COVERAGE PATH PLANNING

SYED MAZHAR
u1988609

Supervisor: Tamara Petrovic
Co-supervisor: Felix

A Final Year Report submitted to School of Physical and Mathematical Sciences, Nanyang Technological University in partial fulfilment of the requirements for the Degree of

Intelligent Field Robotic Systems

June 2024

1. Introduction

In recent years, the intersection of robotics and agriculture has garnered significant attention due to its potential to revolutionize farming practices and address various challenges in crop cultivation and management. One of the critical issues faced by farmers worldwide is the proliferation of weeds in agricultural fields, which not only compete with crops for essential resources but also pose significant threats to livestock health and food safety. In particular, weeds such as Rumex have been identified as major nuisances in grasslands, where their presence can contaminate forage intended for grazing animals, including cows.

The purity of grassland vegetation directly impacts the nutritional quality of forage consumed by livestock, thereby influencing the health and productivity of animals. The ingestion of contaminated forage, tainted with toxic or harmful weed species like Rumex, can lead to adverse health effects in cattle, affecting milk quality and quantity, as well as overall animal welfare. Therefore, the effective removal of weeds, particularly Rumex, from grasslands is imperative to ensure the integrity and safety of livestock feed, as well as the sustainability of agricultural operations.

In response to this pressing agricultural challenge posed by weed infestation in grasslands, 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. By leveraging advancements in robotics, artificial intelligence, and sensing technologies, CPP algorithms can enable autonomous agricultural robots to navigate complex terrain, identify weed-infested areas, and perform targeted weed removal tasks with precision and efficacy.

The primary objective of this research is to develop a coverage path planning (CPP) algorithm tailored for the intelligent and systematic removal of Rumex weeds from grasslands, thereby promoting the production of high-quality forage and safeguarding the health of grazing animals. This algorithm aims to optimize coverage efficiency, ensuring that all targeted weeds are effectively identified and removed by the robotic system. Through the integration of advanced robotics algorithms, the proposed approach seeks to enhance the efficacy of weed management practices while minimizing reliance on chemical herbicides and manual labor.

This thesis will explore various aspects of coverage path planning (CPP) in the context of agricultural robotics, including but not limited to:

- Review of Literature: A comprehensive overview of path planning techniques, including coverage path planning (CPP) and alternatives accommodating non-holonomic constraints. It explores classical and heuristic algorithms, their optimization strategies, and computational complexities, underscoring the importance of effective path planning in agricultural robotics. Through synthesizing diverse research streams, this review informs the development of an innovative CPP algorithm for weed removal.

- Methodology: We provide an in-depth exploration of the proposed CPP algorithm designed specifically for autonomous weed removal operations. We elucidate the underlying design principles governing the algorithm's development, emphasizing its adaptability to varying terrain and distinct dataset. Furthermore, we outline the computational framework, detailing the algorithm's implementation and optimization strategies to ensure efficient coverage and weed removal. Additionally, we discuss sensor integration strategies, elucidating how sensor data is utilized for perception and decision-making during weed removal tasks. Finally, we delve into the decision-making processes guiding the robotic system's actions, encompassing strategies for global and local path planning and obstacle avoidance to maximize efficiency and effectiveness in weed management operations.

- Experimental Setup: This section outlines the key considerations for field deployment and validation of the CPP algorithm in real-world grassland environments. It discusses the selection of robotic hardware, focusing on factors such as mobility, durability, and compatibility with the intended application. Additionally, it elaborates on the sensor configurations essential for environmental perception and obstacle detection during weed removal operations. The section also addresses field testing protocols, detailing the procedures for data collection, performance evaluation, and validation of the CPP algorithm's efficacy under actual operating conditions.

- Results and Analysis: This section presents a comprehensive analysis of experimental findings obtained through both simulation and real-world field trials, encompassing various parameters crucial for field robotics. It includes quantitative assessments of weed removal efficiency, coverage completeness, Energy usage, and to name a few. Furthermore, it compares the performance of the proposed CPP algorithm with other state-of-the-art approaches, considering factors such as path optimality, computational efficiency, and adaptability to different environments. Through a rigorous evaluation framework, this analysis provides valuable insights into the effectiveness and applicability of the CPP algorithm in practical agricultural settings.

- Discussion and Implications: This section engages in a thorough analysis of the research findings in the context of existing literature, elucidating the alignment and disparities between the proposed CPP algorithm and previous studies. It identifies the strengths and limitations of the algorithm, considering factors such as computational efficiency, scalability, and adaptability to diverse agricultural environments. Furthermore, it explores potential applications of the CPP algorithm beyond weed removal, highlighting its broader implications for sustainable agriculture and livestock management practices. By synthesizing empirical evidence with theoretical insights, this discussion offers valuable perspectives on the future trajectory of autonomous robotics in agriculture.

By addressing these key components, this thesis aims to contribute to the advancement of autonomous weed management technologies in the agricultural sector, with a specific focus on improving the health and productivity of grassland ecosystems and ensuring the safety and quality of livestock feed. Through interdisciplinary collaboration and innovative engineering solutions, the integration of CPP algorithms into agricultural robotics holds promise for mitigating weed-related challenges and fostering sustainable farming practices for the benefit of farmers, consumers, and the environment alike.

By meticulously addressing these critical components, this thesis endeavors to significantly propel the evolution of autonomous weed management technologies within the agricultural domain. With a targeted emphasis on enhancing the health and productivity of grassland ecosystems, as well as safeguarding the safety and quality of livestock feed, the research aims to effect tangible improvements across multiple facets of agricultural sustainability.

Through a concerted effort to foster interdisciplinary collaboration and devise innovative engineering solutions, the integration of Coverage Path Planning (CPP) algorithms into agricultural robotics emerges as a beacon of hope for mitigating the myriad challenges posed by weed infestation. This holistic approach not only promises to alleviate immediate concerns for farmers but also extends its benefits to end consumers and the environment at large.

By empowering farmers with efficient and effective weed management tools, the research seeks to enhance agricultural productivity while reducing reliance on conventional, often environmentally detrimental, weed control methods. Moreover, by promoting sustainable farming practices, the integration of CPP algorithms holds the potential to foster long-term environmental stewardship and contribute to the preservation of natural ecosystems.

Ultimately, the overarching goal of this thesis is to usher in a new era of agricultural innovation—one characterized by the harmonious coexistence of technological advancement and environmental conservation. By harnessing the power of robotics and leveraging the principles of sustainable agriculture, the research endeavors to create a brighter future for farmers, consumers, and the planet alike.

need to include the algorithm of the paper as did in the first review paper

2. Problem Statement

We consider a robotic scenario wherein a four-wheel robot equipped with a mechanical CNC-inspired system for extraction operates within a defined area. The robot, characterized as non-holonomic, integrates the mechanical system beneath it, enabling movement along the x and y directions by approximately 60 cm, and vertically until ground contact. 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 (120, 90) 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, utilizing a deep learning model to identify weed locations. This dataset serves as the basis for complete coverage path planning.

Given the robot's mechanical implementation width of 60 cm, the operational region is discretized into points with each point representing an area of 30 cm, facilitating path planning optimization. The robot's velocity is constrained to 0.8 m/s on straight paths and 0.4 m/s on curved paths.

The primary objective of this research 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 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.
- **Field Operation Efficiency:** Facilitate efficient field operations, reducing time spent on weed detection and removal processes.

The algorithm should prioritize finding the shortest path distance to cover all weed points effectively while meeting the aforementioned objectives. The algorithm's performance will be evaluated based on the time taken to generate paths, the distance covered, and the energy consumed during path execution.

3. Methodology

3..1 Brief Description

Start with the algorithm, then you can modify it later.

3..2 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 equipped with a deep learning model capable of identifying and pinpointing the locations of weeds. The drone performs a systematic survey of the field, capturing high-resolution images and employing the deep learning model offline 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 deep learning model, 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 deep learning model'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. Future implementations incorporating the deep learning model will necessitate the adjustment of weed regions based on the uncertainty associated with each data point. 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: Given the robot's extraction width of 60 cm, the field is divided into fixed regions. 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.

