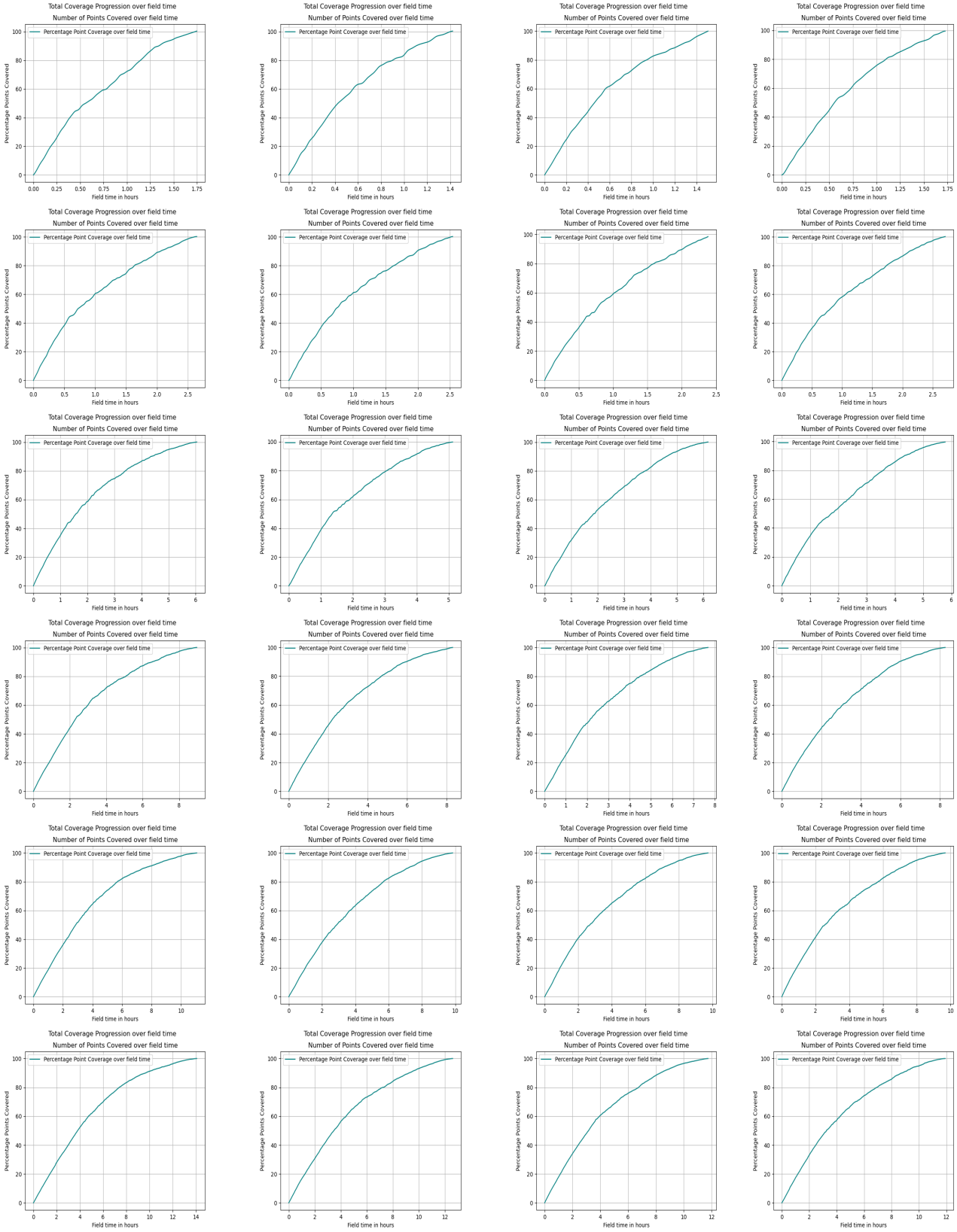


Figure 4: Coverage plots.

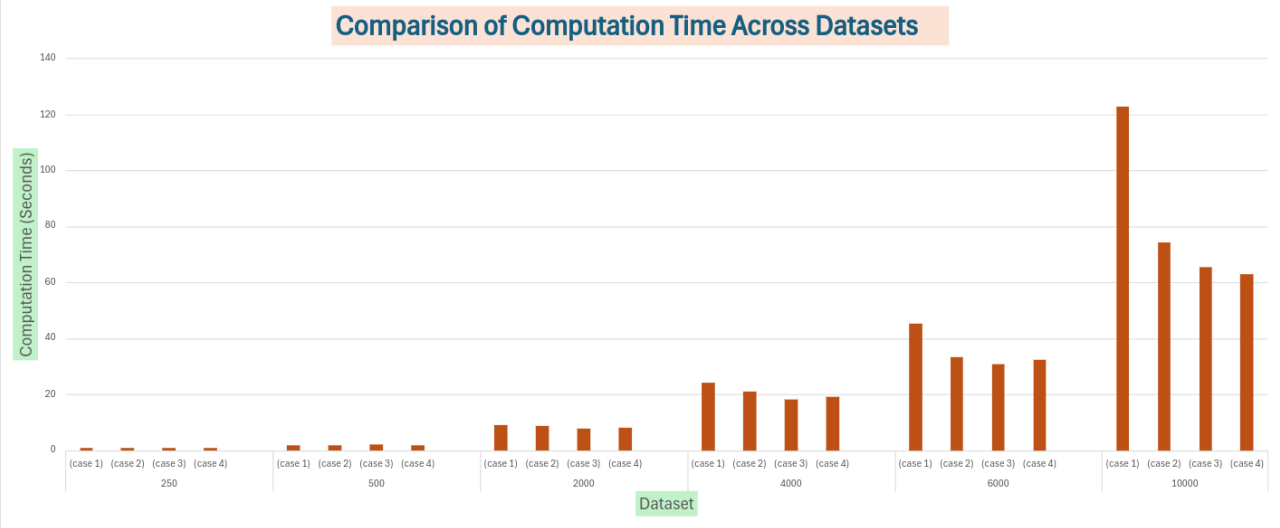


Figure 5: Computation Time.

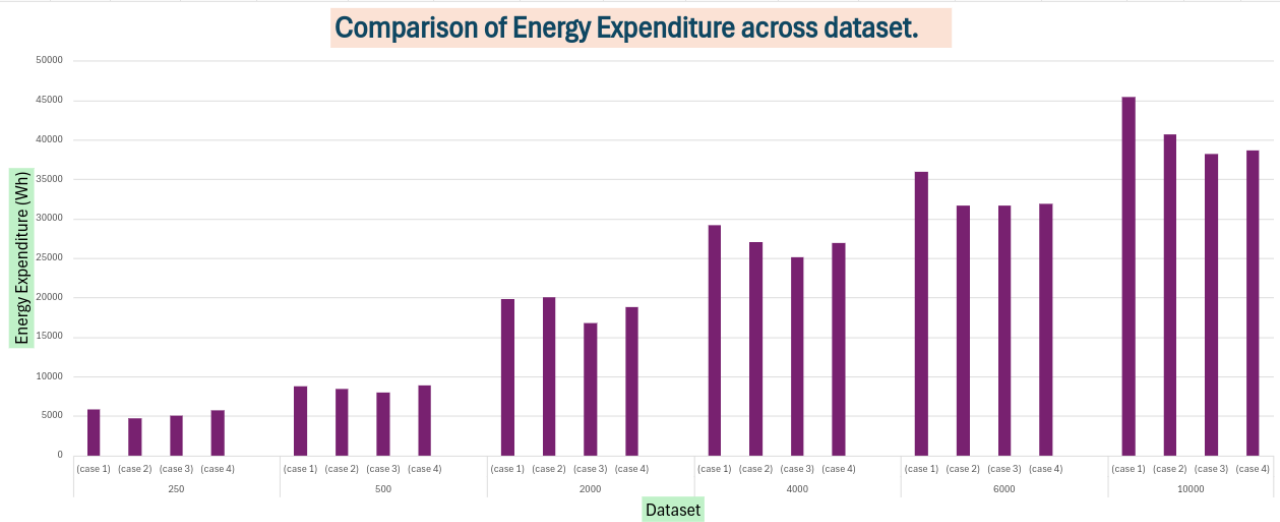


Figure 6: Energy Expenditure.

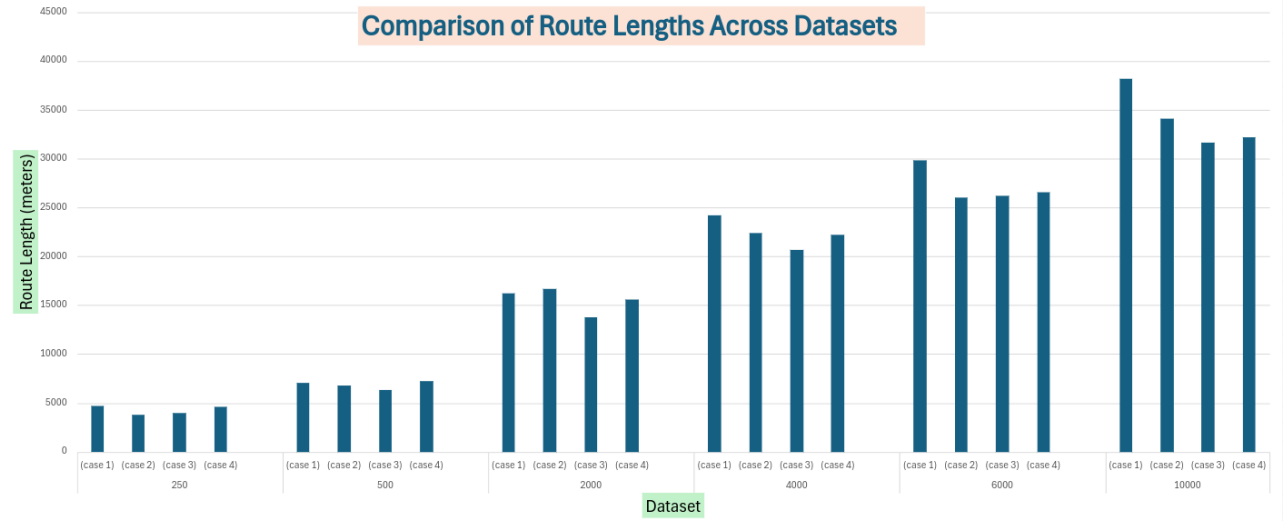


Figure 7: Route Length.

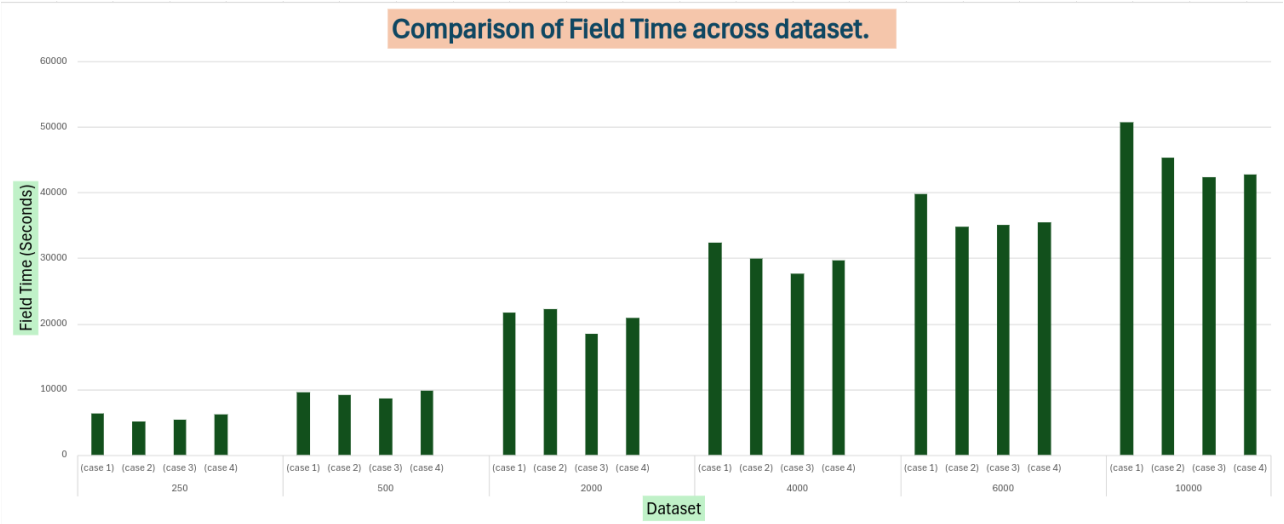


Figure 8: Field Operation Time.