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 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 detailed 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.

I can also discuss two clustering techniques, first one implemented for amny clusters and the second one is the above one. Also write in detail about the procedure followed for the above mentioned steps.

3..3 Algorithm Description

Data Preprocessing

Region Definition and Overlap Management

Given the robot's extraction width of 60 cm, the field is divided into fixed regions. 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.

Let:

- $\mathbf{R}_i$ represent the region associated with data point $\mathbf{p}_i$.
- $\mathbf{C}$ be the set of centroids representing consolidated weed clusters.

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.

Let:

- $\mathbf{O}_{ij}$ denote the overlap region between regions $\mathbf{R}_i$ and $\mathbf{R}_j$.
- $\mathbf{C}_k$ be the centroid representing the consolidated cluster of overlapping regions.

The centroid $\mathbf{C}_k$ can be calculated as the intersection of the overlapping regions:

$$\mathbf{C}_k = \bigcap \mathbf{O}_{ij}$$

Vision Cone Strategy

To effectively prioritize straight paths while minimizing computational complexity and time, we employ a vision cone mechanism for the robot. This vision cone is mathematically defined by two lines extending from the robot at a fixed angle and distance range.

Let:

- θ be the angle of the vision cone.
- d_{min} be the minimum distance of the vision cone.
- d_{max} be the maximum distance of the vision cone.
- $\mathbf{p}_r$ be the current position of the robot.
- $\mathbf{h}_r$ be the heading direction of the robot.

The vision cone angle θ is determined by the robot's minimum turning radius R. 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:

$$\theta = \pm 11^\circ$$

The distance range of the vision cone is defined by d_{min} and d_{max}, where d_{min} ensures that points too close to the robot are ignored, and d_{max} sets the farthest point considered. For this scenario, the values are:

$$d_{\text{min}} = 2 \text{ meters}$$

$$d_{\text{max}} = 100 \text{ meters}$$

The vision cone allows the robot to consider only those points $\mathbf{p}_i$ within this cone from its current position $\mathbf{p}_r$ as potential next travel points. To determine if a point $\mathbf{p}_i$ is within the vision cone, we use the following criteria:

The Euclidean distance between $\mathbf{p}_r$ and $\mathbf{p}_i$ should be within the range $[d_{\text{min}}, d_{\text{max}}]$:

$$d_{\text{min}} \leq \|\mathbf{p}_i - \mathbf{p}_r\| \leq d_{\text{max}}$$

The angle φ between the line connecting $\mathbf{p}_r$ to $\mathbf{p}_i$ and the robot's current heading $\mathbf{h}_r$ should be less than or equal to θ:

$$\varphi \leq \theta$$

Where the angle φ can be computed using the dot product:

$$\varphi = \cos^{-1}\left(\frac{(\mathbf{p}_i - \mathbf{p}_r) \cdot \mathbf{h}_r}{\|\mathbf{p}_i - \mathbf{p}_r\| \|\mathbf{h}_r\|}\right)$$

By narrowing the focus to relevant points within the vision cone, computational efficiency is enhanced. This selective consideration significantly reduces the computational effort required to determine the path, as it disregards points outside the cone. The time complexity of the vision cone strategy for considering all points within the cone from the robot's current position is $O(n)$. Thus, the vision cone mechanism proves to be an intelligent and effective strategy for the robot's navigation.

3..4 Problem formlation and proof

In this section, we aim to address a coverage path planning problem by decomposing it into three behavioral approach subproblems. Each subproblem will be tackled by a dedicated behavioral approach, collectively providing a comprehensive solution to the coverage path planning challenge.

Subproblems Overview

Problem 1: Given a set of points, the objective is to determine the path that covers the maximum number of points and selects the optimal point for turning back towards the centroid to continue the traversal loop.

Consider a set of points $\mathbf{P} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$ in a 2D plane, along with the initial robot position $\mathbf{P}_r$ and orientation $\mathbf{O}_r$. Initially, the algorithm will sample $\mathbf{m}$ temporary orientations $\mathbf{O}_1, \mathbf{O}_2, \ldots, \mathbf{O}_m$. For each orientation $\mathbf{O}_i$, the trajectory $\mathbf{T}_1, \mathbf{T}_2, \ldots, \mathbf{T}_m$ will be computed, and the trajectory $\mathbf{T}^*$ with maximum point coverage will be selected. Following traversal along trajectory $\mathbf{T}^*$, the robot will determine the optimal point to turn back towards the centroid to continue the traversal loop.

theorem 3..1. *Selection of Optimal Point for Maximum Point Coverage*
Given the distance of the points within the vision cone and the distribution of points on either side of the robot's orientation, it is guaranteed that the best combined score will ensure the maximum coverage of points in the entire trajectory.

Proof: Point Visibility Calculation:
Upon selecting a sample orientation $\mathbf{O}_i$ in the initial sampling iteration, the robot, positioned at its current location and orientation, will calculate all visible points within the vision cone. Let $\mathbf{V} = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k\}$ denote the set of visible points.

Point Selection Process:
The robot will then evaluate the visibility of points based on their Euclidean distance from the robot's position, aiming to select points in close proximity to the robot to maximize point coverage in the trajectory. Let $\mathbf{D} = \{d_1, d_2, \ldots, d_k\}$ represent the distances of visible points from the robot's position. Let $\mathbf{D}' = \left\{\frac{\max(\mathbf{D})}{d_1}, \frac{\max(\mathbf{D})}{d_2}, \ldots, \frac{\max(\mathbf{D})}{d_k}\right\}$ denote the normalized distance values.

Distribution of Points:
Concurrently, the robot will analyze the distribution of points on either side of its orientation, contributing to the selection of points that ensure comprehensive coverage across the trajectory. Let $\mathbf{Di} = \{\mathbf{Di_L}, \mathbf{Di_R}\}$ denote the distribution of points on either side, both the values are

normalized to the same scale for a fair comparison. All the points on the left side of the robot's orientation get the value $\mathbf{Di_L}$ and all the points on the right side of the robot's orientation get the value $\mathbf{Di_R}$.

Combination:
To integrate both the distance of points within the vision cone and the distribution of points relative to the robot's orientation, normalized distance and distribution are summed up for all the visible points.
The combined metric $\mathbf{C}' = \mathbf{D}' + \mathbf{Di}'$ is computed.

Optimal Point Selection:
The optimal point $\mathbf{p}^*$ is selected based on the combined metric $\mathbf{C}'$, ensuring maximum coverage of points in the trajectory.

Therefore, this theorem guarantees that the best point selected by the robot ensures maximum coverage of points in the entire trajectory.

Lemma 3..1. *Given the current robot position $\mathbf{p}_r$ and the next best point $\mathbf{p}_{next}$ with a line segment $\mathbf{L}$ connecting them, the intermediate points $\{\mathbf{p}_{int1}, \mathbf{p}_{int2}, \dots, \mathbf{p}_{intk}\}$ respect the non-holonomic constraints and straight path prioritization of the robot if they satisfy the following conditions:*

1. The perpendicular distance $d_{i,L}$ from each intermediate point $\mathbf{p}_{inti}$ to the line $\mathbf{L}$ is less than half of the minimum turning radius $R_{\min}$ of the robot:

$$d_{i,L} < \frac{R_{\min}}{2} \quad \forall i \in \{1, 2, \dots, k\}$$

2. The Euclidean distance $d_{i,j}$ between any two intermediate points $\mathbf{p}_{inti}$ and $\mathbf{p}_{intj}$ and between intermediate points and endpoints of the line segment $\mathbf{p}_r$ and $\mathbf{p}_{next}$ is greater than the minimum turning radius $R_{\min}$:

$$d_{i,j} > R_{\min} \quad \forall i \neq j \quad \textit{and} \quad d_{i,\mathbf{p}_r}, d_{i,\mathbf{p}_{next}} > R_{\min} \quad \forall i \in \{1, 2, \dots, k\}$$

Proof: To ensure the intermediate points respect the non-holonomic constraints and the straight path prioritization of the robot, we must validate the following conditions:

1. **Straight Path Prioritization:** The intermediate points should lie close to the line segment $\mathbf{L}$ connecting $\mathbf{p}_r$ and $\mathbf{p}_{\text{next}}$. This is ensured if the perpendicular distance $d_{i,L}$ from each intermediate point $\mathbf{p}_{\text{int}i}$ to the line $\mathbf{L}$ is less than half of the minimum turning radius $R_{\min}$:

$$d_{i,L} < \frac{R_{\min}}{2} \quad \forall i \in \{1, 2, \dots, k\}$$

If the perpendicular distance $d_{i,L}$ exceeds $\frac{R_{\min}}{2}$, the intermediate point $\mathbf{p}_{\text{int}i}$ deviates significantly from a straight line, thus violating the straight path prioritization.

2. **Non-Holonomic Constraints:** The robot's non-holonomic constraints dictate that the turning radius $R_{\min}$ must be respected to prevent sharp turns or loops. This requires the Euclidean distance $d_{i,j}$ between any two intermediate points $\mathbf{p}_{\text{int}i}$ and $\mathbf{p}_{\text{int}j}$ and the distance between intermediate points and the endpoints of the line segment $\mathbf{p}_r$ and $\mathbf{p}_{\text{next}}$ to be greater than the minimum turning radius $R_{\min}$:

$$d_{i,j} > R_{\min} \quad \forall i \neq j \quad \text{and} \quad d_{i,\mathbf{p}_r}, d_{i,\mathbf{p}_{\text{next}}} > R_{\min} \quad \forall i \in \{1,2,\ldots,k\}$$

If the distance $d_{i,j}$ is less than $R_{\min}$, the intermediate points would necessitate a turning radius smaller than $R_{\min}$, thereby violating the non-holonomic constraints of the robot.

Therefore, the intermediate points that satisfy these conditions will ensure that both the straight path prioritization and the non-holonomic constraints of the robot are respected.

Hence, this lemma proves that if the intermediate points are selected according to the expressions above, the non-holonomic constraints and straight path prioritization of the robot will be maintained.

Lemma 3..2. *Given the best trajectory* $\mathbf{T}^*$ *and its endpoint* $\mathbf{p}_{end}$*, the optimal next point* $\mathbf{p}_{opt}$ *for turning back towards the uncovered points lies within the concurrent region defined by radii* $R_{\min}$ *and* $R_{\max}$*, and has the minimum angle difference with the current orientation* θ_{cur}*. The optimal orientation at* $\mathbf{p}_{opt}$ *is directed towards the centroid of all the points, ensuring maximum coverage in the next trajectory.*

Proof:

Given the endpoint $\mathbf{p}_{\text{end}}$ of the best trajectory $\mathbf{T}^*$, the objective is to compute the next best point $\mathbf{p}_{\text{opt}}$ to initiate a turn that maximizes the coverage of points in the subsequent trajectory.

Concurrent Region Constraint: The robot has a minimum turning radius $R_{\min}$, implying that it can only turn outside of a circular region of radius $R_{\min}$ centered at $\mathbf{p}_{\text{end}}$. To select the next best point to turn, the point should be between $R_{\min}$ and $R_{\max}$. Here, $R_{\max}$ is chosen to limit the search area, ensuring that the robot doesn't make unnecessarily large turns, which would result in longer paths and potential inefficiency. Therefore, the next point $\mathbf{p}_{\text{opt}}$ must lie within a concurrent region defined by the minimum radius $R_{\min}$ and a maximum radius $R_{\max}$:

$$R_{\min} \leq \|\mathbf{p}_{\text{opt}} - \mathbf{p}_{\text{end}}\| \leq R_{\max}$$

where $\|\cdot\|$ denotes the Euclidean distance.

Angle Difference Minimization: To ensure a smooth turn, the angle difference between the robot's current orientation θ_{cur} and the vector $\overrightarrow{\mathbf{p}_{\text{end}}\mathbf{p}_{\text{opt}}}$ should be considered. This angle difference is computed for all points within the concurrent region. The point with the smallest angle difference is chosen as $\mathbf{p}_{\text{opt}}$, as this point is likely to be near the edge or on the boundary of the set of points, ensuring more points are covered in the next trajectory. The angle difference can be expressed as:

$$\theta_{\text{opt}} = \arg \min_{\mathbf{p} \in \mathbf{P}_{\text{concurrent}}} |\theta - \theta_{\text{cur}}|$$

where θ is the angle between the line segment $\overrightarrow{\mathbf{p}_{\text{end}}\mathbf{p}_{\text{opt}}}$ and the positive x-axis, and $\mathbf{P}_{\text{concurrent}}$ is the set of points within the concurrent region. This selection helps ensure that the next point $\mathbf{p}_{\text{opt}}$ is positioned to maximize coverage in the following trajectory.

Optimal Orientation Towards Centroid: To maximize the coverage of points in the next trajectory, the optimal orientation at $\mathbf{p}_{\text{opt}}$ should be directed towards the centroid $\mathbf{C}$ of all the points $\mathbf{P}_{\text{rem}} = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$. The centroid $\mathbf{C}$ is computed as:

$$\mathbf{C} = \left(\frac{1}{n} \sum_{i=1}^{n} x_i, \frac{1}{n} \sum_{i=1}^{n} y_i \right)$$

where $\mathbf{p}_i = (x_i, y_i)$ are the coordinates of the remaining points.

Thus, the optimal point $\mathbf{p}_{\text{opt}}$ is chosen based on the following criteria:

$$\mathbf{p}_{\text{opt}} = \arg \min_{\mathbf{p} \in \mathbf{P}_{\text{concurrent}}} \{|\theta_{\text{opt}} - \theta_{\text{cur}}| \mid R_{\min} \leq \|\mathbf{p} - \mathbf{p}_{\text{end}}\| \leq R_{\max}\}$$

and the optimal orientation at $\mathbf{p}_{\text{opt}}$ is towards the centroid $\mathbf{C}$.

Hence, this lemma proves that selecting $\mathbf{p}_{\text{opt}}$ with the conditions above ensures the robot's non-holonomic constraints are respected and maximum coverage of points in the subsequent trajectory is achieved.

Problem2: Given a set of points, find the path that covers maximum number of points and select the next best point to circumnavigate the centroid and continue the loop.

Problem3: Given a set of points, find the dubins traveling salesman path that covers rest of the points with optimal path.

3..5 Algorithm Description

Behavioral Approach for Coverage Path Planning in Agricultural Fields

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.

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.

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 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. **Initial 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. **Intermediate 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. **Final Behavior:** 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, 100% 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 vision cone mechanism plays a pivotal role in this process, enhancing computational efficiency and enabling intelligent path selection.

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.

3..5.1 First Behavior: Initial Coverage (change its name)

Algorithm 1 CompleteBehavioralAlgorithm

Input: 2D points, initial robot pose, turning radius
Output: Dubins path

```
1:  $clustered\_points, behavior\_change\_perc \leftarrow CentroidsAndAutoBehaviorShift(2Dpoints)$
2:  $remaining\_points \leftarrow clustered\_points$
3:  $covered\_perc \leftarrow 0$
4:  $completed\_path \leftarrow []$
5:  while True do
6:      if $covered\_perc < behavior\_change\_perc$ then
7:          $straight\_path, remaining\_points \leftarrow Behavior\_1(remaining\_points, number\_of\_sample-$
              $-orientations(Ns), robot\_pose, vision\_cone, centroid, concurrent\_region\_radii, step)$
8:          $completed\_path += straight\_path$
9:          $robot\_pose \leftarrow completed\_path[-1]$
10:         $covered\_perc \leftarrow UpdateCoveragePerc(clustered\_points, remaining\_points)$
11:     else
12:         $straight\_path, remaining\_points \leftarrow Behavior\_2(remaining\_points, number\_of\_sample-$
              $-orientations(Ns), robot\_pose, vision\_cone, centroid, concurrent\_region\_radii, step)$
13:         $completed\_path += straight\_path$
14:         $robot\_pose \leftarrow completed\_path[-1]$
15:         $covered\_perc \leftarrow UpdateCoveragePerc(completed\_path, remaining\_points)$
16:     end if
17:     if $len(straight\_path) == 0$ then
18:         break
19:     end if
20: end while
21: $dubins\_path \leftarrow DubinsPath(completed\_path, turning\_radius)$
22: $dotsp\_path \leftarrow DOTSPPath(remaining\_points, turning\_radius)$
23: $complete\_dubins\_path \leftarrow dubins\_path + dotsp\_path$
24: return $complete\_dubins\_path$
```

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.