The proposed algorithm operates by initially generating straight paths and subsequently applying Dubins paths to ensure feasibility given the robot's constraints. The straight paths, as visualized in the initial results, are approximately linear, aiming to cover the maximum number of points while maintaining minimal path length. This demonstrates the algorithm's ability to generate efficient coverage paths.

The Dubins path results further validate the algorithm's capability to model the robot's constraints effectively. An inability to model these constraints accurately would result in numerous infeasible circular paths. The primary objective of employing this behavioral approach is to sustain a high coverage rate from start to finish across various data distributions. The results indicate that the algorithm successfully maintains a high coverage rate irrespective of the distribution type. It is noteworthy that the coverage rate is initially very high and gradually decreases, attributed to the diminishing number of points left to cover after a significant percentage of coverage. Nonetheless, the coverage rate remains high, indicating efficient coverage by the algorithm. The percentage coverage

also suggests that the point at which the algorithm changes behavior is near-optimal; otherwise, a gradual increase in coverage rate would be observed if the change occurred at a different percentage.

The computation time plot across all datasets indicates an expected increase in computation time with the number of points. However, the rise in computation time is notable. For instance, even in the largest case of 10,000 points with random distribution, the computation time is 122.68 seconds, which is remarkably low as compared to current state-of-the-art approaches following the non-holonoomic constraints. This efficiency is further highlighted by the average computation time of 60 seconds for other scenarios, demonstrating the algorithm's computational efficiency. This is particularly significant as no real agricultural field would have a purely random distribution, making the algorithm's performance even more impressive in practical scenarios.

Similarly, the route length increases with the number of points as the robot needs to traverse more straight lines to cover all points. It is noteworthy that the maximum route length for each dataset occurs with random distribution and decreases significantly with fewer clusters, increasing again as the number of clusters rises.

Field operation time and energy consumption are derived from the route length, factoring in the robot's maximum allowable velocity on straight and curved paths. The robot can move at twice the speed on straight paths compared to curved paths without damaging crops. The parameters used for the current robot, as outlined in the table, provide a close approximation of real values, although the exact curved distance is challenging to estimate with this pipeline. Energy consumption is approximated to be four times higher on curved paths than on straight paths. The plots above reflect these estimations.

Simulation Test and Analysis with obstacles

This section presents a comprehensive examination of the proposed complete coverage path planning algorithm, tailored to handle static polygonal obstacles in agricultural field applications. The section is divided into two key parts: Simulation Setup and Results & Analysis.

1 Simulation Setup

This subsection provides a detailed description of the simulation setup, focusing on the experimental framework within which the tests are conducted.

The simulation environment is meticulously crafted to replicate real-world agricultural fields, incorporating static polygonal obstacles to simulate common structures such as trees, rocks, or buildings that serve as keep-out zones. This realistic setup ensures that the algorithm's performance is rigorously evaluated under practical conditions.

This simulation environment is setup utilizing a realistic field data obtained from the field manually and inclusion of the obstacles within the field. While the point distribution remains unchanged, the obstacles' shapes, sizes, and numbers vary across different scenarios. Strategic placement of obstacles tests the algorithm's adaptability and robustness. Five distinct datasets are used to assess the algorithm's performance under varying obstacle configurations, increasing in complexity with each set.

Dataset Description
To thoroughly evaluate the algorithm, five distinct datasets are employed within a flat 120m x 120m field, each containing different numbers and configurations of obstacles:

- Set1: Contains one rectangular obstacle of size 10m x 10m.
- Set2: Contains two rectangular obstacles of sizes 10m x 10m and 4m x 4m.
- Set3: Contains three convex polygonal obstacles of distinct sizes and shapes.
- Set4: Contains six convex polygonal obstacles of different sizes and shapes.
- Set5: Contains six polygonal obstacles of varying sizes and shapes, including concave and extremely long obstacles.

This hierarchical arrangement of datasets, with increasing complexity, ensures comprehensive testing of the algorithm's robustness and adaptability to real-world agricultural environments, where obstacles can vary significantly.

Performance Metrics The algorithm's performance is evaluated using several critical metrics across all datasets:

- Computational Time: The time required for the algorithm to compute the coverage path.
- Field Operation Time: The actual time taken for the robot to complete the coverage task.
- Total Path Length: The total distance traveled by the robot.

- Number of Turns: The total number of turns the robot makes during its operation.
- Energy Consumption: The energy used by the robot to complete the coverage task.
- Coverage Efficiency: The percentage of points covered by the robot within the operational time.

These metrics provide a comprehensive assessment of the algorithm's efficiency, effectiveness, and overall performance under different test scenarios.

Visual Representation of the Dataset

The (Figure 1) illustrates the five different datasets used in the simulation experiments, showcasing the shapes, sizes, numbers, and locations of obstacles within the field. This visual representation aids in understanding the varying complexities introduced by the obstacles.

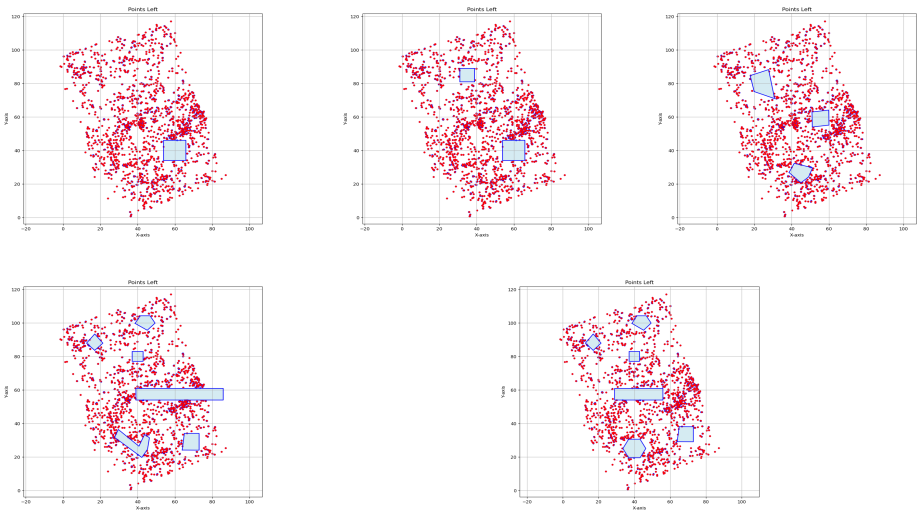


Figure 1: Obstacles Dataset.

2 Simulation Results and Analysis

This section details the results and analysis of simulation experiments designed to evaluate the robustness of the proposed complete coverage path planning algorithm in the presence of static polygonal obstacles. The empirical findings are meticulously analyzed to provide insights into the algorithm's performance across diverse scenarios, offering a comprehensive assessment of its efficiency and effectiveness.

Experimental Consistency To ensure the validity of comparisons and isolate the effects of different datasets on the algorithm's performance, the initial position and orientation of the robot were kept constant throughout all experiments. Additionally, the parameters for both the robot and the algorithm were consistently maintained across all trials.

Robot Constraints and Algorithm Parameters:

The robot's constraints and operational parameters are set to realistic values to ensure the validity of the simulation results. These data can be visualized in (Table 1).

The algorithm parameters are also meticulously defined to ensure consistent performance across all experiments. These parameters are set based on the robot's physical constraints and operational requirements, ensuring that the algorithm operates within realistic boundaries.

Table 1: Constraints and Parameters in presence of obstacles.

Constraints and Parameters	***Values***
Robot initial position	[60.0, 1.0]
Robot initial orientation	45
Search angle of the vision cone	±11
Minimum search radius of the vision cone.	1.25
Maximum search radius of the vision cone.	100
Minimum turning radius allowed for the robot.	2.0
robot velocity on curved paths.	0.4
robot velocity on straight paths.	0.8
Distance used to compute field time taken on curved paths.	7.85
Number of points to consider in the vision cone.	5
Minimum distance between intermediate points, and start and end point.	2.7
Minimum distance between intermediate points.	2
Maximum distance of the intermediate point form the path.	0.6
Graph search extreme angles	±11
buffer distance around obstacles.	1
Extend polygon distance for safe region.	1.4
Graph generation step reduction iteration.	3
Grid diameter.	0.4
Max generation for graph generation.	18
Minimum step length for graph generation.	1.2
Total operational area.	[[-2,0], [90,120]]
Distance between waypoints.	10

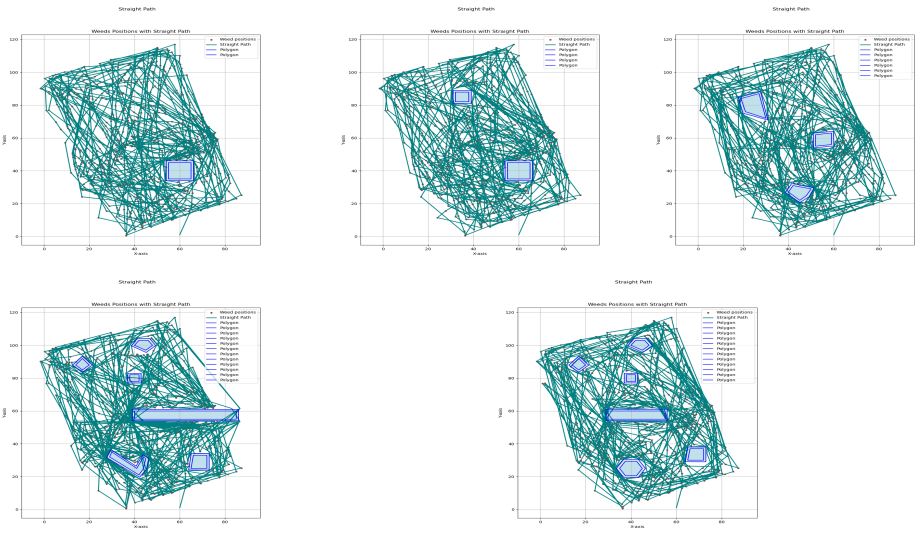


Figure 2: Straight Path.

The algorithm was designed to initially compute the global coverage path using straight lines. Subsequently, Dubins paths are generated for each line segment to ensure the shortest feasible path between each pair of points, optimizing the overall path length. The (Figure 2) illustrate the straight obstacle free paths generated by the algorithm for all the datasets.

The resulting path after applying dubins constraints are depicted in the (Figure 3). These visual representations portray the robot's movement across diverse datasets, providing insight into the algorithm's adaptability amidst varying obstacles.