Description of the notations:

- 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 2 CompleteBehavioralAlgorithm

```
Require: 2D points ($p_{2d}$), initial robot pose ($r_{\mathrm{pos}}$), turning radius ($R_{\mathrm{tu}}$)
Ensure: Dubins path $P_{cd}$
 1: $p_{cl}, \delta_{\mathrm{bc}} \leftarrow$ CentroidsAndAutoBehaviorShift($p_{2d}$)
 2: $p_r \leftarrow p_{cl}$
 3: $\delta_c \leftarrow 0$
 4: $P_c \leftarrow []$
 5: while True do
 6:     if $\delta_c < \delta_{\mathrm{bc}}$ then
 7:         $P_s, p_r \leftarrow$ Behavior_1($p_r$, $N_s$, $r_{\mathrm{pos}}$, $VC$, $C$, $R_{\mathrm{conc}}$, $step$)
 8:     else
 9:         $P_s, p_r \leftarrow$ Behavior_2($p_r$, $N_s$, $r_{\mathrm{pos}}$, $VC$, $C$, $R_{\mathrm{conc}}$, $step$)
10:     end if
11:     $P_c \mathrel{+}= P_s$
12:     $r_{\mathrm{pos}} \leftarrow P_c[-1]$
13:     $\delta_c \leftarrow$ UpdateCoveragePerc($P_c$, $p_r$)
14:     if len($P_s$) $== 0$ then
15:         break
16:     end if
17: end while
18: $P_d \leftarrow$ DubinsPath($P_c$, $R_{\mathrm{tu}}$)
19: $P_{dotsp} \leftarrow$ DOTSPPath($p_r$, $R_{\mathrm{tu}}$)
20: $P_{cd} \leftarrow P_d + P_{dotsp}$
21: return $P_{cd}$
```

Algorithm 3 AutoBehaviorShift

```
    Input: 2D points, centroid, min_radius, max_radius, number of concurrent circles ($N$).
    Output: behavior_change_percent

 1: $total\_perc \leftarrow 0$
 2: $threshold\_perc \leftarrow 50$
 3: $percent\_list \leftarrow$ GeneratePercentageList$(30, 80, N)$    ▷ List from 30 to 80% with N steps.
 4: $concentric\_circles \leftarrow$ ConcentricCircles$(points, min\_radius, max\_radius, N)$
 5: for $i, curr\_radius \in concentric\_circles$ do
 6:     $curr\_perc\_points \leftarrow$ PointsPercentageInSubregion$(curr\_radius)$
 7:     $total\_perc \leftarrow total\_perc + curr\_perc\_points$
 8:     if $total\_perc > threshold\_perc$ then
 9:         $behavior\_change\_percent \leftarrow percent\_list[i]$
10:         break
11:     end if
12: end for
13: return $behavior\_change\_percent$
```

Description of the notations:

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

Algorithm 4 AutoBehaviorShift

```
    Input: $P_{2d}$, $R_{\min}$, $R_{\max}$, $N$
    Output: $\delta_{bc}$

 1: $\delta_{to} \leftarrow 0$
 2: $\delta_{thp} \leftarrow 50$
 3: $\delta_{list} \leftarrow$ GeneratePercentageList$(30, 80, N)$    ▷ List from 30 to 80% with N steps.
 4: $cc \leftarrow$ ConcentricCircles$(P_{2d}, R_{\min}, R_{\max}, N)$
 5: for $i, R_{cu}$ in $cc$ do
 6:     $p_{cu_\delta} \leftarrow$ PointsPercentageInSubregion$(R_{cu})$
 7:     $\delta_{to} \leftarrow \delta_{to} + p_{cu_\delta}$
 8:     if $\delta_t o > \delta_{thp}$ then
 9:         $\delta_{bc} \leftarrow \delta_{list}[i]$
10:         break
11:     end if
12: end for
13: return $\delta_{bc}$
```

Algorithm 5 Behavioral1

```
Input: 2D points, number of sample orientations $Ns$, robot pose, vision cone, centroid, concurrent region radii, step.
Output: Complete path, remaining points.

1: $complete\_path \leftarrow []$
2: $temporary\_path \leftarrow [[],[],[],...,Ns]$
3: $points\_covered \leftarrow [[],[],[],...,Ns]$
4: $sample\_orientations \leftarrow SampleTheOrientations(robot\_pose[2], Ns, step)$
5: for $i, orientation$ in $sample\_orientations$ do
6:     $robot\_pose[2] \leftarrow orientation$
7:     while no point is visible do
8:         $visible\_points \leftarrow ComputeVisionConePoints(robot\_pose, vision\_cone)$
9:         $potential\_point \leftarrow FindPotentialPoint(robot\_pose, visible\_points)$
10:        if potential_point is None then
11:            break
12:        end if
13:        $new\_orientation \leftarrow FromCurrentPoseToPotentialPoint(robot\_pose, potential\_point)$
14:        $temporary\_path[i].append([potential\_point, new\_orientation])$
15:        $Intermediate\_points \leftarrow CheckIntermediatePoints()$
16:        $temporary\_path[i].append(Intermediate\_points)$
17:        $points\_covered[i].append(len(temporary\_path))$
18:        $remaining\_points \leftarrow all\_points - temporary\_path$
19:        $robot\_pose \leftarrow [potential\_point, new\_orientation]$
20:    end while
21: end for
22: $best\_orientation\_index \leftarrow \text{argmax}(points\_covered[:])$
23: $complete\_path \leftarrow temporary\_path[best\_orientation\_index]$
24: $robot\_pose \leftarrow complete\_path[-1]$
25: $turn\_point \leftarrow PotentialPointToTurn(remaining\_points, concurrent\_region\_radii, robot\_pose)$
26: $best\_orientation \leftarrow TowardsCentroid(turn\_point, centroid)$
27: $complete\_path.append([turn\_point, best\_orientation])$
28: $remaining\_points.remove([turn\_point])$
29: return $complete\_path, remaining\_points$
```

Description of the notations:

- P_{cs} : $Completestraightpath$
- P_t : $Temporarypath$
- p_{co} : $Pointscovered$
- O_{sa} : $Sampledorientations$
- p_v : $Visiblepoints$
- p_{po} : $Potentialpoint$
- O_n : $Neworientation$
- p_{int} : $Intermediatepoints$
- O_{bi} : $Bestorientationindex$
- P_{tu} : $Turnpoint$
- O_b : $Bestorientation$

Algorithm 6 Behavioral1

```
Require: Set of 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 ($R_{\text{conc}}$), step size ($S$)
Ensure: Complete straight path $P_{cs}$, remaining points $p_r$
 1: $P_{cs} \leftarrow []$
 2: $P_t \leftarrow [[] \text{ for \_ in range}(N_s)]$
 3: $p_{co} \leftarrow [[] \text{ for \_ in range}(N_s)]$
 4: $O_{sa} \leftarrow$ SampleTheOrientations($R_{pos}[2]$, $N_s$, $S$)
 5: for $i, O$ in $O_{sa}$ do
 6:     $R_{pos}[2] \leftarrow O$
 7:     while no point is visible do
 8:         $p_v \leftarrow$ ComputeVisionConePoints($r_{pos}$, $VC$)
 9:         $p_{po} \leftarrow$ FindPotentialPoint($r_{pos}$, $p_v$)
10:         if $p_{po}$ is None then
11:             break
12:         end if
13:         $O_n \leftarrow$ FromCurrentPoseToPotentialPoint($r_{pos}$, $p_{po}$)
14:         $P_t[i]$.append($[p_{po}, O_n]$)
15:         $p_{int} \leftarrow$ CheckIntermediatePoints($p_r$)
16:         $P_t[i]$.append($p_{int}$)
17:         $p_c[i]$.append(len($P_t$))
18:         $p_r \leftarrow p_{cl} - P_t$
19:         $r_{pos} \leftarrow [p_{po}, O_n]$
20:     end while
21: end for
22: $O_{bi} \leftarrow$ argmax($p_c[:]$)
23: $P_{cs} \leftarrow P_t[O_{bi}]$
24: $r_{pos} \leftarrow P_{cs}[-1]$
25: $p_{tu} \leftarrow$ PotentialPointToTurn($p_r$, $R_{\text{conc}}$, $r_{pos}$)
26: $O_b \leftarrow$ TowardsCentroid($p_{tu}$, $C$)
27: $P_{cs}$.append($[p_{tu}, O_b]$)
28: $p_r$.remove($[p_{tu}]$)
29: return $P_{cs}$, $p_r$
```