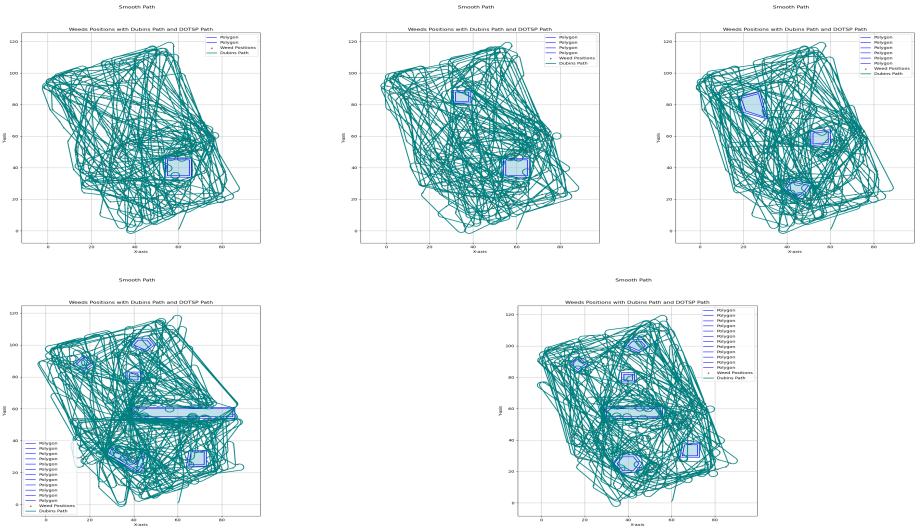


Figure 3: Dubins' Path.

The (Table 2) summarizes the performance metrics for all datasets, offering valuable insights into the algorithm's computational efficiency, operational effectiveness, and overall robustness.

Table 2: Results for the performance metrics.

Cases	*Computation Time in secs.*	*Field Operation Time in sec.*	*Route length in meters*	*No. of Turns.*	*Energy Consumption in KWh.*
case 1	13.32	14612.4	10637.003	134	13792.7026
case 2	20.762	15170.4	11028.086	141	14348.6364
case 3	35.71	15094.8	11079.932	127	14070.7815
case 4	93.918	18694.8	13738.529	155	17388.7787
case 5	58.307	16160.4	12010.546	117	14765.8961

Additionally, the coverage rate plot over the field time provides a visual depiction of the algorithm's performance across all datasets. This plot illustrates how efficiently the algorithm achieves coverage over time, facilitating a comparative analysis of its efficiency under varying scenarios. Please refer to (Figure 4) for visualization of the coverage plots.

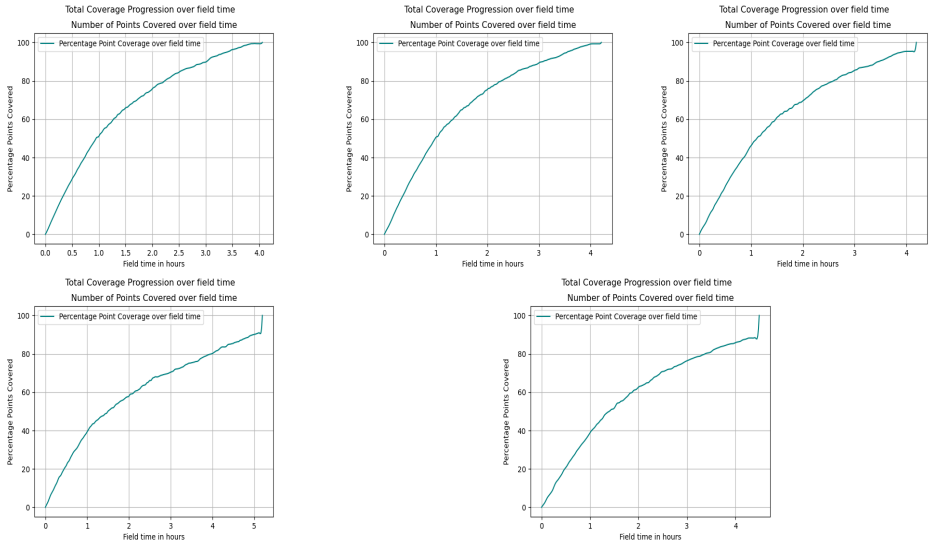


Figure 4: Coverage plots.

The remaining points after the completion of the second behavior in presence of obstacles are depicted in the (Figure 5).

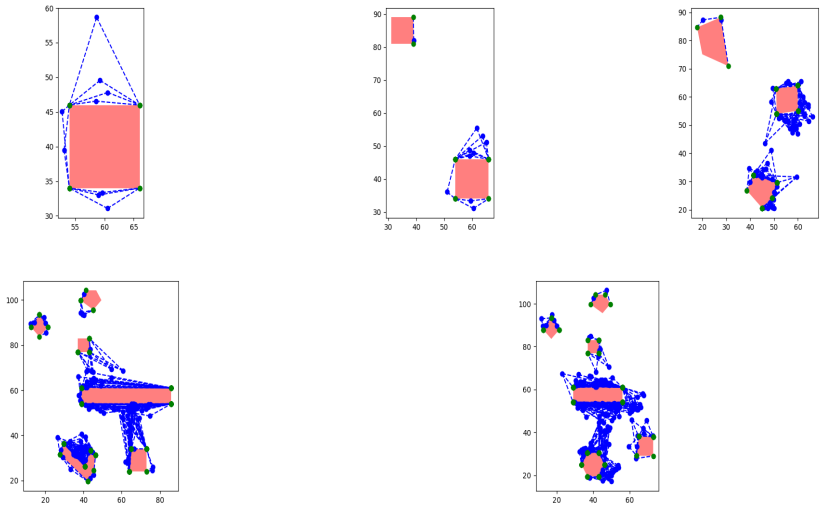


Figure 5: Remaining points after second behavior.

The path obtained for the remaining points utiizing the third behavior can be visualized in the (Figure 6).

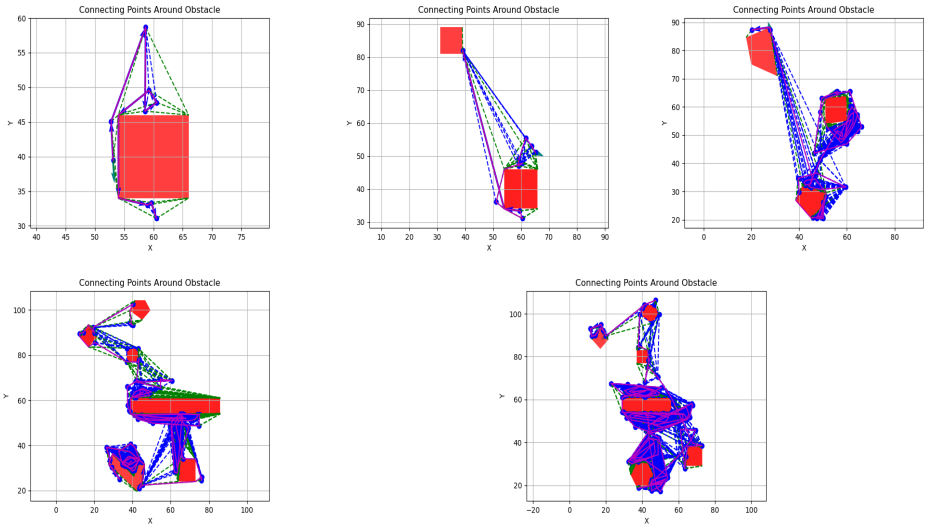


Figure 6: Path for remaining points.

Each perfromance metrics for all the datasets are plotted for better visualization and analysis of the algorithm's performance. The plots are depicted in the (Figure 7), (Figure 8), (Figure 9), (Figure 10), and (Figure 11).

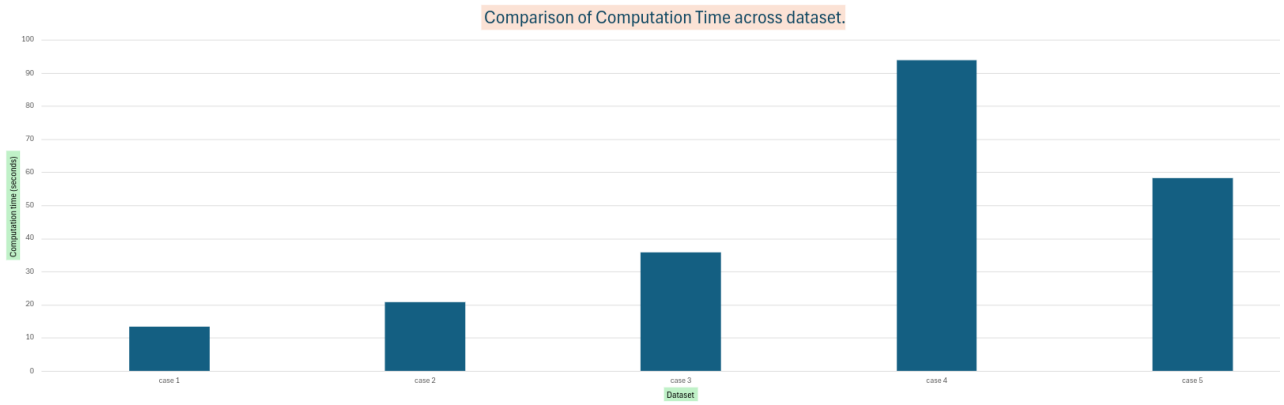


Figure 7: Computation Time.

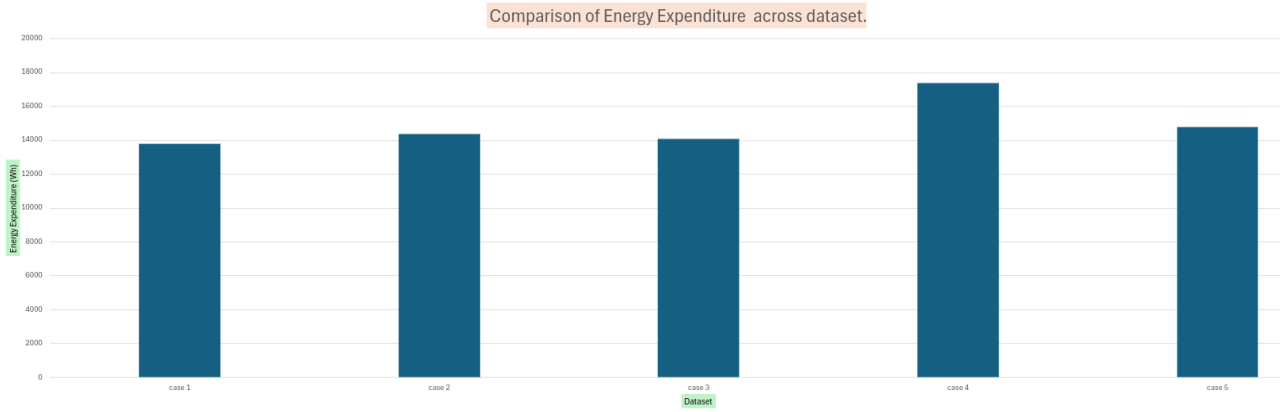


Figure 8: Energy Expenditure.

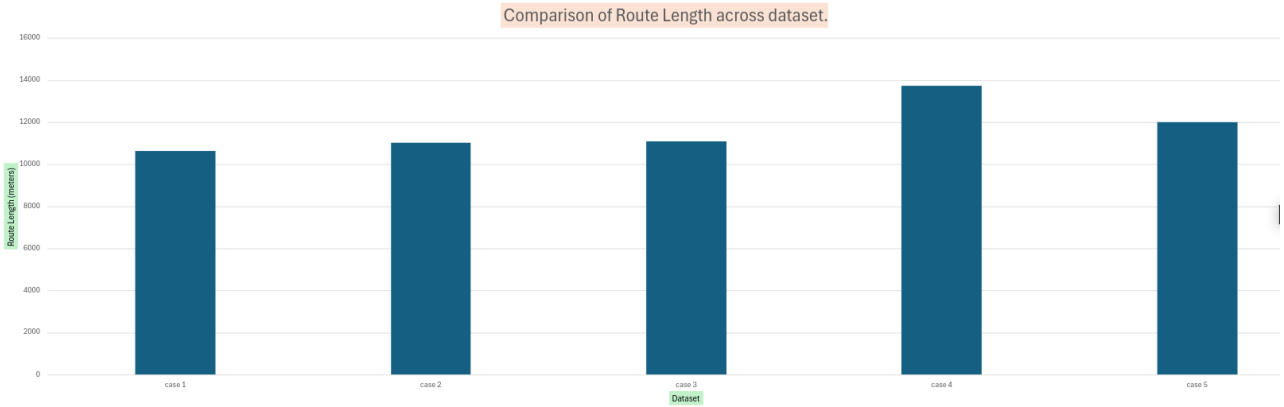


Figure 9: Route Length.

Figure 10: Field Operation Time.

Figure 11: Number of Turns.

Analysis:

From the above plots for computational time, it is evident that computational time increases as the number and complexity of obstacles in the area increase. This is due to the hybrid space approach employed (combining continuous and discrete methods). As is well known, discrete approaches generally require more computation time compared to continuous methods. In our case, an increase in the number or size of obstacles results in a larger discrete space region, thereby increasing computational time. Additionally, as the number of obstacles increases, the likelihood of the path colliding with obstacles rises, necessitating repeated graph generation and path computation for each collision. Therefore, more obstacles and more complex shapes contribute to increased computational time.

However, despite these challenges, the algorithm consistently computes paths within a reasonable timeframe. Even in the most extreme case (case 4, characterized by extensively large, convex, and concave obstacles), the maximum computation time recorded was 93.91 seconds (1.56 minutes). This is well within the acceptable range for agricultural field applications and demonstrates greater efficiency compared to most existing coverage path planning approaches with obstacle avoidance. The efficiency of the algorithm can be attributed to the automatic selection of the step length during graph generation, which is dynamically adjusted to expedite the process. Even in scenarios where frequent path collisions with obstacles occur, the algorithm adaptively modifies the step length to circumvent these obstacles, thereby reducing overall computation time.

Regarding route length, energy expenditure, and field operation time, the results indicate that for the same data distribution with varying number and size of the obstacles, these performance metrics remain relatively consistent across different cases. This highlights the robustness of the algorithm; despite the increased number and complexity of obstacles, the algorithm can compute near-optimal, smooth, natural paths that minimize energy consumption, field operation time, and the number of turns.

Real robot testing and analysis

This section details the implementation of the proposed Complete Coverage Path Planning (CCP) algorithm on a real robot platform, addressing both scenarios with and without obstacles. The subsequent analysis of the experimental results is provided. The section is organized into two main parts: Experimental Setup and Results & Analysis.

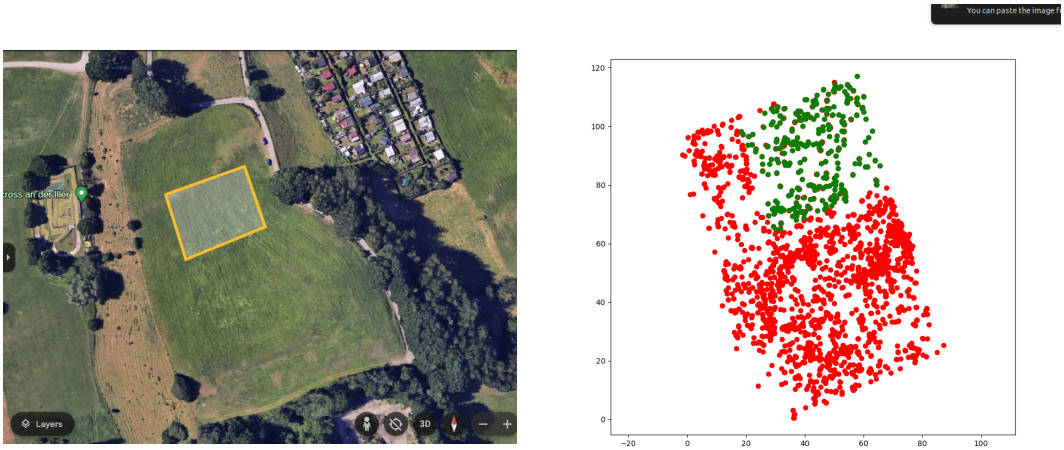
1 Experimental Setup

This subsection provides a comprehensive description of the real robot setup, emphasizing the experimental framework within which the tests are conducted.

The real-world experiments were conducted using a specially designed robot platform equipped with a camera for weed detection upon weed detection by the drone for accuate extractionn, and a GPS module for localization of the robot itself. This robot is specifically engineered to operate in grass fields. The robot can be visualized in (Figure 1).

Figure 1: Real robot.

To perform the tests on the real robot platform, a small region of the complete field was selected, and only the weed positions within this area were considered. Testing on the entire field was impractical due to its extensive size and the significant time required for complete coverage. Therefore, a smaller region was chosen for practical testing purposes. Additionally, complete coverage was not pursued due to time constraints. Instead, the robot was programmed to cover 42% of the points, with the expectation that the robot would behave similarly even with full coverage. This assumption is based on the robot's ability to mimic its behavior in the simulation accurately. The selected region for experimentation and the specific points utilized for the tests are demonstrated in the (Figure 2).

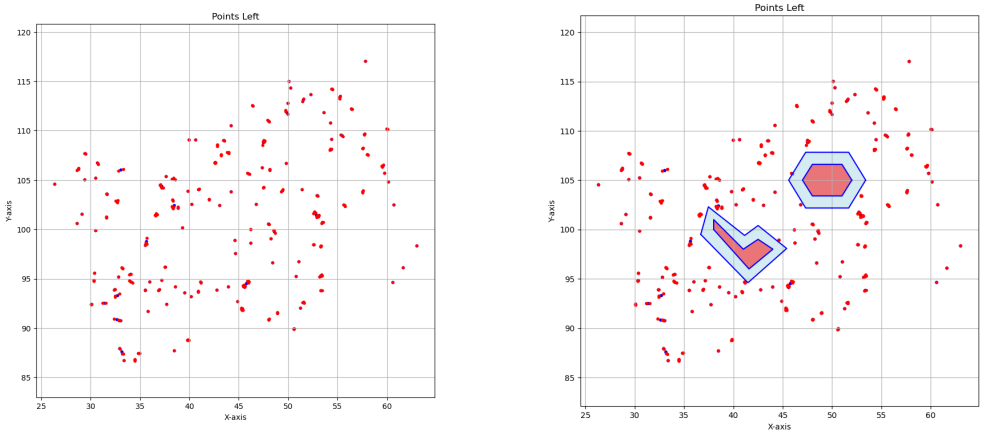


(a) Selected Field Region

(b) Points in the Region (Green)

Figure 2: Selected field and points in the region.

Apart from these adjustments, all the robot constraints and parameters were kept consistent with those discussed in the simulation setup. Two experimental cases were considered: one without obstacles and the other with obstacles. For the scenario involving obstacles, two polygonal obstacles (one concave and one convex) were introduced. The (Figure 3) depict both experimental scenarios.



(a) No obstacle scenario.

(b) Two obstacles scenario.

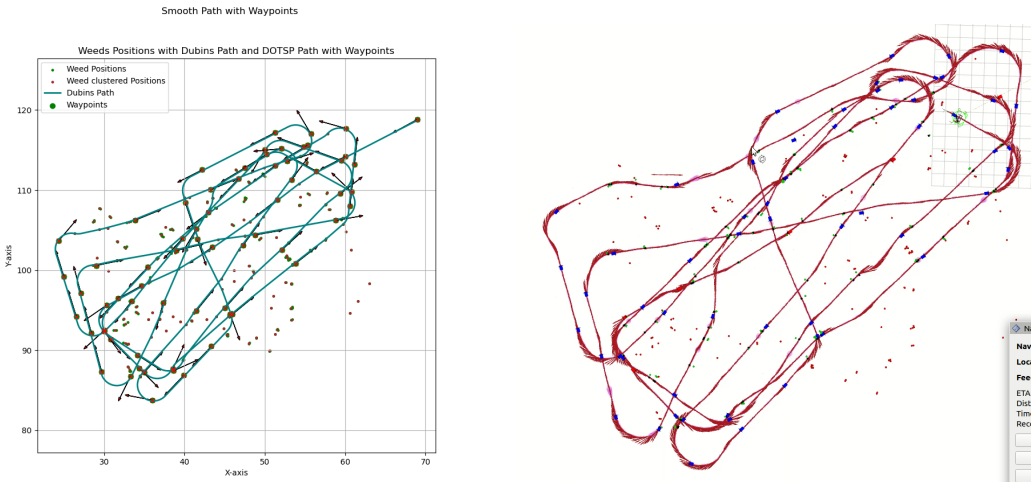
Figure 3: Distinct Environments.

The objective of demonstrating the CCP algorithm on a real robot platform is twofold: to showcase the alignment between the simulated global trajectory and the actual trajectory followed by the robot, and to evaluate the algorithm's effectiveness in weed removal. Specifically, the analysis focuses on how many weeds the robot successfully removes compared to the number planned by the global planner. Since the robot relies on its local planner, any deviation from the global planner's trajectory could result in missed weeds or the coverage of additional, unintended weed points.

This approach aims to highlight the practical applicability and robustness of the CCP algorithm in real-world agricultural settings, emphasizing its ability to adapt to various environmental conditions and obstacles while maintaining efficient operational performance.

2 Real Robot Results and Analysis:

Initially, the real-world experiment was conducted on the first case without obstacles. Given that only 42% coverage was used to demonstrate the results on the real robot, the global trajectory obtained from the simulation is depicted in (Figure 4a). After executing the test on the real robot in the field, the actual path followed by the robot using the local planner is shown in (Figure 4b).



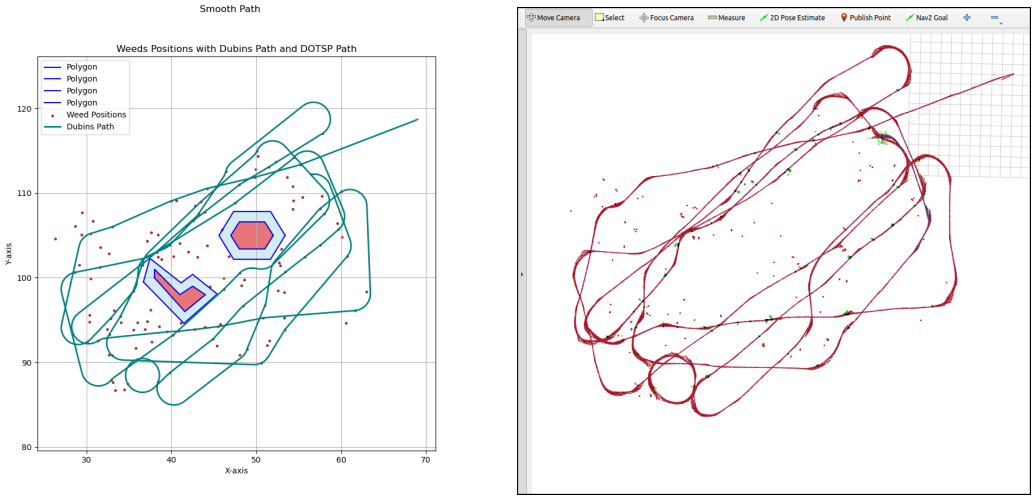
(a) Planned Path

(b) Real Robot Path

Figure 4: No obstacle Trajectories.

The local planner initially identified 118 weeds for extraction, whereas the robot successfully removed 109 of these planned weeds. The discrepancy between the planned and actual removals arises from deviations in the local trajectories relative to the global trajectory. These deviations are primarily due to significant orientation differences between waypoints, which cause variations in the path of the local planner. Additionally, slight orientation adjustments made by the robot while navigating towards the waypoints and extracting weeds contribute to this difference.

Similarly, the same procedure was performed for the second case with obstacles. The global planned trajectory and the local trajectory from the real robot are illustrated in (Figure 5).



(a) Planned Path

(b) Real Robot Path

Figure 5: obstacle Trajectories.