Algorithm 7 Behavioral2

Input: Set of 2D points, number of sample orientations Ns, robot pose, vision cone, centroid, concurrent region radii, step
Output: Complete path, remaining points

```
1: $complete\_path \leftarrow []$
2: $temporary\_path \leftarrow [[],[],[],...,Ns]$
3: $points\_covered \leftarrow [[],[],[],...,Ns]$
4: $sample\_orientations \leftarrow SampleTheOrientations(robot\_pose[2], Ns, step)$
5: for $i, orientation$ in $sample\_orientations$ do
6:     $robot\_pose[2] \leftarrow orientation$
7:     while no point is visible do
8:         $visible\_points \leftarrow ComputeVisionConePoints(robot\_pose, vision\_cone)$
9:         $potential\_point \leftarrow FindPotentialPoint(robot\_pose, visible\_points)$
10:        if potential_point is None then
11:            break
12:        end if
13:        $new\_orientation \leftarrow FromCurrentPoseToPotentialPoint(robot\_pose, potential\_point)$
14:        $temporary\_path[i].append([potential\_point, new\_orientation])$
15:        $Intermediate\_points \leftarrow CheckIntermediatePoints()$
16:        $temporary\_path[i].append(Intermediate\_points)$
17:        $points\_covered[i].append(len(temporary\_path))$
18:        $remaining\_points \leftarrow all\_points - temporary\_path$
19:        $robot\_pose \leftarrow [potential\_point, new\_orientation]$
20:    end while
21: end for
22: $best\_orientation\_index \leftarrow \text{argmax}(points\_covered[:])$
23: $complete\_path \leftarrow temporary\_path[best\_orientation\_index]$
24: $turn\_point \leftarrow PotentialPointToTurnCCW(remaining\_points, concurrent\_region\_radii, robot\_pose)$
25: $best\_orientation \leftarrow VectorsTowardsNextCCWPoint(turn\_point, remaining\_points)$
26: $complete\_path.append([turn\_point, best\_orientation])$
27: return $complete\_path, remaining\_points$
```

Algorithm 8 Behavioral2

```
Require: Set of 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 ($R_{\text{conc}}$), step size ($S$)
Ensure: Complete straight path $P_{cs}$, remaining points $p_r$
 1: $P_{cs} \leftarrow []$
 2: $P_t \leftarrow [[]$ for _ in range$(N_s)]$
 3: $p_{co} \leftarrow [[]$ for _ in range$(N_s)]$
 4: $O_{sa} \leftarrow$ SampleTheOrientations($R_{pos}[2]$, $N_s$, $S$)
 5: for $i, O$ in $O_{sa}$ do
 6:     $R_{pos}[2] \leftarrow O$
 7:     while no point is visible do
 8:         $p_v \leftarrow$ ComputeVisionConePoints($r_{pos}$, $VC$)
 9:         $p_{po} \leftarrow$ FindPotentialPoint($r_{pos}$, $p_v$)
10:         if $p_{po}$ is None then
11:             break
12:         end if
13:         $O_n \leftarrow$ FromCurrentPoseToPotentialPoint($r_{pos}$, $p_{po}$)
14:         $P_t[i]$.append$([p_{po}, O_n])$
15:         $p_{int} \leftarrow$ CheckIntermediatePoints($p_r$)
16:         $P_t[i]$.append$(p_{int})$
17:         $p_c[i]$.append(len$(P_t)$)
18:         $p_r \leftarrow p_{cl} - P_t$
19:         $r_{pos} \leftarrow [p_{po}, O_n]$
20:     end while
21: end for
22: $O_{bi} \leftarrow$ argmax$(p_c[:])$
23: $P_{cs} \leftarrow P_t[O_{bi}]$
24: $r_{pos} \leftarrow P_{cs}[-1]$
25: $p_{tu} \leftarrow$ PotentialPointToTurnCCW($p_r$, $R_{\text{conc}}$, $r_{pos}$)
26: $O_b \leftarrow$ VectorsTowardsNextCCWPoint($p_{tu}$, $p_r$)
27: $P_{cs}$.append$([p_{tu}, O_b])$
28: $p_r$.remove$([p_{tu}])$
29: return $P_{cs}$, $p_r$
```

Algorithm 9 DOTSPPath

```
    Input: Set of 2D points, initial robot position, turning radius
    Output: Dubins path

1: $start\_node \leftarrow initial\_robot\_position$
2: $Graph \leftarrow GenerateGraph(2D\_points)$
3: $Closed\_path \leftarrow SolveClosedTSP(Graph, start\_node)$
4: $open\_path \leftarrow RemoveEdgeWithMaxWeightFromStartNode(Graph, start\_node)$
5: $dubins\_path \leftarrow DubinsPath(open\_path, turning\_radius)$
6: return $dubins\_path$
```

Description of the notations:

- sn: Start node
- G: Graph
- P_{clo}: Closed path
- P_{op}: Open path
- P_d: Dubins path

Algorithm 10 DOTSPPath

```
Require: Set of 2D points $P_{2d}$, initial robot position $r_{\text{pos}}$, turning radius $R_{\text{tu}}$
Ensure: Dubins path $P_d$
1: $sn \leftarrow r_{\text{pos}}$
2: $G \leftarrow$ GenerateGraph($P_{2d}$)
3: $P_{clo} \leftarrow$ SolveClosedTSP($G$, $sn$)
4: $P_{op} \leftarrow$ RemoveEdgeWithMaxWeightFromStartNode($G$, $sn$)
5: $P_d \leftarrow$ DubinsPath($P_{op}$, $R_{\text{tu}}$)
6: return $P_d$
```

Convention to be followed for the algorithms:

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

Algorithm 11 CompleteBehavioralObstacleAvoidance

```
Input: 2D points, initial robot pose, turning radius, polygonal obstacles, grid diameter, vision cone
Output: Dubins path

1: $extended\_obstacles, grids, buffer\_points \leftarrow SetupObstacles(polygonal\_obstacles,$
       $grid\_diameter, vision\_cone)$
2: $extended\_obstacles, grids, buffer\_points \leftarrow SetupObstacles(polygonal\_obstacles,$
       $grid\_diameter, vision\_cone)$
3: $clustered\_points, behavior\_change\_percentage \leftarrow CentroidsAndAutoBehaviorShift(2Dpoints)$
4: $remaining\_points \leftarrow clustered\_points$
5: $covered\_percentage \leftarrow 0$
6: $completed\_path \leftarrow []$
7: while True do
8:     if $covered\_percentage < behavior\_change\_percentage$ then
9:         $straight\_path, remaining\_points \leftarrow Behavioral1(extended\_obstacles, grids, buffer\_points,$
       $remaining\_points, Ns, robot\_pose, vision\_cone, centroid, concurrent\_region\_radii, step)$
10:        $completed\_path += straight\_path$
11:        $robot\_pose \leftarrow completed\_path[-1]$
12:        $covered\_percentage \leftarrow UpdateCoveragePercentage(clustered\_points, remaining\_points)$
13:    else
14:        $straight\_path, remaining\_points \leftarrow Behavioral2(extended\_obstacles, grids, buffer\_points,$
       $remaining\_points, Ns, robot\_pose, vision\_cone, centroid, concurrent\_region\_radii, step)$
15:        $completed\_path += straight\_path$
16:        $robot\_pose \leftarrow completed\_path[-1]$
17:        $covered\_percentage \leftarrow UpdateCoveragePercentage(completed\_path, remaining\_points)$
18:    end if
19:    if len$(straight\_path) == 0$ then
20:        break
21:    end if
22: end while
23: $dubins\_path \leftarrow DubinsPath(completed\_path, turning\_radius)$
24: $remaining\_path \leftarrow PathAroundObstaclesAlgorithm(remaining\_points, turning\_radius)$
25: $complete\_dubins\_path \leftarrow dubins\_path + remaining\_path$
26: return $complete\_dubins\_path$
```