In this scenario, the local planner identified 101 weeds for extraction, whereas the robot successfully removed 84 of these weeds. The higher discrepancy in the number of weeds not covered in the presence of obstacles is attributed to the robot's need to navigate around these obstacles. This navigation often involves making curves, which leads to deviations between the planned path and the executed path, ultimately resulting in a lower success rate compared to scenarios without obstacles. However, despite this decrease, the success rate remains high, demonstrating the robustness of the algorithm, as it performs only slightly less efficiently than in obstacle-free scenarios.

The alignment between the two trajectories is illustrated in (Figure 6). Typically, the success rate of weed removal is higher when the path is approximately straight, which aligns with the primary objective of maintaining a natural and approximately straight path.

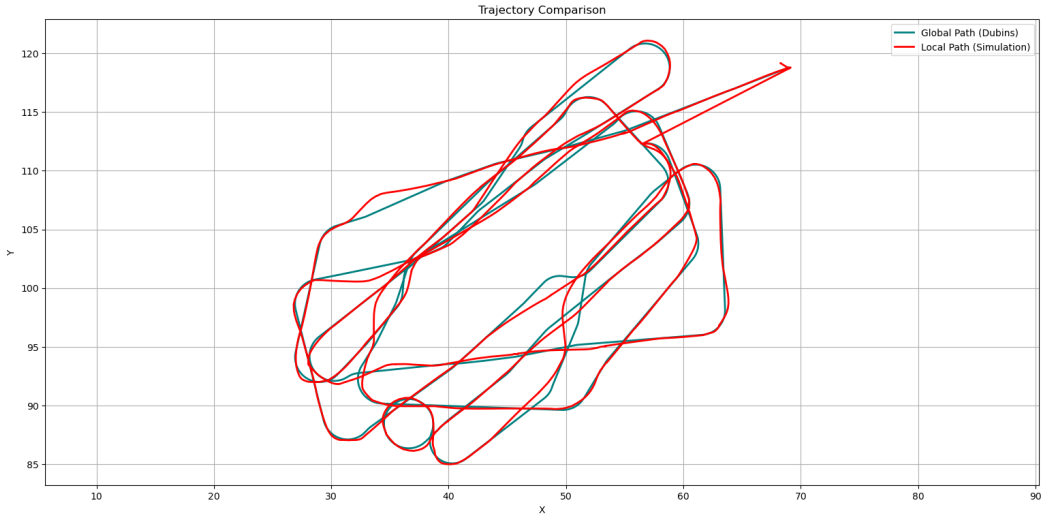
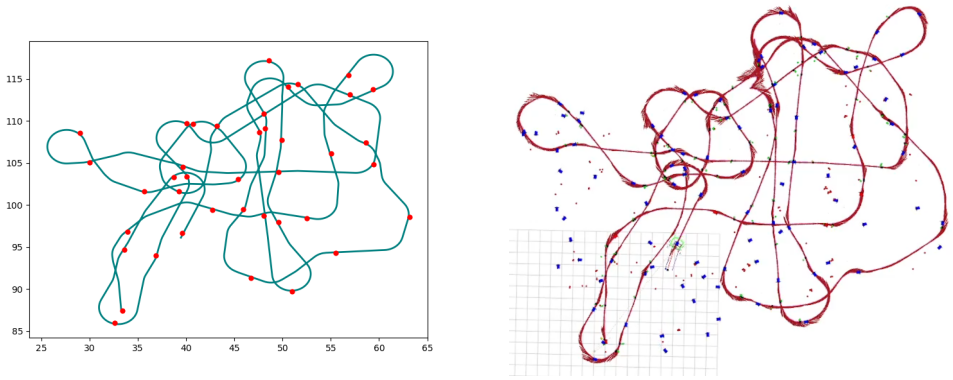


Figure 6: Trajector comparison.

Hence, it is evident from the above results that the robot is capable of closely following the global trajectory because the constraints are so well modelled in the global path to account the behavior of the local planner, thereby ensuring efficient and effective weed removal. The alignment between the global and local trajectories is crucial for achieving complete coverage, as any deviation could result in missed weeds or the coverage of unintended points. By minimizing this deviation, the proposed CCP algorithm can achieve optimal coverage, thus validating its practical applicability in real-world agricultural scenarios. The detailed analysis underscores the importance of trajectory alignment in achieving the desired operational outcomes.

To study the effect of non approximate straight paths on the planned and actually extracted weed points. The results from the graph search approach is also provided here. The scenario is the same as the one without obstacles we discussed earlier. The planned path obtained and the actual path followed by the robot are shown in the (Figure 7). The red dots represent the equidistant waypoints which robot follows to extract the weeds.



(a) Planned Path (b) Real Robot Path

Figure 7: Graph search no obstacle trajectories.

In this scenario, the number of weeds planned for extraction by the local planner was 191, while the number of weeds actually removed by the robot was 150. This discrepancy highlights the impact of non-approximately straight paths on the planned and actual weed extraction points. Curved paths result in greater orientation differences between waypoints, necessitating more adjustments by the robot while removing weeds and navigating these curves. Consequently, this leads to a larger deviation between the planned and actual weed extraction points, ultimately reducing the success rate of weed removal.

The graph search approach respects non-holonomic constraints but does not inherently prioritize approximately straight paths. One potential method to better align the trajectories in such cases is to provide more waypoints with shorter distances between them. However, this would require the robot to stop more frequently to adjust its orientation, select the next points for extraction by the local planner, and move to the next waypoint. While this could improve trajectory alignment, it would also increase field operation time and energy consumption.

The alignment between the two trajectories is illustrated in the (Figure 8).

Analysis:

From the results, it can be summarized that the local planner (real robot) closely follows the global planner, demonstrating the accuracy of the global planner in modeling the robot's constraints in a manner that aligns well with the local planner. A key focus is the comparison between the number of

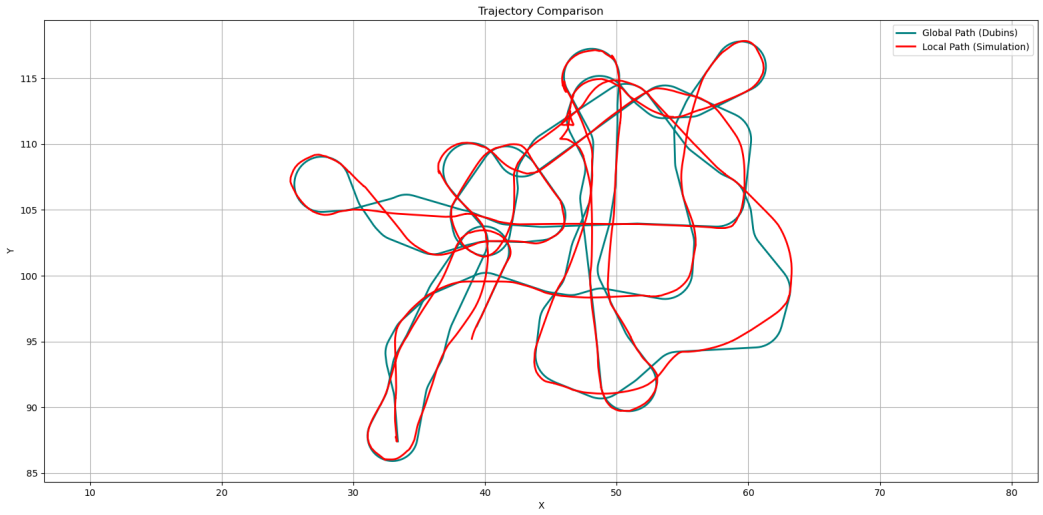


Figure 8: Trajector comparison.

weeds planned for extraction and the number of weeds actually removed. Ideally, these numbers should match as closely as possible, depending on the alignment between the local and global trajectories.

If the robot accurately follows the global trajectory, it will cover and extract all the planned weeds. Conversely, if there is a deviation, the robot may fail to extract some of the planned weeds and instead remove others that were not targeted. These additional weeds would have been covered in subsequent turns, and the missed weeds would remain uncovered, resulting in incomplete coverage.

Thus, the alignment between the global and local trajectories is crucial and can be assessed using the absolute trajectory error and relative pose error. These trajectory error metrics indirectly indicate the efficiency of the algorithm and the alignment between the global and local planners. Therefore, complete coverage is more likely if the trajectory deviation is minimal, whereas greater deviation compromises coverage completeness, leaving some weed points untreated.

By ensuring minimal deviation, the proposed CCP algorithm can achieve efficient and effective coverage, thus validating its applicability in real-world agricultural scenarios. The detailed analysis underscores the importance of trajectory alignment in achieving the desired operational outcomes.

Comparison with other approaches.

In this section, we will conduct a comparative analysis of the proposed complete coverage path planning (CCP) algorithm against two alternative approaches: the lawnmower approach, which serves as the baseline for this thesis, and the graph search approach. The comparison will be conducted under the same experimental conditions described in the simulation section without considering the obstacles, utilizing identical datasets and performance metrics to ensure consistency and fairness in evaluation.

Simulation Setup: As with the initial simulations, the setup is designed to emulate the operational environment of an agricultural field. The same datasets used in the evaluation of the proposed CCP algorithm will be employed here to facilitate a direct comparison. These datasets are characterized by different point distributions and densities, ensuring a comprehensive assessment of each approach's performance.

The key performance metrics for comparison include computational time, field operation time, route length, and energy consumption. These metrics will provide a detailed view of each algorithm's efficiency, effectiveness, and overall robustness.

Lawnmower Approach: The lawnmower approach, often referred to as the boustrophedon path, is a widely used baseline method in coverage path planning. It involves the robot traversing the field in parallel lines, akin to a lawnmower, ensuring that the entire area is covered systematically. This method is straightforward but may not be optimal in terms of path length and energy consumption, particularly in fields with irregular plant distributions or obstacles.

Graph Search Approach: The graph search approach utilizes a graph-based representation of the field, where points of interest (weeds and clusters of weeds, in this context) are treated as nodes, and paths between them are edges. After graph generation, a graph search algorithm is applied to find an optimal path that covers all nodes while adhering to non-holonomic constraints. However, this approach does not prioritize approximately straight paths but ensures compliance with the non-holonomic constraints.

Performance Metrics: The performance metrics for all datasets and three approaches has been summarized in the (Table 1). These metrics will provide a comprehensive view of each algorithm's performance under different scenarios, enabling a detailed comparison of their efficiency and effectiveness. The results will be analyzed to identify the strengths and weaknesses of each approach, facilitating an informed decision on the most suitable method for complete coverage path planning in agricultural fields.

The lawnmover approach and behavioral approach was performed on the same device. However, the graph search approach was performed on a different device with the following specifications:
Processor: 11th Gen Intel(R) Core(TM) i7-1165G7
Processing Speed: 2.80 GHz, Number of Cores: 8 and RAM: 32GB

The plots showcasing the results of the comparison among the proposed behavioral algorithm, the lawnmower approach, and the graph search approach are presented in the following plots ((Figure 1), (Figure 2), (Figure 3), and (Figure 4)). Each plot represents a specific performance metric, such as computational time, energy consumption, route length, and field operation time for all the datasets.

No. of points	Cases	Route Length (meters)			Computation Time (seconds)			Field Time (seconds)			Energy Expenditure (units)		
		Behavioral Approach	Graph search Approach	Baseline (Lawnmover)	Behavioral Approach	Graph search Approach	Baseline (Lawnmover)	Behavioral Approach	Graph search Approach	Baseline (Lawnmover)	Behavioral Approach	Graph search Approach	Baseline (Lawnmover)
250	(case 1)	4652.945	2288.31	9896.291776	0.858553886	4.435761562	0.601133347	6282.400938	1741.022767	13910.92722	5783.34459	3632.750509	13593.64178
	(case 2)	3769.005	2083.73	8253.784905	0.775789261	4.341929366	0.587094307	5125.168209	1576.077269	11632.10613	4734.555207	3447.362011	11409.4849
	(case 3)	3985.978	2092.1	7564.831982	0.788151503	4.599930199	0.590665579	5413.822753	1572.229368	10819.97748	5022.177564	3613.97386	10838.28198
	(case 4)	4620.695	2061.77	8564.113438	0.825942278	4.318975639	0.613069534	6216.01818	1546.992294	12186.8293	5656.894864	3598.218862	12120.16344
500	(case 1)	7027.363	3710.8	17071.30153	1.842700005	8.956043648	0.758333683	9520.140257	2806.494905	23468.43942	8793.612844	6143.002153	22181.65153
	(case 2)	6788.619	3248.08	11616.47502	1.696401358	8.61580853	0.695278645	9151.211911	2442.859621	16679.34377	8366.468569	5582.217349	16797.47502
	(case 3)	6336.544	3304.11	12034.99812	1.940592766	8.740781215	0.669975042	8589.717644	2469.994472	16947.37264	7914.394434	5903.641572	16603.69812
	(case 4)	7250.2	3215.45	11852.70452	1.609200478	9.350487668	0.683098078	9761.624722	2396.719655	16945.19315	8898.700417	5850.162864	16963.05452
2000	(case 1)	16189.858	655.67	28265.5625	9.04812026	40.44517501	1.046939611	21751.64778	6456.38521	37873.39062	19816.55759	15678.01424	34365.0125
	(case 2)	16650.701	7850.07	25402.09418	8.591621161	36.87743123	1.091753483	22226.37676	5819.37001	34519.74273	20041.90077	14760.4663	32043.19418
	(case 3)	13728.721	7257.3	21144.39368	7.65038085	34.6019571	0.995180845	18446.73883	5373.163304	29256.4921	16813.77138	13747.4777	27926.79368
	(case 4)	15547.469	7828.97	23561.53673	8.155776024	36.02791135	1.016644001	20839.67217	5806.381665	32091.48341	18820.91894	14680.92948	29896.48673
4000	(case 1)	24191.259	13033.92	31436.1282	24.18394971	80.41146149	1.38740325	32301.69933	9654.118538	41846.41025	29136.75914	24629.20129	37559.1282
	(case 2)	22356.011	11583.92	28932.5249	20.85941482	71.84240132	1.366778374	29871.08874	8577.056512	39050.53112	27018.91147	21935.23424	35856.2249
	(case 3)	20620.054	10643.04	26886.435	18.23963118	66.78159156	1.275558949	27659.4681	7849.986297	36257.41874	25141.65448	20609.80285	33244.935
	(case 4)	22222.31	11076.75	28340.7056	19.22349644	68.28330584	1.293629885	29701.58872	8162.427239	38310.757	26932.31049	21561.45279	35264.4056
6000	(case 1)	29777.382	16816.83	33040.58685	45.23673987	286.3729786	1.66720438	39851.80224	12440.13841	43851.98356	35947.48179	32016.74242	39163.58685
	(case 2)	26039.854	14100.83	30366.86311	33.23154497	246.6400248	1.570202589	34821.16749	10412.05816	40500.01639	31597.65399	27130.02134	36466.31311
	(case 3)	26223.169	13599	31659.26981	30.80121374	237.7119713	1.675028086	35035.83592	9984.599781	42135.14976	31639.66906	27018.06546	37805.81981
	(case 4)	26570.832	14609.54	32148.11261	32.34737062	244.7778758	1.63052392	35496.36509	10738.4984	43227.01576	31893.13207	28846.61331	39448.61261
10000	(case 1)	38128.675	21623.33	35598.33764	122.6808474	490.2239986	2.21295166	50681.89334	15947.4934	47049.17205	45382.07468	41890.90133	41721.33764
	(case 2)	34118.252	18820.21	36001.3554	74.37674475	411.4919939	2.168357611	45367.87779	13852.97698	48288.88175	40641.60223	36868.19144	43890.6054
	(case 3)	31650.696	17269.61	33119.52466	65.54510021	376.7545004	2.029067278	42291.24514	12686.21149	44245.03083	38197.59611	34211.77633	39949.02466
	(case 4)	32154.031	17527.4	34293.53829	62.76576424	375.6292722	2.238646507	42801.16425	12876.94731	45859.73536	38606.73138	34702.42134	41476.28829

Table 1: Performance metrics for comparison of all approaches.

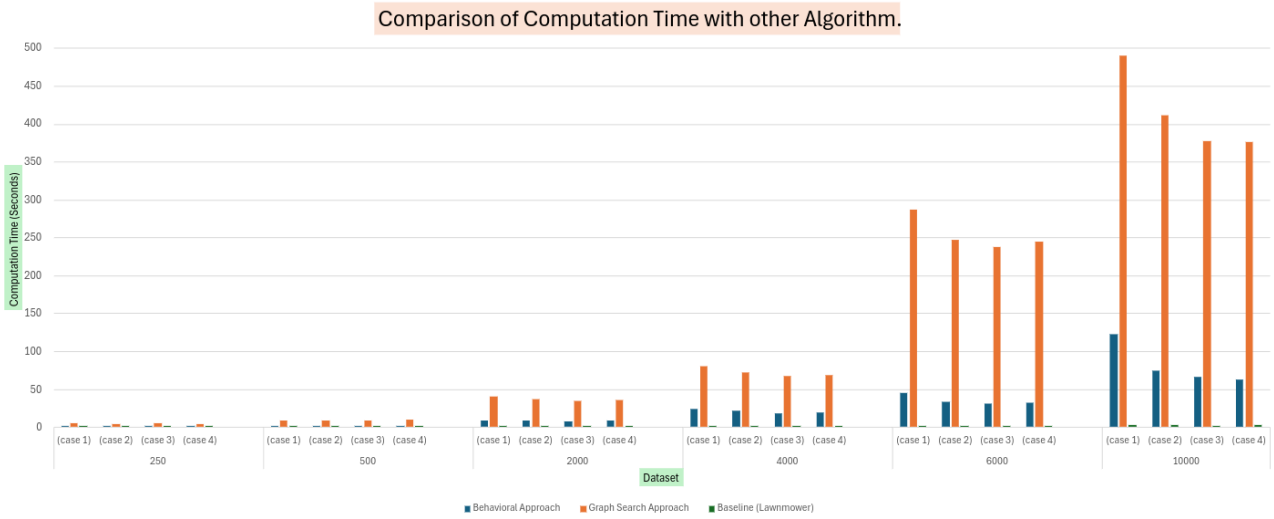


Figure 1: Computation Time.

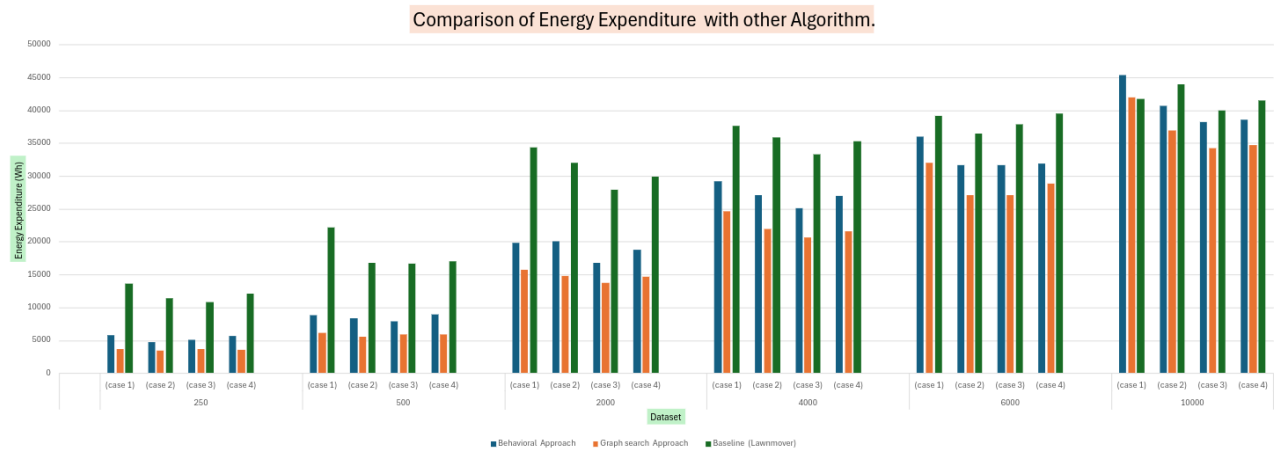


Figure 2: Energy Expenditure.

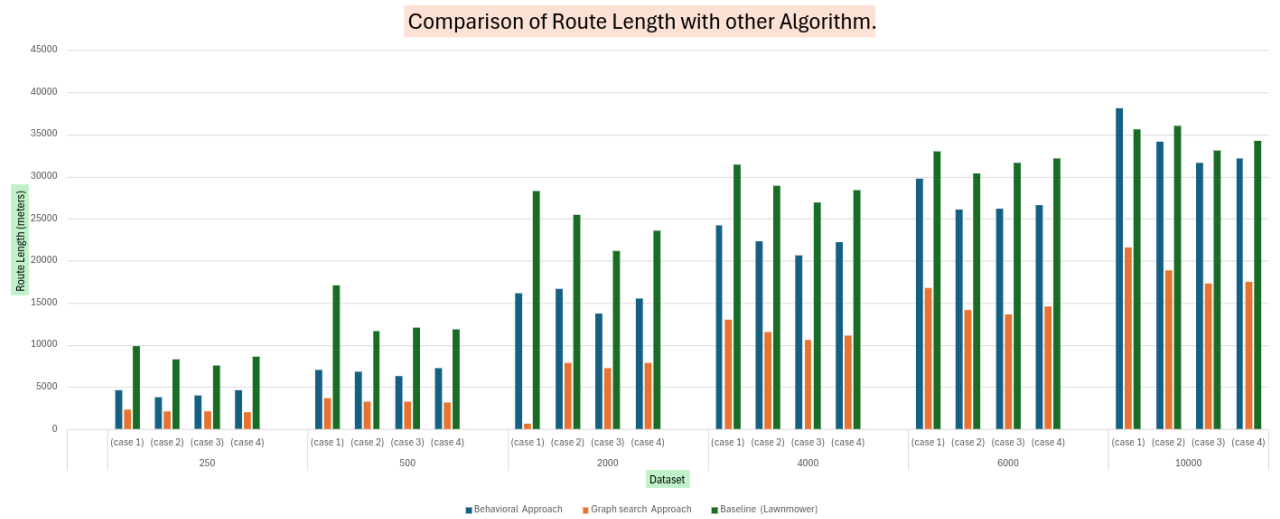


Figure 3: Route Length.

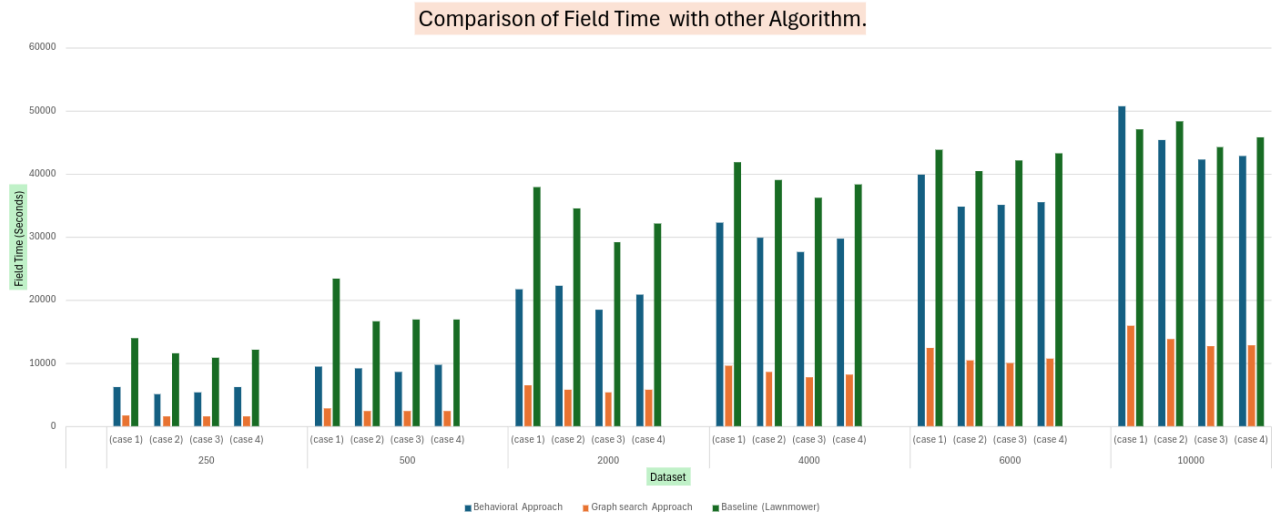


Figure 4: Field Operation Time.