Algorithm 12 CompleteBehavioralAlgorithm

```
Require: 2D points ($p_{2d}$), initial robot pose ($r_{\text{pos}}$), turning radius ($R_{\text{tu}}$), polygonal obstacles
    ($OB_p$), grid diameter ($g_d$), vision cone ($VC$)
Ensure: Dubins path $P_{cd}$
 1: $OB_e, g_s, p_{bu} \leftarrow$ SetupObstacles($OB_p$, $g_d$, $VC$)
 2: $p_{cl}, \delta_{\text{bc}} \leftarrow$ CentroidsAndAutoBehaviorShift($p_{2d}$)
 3: $p_r \leftarrow p_{cl}$
 4: $\delta_c \leftarrow 0$
 5: $P_c \leftarrow []$
 6: while True do
 7:     if $\delta_c < \delta_{\text{bc}}$ then
 8:         $P_s, p_r \leftarrow$ Behavior_1($OB_p$, $g_d$, $p_{bu}$, $p_r$, $N_s$, $r_{\text{pos}}$, $VC$, $C$, $R_{\text{conc}}$, $step$)
 9:     else
10:         $P_s, p_r \leftarrow$ Behavior_2($OB_p$, $g_d$, $p_{bu}$, $p_r$, $N_s$, $r_{\text{pos}}$, $VC$, $C$, $R_{\text{conc}}$, $step$)
11:     end if
12:     $P_c += P_s$
13:     $r_{\text{pos}} \leftarrow P_c[-1]$
14:     $\delta_c \leftarrow$ UpdateCoveragePerc($P_c$, $p_r$)
15:     if len($P_s$) $== 0$ then
16:         break
17:     end if
18: end while
19: $P_d \leftarrow$ DubinsPath($P_c$, $R_{\text{tu}}$)
20: $P_{re} \leftarrow$ PathAroundObstaclesAlgorithm($p_r$, $R_{\text{tu}}$)
21: $P_{cd} \leftarrow P_d + P_{re}$
22: return $P_{cd}$
```

Algorithm 13 Behavioral1

```
Input: Extended polygonal obstacles, grids, set of 2D points, number of sample orientations $Ns$, robot pose, vision cone, centroid, concurrent region radii, step
Output: Complete path, remaining points

1: $complete\_path \leftarrow []$
2: $temporary\_path \leftarrow [[],[],[],\dots,Ns]$
3: $points\_covered \leftarrow [[],[],[],\dots,Ns]$
4: $sample\_orientations \leftarrow SampleTheOrientations(robot\_pose[2], Ns, step)$
5: for $i, orientation \in sample\_orientations$ do
6:     $robot\_pose[2] \leftarrow orientation$
7:     while no point is visible do
8:         $visible\_points \leftarrow ComputeVisionConePoints(robot\_pose, vision\_cone)$
9:         $potential\_point \leftarrow FindPotentialPoint(robot\_pose, visible\_points)$
10:        if $potential\_point$ is None then
11:            break
12:        end if
13:        $new\_orientation \leftarrow FromCurrentPoseToPotentialPoint(robot\_pose, potential\_point)$
14:        $curr\_path \leftarrow [[robot\_pose, potential\_point]]$
15:        $obstacle\_free\_path \leftarrow ComputeObstacleFreePath(curr\_path, new\_orientation)$
16:        $temporary\_path[i] \mathrel{+}= obstacle\_free\_path$
17:        $Intermediate\_points \leftarrow CheckIntermediatePoints()$
18:        $temporary\_path[i].append(Intermediate\_points)$
19:        $points\_covered[i].append(len(temporary\_path))$
20:        $remaining\_points \leftarrow all\_points - temporary\_path$
21:        $robot\_pose \leftarrow obstacle\_free\_path[-1]$
22:    end while
23: end for
24: $best\_orientation\_index \leftarrow argmax(points\_covered[:])$
25: $complete\_path \leftarrow temporary\_path[best\_orientation\_index]$
26: $robot\_pose \leftarrow complete\_path[-1]$
27: $turn\_point \leftarrow PotentialPointToTurn(remaining\_points, concurrent\_region\_radii, robot\_pose)$
28: $best\_orientation \leftarrow TowardsCentroid(turn\_point, centroid)$
29: $curr\_path \leftarrow [[robot\_pose, turn\_point]]$
30: $obstacle\_free\_path \leftarrow ComputeObstacleFreePath(curr\_path, best\_orientation)$
31: $complete\_path \mathrel{+}= obstacle\_free\_path$
32: $remaining\_points.remove(obstacle\_free\_path)$
33: return $complete\_path, remaining\_points$
```

Description of the notations:

- P_{cu}: Current path
- P_{OBfree}: Obstacle free path

Algorithm 14 Behavioral1

```
Require: Extended polygonal obstacles ($OB_p$), grids ($g_d$), buffer points ($p_{bu}$), Set of 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 ($R_{\text{conc}}$), step size ($S$)
Ensure: Complete straight path $P_{cs}$, remaining points $p_r$
 1: $P_{cs} \leftarrow []$
 2: $P_t \leftarrow [[]$ for _ in range($N_s$)]
 3: $p_{co} \leftarrow [[]$ for _ in range($N_s$)]
 4: $O_{sa} \leftarrow$ SampleTheOrientations($R_{pos}[2]$, $N_s$, $S$)
 5: for $i, O$ in $O_{sa}$ do
 6:     $R_{pos}[2] \leftarrow O$
 7:     while no point is visible do
 8:         $p_v \leftarrow$ ComputeVisionConePoints($r_{pos}$, $VC$)
 9:         $p_{po} \leftarrow$ FindPotentialPoint($r_{pos}$, $p_v$)
10:         if $p_{po}$ is None then
11:             break
12:         end if
13:         $O_n \leftarrow$ FromCurrentPoseToPotentialPoint($r_{pos}$, $p_{po}$)
14:         $P_{cu} \leftarrow [[r_{pos}, p_{po}]]$
15:         $P_{OBfree} \leftarrow$ ComputeObstacleFreePath($P_{cu}$, $O_n$)
16:         $P_t[i] += P_{OBfree}$
17:         $p_{int} \leftarrow$ CheckIntermediatePoints($p_r$)
18:         $P_t[i]$.append($p_{int}$)
19:         $p_c[i]$.append(len($P_t$))
20:         $p_r \leftarrow p_{cl} - P_t$
21:         $r_{pos} \leftarrow P_{OBfree}[-1]$
22:     end while
23: end for
24: $O_{bi} \leftarrow$ argmax($p_c[:]$)
25: $P_{cs} \leftarrow P_t[O_{bi}]$
26: $r_{pos} \leftarrow P_{cs}[-1]$
27: $p_{tu} \leftarrow$ PotentialPointToTurn($p_r$, $R_{\text{conc}}$, $r_{pos}$)
28: $O_b \leftarrow$ TowardsCentroid($p_{tu}$, $C$)
29: $P_{cu} \leftarrow [[r_{pos}, p_{tu}]]$
30: $P_{OBfree} \leftarrow$ ComputeObstacleFreePath($P_{cu}$, $O_b$)
31: $P_{cs} += P_{OBfree}$
32: $p_r$.remove($[P_{OBfree}]$)
33: return $P_{cs}$, $p_r$
```