Analysis of the comparison results

From the first plot of computation time, it is evident that the lawnmower approach exhibits the least and most constant computation time across all datasets. This is attributed to its simplicity, merely forming parallel lines without extensive computations. In contrast, the behavioral approach maintains a similar computation time to the lawnmower approach for smaller datasets (up to 500 points) but gradually increases as the number of points grows. The graph search approach shows the highest computation time across all datasets, with a significant increase as the number of points rises, making it considerably higher than the other two approaches.

In terms of energy efficiency, the lawnmower approach consistently demonstrates the highest energy consumption due to the extensive parallel line formation. Conversely, the graph search algorithm exhibits the least energy consumption across all datasets. The behavioral approach, while prioritizing straight paths, achieves energy consumption levels quite close to the graph search approach, indicating its efficiency.

The route length plot reveals that the lawnmower approach has the highest route length for all datasets, while the graph search approach maintains the shortest route length. The behavioral approach falls between the other two approaches.

The field operation time plot indicates that the graph search approach has the shortest field operation time across all datasets. Initially, the lawnmower approach shows the highest field operation time for smaller datasets, with the behavioral approach positioned between the two. However, as the number of points increases, the field operation time for the behavioral approach escalates and nearly matches that of the lawnmower approach.

Performance Metrics.	***Types of coverage path planning algorithms***							
	Randomized algorithms	***Spanning tree coverage***	***Artificial potential field***	***Sampling-based planning***	***Graph search algorithms***	***Evolutionary algorithms***	**Human-inspired algorithms**	**Behavioral algorithm (ours)**
Fast Searching Time	✓		✓					✓
Collision Avoidance			✓	✓	✓	✓	✓	✓
Complete Coverage		✓		✓			✓	✓
The Shortest Path between two Points				✓	✓		✓	✓
TSP Optimization						✓	✓	✓
Large-Scale Structural Coverage				✓			✓	✓
Low Computational Cost	✓	✓	✓		✓			✓
Global Path		✓		✓	✓	✓	✓	✓
Local Path	✓	✓	✓	✓				✓
Experimentally Sufficient	✓	✓			✓			✓
Potential Future Search				✓	✓	✓	✓	✓
Less Number of Turns.		✓			✓	✓	✓	✓
Low Energy Consumption					✓	✓	✓	✓
Scalable Algorithm.	✓	✓		✓				✓
Low Trajectory Error between local and global paths								✓
Naturally Straight Path Approximations								✓
Grid Operational Space		✓	✓					✓
Continuous Operational Space	✓			✓	✓	✓	✓	✓
Hybrid Operational Space								✓

Table 2: Comparison of various path planning algorithms: Performance and analysis.

Discussion

The results presented in the previous section shed light on the performance of three distinct approaches – the lawnmower approach, behavioral approach, and graph search approach – for weed removal in agricultural fields. Each approach offers unique advantages and challenges, contributing to a comprehensive understanding of their suitability for real-world implementation.

Lawnmower Approach

The lawnmower approach is characterized by its simplicity, resulting in consistent and low computation times across all datasets. Its linear path formation facilitates straightforward trajectory planning, making it suitable for smaller field sizes and less complex terrains. However, as the complexity of the field increases, the scalability of the lawnmower approach becomes limited. Furthermore, the linear path formation contributes to longer route lengths, higher field operation times, and increased energy consumption, ultimately impacting operational efficiency. Despite these limitations, the lawnmower approach remains a robust and reliable option for weed removal tasks in simpler agricultural environments. Therefore, if the field is relatively straightforward and compact, and the primary concern is not optimizing route length, minimizing field time, or reducing energy consumption, then the lawnmower approach presents a viable choice.

Graph Search Approach

The graph search approach is centered around generating optimal paths by conceptualizing the field as a graph, with nodes representing key points of interest. By leveraging graph search algorithms, this approach achieves the shortest route lengths and field operation times, ideal for maximizing efficiency in weed removal operations. However, its notable computational overhead, particularly with larger datasets, poses a significant challenge. Despite its computational demands, the graph search approach prioritizes optimality over simplicity, resulting in longer computation times but ensuring minimal energy consumption for smaller datasets. Nonetheless, its real-time adaptability in dynamic environments is limited, necessitating extensive computational resources for path planning. Despite these challenges, the ability of the graph search approach to identify optimal paths makes it a promising option for weed removal tasks in controlled agricultural settings. It is a suitable choice when computational time and naturally straight paths are not primary concerns, and the main focus is on minimizing route length, energy consumption, and field operation time. However, for larger datasets, the graph search approach will exhibit similar energy consumption as of other aproaches due to the increased presence of curved paths. Therefore, it may not be the most suitable option if energy consumption and computation time are significant concerns, particularly for larger datasets.

Behavioral Approach

The behavioral approach offers a versatile and adaptable solution, capable of dynamically adjusting trajectories based on field conditions and encountered obstacles. By prioritizing straight paths, it achieves competitive computation times, especially effective for smaller datasets. However, scalability becomes a concern as the number of points increases, resulting in longer computation times compared to the lawnmower approach but significantly lower than the graph search approach. Despite this, the behavioral approach strikes a pragmatic balance between route length and field operation time, presenting a practical compromise between simplicity and optimality. Its inherent adaptability to diverse field topologies renders it suitable for a broad spectrum of agricultural applications.

In scenarios where straightness is not a primary concern, the behavioral approach can still yield comparable results to the graph search approach by adjusting the region of the robot's vision cone. Thus, the behavioral approach emerges as a promising option for weed removal tasks in complex and varied agricultural environments, where adaptability and operational efficiency are paramount. It serves as a suitable choice when computational time and energy consumption are significant considerations, with a primary emphasis on achieving a balance between route length and field operation time.

However, the current implementation's prioritization of straight paths may not be optimal when straightness and computational time are less critical, and energy consumption, route length, and field operation time take precedence, particularly for smaller datasets. Nonetheless, for larger datasets, the behavioral approach remains a viable option, offering comparable results to the graph search approach and demonstrating its efficacy in addressing the challenges of agricultural field operations.

Future Directions

The findings from the comparative analysis underscore the need for further exploration and refinement of the behavioral approach in agricultural and other path planning tasks. The following future research directions are proposed to address key challenges and capitalize on opportunities for improvement:

- **Enhanced Obstacle Avoidance:** The current obstacle avoidance strategy in the behavioral approach primarily focuses on checking obstacles during the generation of straight paths, potentially resulting in trajectories that approach the corners of obstacles. To address this limitation, an advanced obstacle avoidance methodology could be implemented, integrating obstacle checking directly into the Dubins path generation phase. This enhancement would involve seamlessly incorporating obstacle detection and avoidance mechanisms throughout the trajectory planning process. By doing so, the algorithm would yield smoother and more efficient paths, particularly in environments characterized by intricate obstacle layouts, while ensuring avoidance of obstacle corners.

- **Dynamic Obstacle Adaptation:** Extending the behavioral approach to dynamically adapt to moving obstacles presents an exciting avenue for future research. While the current implementation successfully generates grids for static obstacles, enhancing the algorithm to account for dynamic obstacles would require associating grids with other robots or moving objects whose positions are known. By integrating real-time sensor data and predictive modeling techniques, the algorithm could dynamically adjust its trajectory to avoid collisions with moving obstacles, thus enabling multi-robot coordination and dynamic obstacle avoidance in agricultural and other environments.

- **Sensor-Based Obstacle Avoidance:** Leveraging sensor data for obstacle detection and avoidance represents another promising direction for enhancing the behavioral approach. By integrating sensors such as LiDAR or cameras, the algorithm can detect obstacles in its environment and autonomously navigate around them. In scenarios where moving obstacles with unknown positions are encountered, an automatic grid generation mechanism could be invoked to guide the robot away from potential collision points. This sensor-based approach offers increased adaptability and robustness, enabling the algorithm to effectively navigate complex and dynamic environments.

- **Generalization to Other Planning Problems:** Beyond its application in coverage path planning, the behavioral approach holds potential for addressing a broader range of planning problems. Given its efficient computational performance and adaptive nature, slight modifications to the algorithm could render it suitable for tasks such as trajectory planning, path optimization, and path following in various domains. Leveraging the algorithm's inherent flexibility and scalability opens avenues for exploring its applicability to various planning challenges, thereby driving the advancement of autonomous systems in agriculture and beyond.

In summary, the future directions outlined above aim to refine and extend the capabilities of the behavioral approach, paving the way for more robust, adaptive, and versatile solutions in motion planning.

Conclusion

This thesis presents a detailed investigation into the development of a novel coverage path planning (CPP) algorithm for autonomous agricultural robots, specifically aimed at the efficient removal of Rumex weeds in grasslands. The central focus was to create a behavioral algorithm that adheres to non-holonomic constraints, prioritizes straight paths, is computationally efficient, and enhances coverage efficiency.

The proposed behavioral algorithm dynamically adapts its behaviors to ensure near-optimal coverage and efficient navigation in complex environments. Research findings demonstrate that this algorithm consistently outperforms other methods regarding computation time, path straightness, coverage rate, route length, energy consumption, and weed extraction efficiency. Its adaptability allows the strategy to evolve throughout the coverage process, ensuring optimal performance across various environments.

This approach contributes to CPP by introducing several innovations: automatic behavior changes, the use of a hybrid space (continuous and discrete), dynamic grid generation based on obstacle sizes, and generating approximate straight paths, even in complex environment.

Two algorithm variants were developed: one for environments without obstacles and another for those with obstacles. Comparative analyses with other methods such as graph search and the lawnmower approach showed that the proposed behavioral algorithm maintains a higher coverage rate and efficiently extracts planned weeds.

In addition to its technical merits, the algorithm advances the field of CPP by offering a flexible, efficient, and adaptive solution. This research establishes a strong foundation for future work, encouraging further exploration and refinement of these algorithms. Future research should focus on applicability to dynamic obstacles and multi-robot systems, further enhancing the capabilities of autonomous systems in both agricultural and non-agricultural domains.

Bibliography

[1] Dubins, L.E., 1957. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. American Journal of mathematics, 79(3), pp.497-516.

[2] Tan, C.S., Mohd-Mokhtar, R. and Arshad, M.R., 2021. A comprehensive review of coverage path planning in robotics using classical and heuristic algorithms. IEEE Access, 9, pp.119310-119342.

[3] Reeds, J. and Shepp, L., 1990. Optimal paths for a car that goes both forwards and backwards. Pacific journal of mathematics, 145(2), pp.367-393.

[4] Dumitrescu, A. and Mitchell, J.S., 2003. Approximation algorithms for TSP with neighborhoods in the plane. Journal of Algorithms, 48(1), pp.135-159.

[5] Savla, K., Frazzoli, E. and Bullo, F., 2005, June. On the point-to-point and traveling salesperson problems for Dubins' vehicle. In Proceedings of the 2005, American Control Conference, 2005. (pp. 786-791). IEEE.

[6] Váňa, P. and Faigl, J., 2015, September. On the dubins traveling salesman problem with neighborhoods. In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 4029-4034). IEEE.