Algorithm 15 BehavioralAlgorithm2

```
Input: Extended polygonal obstacles, grids, set of 2D points, number of sample orientations
Ns, robot pose, vision cone, centroid, concurrent region radii, step
Output: Complete path, remaining points

1: $complete\_path \leftarrow []$
2: $temporary\_path \leftarrow [[],[],[],\dots,Ns]$
3: $points\_covered \leftarrow [[],[],[],\dots,Ns]$
4: $sample\_orientations \leftarrow SampleTheOrientations(robot\_pose[2], Ns, step)$
5: for $i, orientation \in sample\_orientations$ do
6:     $robot\_pose[2] \leftarrow orientation$
7:     while no point is visible do
8:         $visible\_points \leftarrow ComputeVisionConePoints(robot\_pose, vision\_cone)$
9:         $potential\_point \leftarrow FindPotentialPoint(robot\_pose, visible\_points)$
10:        if $potential\_point$ is None then
11:            break
12:        end if
13:        $new\_orientation \leftarrow FromCurrentPoseToPotentialPoint(robot\_pose, potential\_point)$
14:        $curr\_path \leftarrow [[robot\_pose, potential\_point]]$
15:        $obstacle\_free\_path \leftarrow ComputeObstacleFreePath(curr\_path, new\_orientation)$
16:        $temporary\_path[i] \mathrel{+}= obstacle\_free\_path$
17:        $Intermediate\_points \leftarrow CheckIntermediatePoints()$
18:        $temporary\_path[i].append(Intermediate\_points)$
19:        $points\_covered[i].append(len(temporary\_path))$
20:        $remaining\_points \leftarrow all\_points - temporary\_path$
21:        $robot\_pose \leftarrow obstacle\_free\_path[-1]$
22:    end while
23: end for
24: $best\_orientation\_index \leftarrow argmax(points\_covered[:])$
25: $complete\_path \leftarrow temporary\_path[best\_orientation\_index]$
26: $robot\_pose \leftarrow complete\_path[-1]$
27: $turn\_point \leftarrow PotentialPointToTurnCCW(remaining\_points, concurrent\_region\_radii, robot\_pose)$
28: $best\_orientation \leftarrow VectorsTowardsNextCCWPoint(turn\_point, remaining\_points)$
29: $curr\_path \leftarrow [[robot\_pose, turn\_point]]$
30: $obstacle\_free\_path \leftarrow ComputeObstacleFreePath(curr\_path, best\_orientation)$
31: $complete\_path \mathrel{+}= obstacle\_free\_path$
32: $remaining\_points.remove(obstacle\_free\_path)$
33: return $complete\_path, remaining\_points$
```

Algorithm 16 BehavioralAlgorithm2

```
Require: Extended polygonal obstacles ($OB_e$), grids ($g_d$), buffer points ($p_{bu}$), Set of 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 ($R_{\text{conc}}$), step size ($S$)
Ensure: Complete straight path $P_{cs}$, remaining points $p_r$
1: $P_{cs} \leftarrow []$
2: $P_t \leftarrow [[]$ for _ in range$(N_s)]$
3: $p_{co} \leftarrow [[]$ for _ in range$(N_s)]$
4: $O_{sa} \leftarrow$ SampleTheOrientations($R_{pos}[2]$, $N_s$, $S$)
5: for $i, O$ in $O_{sa}$ do
6:     $R_{pos}[2] \leftarrow O$
7:     while no point is visible do
8:         $p_v \leftarrow$ ComputeVisionConePoints($r_{pos}$, $VC$)
9:         $p_{po} \leftarrow$ FindPotentialPoint($r_{pos}$, $p_v$)
10:        if $p_{po}$ is None then
11:            break
12:        end if
13:        $O_n \leftarrow$ FromCurrentPoseToPotentialPoint($r_{pos}$, $p_{po}$)
14:        $P_{cu} \leftarrow [[r_{pos}, p_{po}]]$
15:        $P_{OBfree} \leftarrow$ ComputeObstacleFreePath($P_{cu}$, $O_n$)
16:        $P_t[i] \mathrel{+}= P_{OBfree}$
17:        $p_{int} \leftarrow$ CheckIntermediatePoints($p_r$)
18:        $P_t[i]$.append$(p_{int})$
19:        $p_c[i]$.append(len$(P_t)$)
20:        $p_r \leftarrow p_{cl} - P_t$
21:        $r_{pos} \leftarrow P_{OBfree}[-1]$
22:    end while
23: end for
24: $O_{bi} \leftarrow$ argmax$(p_c[:])$
25: $P_{cs} \leftarrow P_t[O_{bi}]$
26: $r_{pos} \leftarrow P_{cs}[-1]$
27: $p_{tu} \leftarrow$ PotentialPointToTurnCCW($p_r$, $R_{\text{conc}}$, $r_{pos}$)
28: $O_b \leftarrow$ VectorsTowardsNextCCWPoint($p_{tu}$, $p_r$)
29: $P_{cu} \leftarrow [[r_{pos}, p_{tu}]]$
30: $P_{OBfree} \leftarrow$ ComputeObstacleFreePath($P_{cu}$, $O_b$)
31: $P_{cs} \mathrel{+}= P_{OBfree}$
32: $p_r$.remove$([P_{OBfree}])$
33: return $P_{cs}$, $p_r$
```

Algorithm 17 SetupObstacleAlgorithm

```
    Input: Polygonal obstacles, grid diameter, vision cone, safe margin
    Output: Extended polygonal obstacles, occupancy grids, buffer points

1: $distance\_x, distance\_y \leftarrow ComputeDistancesFor90DegreeTurn(vision\_cone)$
2: $free\_grid\_distance \leftarrow 1.5 \times distance\_y$
3: $extended\_obstacles \leftarrow ExtendPolygonForSafeMargin(polygonal\_obstacles, safe\_margin)$
4: $grids \leftarrow ComputeGrid(extended\_obstacles, grid\_diameter, free\_grid\_distance)$
5: $buffer\_points \leftarrow PointsSurroundingObstacles(extended\_obstacles)$
6: return $extended\_obstacles, grids, buffer\_points$
```

Description of the notations:

- dx: Distance in the x direction at 90degree turn.
- dy: Distance in the y direction at 90degree turn.
- fd: Free grid space from obstacle edge.

Algorithm 18 SetupObstacleAlgorithm

```
Require: polygonal obstacles ($OB_p$), grid diameter ($g_d$), vision cone ($VC$), safe margin ($sm$)
Ensure: 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 $OB_e$, $g_s$, $p_{bu}$
```

Algorithm 19 ComputeObstacleFreePath

Input: Extended obstacles, grids, buffer points, complete path, current path, current orientation
Output: Obstacle-free path

```
1: $robot\_pose \leftarrow curr\_path[-1]$
2: $is\_path\_in\_obstacle, obs\_idx \leftarrow CheckPath(curr\_path, extended\_obstacles)$
3: if is_path_in_obstacle then
4:     $obstacle \leftarrow extended\_obstacles[obs\_idx]$
5:     $grid \leftarrow grids[obs\_idx]$
6:     $salient\_points \leftarrow ExtractSalientPoints(obstacle, grid, buffer\_points, robot\_pose)$
7:     $G.initialize()$
8:     $step\_length \leftarrow AutoSelectStepLength(robot\_pose, salient\_points)$
9:     $G, goal\_salient\_point, is\_goal\_found \leftarrow GenerateGraph(G, robot\_pose, salient\_points,$
                                                  $step\_length, max\_generation)$
10:    if is_goal_found then
11:        $shortest\_path \leftarrow AStarSearch(G, robot\_pose, goal\_salient\_point)$
12:        $obstacle\_free\_path \leftarrow shortest\_path$
13:        return $obstacle\_free\_path$
14:    else
15:        $robot\_pose \leftarrow complete\_path[-1]$              ▷ move one step back in the path
16:        goto GenerateGraph
17:    end if
18: else
19:    return $[[curr\_path[-1], curr\_orientation]]$
20: end if
```

Description of the notations:

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

Algorithm 20 ComputeObstacleFreePath

```
Require: Extended obstacles ($OB_e$), grids ($g_s$), buffer points ($p_{bu}$), complete path ($P_c$), current
    path ($P_{cu}$), current orientation ($O_{cu}$)
Ensure: 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
```

Algorithm 21 PathAroundObstaclesAlgorithm

```
    Input: Extended obstacles, 2D points, robot pose, turning radius
    Output: Path

 1: $points\_list, robot\_obs\_idx \leftarrow PointsCloseToObstacle(extended\_obstacles, 2D\_points)$
 2: $curr\_points \leftarrow points\_list[robot\_obs\_idx]$        ▷ Robot close to an obstacle will be the first.
 3: $curr\_obs \leftarrow extended\_obstacles[robot\_obs\_idx]$
 4: $Path \leftarrow []$
 5: for $i$ in range(len(extended_obstacles)) do
 6:     $G.initialize()$
 7:     $G \leftarrow GenerateVisibilityGraph(curr\_obs, curr\_points, robot\_pose)$
 8:     $path\_through\_all\_points \leftarrow CoverageGraphSearch(G, robot\_pose)$
 9:     $Path += path\_through\_all\_points$
10:     $robot\_pose \leftarrow path\_through\_all\_points[-1]$
11:     $curr\_obs, obs\_idx \leftarrow FindNearestObstacle(robot\_pose, points\_list)$
12:     $curr\_points \leftarrow points\_list[obs\_idx]$
13: end for
14: return $Path$
```

Description of the notations:

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

Algorithm 22 PathAroundObstaclesAlgorithm

```
Require: Extended obstacles ($OB_e$), 2D points ($P_{2d}$), robot pose ($r_{pos}$), turning radius ($R_{tu}$)
Ensure: Path ($P$)
 1: $p_{list}, robot\_obs\_idx \leftarrow$ PointsCloseToObstacle($OB_e$, $P_{2d}$)
 2: $p_{cu} \leftarrow p_{list}[robot\_obs\_idx]$        ▷ Robot close to an obstacle will be the first.
 3: $OB_{cu} \leftarrow OB_e[robot\_obs\_idx]$
 4: $P \leftarrow []$
 5: for $i$ in range(len($OB_e$)) do
 6:     $G$.initialize()
 7:     $G \leftarrow$ GenerateVisibilityGraph($OB_{cu}$, $p_{cu}$, $r_{pos}$)
 8:     $P_{cov} \leftarrow$ CoverageGraphSearch($G$, $r_{pos}$)
 9:     $P += P_{cov}$
10:     $r_{pos} \leftarrow P_{cov}[-1]$
11:     $OB_{cu}, obs\_idx \leftarrow$ FindNearestObstacle($r_{pos}$, $p_{list}$)
12:     $p_{cu} \leftarrow p_{list}[obs\_idx]$
13: end for
14: return $P$
```

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 intermediate potential points. For each of these intermediate 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 intermediate point based on the distance from the robot and the distribution of points in the long run. Both the distance and the distribution are normalized before calculating the combined score. The 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. At this point, the number of points covered in this orientation is recorded.

Returning to Initial Position and Orientation

After completing the initial coverage path, 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 robot's operation.

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.

Coverage Efficiency and Convergence Rate

Although this behavior can cover most or all points in the data, the coverage rate diminishes, and convergence takes longer. This limitation highlights the need for a hierarchical approach with multiple behaviors. By transitioning between behaviors, the algorithm can maintain high efficiency and improve the convergence rate, ensuring comprehensive and timely coverage of all points.

Second behavior: Intermediate behavior:

Boundary-Focused Coverage Strategy

The second behavior of the algorithm 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 this behavior lies in the method for selecting 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. Within this region, the algorithm evaluates all potential points, applying refined selection criteria. The criteria prioritize points with the smallest absolute orientation difference from the robot's current orientation and the shortest distance from the robot's current position. 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.

By following a systematic circular pattern, the robot can cover most of the points in the data set. This transition from centroid-focused coverage to boundary-focused coverage allows the algorithm to address the limitations of the first behavior, which tends to become less efficient as more points near the centroid are covered. The hierarchical combination of these behaviors ensures that the robot effectively covers the entire field, optimizing both the coverage rate and the overall efficiency of the operation.

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. Regardless of the distance between the two primary points on the path, these intermediate points are included in the coverage plan.

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 two primary 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 final stages of the coverage process by allowing slight deviations from straight paths to cover more points efficiently, thus reducing the overall path length and energy consumption.

Final Behavior: Completing Coverage:

Final Behavior: Completing Coverage The third and final behavior of the algorithm addresses the coverage of the few 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), leveraging the concepts described in a seminal paper by X.

Overview of the Traveling Salesman Problem (TSP) The traveling salesman problem is a classic optimization challenge in computer science and operations research. It seeks to find the shortest route that visits a set of cities and returns to the original city, with the primary challenge being to determine the optimal sequence of cities to minimize total travel distance. This problem is NP-hard, meaning there is no known polynomial-time algorithm that can solve it exactly for large instances.

Open Traveling Salesman Problem (Open TSP) The open TSP is a variant of the classic TSP where the salesman does not need to return to the starting city after visiting all other cities. This relaxation allows for more flexibility in route planning and can lead to different optimal solutions. The open TSP is particularly useful in scenarios where a return trip is unnecessary or infeasible, such as delivery routes or exploration missions. In our case, the robot does not need to return to the starting point, making the open TSP more suitable for our problem. Nevertheless, the non-holonomic constraints of the robot must be considered.

Dubins Open Traveling Salesman Problem (Dubins Open TSP) The Dubins open TSP 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. The Dubins path, named after mathematician L. F. Dubins, provides the shortest path for vehicles with a fixed turning radius. By applying Dubins paths to the open TSP, the algorithm can generate efficient and feasible routes for vehicles with non-holonomic constraints.

Application to the Final Coverage The final behavior leverages the Dubins open TSP to efficiently cover the remaining points. 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.

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.

Hierarchical Behavioral Algorithm for Coverage Path Planning These three behaviors are meticulously designed to optimize the coverage of points in an area while minimizing the path length and energy consumption of the robot. The algorithm demonstrates a sophisticated approach to coverage path planning, particularly suited to agricultural fields, but adaptable to other domains as well.

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.

Computing the Optimal Coverage 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. This adaptive approach significantly improves the algorithm's performance, as demonstrated in the subsequent results.

3..6 Obstacle Avoidance

Obstacle Avoidance in Coverage Path Planning 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. This discussion focuses on the avoidance of static obstacles within the scope of coverage path planning.

The CPP algorithm considers several inputs and constraints. The robot operates with non-holonomic constraints, meaning it cannot move directly sideways and must follow a path that considers its motion limitations. The total area to be covered is defined, ensuring the robot stays within the designated field. Obstacles are represented as polygons of various shapes and sizes, which the robot must avoid. Additionally, the vision cone defines the area that the robot can see and use for straight-path approximation, aiding in real-time obstacle detection and avoidance.

Prior to executing the algorithm, comprehensive data about the field is collected using a drone. The drone flies over the field, capturing images to generate a detailed map. This map is instrumental in defining the grid for path planning and identifying and mapping out obstacles accurately. By having all the necessary data beforehand, the CPP algorithm can be initialized with a complete overview of the field, including obstacles.

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. By focusing on these efficiency improvements.

Obstacle Avoidance in Coverage Path Planning The obstacle avoidance algorithm in coverage path planning (CPP) consists of two main components:

Initial setup once all the data is received. Real-time operation where the robot follows the path generated by the algorithm.

Initial Setup The initial setup begins once the algorithm receives the total area, obstacles represented by their vertices, and the vision cone. 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.

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.

The setup includes two main components:

Occupancy Grid: Represents the grid cells. Grid Centers: Points within the grid cells used to obtain salient points for obstacle avoidance in the world space.

For each obstacle, point data 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.

Dynamic Obstacle Avoidance in Coverage Path Planning 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 necessary grids, the algorithm proceeds to follow a regular behavioral approach for path planning.

The path generation process begins as described previously. However, this time, the algorithm includes a mechanism to detect 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.

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.

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.

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.

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.

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.

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.