[7] Isaacs, J.T. and Hespanha, J.P., 2013. Dubins traveling salesman problem with neighborhoods: A graph-based approach. Algorithms, 6(1), pp.84-99.

[8] Choudhary, A., Kobayashi, Y., Arjonilla, F.J., Nagasaka, S. and Koike, M., 2021, January. Evaluation of mapping and path planning for non-holonomic mobile robot navigation in narrow pathway for agricultural application. In 2021 IEEE/SICE International Symposium on System Integration (SII) (pp. 17-22). IEEE.

[9] Kegeleirs, M., Garzón Ramos, D. and Birattari, M., 2019, July. Random walk exploration for swarm mapping. In Annual conference towards autonomous robotic systems (pp. 211-222). Cham: Springer International Publishing.

[10] Zhou, P., Wang, Z.M., Li, Z.N. and Li, Y., 2012, September. Complete coverage path planning of mobile robot based on dynamic programming algorithm. In 2nd International Conference on Electronic & Mechanical Engineering and Information Technology (pp. 1837-1841). Atlantis Press.

[11] Yan, J., Li, Y., Song, Z. and Zhang, Q., 2019. Research on UAV coverage path planning algorithm based on improved artificial potential field method. Oper. Res. Fuzziol., 9(4), pp.264-270.

[12] Gabriely, Y. and Rimon, E., 2001. Spanning-tree based coverage of continuous areas by a mobile robot. Annals of mathematics and artificial intelligence, 31, pp.77-98.

[13] Lin, Y. and Saripalli, S., 2017. Sampling-based path planning for UAV collision avoidance. IEEE Transactions on Intelligent Transportation Systems, 18(11), pp.3179-3192.

[14] Karaman, S., 2012. Sampling-based algorithms for optimal path planning problems (Doctoral dissertation, Massachusetts Institute of Technology).

[15] Pivtoraiko, M., Knepper, R.A. and Kelly, A., 2007. Optimal, smooth, nonholonomic mobile robot motion planning in state lattices. Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-15.

[16] Galceran, E. and Carreras, M., 2013. A survey on coverage path planning for robotics. Robotics and Autonomous systems, 61(12), pp.1258-1276.

[17] Muthukumaran, S., Ganesan, M., Dhanasekar, J. and Babu Loganathan, G., 2021. Path Planning Optimization for Agricultural Spraying Robots Using Hybrid Dragonfly–Cuckoo Search Algorithm. Alinteri Journal of Agriculture Sciences, 36(1), pp.2564-7814.

[18] Gajjar, S., Bhadani, J., Dutta, P. and Rastogi, N., 2017, May. Complete coverage path planning algorithm for known 2d environment. In 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT) (pp. 963-967). IEEE.

[19] Kalaivanan, S. and Kalpana, R., 2017, June. Coverage path planning for an autonomous robot specific to agricultural operations. In 2017 International Conference on Intelligent Computing and Control (I2C2) (pp. 1-5). IEEE.

[20] Victerpaul, P., Saravanan, D., Janakiraman, S. and Pradeep, J., 2017. Path planning of autonomous mobile robots: A survey and comparison. Journal of Advanced Research in Dynamical and Control Systems, 9(12), pp.1535-1565.

[21] Zhao, S. and Hwang, S.H., 2024. Complete coverage path planning scheme for autonomous navigation ROS-based robots. ICT Express, 10(1), pp.83-89.

[22] Sánchez-Ibáñez, J.R., Pérez-del-Pulgar, C.J. and García-Cerezo, A., 2021. Path planning for autonomous mobile robots: A review. Sensors, 21(23), p.7898.

[23] Khan, A., Noreen, I. and Habib, Z., 2017. On Complete Coverage Path Planning Algorithms for Non-holonomic Mobile Robots: Survey and Challenges. J. Inf. Sci. Eng., 33(1), pp.101-121.

[24] Pěnička, R., Faigl, J., Váňa, P. and Saska, M., 2017, June. Dubins orienteering problem with neighborhoods. In 2017 International Conference on Unmanned Aircraft Systems (ICUAS) (pp. 1555-1562). IEEE.

[25] aigl, J. and Pěnička, R., 2017, September. On close enough orienteering problem with dubins vehicle. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 5646-5652). IEEE.

[26] ěnička, R., Faigl, J., Váňa, P. and Saska, M., 2017. Dubins orienteering problem. IEEE Robotics and Automation Letters, 2(2), pp.1210-1217.

[27] rchal, J., Faigl, J. and Váňa, P., 2020. WiSM: Windowing surrogate model for evaluation of curvature-constrained tours with Dubins vehicle. IEEE Transactions on Cybernetics, 52(2), pp.1302-1311.

[28] ána, P. and Faigl, J., 2018, June. Optimal Solution of the Generalized Dubins Interval Problem. In Robotics: Science and Systems.

[29] aigl, J., Váňa, P., Saska, M., Báča, T. and Spurný, V., 2017, September. On solution of the Dubins touring problem. In 2017 European Conference on Mobile Robots (ECMR) (pp. 1-6). IEEE.

[30] aigl, J., Váňa, P. and Drchal, J., 2020, October. Fast sequence rejection for multi-goal planning with dubins vehicle. In 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 6773-6780). IEEE.

[31] áňa, P. and Faigl, J., 2015. On sampling based methods for the dubins traveling salesman problem with neighborhoods.

[32] aigl, J., Váňa, P. and Deckerová, J., 2019. Fast heuristics for the 3-D multi-goal path planning based on the generalized traveling salesman problem with neighborhoods. IEEE Robotics and Automation Letters, 4(3), pp.2439-2446.

[33] apoutsis, A.C., Chatzichristofis, S.A. and Kosmatopoulos, E.B., 2017. DARP: Divide areas algorithm for optimal multi-robot coverage path planning. Journal of Intelligent & Robotic Systems, 86, pp.663-680.

[34] afar, M.N. and Mohanta, J.C., 2018. Methodology for path planning and optimization of mobile robots: A review. Procedia computer science, 133, pp.141-152.

[35] andey, A., Pandey, S. and Parhi, D.R., 2017. Mobile robot navigation and obstacle avoidance techniques: A review. Int Rob Auto J, 2(3), p.00022.

[36] hołodowicz, E. and Figurowski, D., 2017. Mobile robot path planning with obstacle avoidance using particle swarm optimization. Pomiary Automatyka Robotyka, 21(3), pp.59-68.

[37] slam, M.R., Protik, P., Das, S. and Boni, P.K., 2021. Mobile robot path planning with obstacle avoidance using chemical reaction optimization. Soft Computing, 25, pp.6283-6310.

[38] ameed, I.A., 2014. Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain. Journal of Intelligent & Robotic Systems, 74(3), pp.965-983.

[39] elek, A., Seder, M., Brezak, M. and Petrović, I., 2022. Smooth complete coverage trajectory planning algorithm for a nonholonomic robot. Sensors, 22(23), p.9269.

[40] etereit, J., 2017. Adaptive State× Time Lattices: A contribution to mobile robot motion planning in unstructured dynamic environments (Vol. 27). KIT Scientific Publishing.

[41] hoset, H., Lynch, K.M., Hutchinson, S., Kantor, G.A. and Burgard, W., 2005. Principles of robot motion: theory, algorithms, and implementations. MIT press.

[42] alaivanan, S. and Kalpana, R., 2017, June. Coverage path planning for an autonomous robot specific to agricultural operations. In 2017 International Conference on Intelligent Computing and Control (I2C2) (pp. 1-5). IEEE.

[43] ormann, R., Jordan, F., Hampp, J. and Hägele, M., 2018, May. Indoor coverage path planning: Survey, implementation, analysis. In 2018 IEEE International Conference on Robotics and Automation (ICRA) (pp. 1718-1725). IEEE.

[44] andey, A., Pandey, S. and Parhi, D.R., 2017. Mobile robot navigation and obstacle avoidance techniques: A review. Int Rob Auto J, 2(3), p.00022.

[45] ümpel, J., Sparbert, J., Tibken, B. and Hofer, E.P., 1999, August. Quadtree decomposition of configuration space for robot motion planning. In 1999 European Control Conference (ECC) (pp. 277-282). IEEE.

[46] ahiya, C. and Sangwan, S., 2018. Literature review on travelling salesman problem. International Journal of Research, 5(16), pp.1152-1155.

[47] an, X., Zhang, C., Luo, W., Li, W., Chen, W. and Liu, H., 2012. Solve traveling salesman problem using particle swarm optimization algorithm. International Journal of Computer Science, 9(6), pp.264-271.

[48] aman, V. and Gill, N.S., 2017. Review of different heuristic algorithms for solving Travelling Salesman Problem. International Journal of Advanced Research in Computer Science, 8(5).

[49] áňa, P. and Faigl, J., 2020. Optimal solution of the Generalized Dubins Interval Problem: finding the shortest curvature-constrained path through a set of regions. Autonomous Robots, 44(7), pp.1359-1376.

[50] ankaranarayanan, A. and Vidyasagar, M., 1990, May. A new path planning algorithm for moving a point object amidst unknown obstacles in a plane. In Proceedings., IEEE International Conference on Robotics and Automation (pp. 1930-1936). IEEE.

[51] urynek, P., 2009, May. A novel approach to path planning for multiple robots in bi-connected graphs. In 2009 IEEE international conference on robotics and automation (pp. 3613-3619). IEEE.

[52] saacs, J.T. and Hespanha, J.P., 2013. Dubins traveling salesman problem with neighborhoods: A graph-based approach. Algorithms, 6(1), pp.84-99.

[53] eng, C. and Isler, V., 2020. Visual coverage path planning for urban environments. IEEE Robotics and Automation Letters, 5(4), pp.5961-5968.

[54] iet, H.H., Dang, V.H., Laskar, M.N.U. and Chung, T., 2013. BA*: an online complete coverage algorithm for cleaning robots. Applied intelligence, 39, pp.217-235.

[55] ameed, I.A., 2017, July. Coverage path planning software for autonomous robotic lawn mower using dubins' curve. In 2017 IEEE International Conference on Real-time Computing and Robotics (RCAR) (pp. 517-522). IEEE.

[56] ameed, I.A., 2014. Intelligent coverage path planning for agricultural robots and autonomous machines on three-dimensional terrain. Journal of Intelligent & Robotic Systems, 74(3), pp.965-983.

[57] uo, G., Zhang, P. and Qiao, J., 2010, March. Path planning algorithm based on sub-region for agricultural robot. In 2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics (CAR 2010) (Vol. 2, pp. 197-200). IEEE.

[58] arrientos, A., Colorado, J., Cerro, J.D., Martinez, A., Rossi, C., Sanz, D. and Valente, J., 2011. Aerial remote sensing in agriculture: A practical approach to area coverage and path planning for fleets of mini aerial robots. Journal of Field Robotics, 28(5), pp.667-689.

[59] ameed, I.A., Bochtis, D. and Sørensen, C.A., 2013. An optimized field coverage planning approach for navigation of agricultural robots in fields involving obstacle areas. International journal of advanced robotic systems, 10(5), p.231.

[60] hou, K., Jensen, A.L., Sørensen, C.G., Busato, P. and Bothtis, D.D., 2014. Agricultural operations planning in fields with multiple obstacle areas. Computers and electronics in agriculture, 109, pp.12-22.

[61] andamurthy, K. and Ramanujam, K., 2020. A hybrid weed optimized coverage path planning technique for autonomous harvesting in cashew orchards. Information Processing in Agriculture, 7(1), pp.152-164.

Appendix

sectionExperiment and Metrics Links

1. Effect of Behavior Change Percent: Click Here
2. Lawnmower Algorithm Performance Metrics: Click Here
3. Behavioral Algorithm Performance Metrics: Click Here
4. Obstacle Avoidance Algorithm Performance Metrics: Click Here

sectionVideos

1. Simulation Videos link: Click Here
2. Real Robot Videos Link: Click Here